Electrical Engineering and Computer Science Dept
College of Engineering

MODEL BASED TESTING

Prepared by Prof. Christopher S. Cerqueira

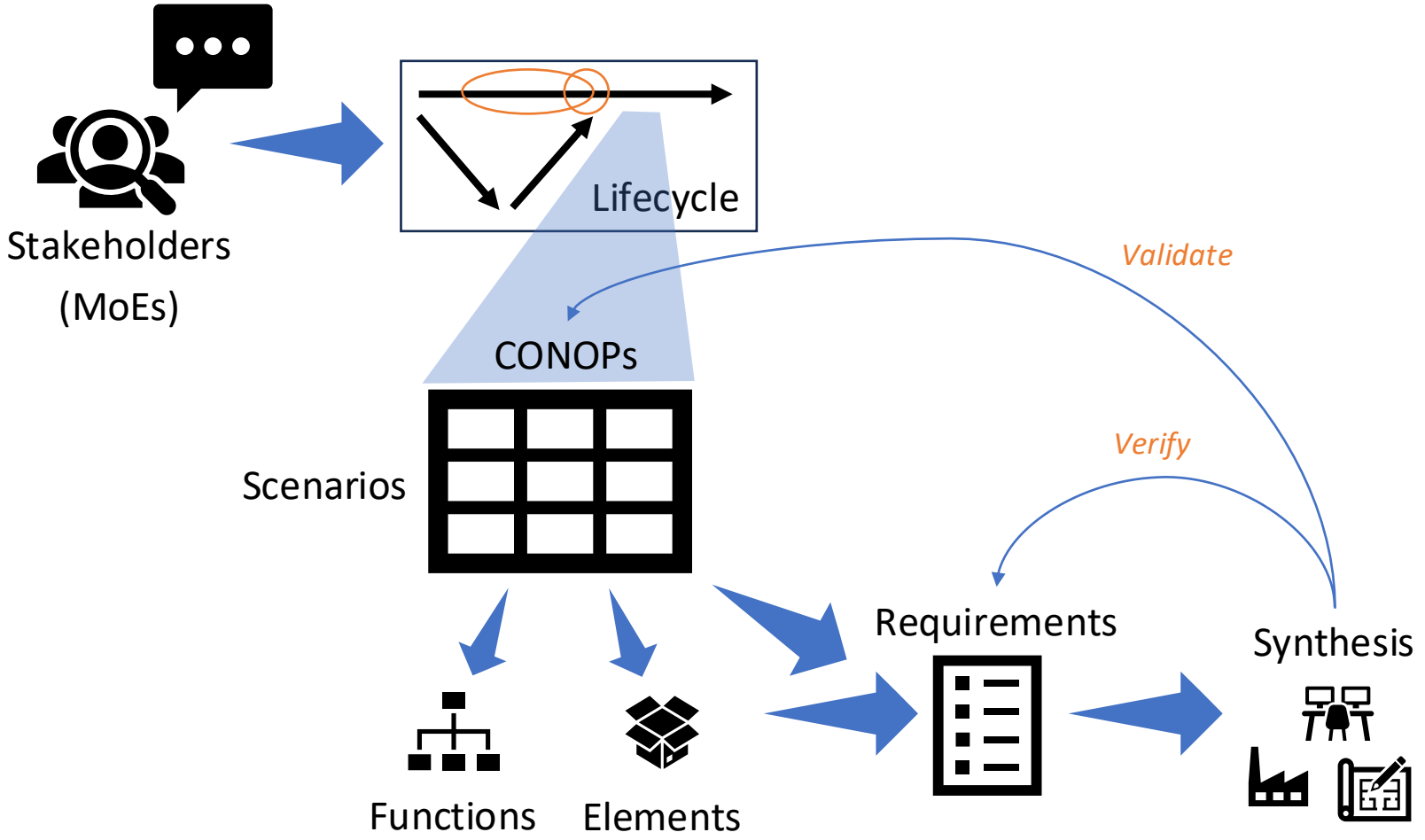
2025

W	Dates	Topic	Ref	Activity
35	26-Aug	Course Introduction Systems Engineering Review Requirement Engineering (Generic Process)	[1][6] [7][9]	Groups & Names
36	2-Sep	Lego exercise 01 - First assessment	[2] [10]	Exercise Report
37	9-Sep	Definition and Structure of Requirements Functional and non-functional Requirements	[8][6] [10]	Exercise List
38	16-Sep	Elicitation Techniques (Interview / Rich Picture / use Cases)		Group Project Delivery
39	23-Sep	Rules for need and requirement statements	[2][3]	Exercise List
40	30-Sep	Attributes of need and requirement statements Boilerplates/patterns	[2]	Exercise List
41	7-Oct	[Carrer Explo]		* Reading activity
42 *	14-Oct	Midterm Session		Self-paced concept questions
		[Fall break]		

W	Dates	Topic	Ref	Activity
43	21-Oct	V&V // DVM // Success Criteria	[4]	Group Project Delivery
44	28-Oct	MBSE	[1] **	Group Project Delivery
45	4-Nov	Model Based Testing		Group Project Delivery
46	11-Nov	[Veterans Day]		* reading activity
47	18-Nov	Lego Exercise 02		Exercise Report
48	25-Nov	System Requirement Review Presentation (G1-G4)		
		[Thanksgiving]		
48	2-Dec	System Requirement Review Presentation (G5-G8)		
49	9-Dec	Finals		Self-paced concept questions and a small

v2

We know the basic SE Framework





https://radio.wosu.org/post/ai-more-aieeee-first-time-robot-can-feel-pain#stream/0



Context of reactive systems

Reactive system definition

- A reactive system is a system that, **when switched on, is able to create desired effects in its environment by enabling, enforcing, or preventing events in the environment.**
- Properties:
  - **Continuous interaction** (nonterminating)
  - System will respond to **external stimuli**, and
  - the **response depends of its current state**

Transformacional systems

- Reactive sytems can be constracted with transformal systems, which exist to transform na input into an output.

- Properties:
  - **Only to transform** na input to the output.
  - After transformation it terminates
  - **Has no states** that is relevant for external entities

REACTIVE SYSTEMS EXAMPLES

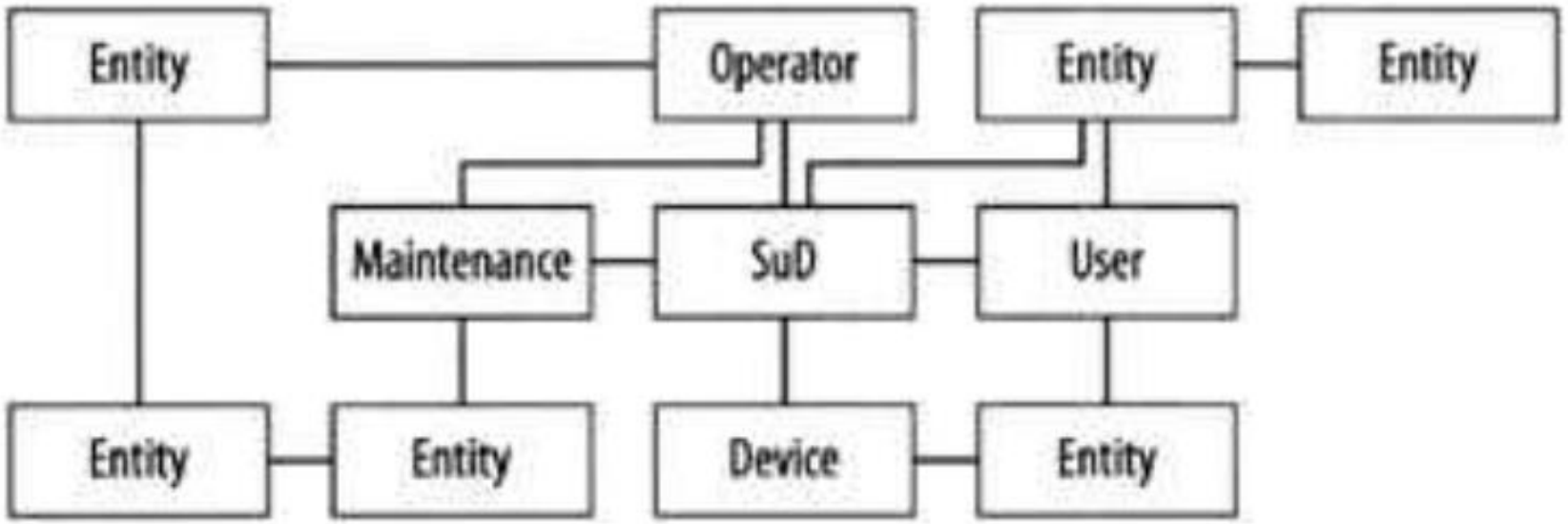
- **REAL TIME SYSTEMS** – the answer depends on the time. Ex.: control software, elevator control, ATMs
- **SAFETY-CRITICAL SYSTEMS** – malfunctional behaviors can lead to lost of lives. Ex.: bio-physical systems, onboard computers (car/aircraft/spacecraft/ship)
- **EMBEDDED SYSTEMS** – implementation restrictions. Ex.: mobile, onboard software-firmware, IoT.

- Systems that manage critical infrastructures: air management, train, nuclear reactors, so on.

THE ENVIRONMENT

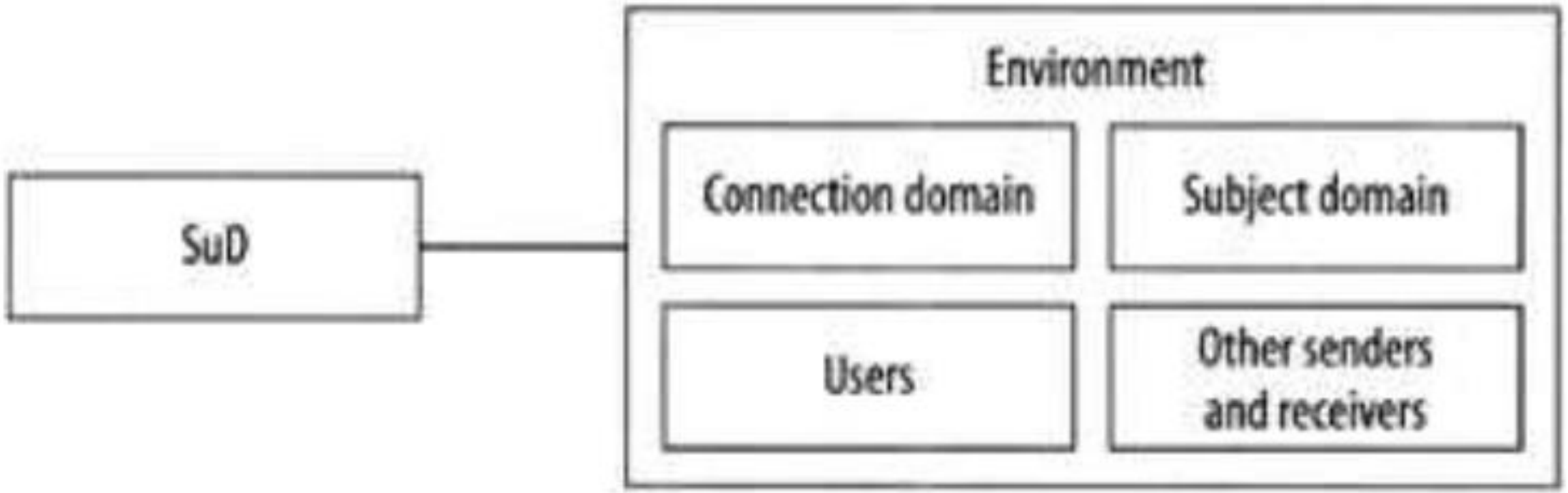
Environment

- We need to identify the part of the world that is relevant as **system environment. – Identify the SuD interfaces**
- **The system interfaces might change according the objective or test phase.**



Environment

- The environment of a system contains various **subjects and connected domains**.



Subject domain

- DOMAIN – knowledge domain that defines the **dimensions of the environment** (attitude control domain: reaction wheel, magnetometer, solar sensor, etc.)
  - Domain entities: reaction wheel, magnetometer, etc..
  - Events: torquer application, measurement reading

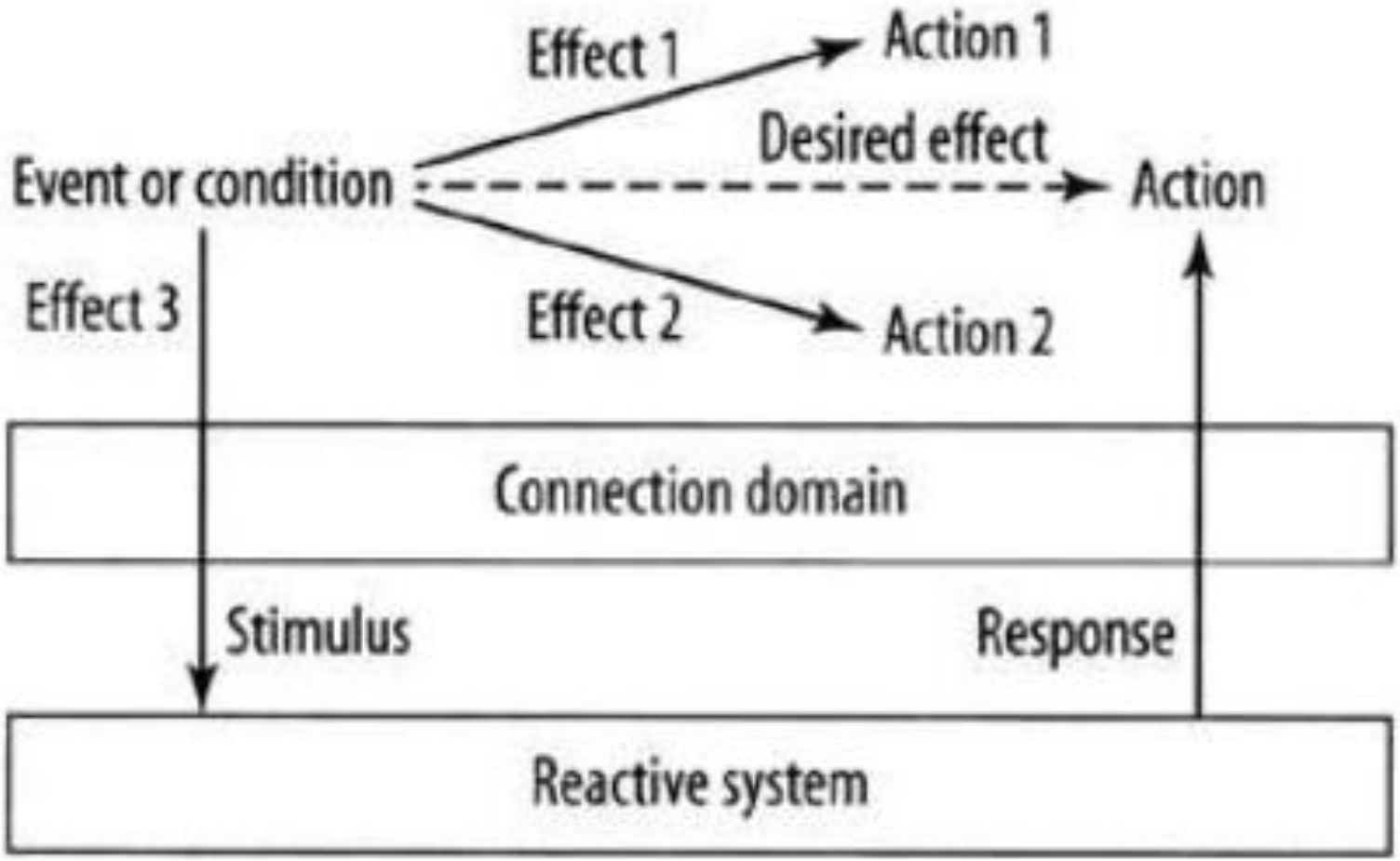
Connection domain

- Communication channels that **allows the exchange of messages** among the SuT and the Environment.
- We need to model the comm channel because it:
  - Can lead to delays, corrupt data, so on – a lot of error happens in the interfaces.
  - Identify the inbound/outbound flows.

STIMULUS-RESPONSE BEHAVIOR

CAUSE AND EFFECT CHAINS

- The function of a reactive system is to **respond to the occurrence of events or conditions in the environment by causing desirable changes** in the environment.



Events, conditions and actions

- EVENT – something that **happens** in the world.
  - EXTERNAL EVENTS: discrete change in the condition of the environment
  - TEMPORAL EVENTS: passage of a significant time to which the system is expected to respond
- CONDITION – **state of the world** that persists for some nonzero period of time.
- ACTIONS – events from the point of view of the initiator

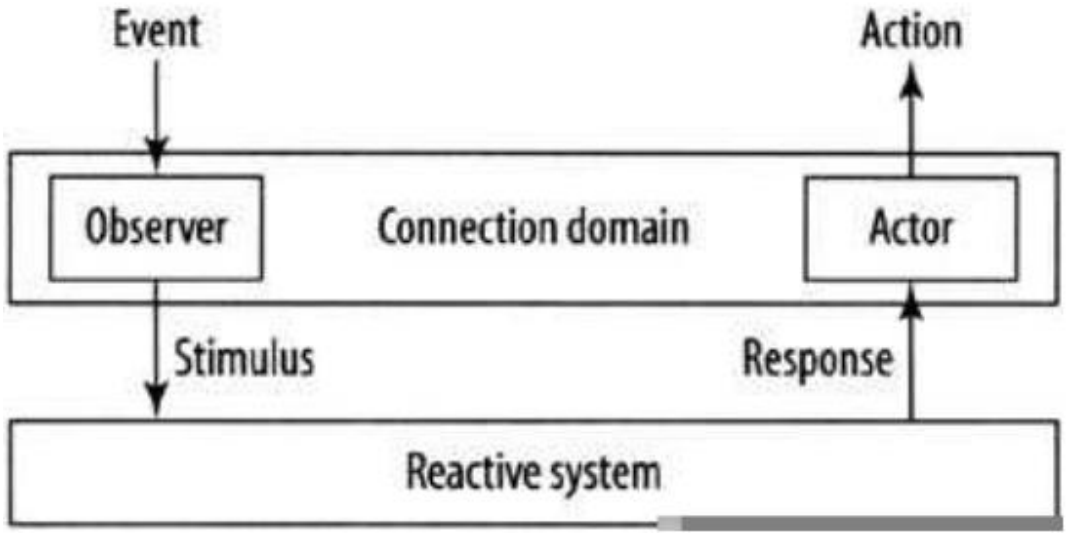
stimuli

- The stimulus of a system is an **event at the interface of the system caused by the environment**.
- External events occur somewhere in the environment; **stimuli occur at the interface of the system**

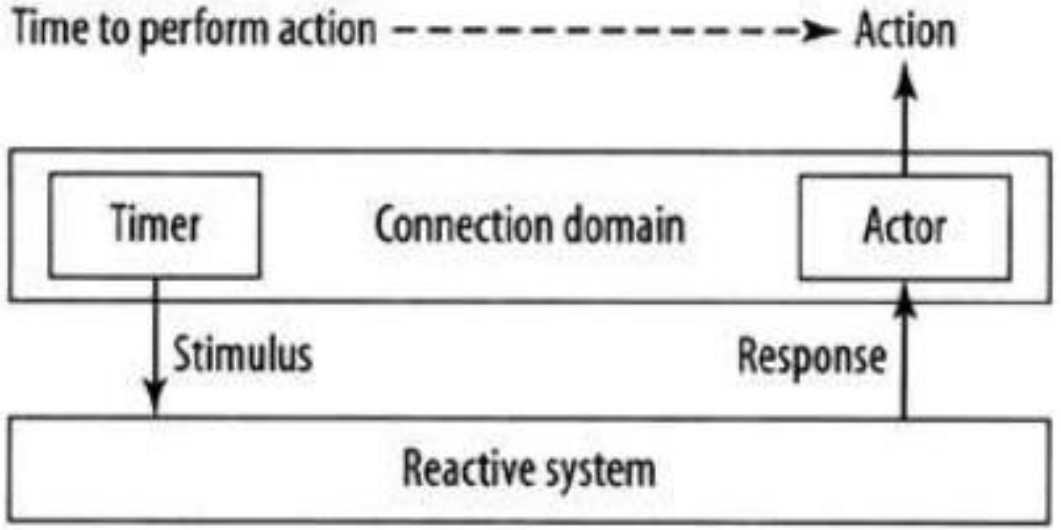
response

- The reactive system **responds to an external** or temporal event by updating its state or producing an output.
- An output reponse of a system is an **event at the interface** of the system, caused by the system.
- The desired effect of the stimulus may consist of **one or more desired actions**, to be caused by several responses.

summary



Events-Stimuli



Temporal Events

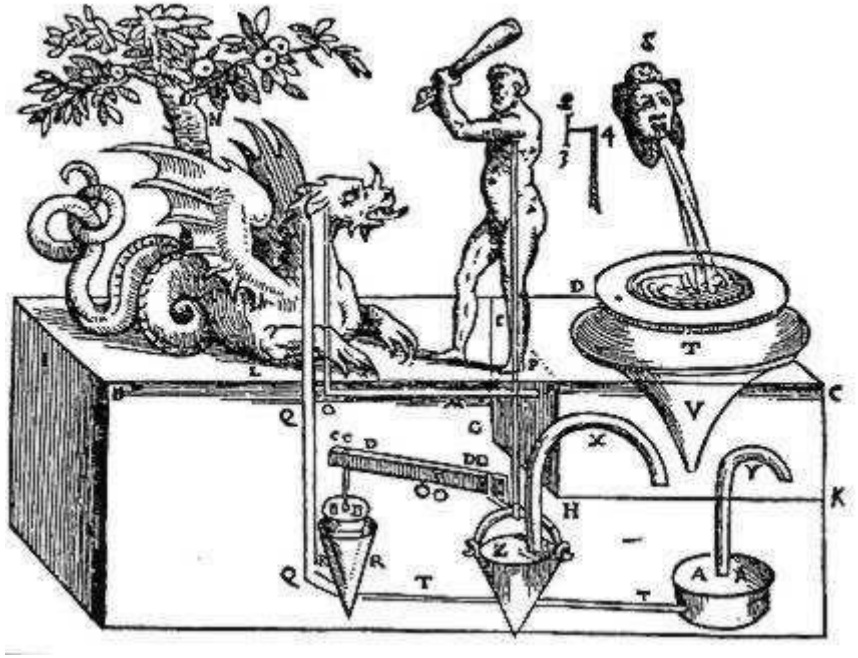


Concepts of State Machines

AUTOMATON

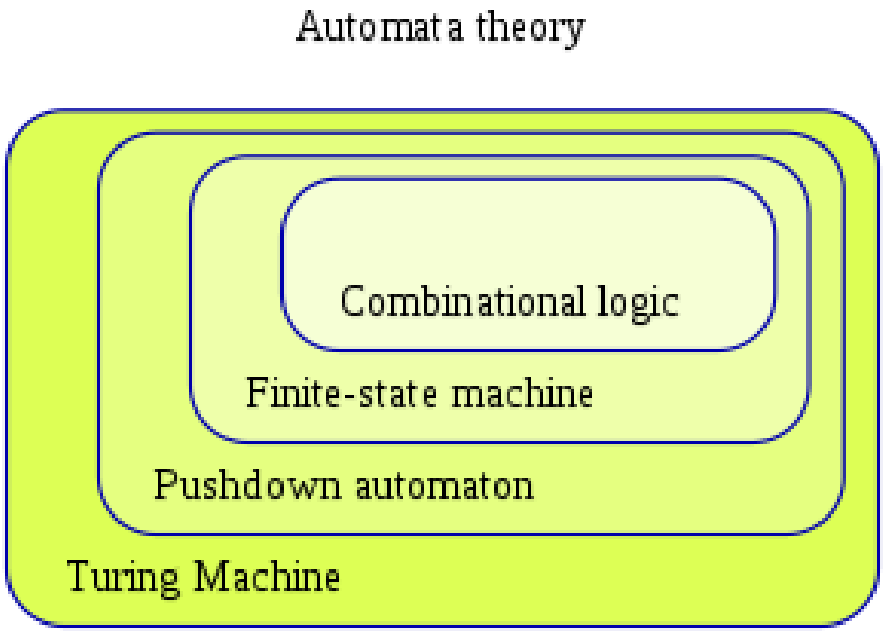
- The word **automaton** itself, closely related to the word "automation", denotes automatic processes carrying out the production of specific processes. Simply stated, automata theory deals with the logic of computation with respect to simple machines, referred to as **automata**.
- **Automatons** are abstract models of machines that perform computations on an input by moving through a series of states or configurations. At each state of the computation, a transition function determines the next configuration on the basis of a finite portion of the present configuration.

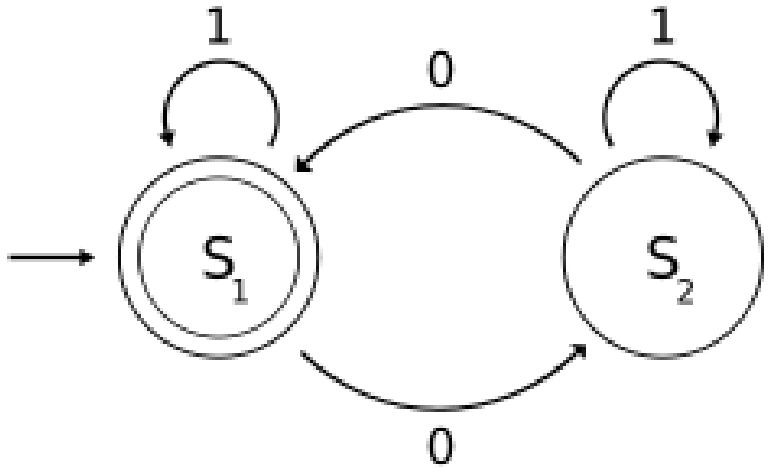




Automata theory

- A deterministic finite automaton is represented formally by a 5-tuple <Q,Σ,δ,q0,F>, where:
  - **Q** is a finite set of states.
  - **Σ** is a finite set of symbols, called the alphabet of the automaton.
  - **δ** is the transition function, that is, δ: Q × Σ → Q.
  - **q0** is the start state, that is, the state of the automaton before any input has been processed, where q0∈ Q.
  - **F** is a set of states of Q (i.e. F⊆Q) called accept states.

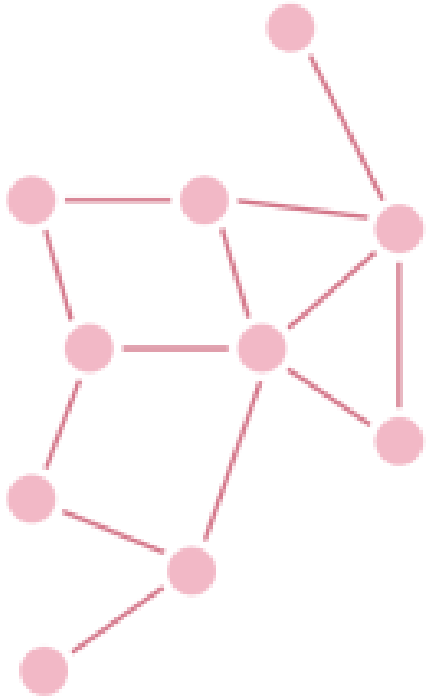




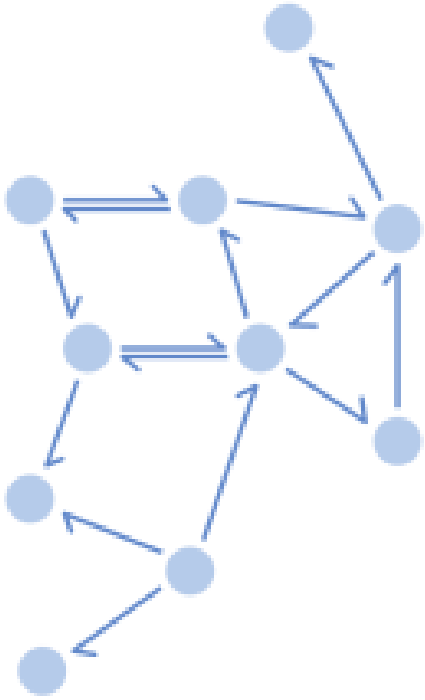
https://en.wikipedia.org/wiki/Automata_theory

Types of graphs

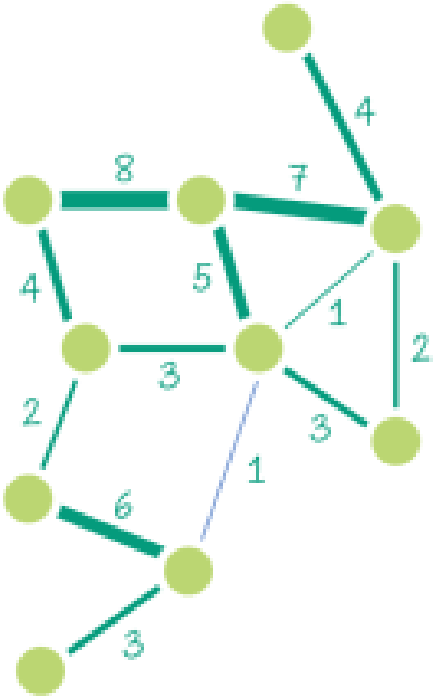
undirected



directed



weighted





State machines

Tool: state machines

- A **finite-state machine** (FSM) or **finite-state automaton** (FSA, plural: automata), finite automaton, or simply a state machine, is a mathematical model of computation.
- **It is an abstract machine that can be in exactly one of a finite number of states at any given time.**
- The FSM can **change from one state to another** in response to some external inputs; the change from one state to another is called a **transition**.
- An FSM is defined by a list of its states, its initial state, and the conditions for each transition.

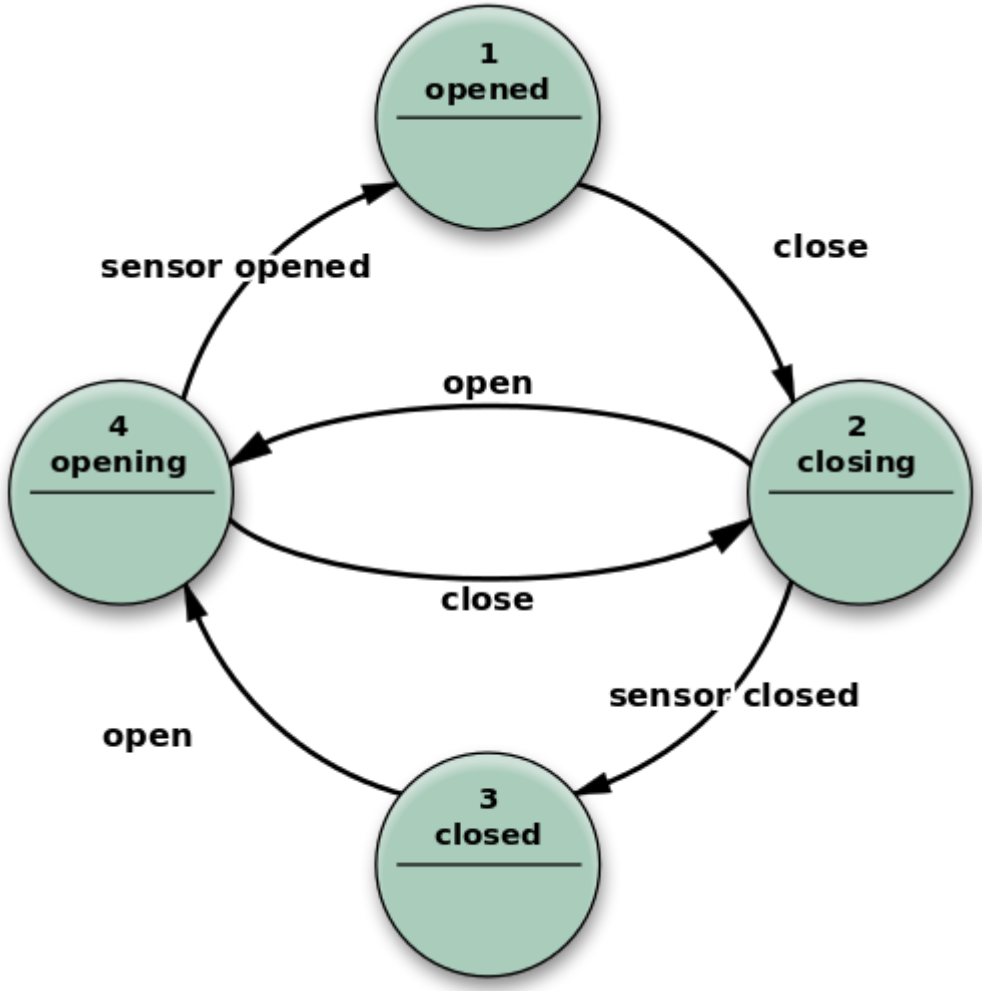
https://en.wikipedia.org/wiki/Finite-state_machine

Main elements of a state machine

- **State** – abstraction that agreggates the needed information of the events
- **Transition** – specifies a source-destination causal event.
- **Event** – input that causes “something” in the SuD
- **Action** – outputs after the event

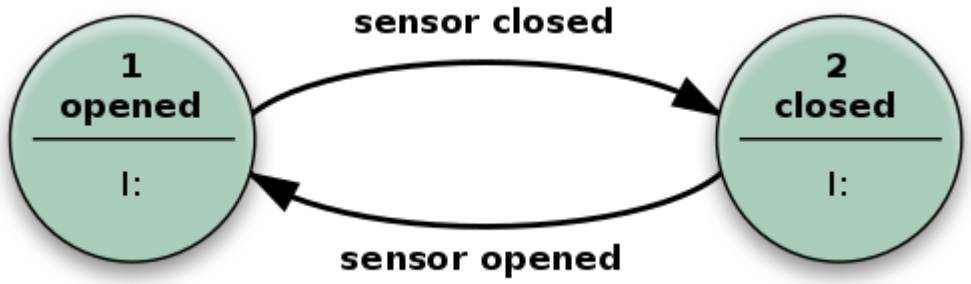
Moore Machine

- The FSM uses only entry actions, i.e., output depends only on the state.
- The advantage of the Moore model is a simplification of the behaviour.
- Consider an elevator door.
  - The state machine recognizes two commands: "command_open“ and "command_close", which trigger state changes.
  - The entry action (E:) in state "Opening" starts a motor opening the door, the entry action in state "Closing" starts a motor in the other direction closing the door.
  - States "Opened" and "Closed" stop the motor when fully opened or closed.
  - They signal to the outside world (e.g., to other state machines) the situation: "door is open" or "door is closed"



Mealy Machine

- The FSM also uses input actions, i.e., output depends on input and state.
- The use of a Mealy FSM leads often to a reduction of the number of states.
- The example in Figure shows a Mealy FSM implementing the same behaviour as in the Moore example.
  - There are two input actions (I:): "start motor to close the door if command_close arrives" and "start motor in the other direction to open the door if command_open arrives".
  - The "opening“ and "closing" intermediate states are not shown.

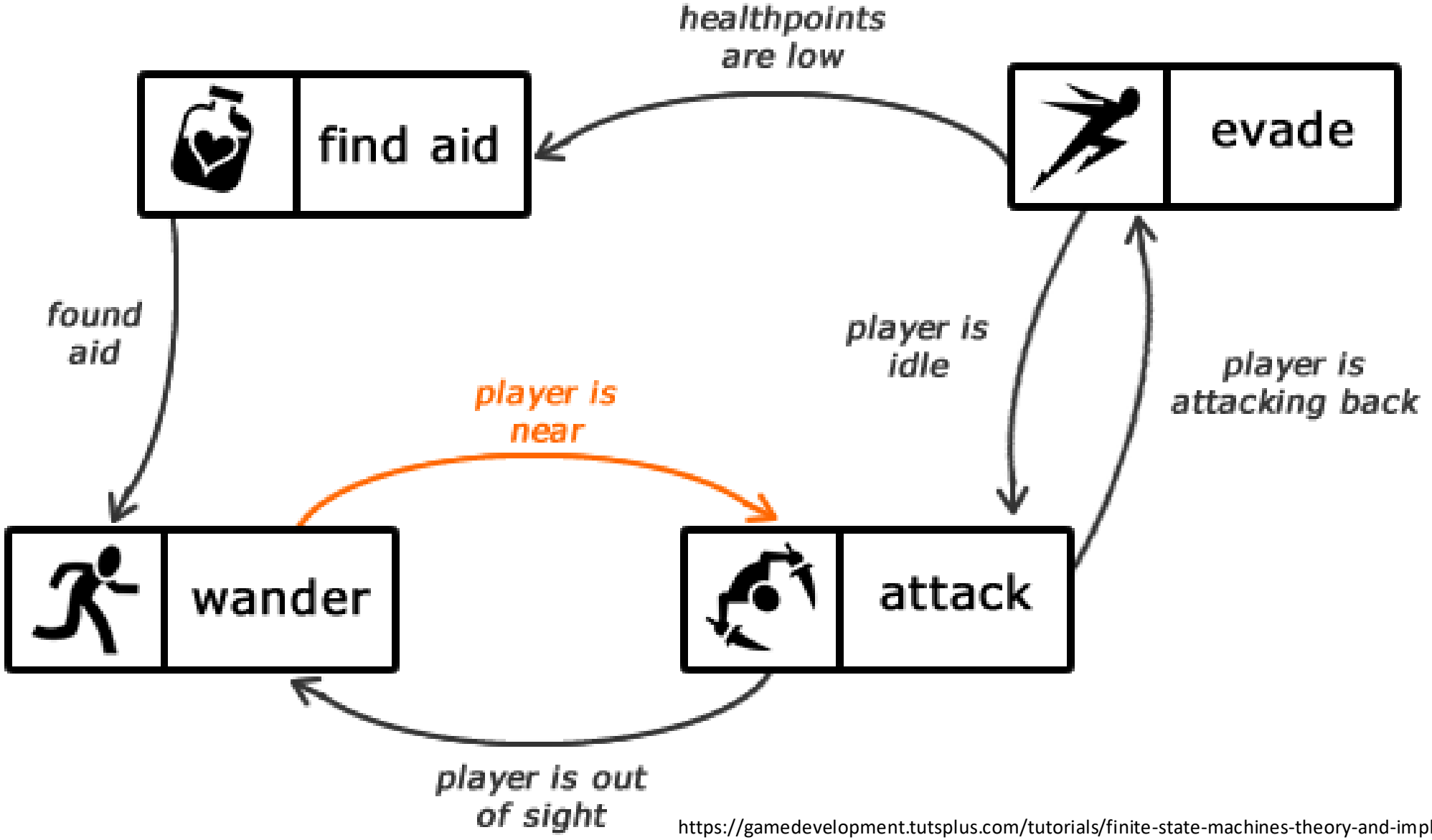


18:03

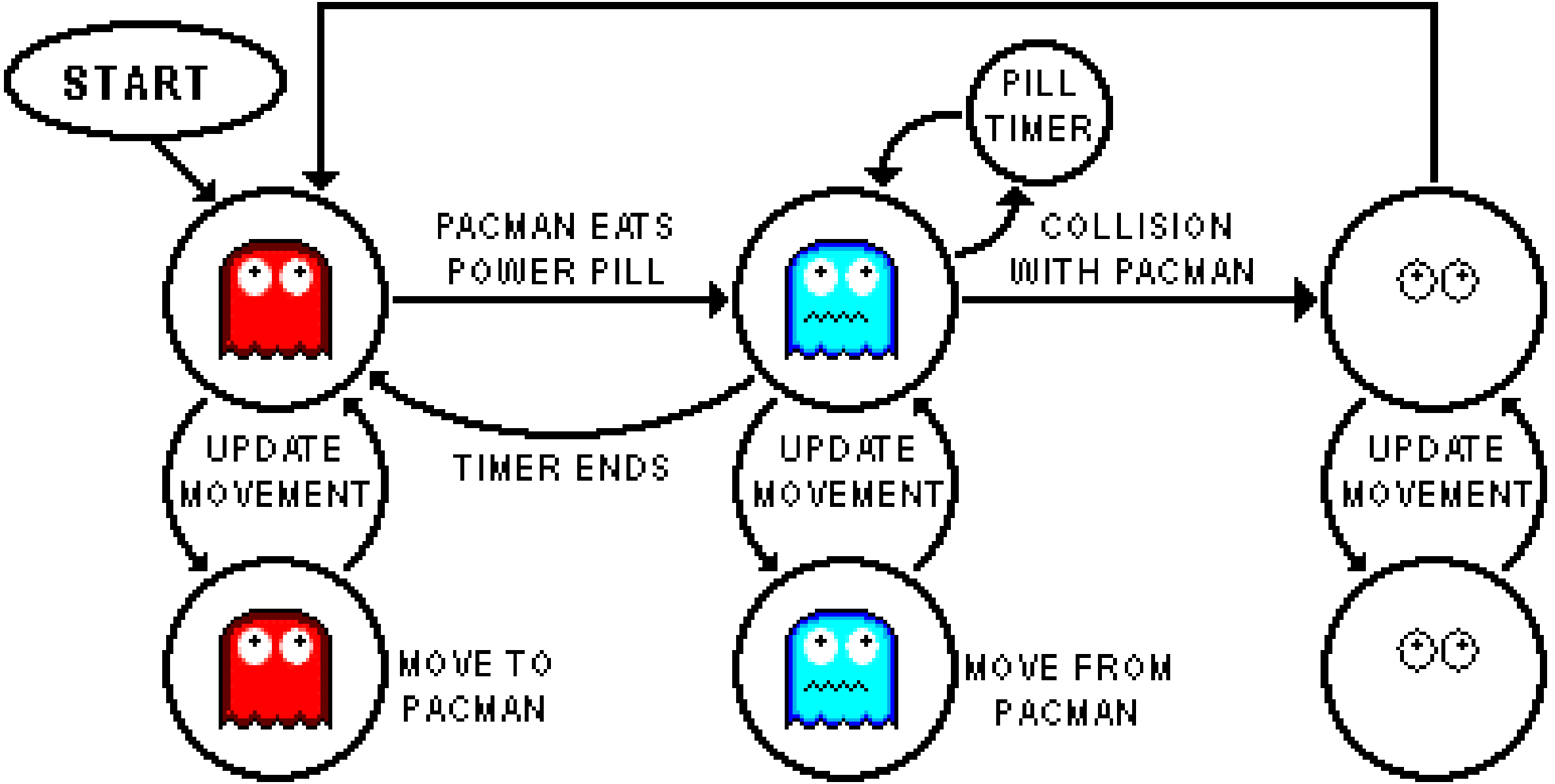
examples

A game example

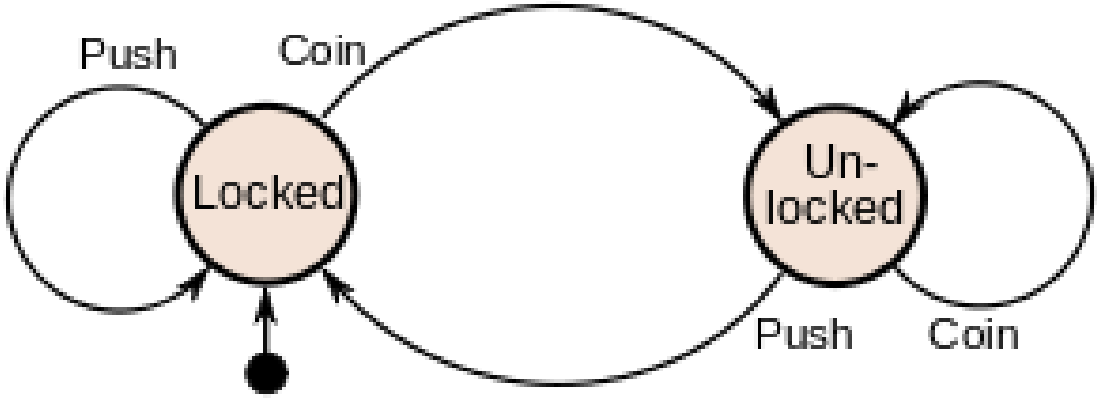
FSM representing the brain of an enemy.



https://gamedevelopment.tutsplus.com/tutorials/finite-state-machines-theory-and-implementation--gamedev-11867


START
PILL TIMER
PACMAN EATS POWER PILL
COLLISION WITH PACMAN
UPDATE MOVEMENT
TIMER ENDS
UPDATE MOVEMENT
UPDATE MOVEMENT
MOVE TO PACMAN
MOVE FROM PACMAN

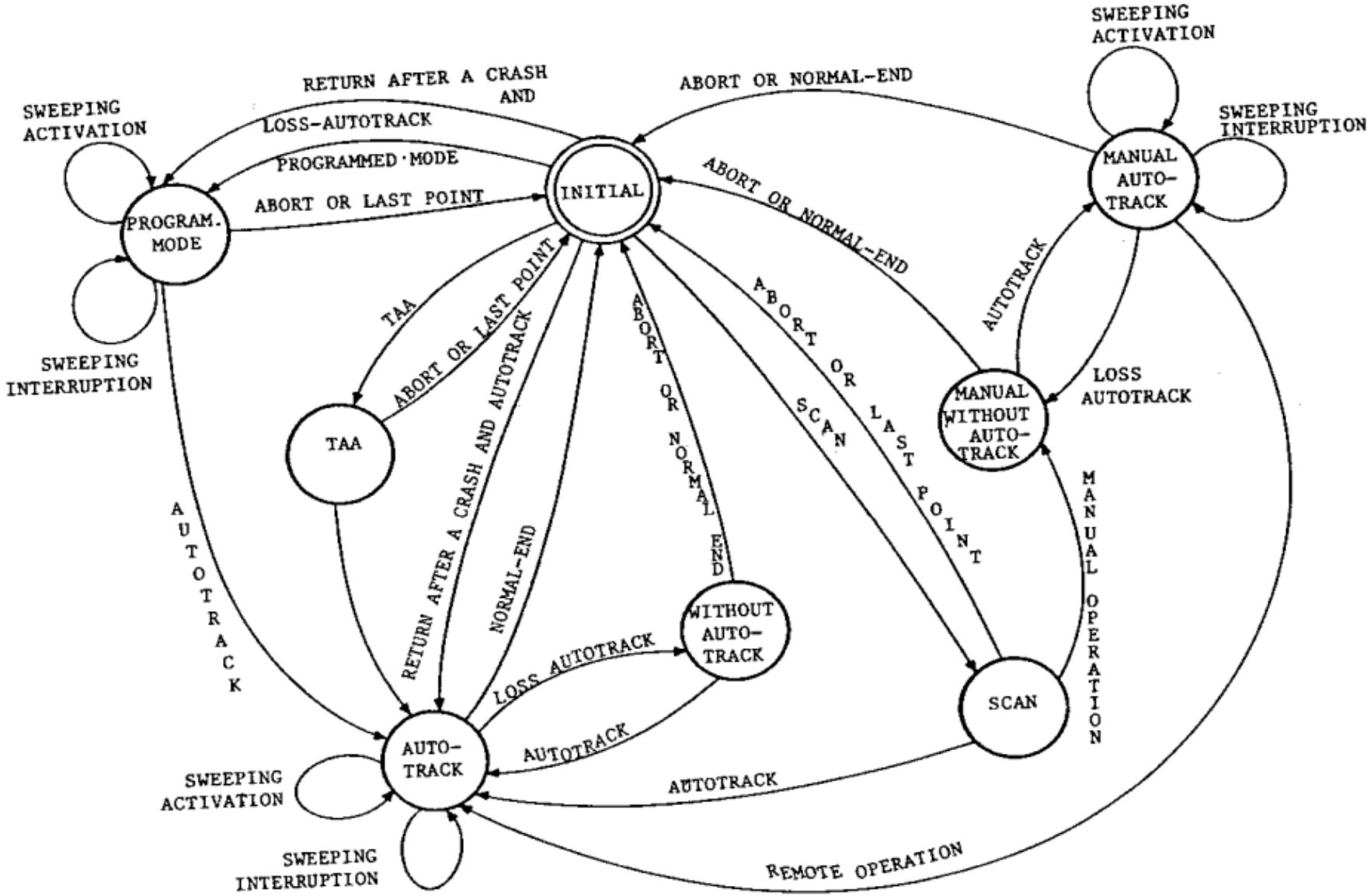
Example



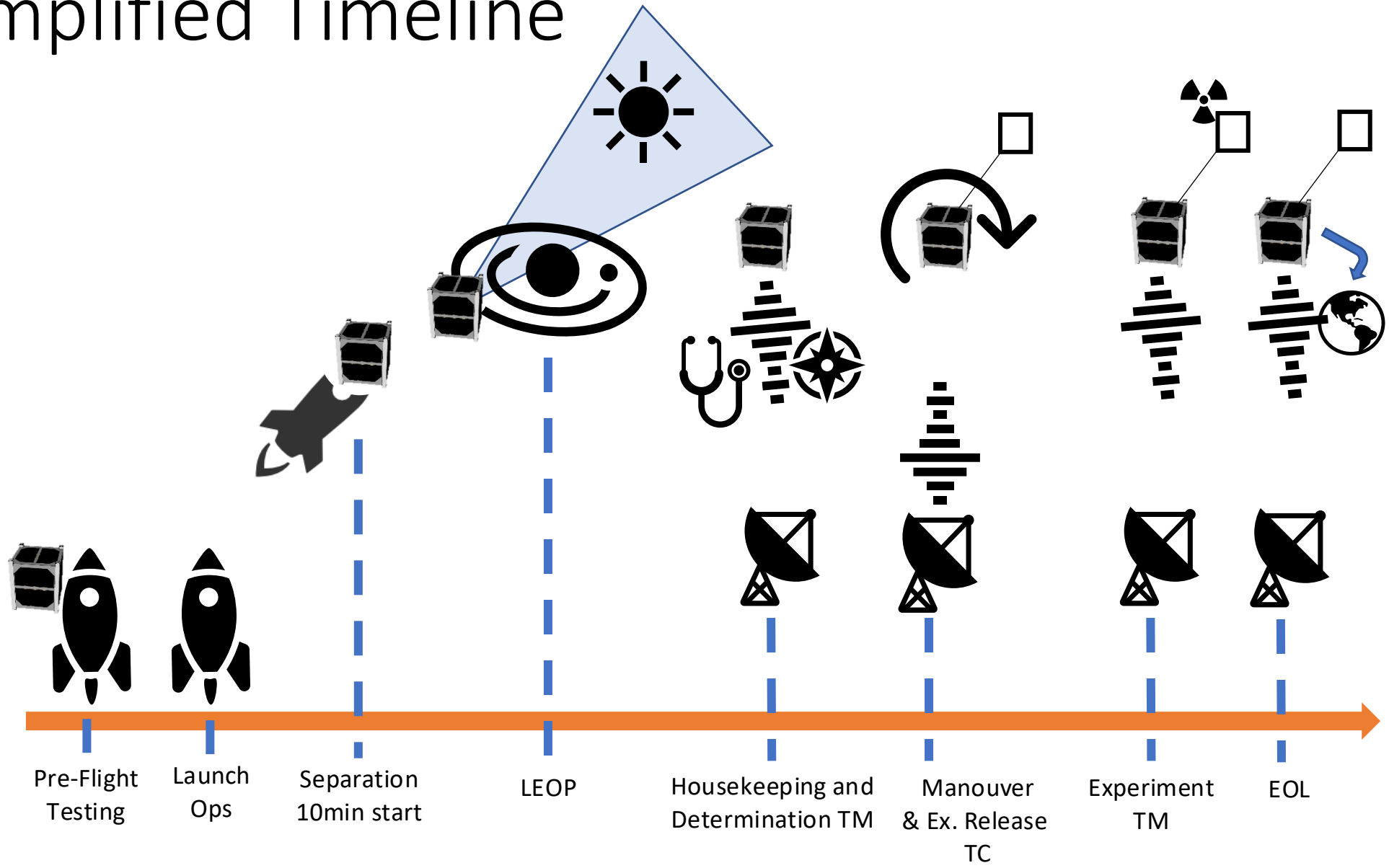
Current State	Input	Next State	Output
Locked	coin	Unlocked	Unlocks the turnstile so that the customer can push through.
	push	Locked	None
Unlocked	coin	Unlocked	None
	push	Locked	When the customer has pushed through, locks the turnstile.



Example: Satellite Groundstation Tracking

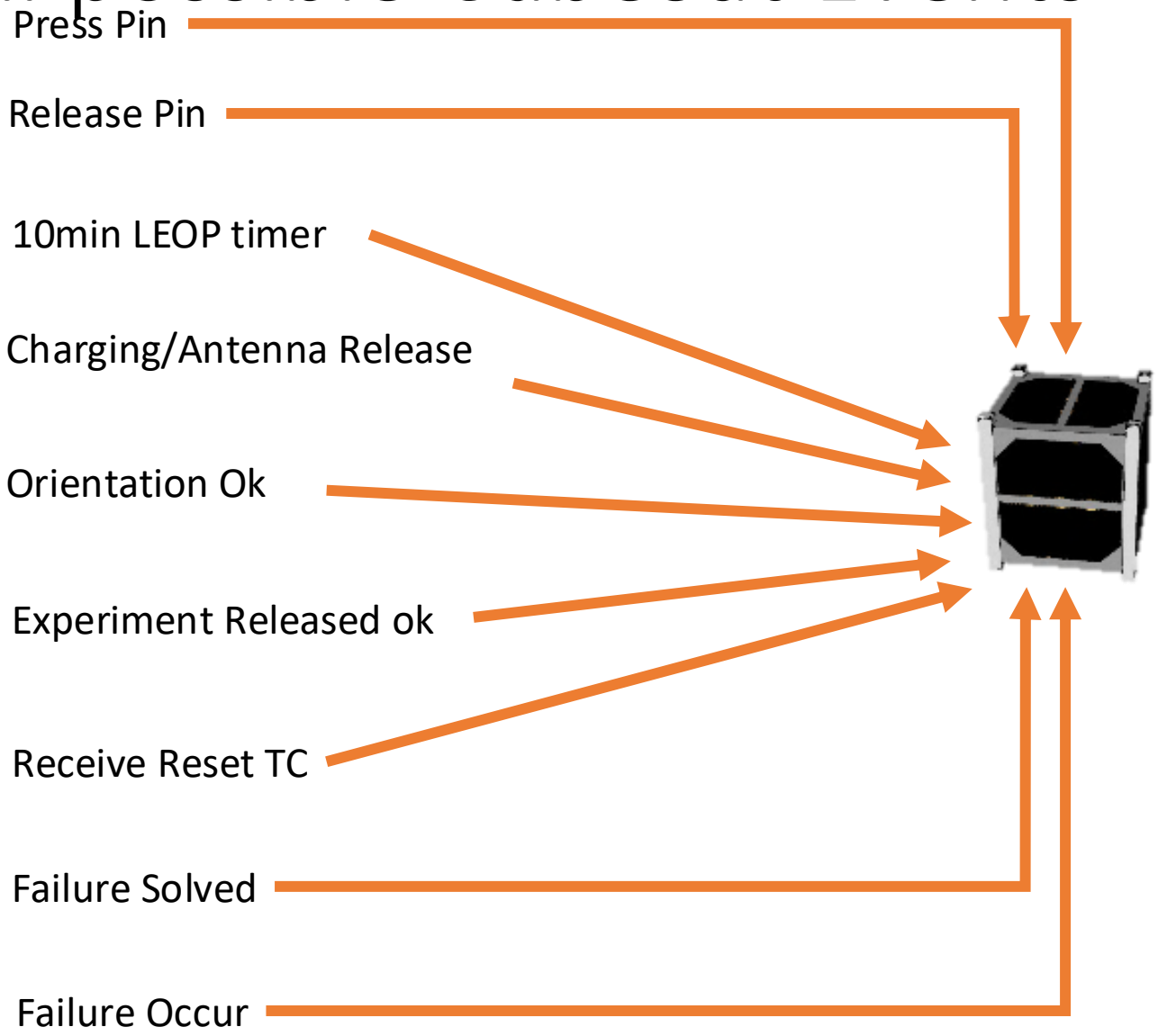


Simplified Timeline



cscerqueira.com.br

List all possible CubeSat Events



cscerqueira.com.br

Select a few events to model

Pin

- ON
- OFF

Software

- OFF
- OK
- FAILURE

Telecommand

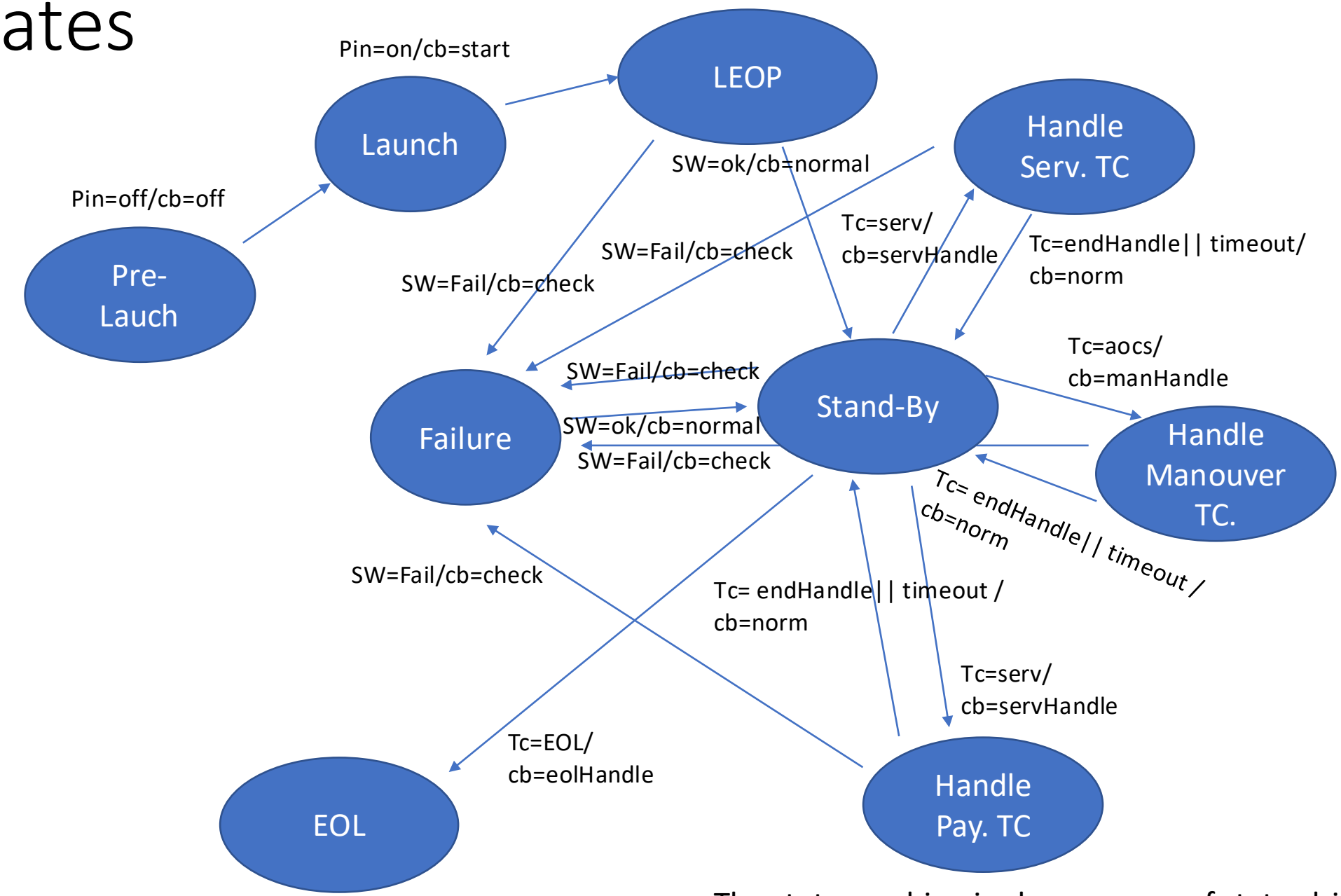
- NONE
- SERVICE
- PAYLOAD
- MANOUVER
- EOL
- END_HANDLE

2*3*6 = 36 states

All are possibles?
All are feasibles?
All should exist?
All are achievables?

cscerqueira.com.br

States



The state machine is also a source of state-driven requirements

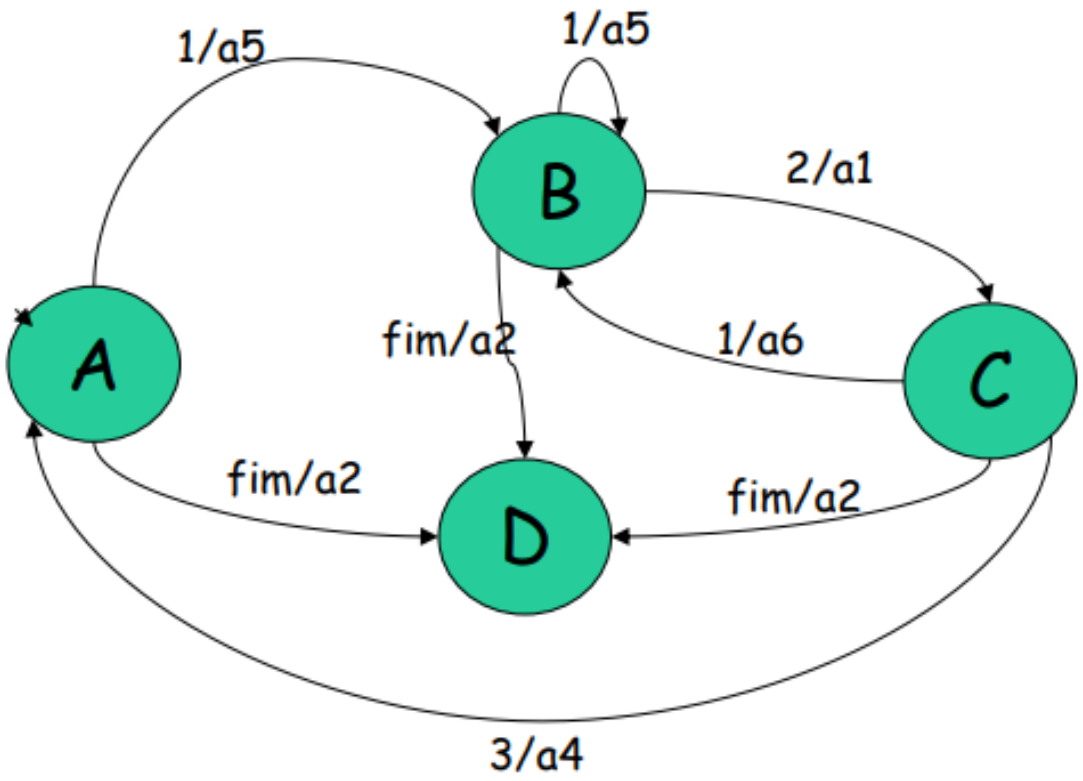
Events table: (it is not a DSM)

	Pre-Flight Testing	Launch Ops	LEOP	Stand-by	Handle Service TC	Handle Payload TC	Handle Manouver TC	EOL	Failure
Pre-Flight Testing	x	Pin = OFF							
Launch Ops		x	Pin = ON						
LEOP			x	SW = OK					SW = Failure
Stand-by				x	TC = serv	TC = pay	TC = man	TC = eol	SW = Failure
Handle Service TC				TC = endHandle || timeout	x				SW = Failure
Handle Payload TC				TC = endHandle || timeout		x			SW = Failure
Handle Manouver TC				TC = endHandle || timeout			x		SW = Failure
EOL								x	
Failure				SW = OK					x

State machine properties

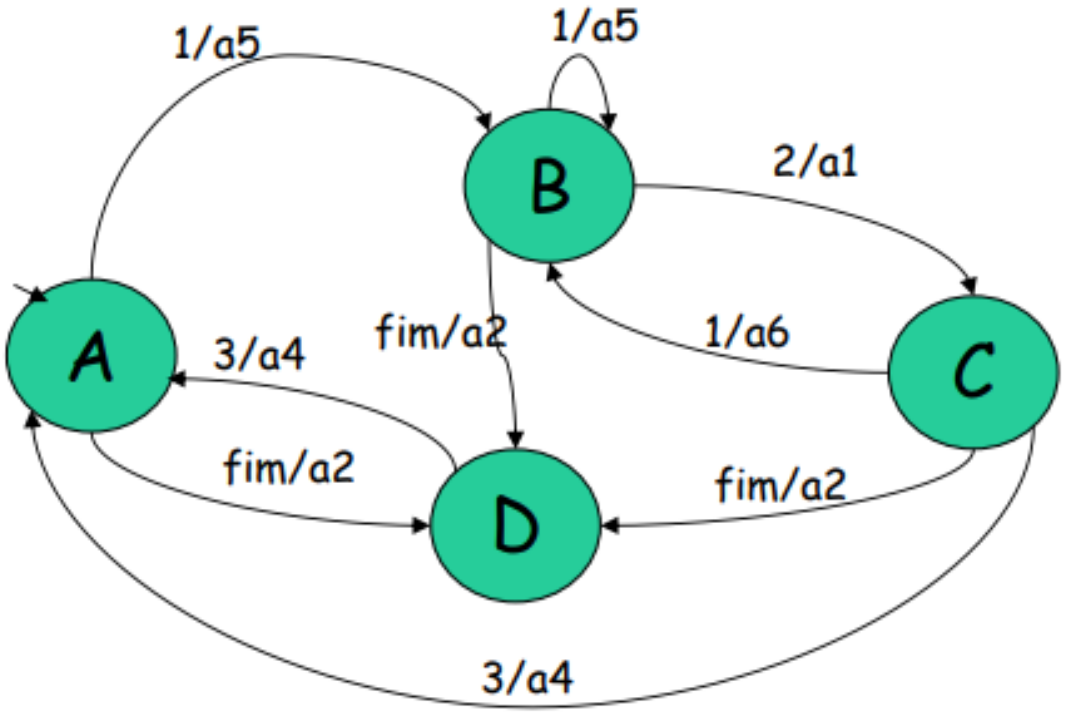
connectivity

- A statemachine is initially connected if **all states are achievable** from the initial state



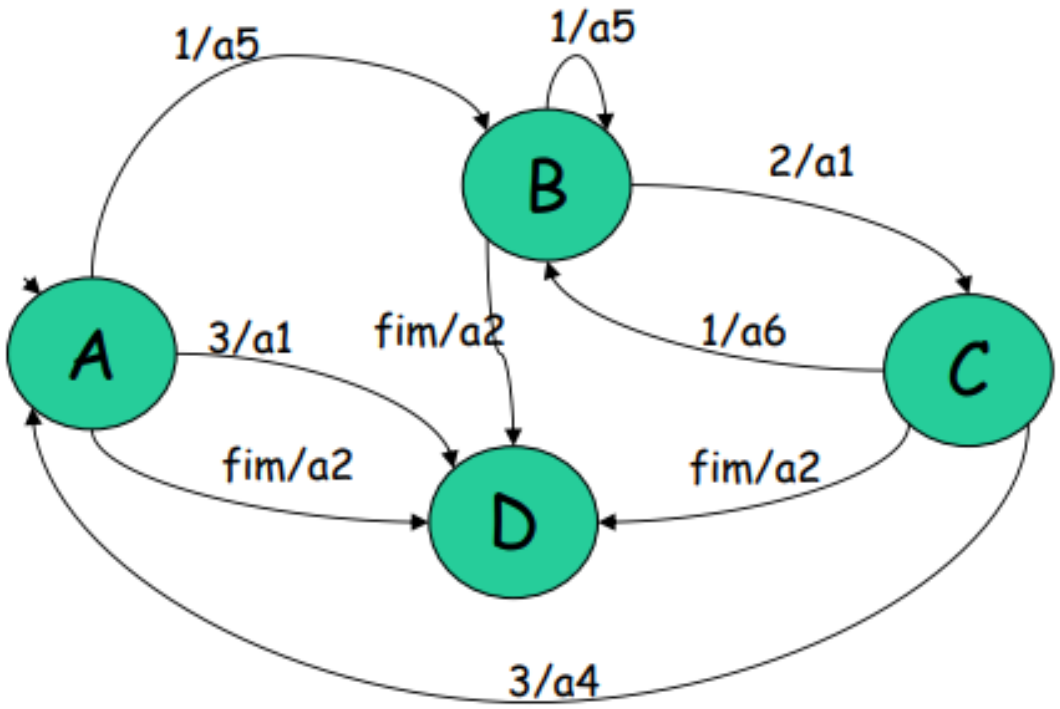
Strongly connected

- A statemachine is strongly connected if from **any state we can achieve the remaining states**.



deterministic

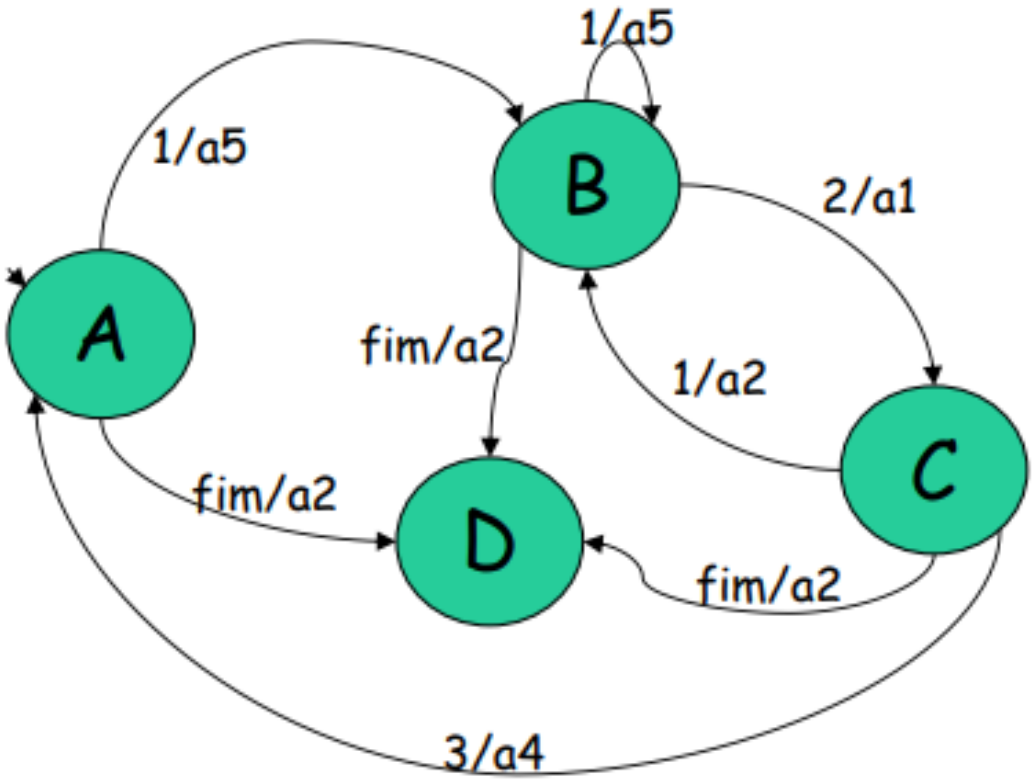
- The machine is deterministic if **each pair event-state is associated to only one transition**.



Reduced / minimum

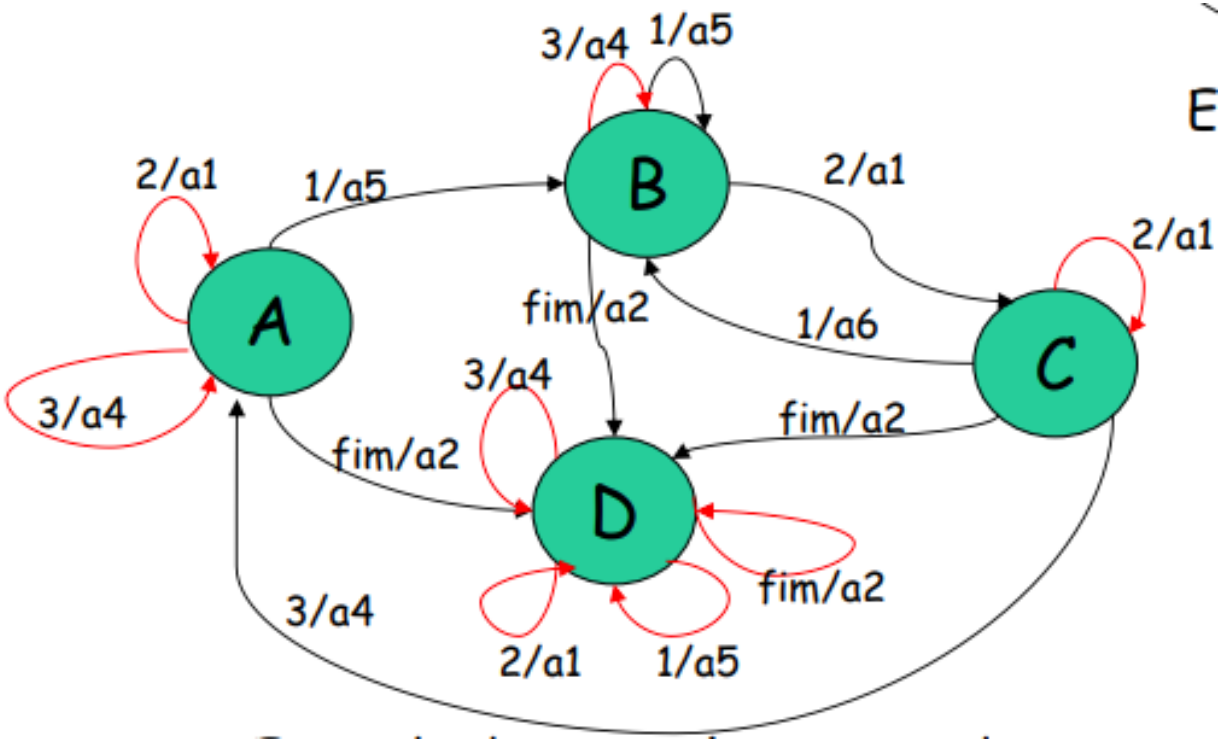
- A statemachine is reduced/minimun if **do not have equivalente states**

- Equivalent states are those that the same sequence of events led to the same actions



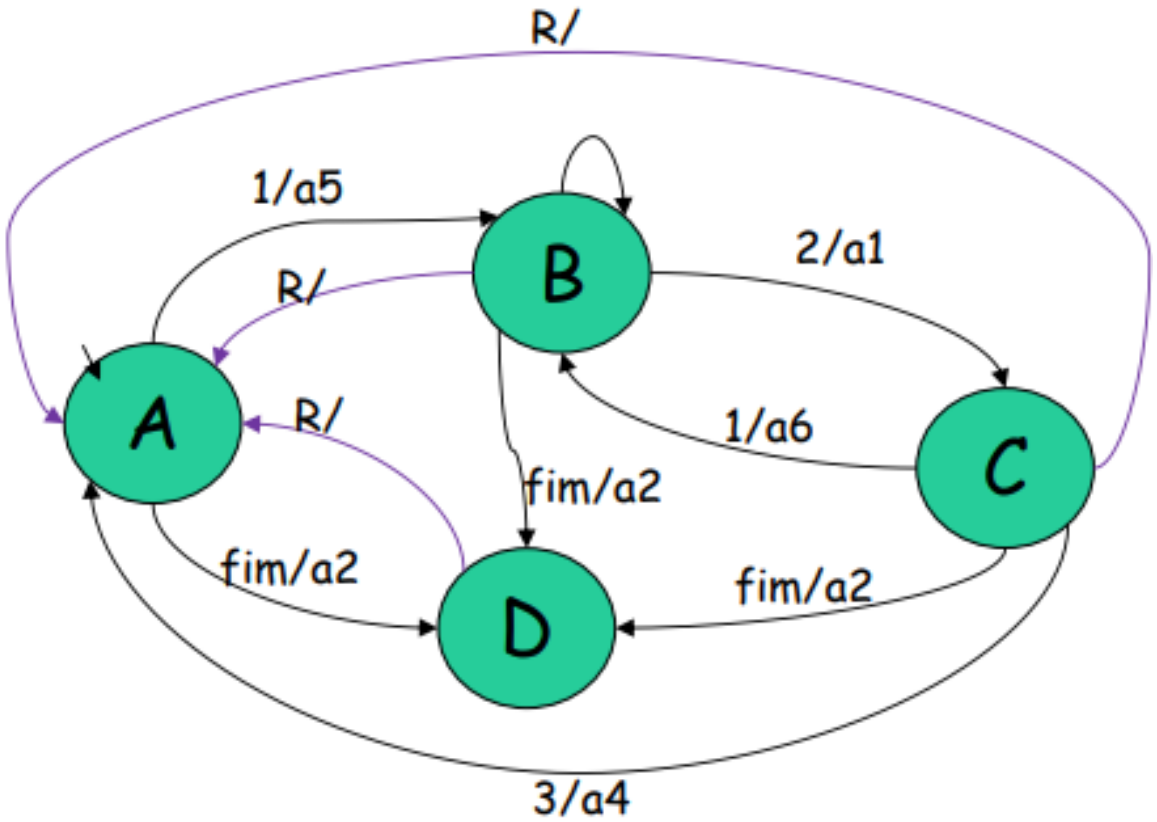
complete

- A statemachine is complete **if to each pair event-state exist one transition.**



Reset capability

- There is a special event that **leds the machine to a defined state**.



Exercises if you want to practice...

Walking in the statemachine

- Find the output to each input sequence:
  - WOOLOOMOOLOO
  - WLLOMWOOLLM
  - MLWWLOOMMOO
- Draw the state diagram

State	Transition				Exit			
	W	L	M	O	W	L	M	O
→ 0	0	2	1	2	t	a	A	a
1	2	0	2	1	a	t	T	a
2	1	0	2	1	n	n	n	n

Properties

- Check the completness.
- Change the machine so it is not deterministic.
- Illustrate a situation where the machine is not minimum.
- Illustrate a situation where the machine is not initially conected.

State	Transition				Exit			
	W	L	M	O	W	L	M	O
→ 0	0	2	1	2	t	a	a	a
1	2	0	-	1	a	t	-	a
2	1	0	2	1	n	n	n	n



State Model Based Testing

[Chow 78] T. S. Chow. “Testing software design modeled by finite-state machines”, IEEE Transactions on Software Engineering, v.4, n.3, p 178-187, 1978.

DEFINITION

- **Test based on finite state machines** relies **on systems that can be modelled as reactive systems**, and consists in a **generation of test cases from the state machine model in order to find implementation defects**.

Test Generation methods

- DS (Distinguishing Sequence) (Gonec- 1970),
- **TT (transition tour) (Chow 1978)**
- W – (Chow – 1978),
- UIO (Unique Input/Output) – Sabnani e Dahbura – 1988)
- UIOv – (Yuong – 1989)
- HSI (Harmonized State Identification) (Petronko et al. – 1993)
- HIS (Harmonized Identifiers) – (Luo et al., 1995)
- State-Counting – (Petrenko e Yevtushenko, 2005)
- HSY – (Adenilso Simão et al. – 2011)



TRANSITION TOUR EXAMPLE

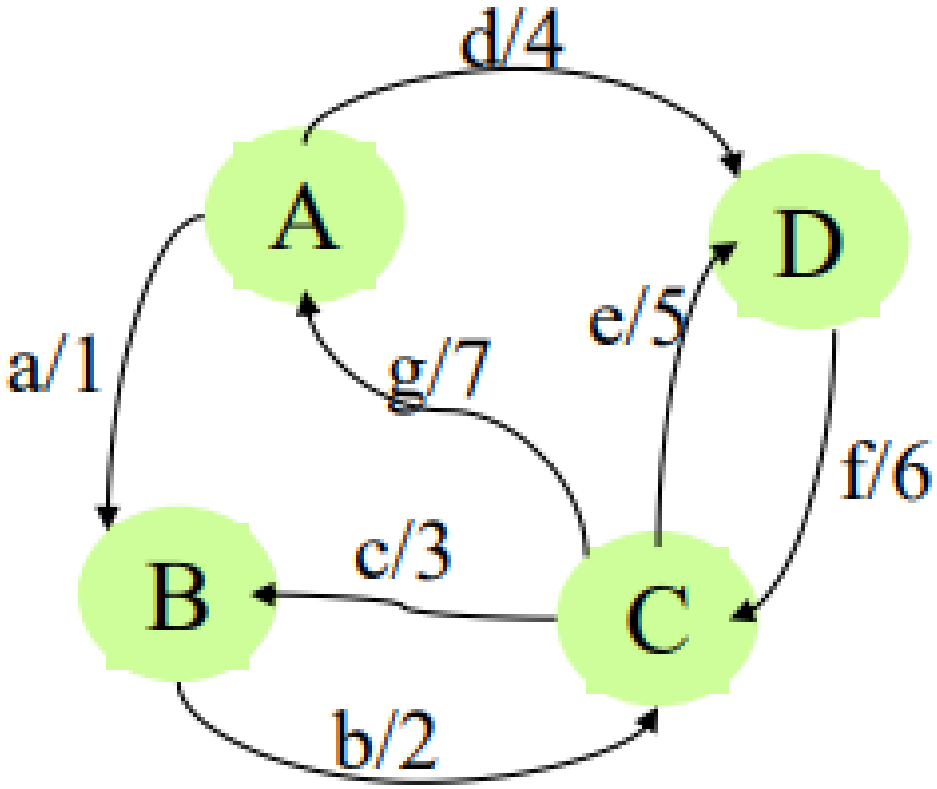
Lets first learn how to “walk” in the statemachine

TRANSITION TOUR

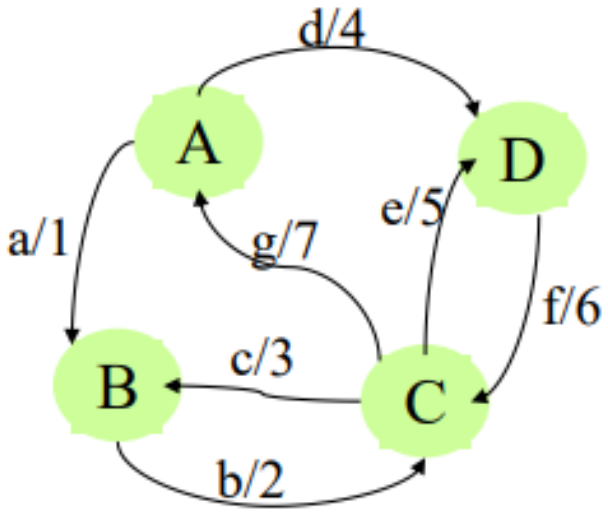
- This method garantes that all outputs are tested as all transitions are run at least one time.
- Using this method, the best test sequence is the one that starts from the initial state and run the transition only one time (Euler Tour).

example

- 1) a b e; d f g ; c
- 2) a; b; c; d; e; f; g



From the test cases we can design a test tree





- a - b - g -> 1 - 2 - 7
- a - b - c -> 1 - 2 - 3
- a - b - e -> 1 - 2 – 5
- d - f -> 4 - 5



MUTANTS

R. A. DeMillo, R. J. Lipton, and F. G. Sayward. Hints on test data selection: Help for the practicing programmer. IEEE Computer, 11(4):34–43, April 1978.

T. A. Budd. Mutation Analysis: Ideas, Example, Problems and Prospects, chapter Computer Program Testing. North-Holand Publishing Company, 1981.

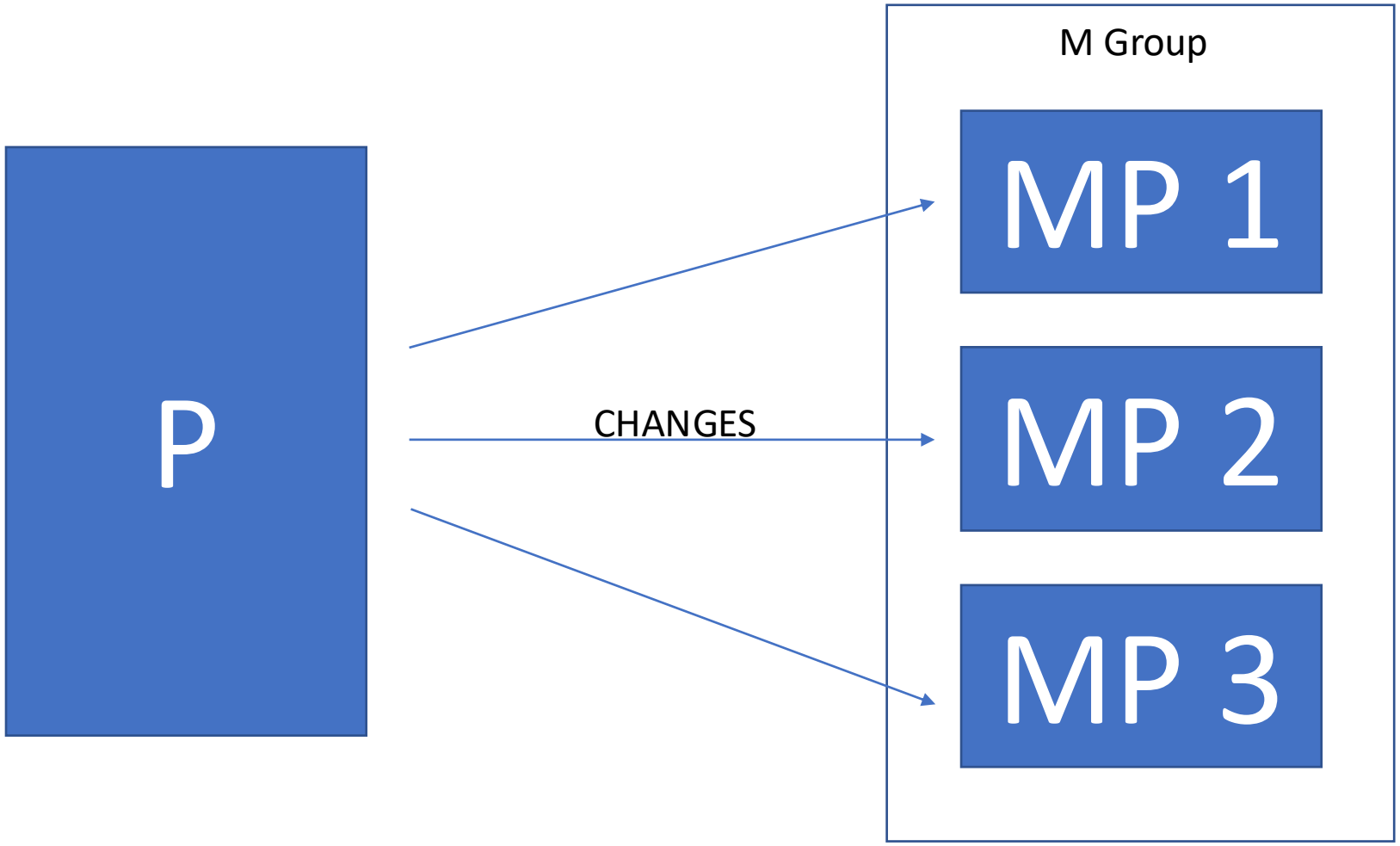
Definition

- A mutante is a slight change in the model made by inclusion, alteration or removal of a characteristic.
- A mutante tries to model a simple defect that could be made by the system developer.

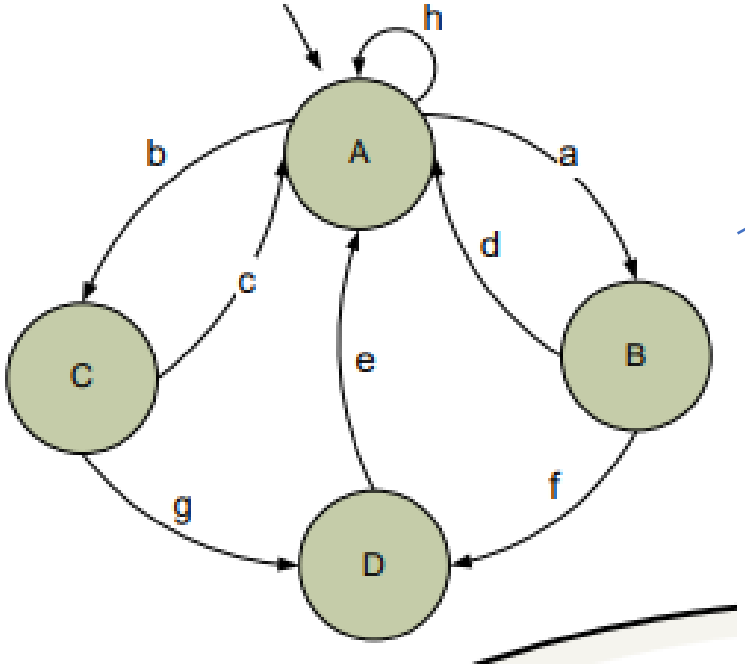
Mutant generation

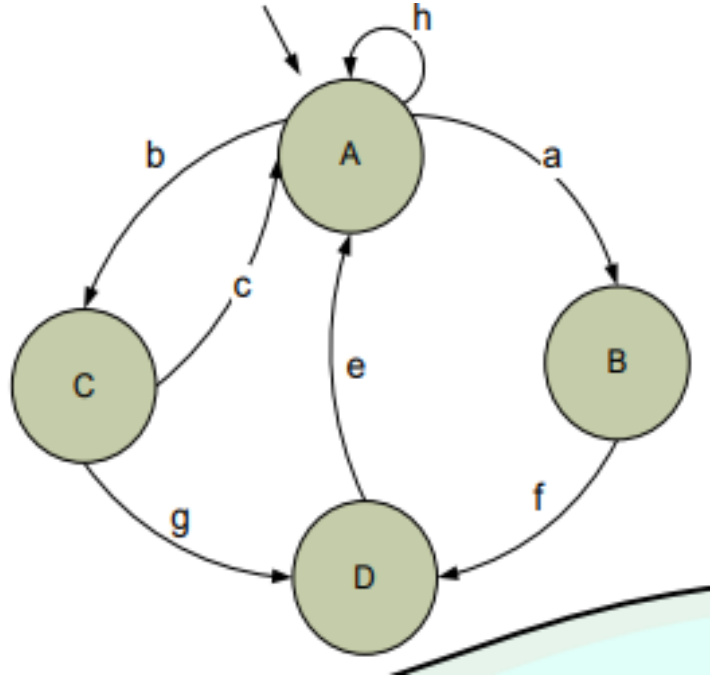
- From a product P, it is built a set of products with alterations in P.
- Each product is named “mutant of P”
- The group M, named neighborhood of P, is the group of mutants of P.

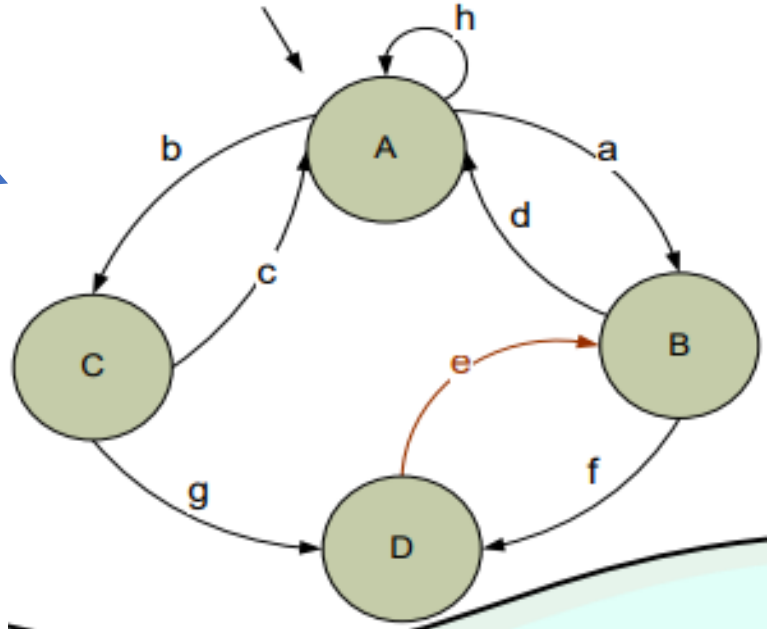
MUTANT GROUP



example







Where the problem starts

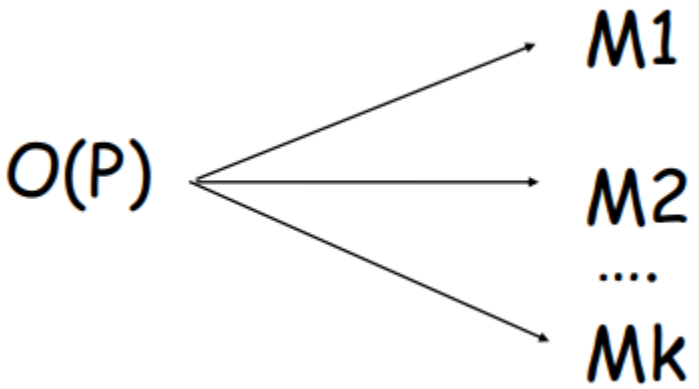
- There is a problem situation regarding the specification of system and the implementation of a system.
- The specification is the desired behavior proposed by the system developers (not only on the systems engineering field)
- The implementation is the understanding of the specification done by the supplier, which in most of the cases use requirements as the only source of information.

Mutant hypothesis

- A implemented system is correct or near correct.
- And due this fact, the neighborhood of P, which are products close to the expect behavior of P can be derived.
- So, if the neighborhood of P is not readed by the system, we can assume that the system is resiliente to the proposed neighborhood defects.
- Then the group of test cases T, that will test P may find the neighborhood of P and find-out the defects.

Mutant operators

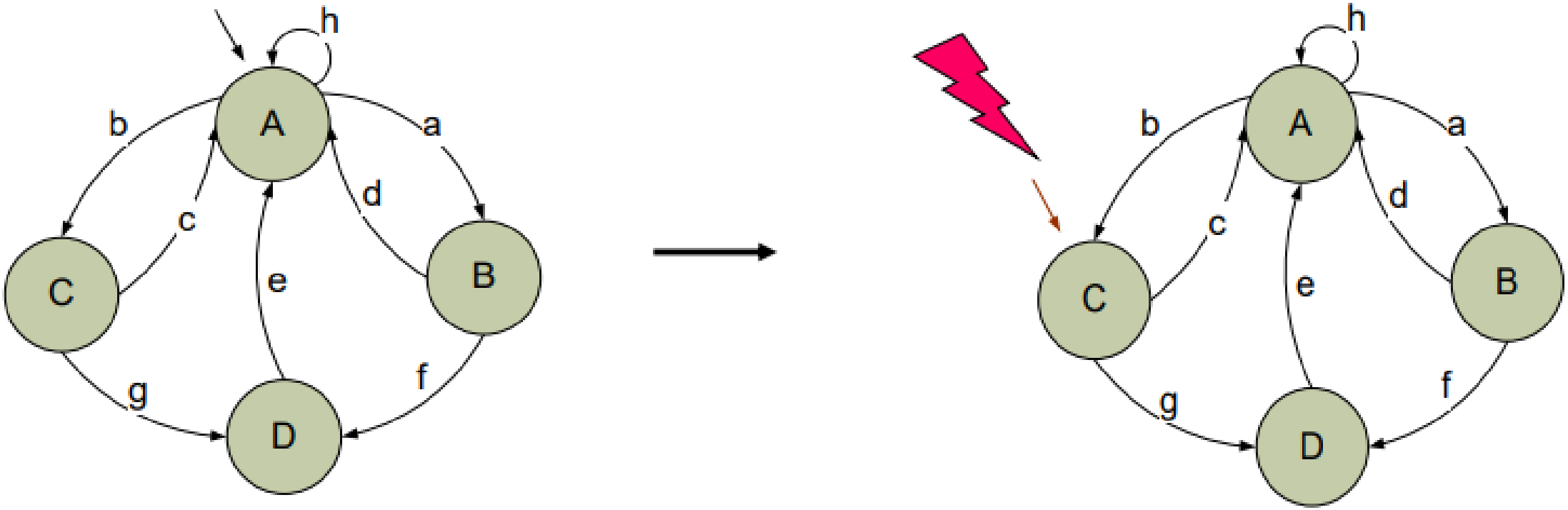
- A mutant operator O is a function that maps the test program in a group of K mutants of P.
- A mutant operator is a generalization of the set of defects that a system may have.



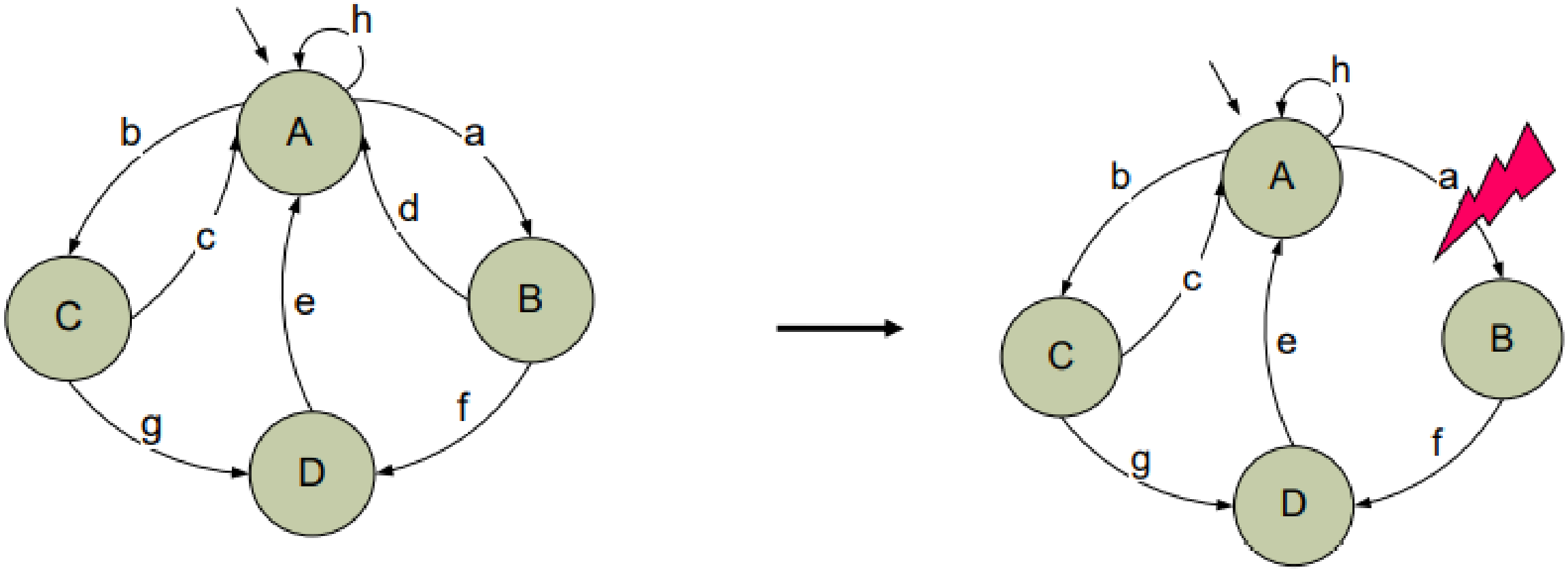
OPERATORS

- TraIniStaAlt – Change of the initial state
- TraArcDel – Deletion of a transition
- TraEveDel – Delete event
- TraEveIns – Add a event
- TraEveAlt – Change the event
- TraDesStaAlt – Change the transition destination
- OutDel – Output missing
- OutAlt – Change output
- StaDel – Delete state

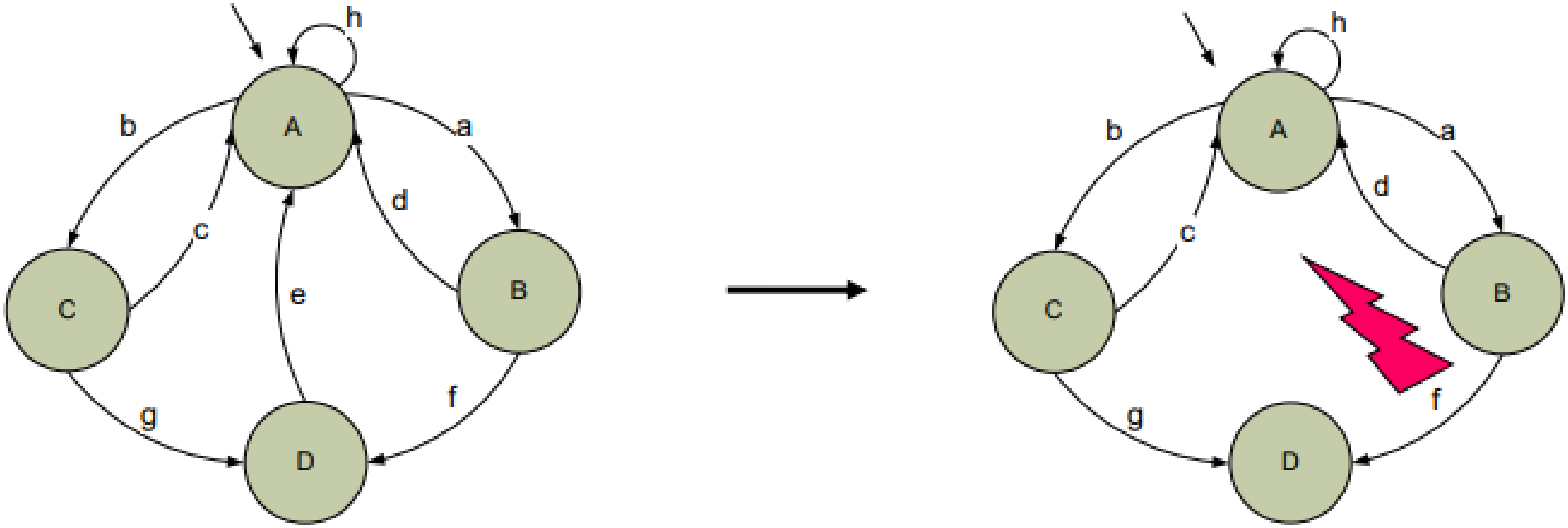
Trainistaalt



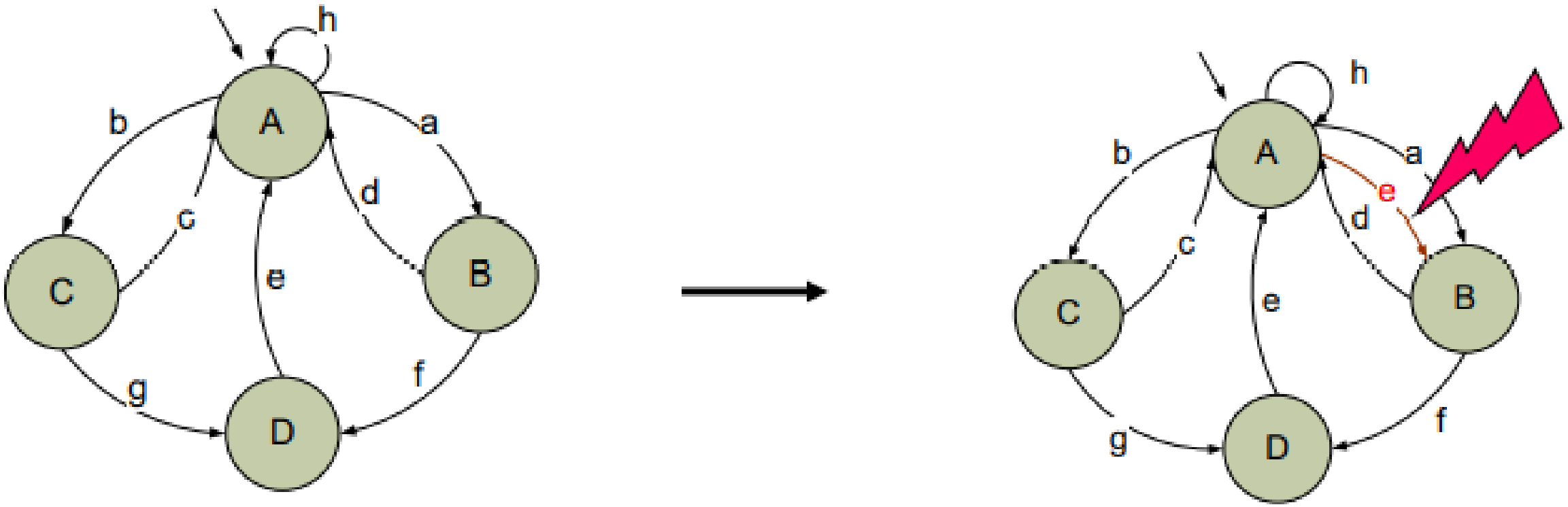
traarcdel



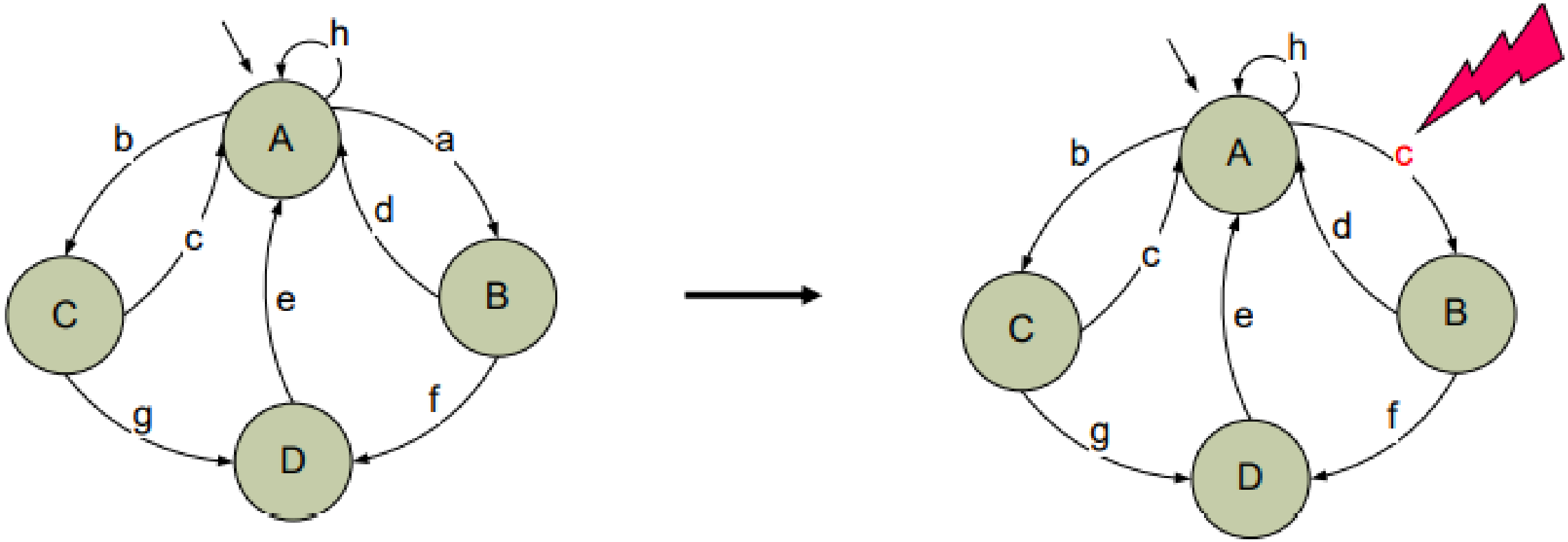
traevedel



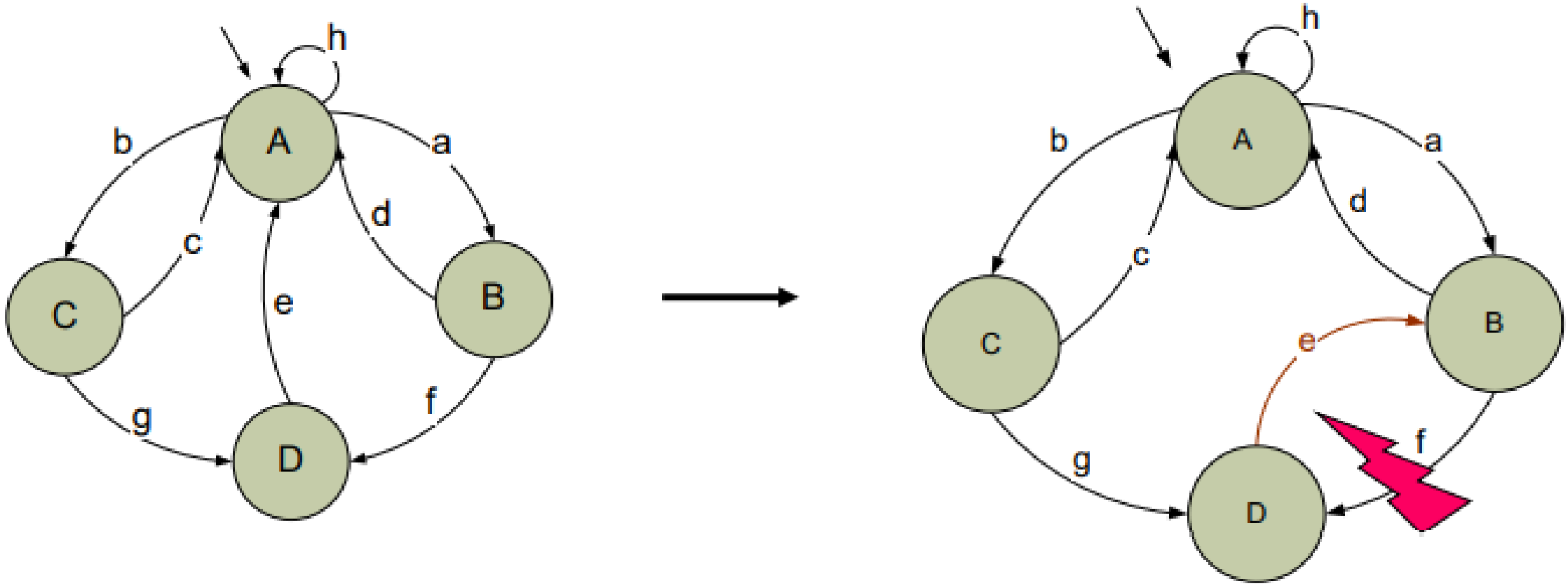
traeveins



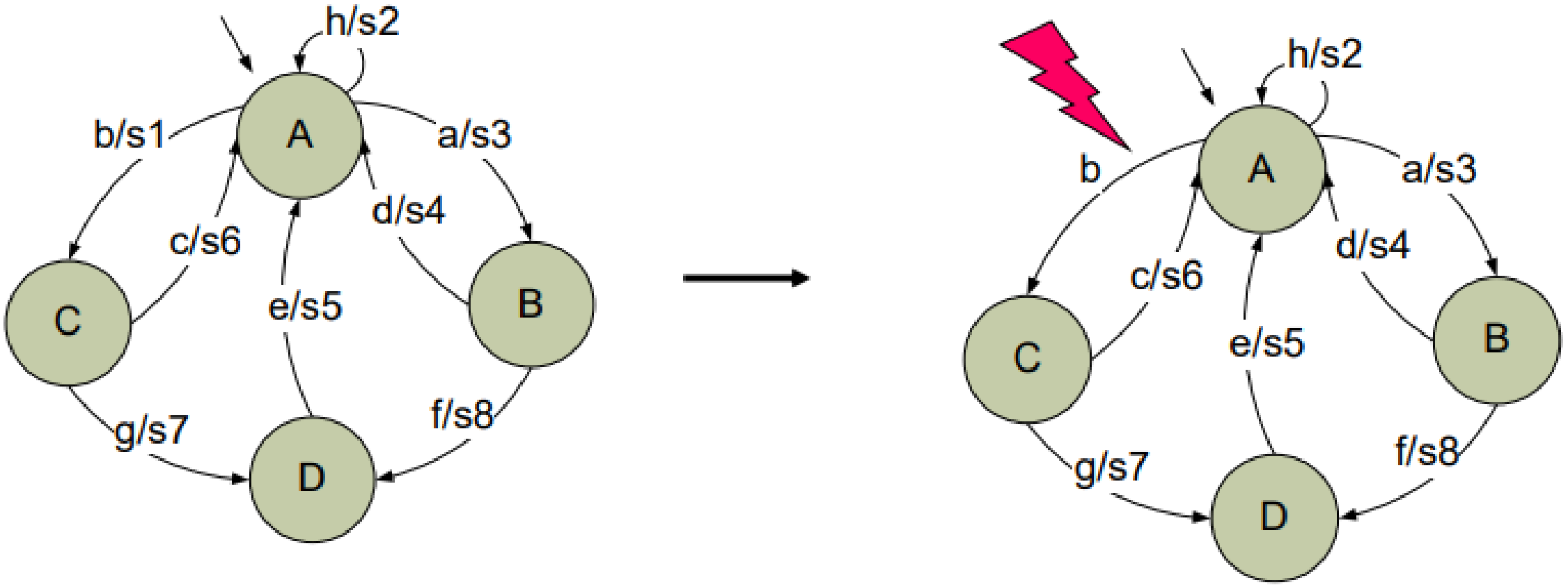
traevealt



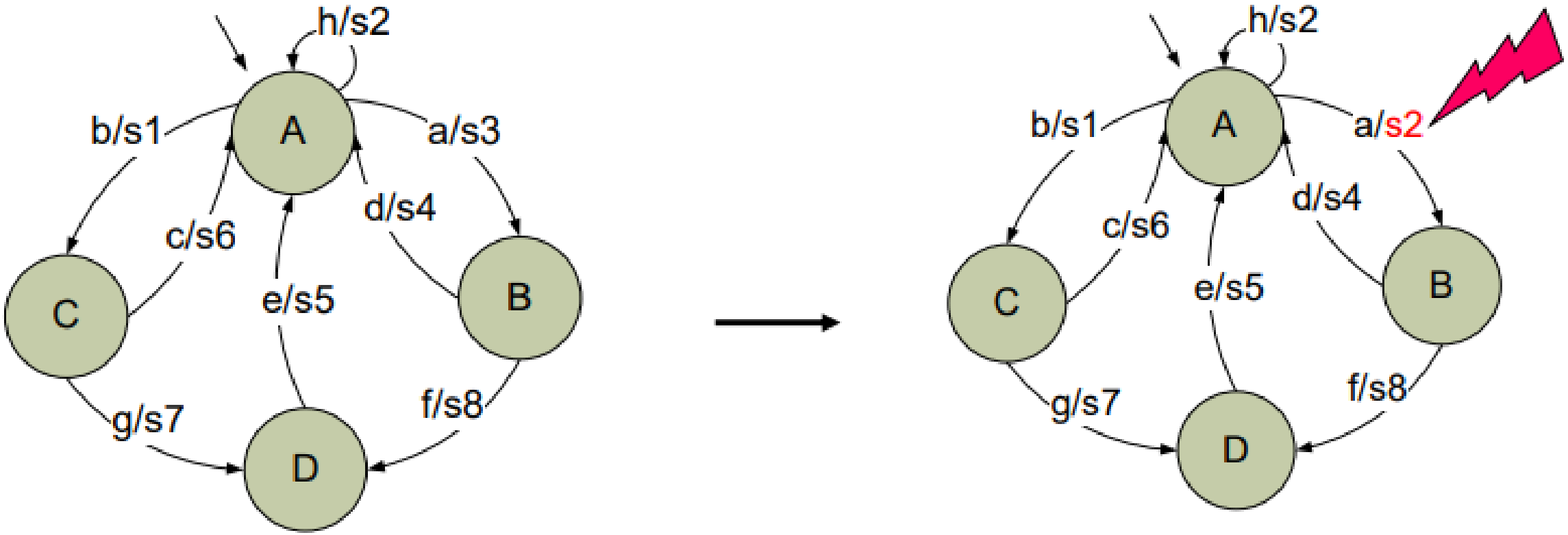
tradesstaalt



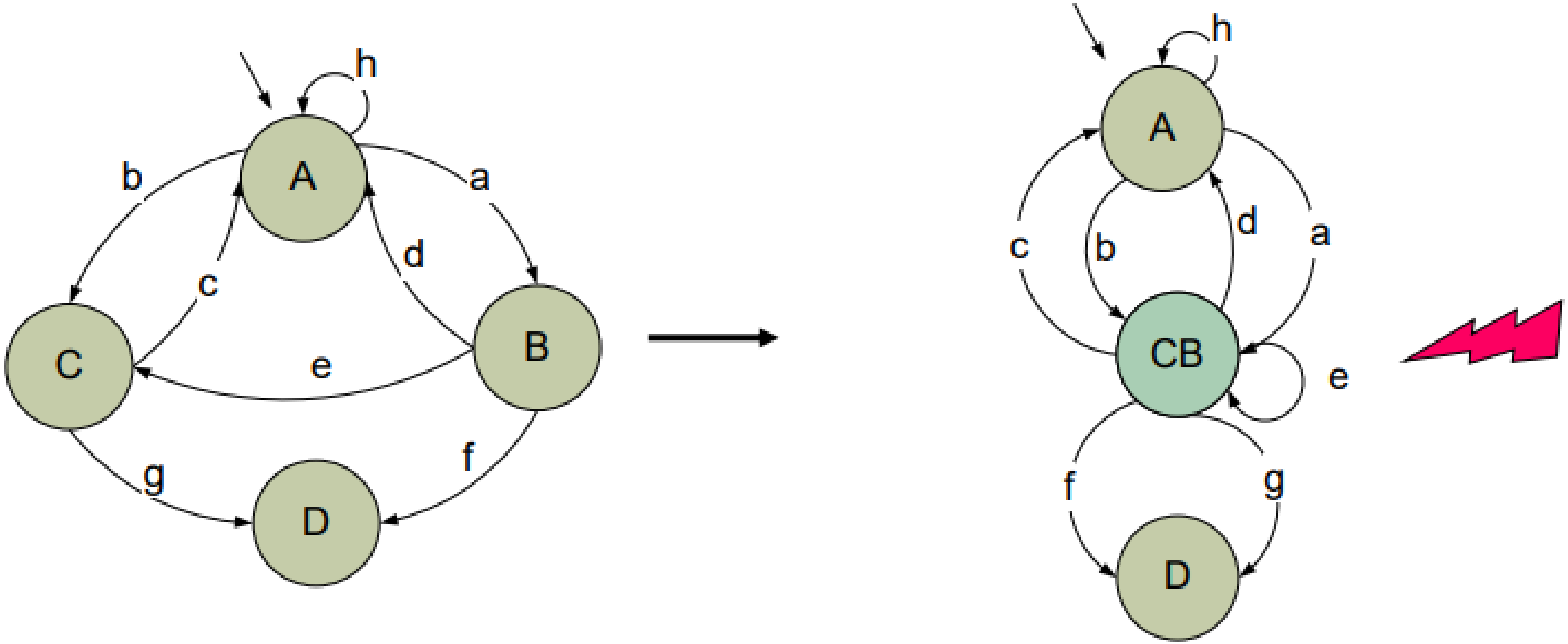
outdel


h/s2
b/s1
A
a/s3
d/s4
c/s6
e/s5
C
B
g/s7
f/s8
D
h/s2
b
A
a/s3
c/s6
d/s4
e/s5
C
B
g/s7
f/s8
D

outalt


h/s2
b/s1
A
a/s3
d/s4
c/s6
e/s5
C
B
g/s7
f/s8
D
h/s2
b/s1
A
a/s2
d/s4
c/s6
e/s5
C
B
g/s7
f/s8
D

stadel



Mutant analysis

- It is a criteria to evaluate the group of test cases T with the model P.
- To do the analysis, it is used the group of mutants created from the original model P.

- Main question: what is the the object to do the analysis?

Mutant analysis objective

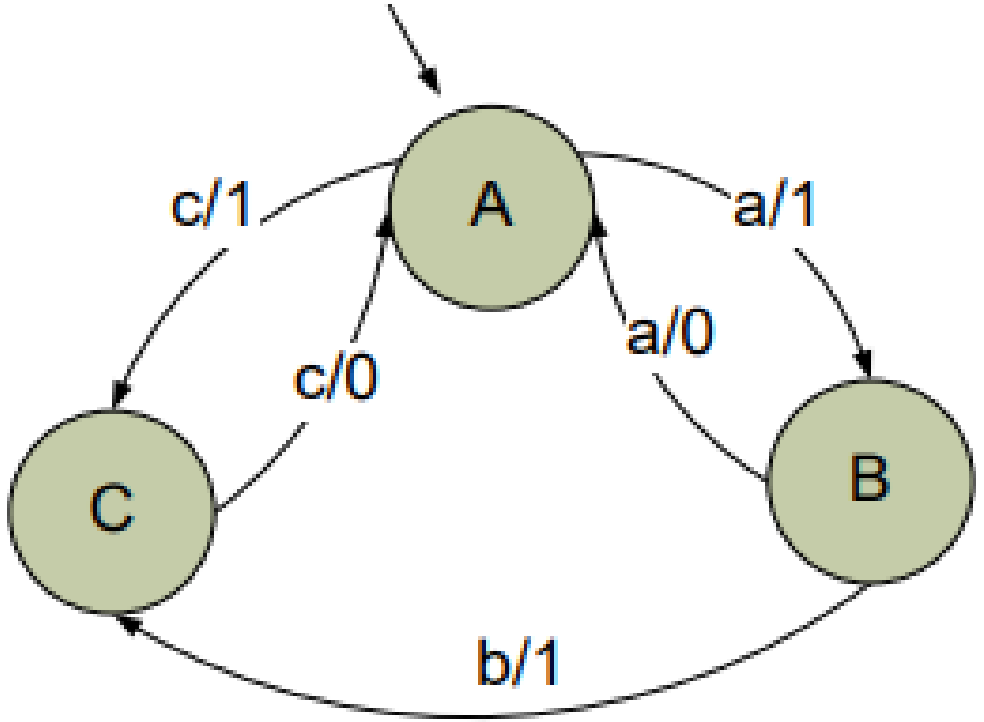
- The group of test cases must show, through the execution o P, the behavior diferences existing among P and the mutants.
- So:
  - Generate the mutants
  - Execute P with the T test cases
  - Execute group of mutants M with the T test cases
  - Analyse the mutants
  - Reports

Dead mutante

- With the execution of the test cases into the mutante, this mutante will generate a result diferente from the generated by the original P.
- This understanting indicates the conclusion that the test cases are sensible and where able to show the diferences between P and the mutants.
- If the mutante survives, it indicates that T is not adequate to test P, so new test cases must be created into T.

Practice exercise

- List the states, input and outputs
- Determine the properties of the state machine
- Draw one mutante to each operator



Theme: **Smart City Traffic Light Coordination -** Use inputs from sensors, manual override, pedestrian crossing requests.

- The mission of the Smart City Traffic Light Coordination System is to enhance the safety, efficiency, and sustainability of urban mobility. The system shall manage multiple intersections by integrating real-time data from road sensors, operator commands, and pedestrian crossing requests. It shall dynamically adjust traffic signal patterns to reduce congestion, minimize travel delays, and provide safe crossing opportunities. The system shall accommodate manual overrides for emergency and special events, giving city operators the ability to impose priorities when necessary. It shall also ensure rapid clearance routes for emergency vehicles, while balancing the needs of everyday commuters and pedestrians. Operating within the constraints of city regulations, legacy infrastructure, budget limitations, and environmental concerns, the system shall aim to reduce vehicle emissions and improve the overall quality of life for urban residents.

Rich picture



Group / Case:

X / **Smart City Traffic Light Coordination**

Analize 01 (3 actors):

City Traffic Operators (control center staff)

Pedestrians (safety + crossing requests)

Drivers (road users affected by congestion)

Analize 02 (3 dimensions):

Political: City council demands congestion reduction and improved safety.

Economic: Traffic delays cost fuel, time, and productivity.

Social: Public pressure for fair pedestrian treatment and smoother traffic flow.

Analize 03:

City Traffic Operators → can configure, activate overrides, and set emergency priorities.

Pedestrians → can request crossings, influence political for safeness.

Drivers → create demand, cause congestion, and exert pressure for efficiency

C **Customers (C):** Drivers and pedestrians benefit from smoother flow and safer crossings.

A **Actors (A):** City traffic operators configure and monitor the system.

T **Transformation (T): From uncoordinated traffic lights to coordinated signals across intersections.**

W **Weltanschauung (W):** Coordinated signals reduce congestion and delays, creating safer, more efficient mobility.

O **Owners (O):** City government / transport authority decides whether the system is implemented.

E **Environment (E):** Legacy infrastructure, limited budget, city regulations.

C **Customers (C):** Pedestrians (shorter, safer waits); drivers (less risky jaywalking).

A **Actors (A):** City traffic operators and automated timing algorithms.

T **Transformation (T): From excessive pedestrian waiting at crossings to balanced waiting times that ensure safety and mobility.**

W **Weltanschauung (W):** Balancing pedestrian needs with vehicle flow improves safety, fairness, and public trust in the system.

O **Owners (O):** City government / transport authority.

E **Environment (E):** Minimum legal crossing times, high vehicle demand at rush hour, weather conditions.

Root Definition:

A city-operated system that coordinates traffic light signals across intersections using operator inputs, in order to reduce congestion and delays for drivers and provide safer crossings for pedestrians, within the constraints of city regulations, legacy infrastructure, and budget limitations.

Needs:

Need 1 - **The city government** needs the traffic light coordination system to reduce congestion and delays for drivers across multiple intersections.

Need02 - **The pedestrians** need the traffic light coordination system to provide safer and more predictable crossing opportunities.

Root Definition:

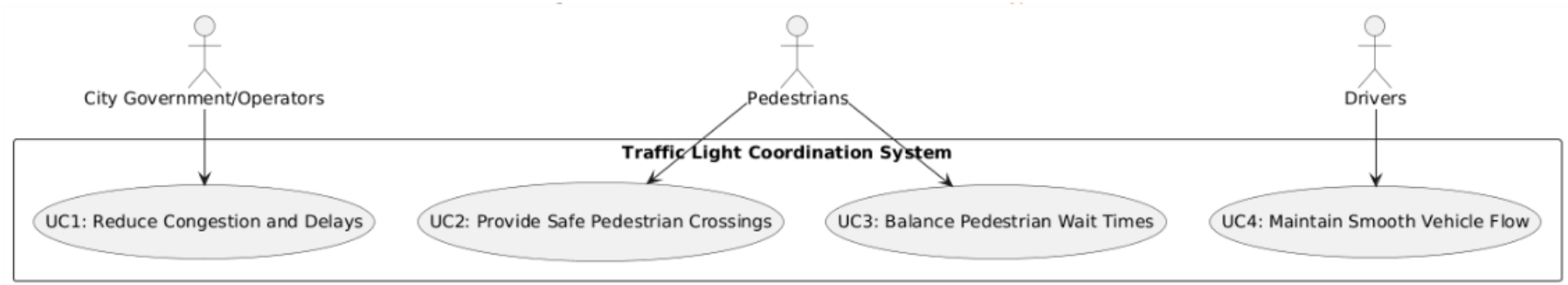
A city-operated system that balances pedestrian wait times at crossings through operator actions and automated timing algorithms, in order to improve safety and fairness for pedestrians while supporting vehicle mobility, within the constraints of legal crossing requirements, traffic demand, and environmental conditions.

Needs:

*Need03 - **The pedestrians** need the traffic light coordination system to balance waiting times so that crossing opportunities occur within acceptable limits.*

*Need04 - **The drivers** need the traffic light coordination system to maintain smooth vehicle flow while accommodating pedestrian safety and fairness.*

Use Case Diagram



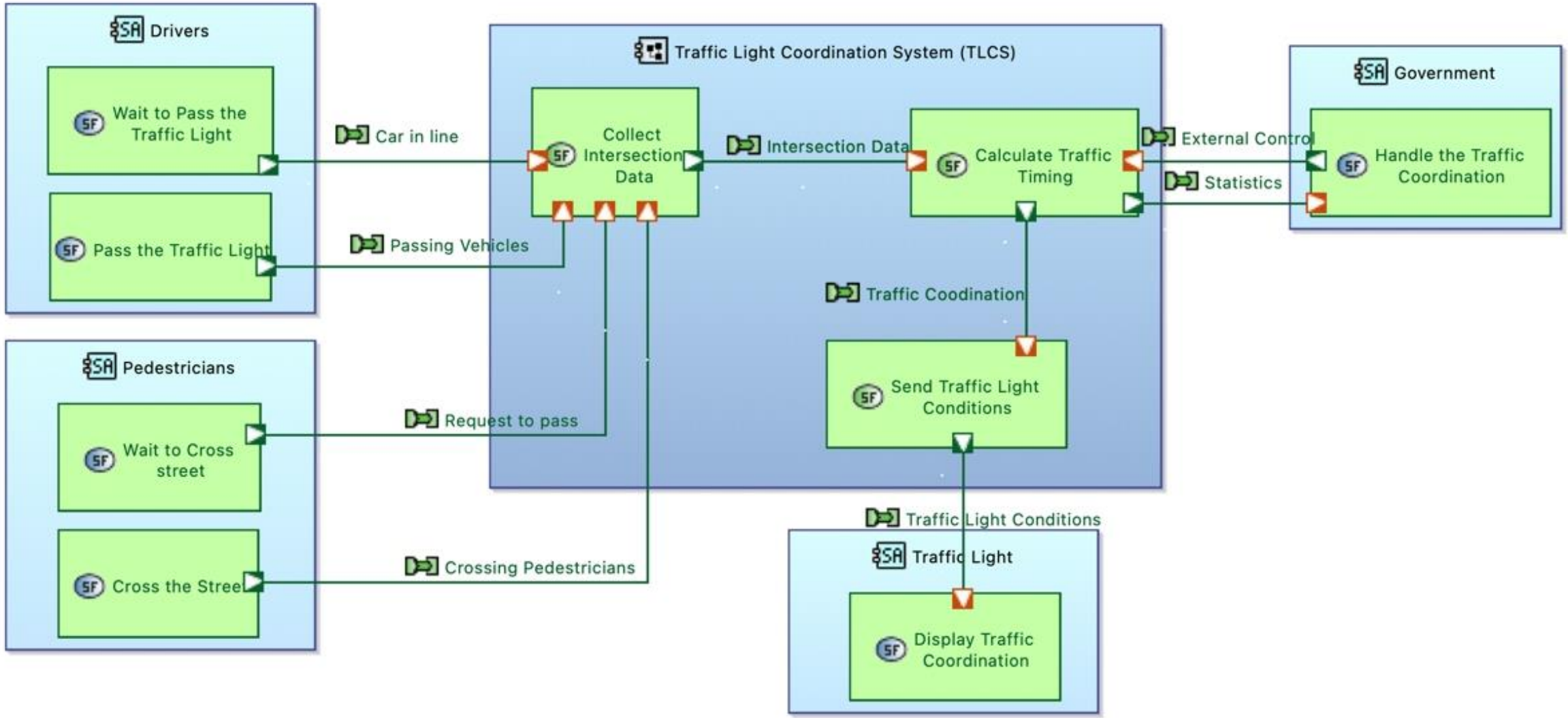
UML Use Case Diagram (PlantUML)

plantuml Copy

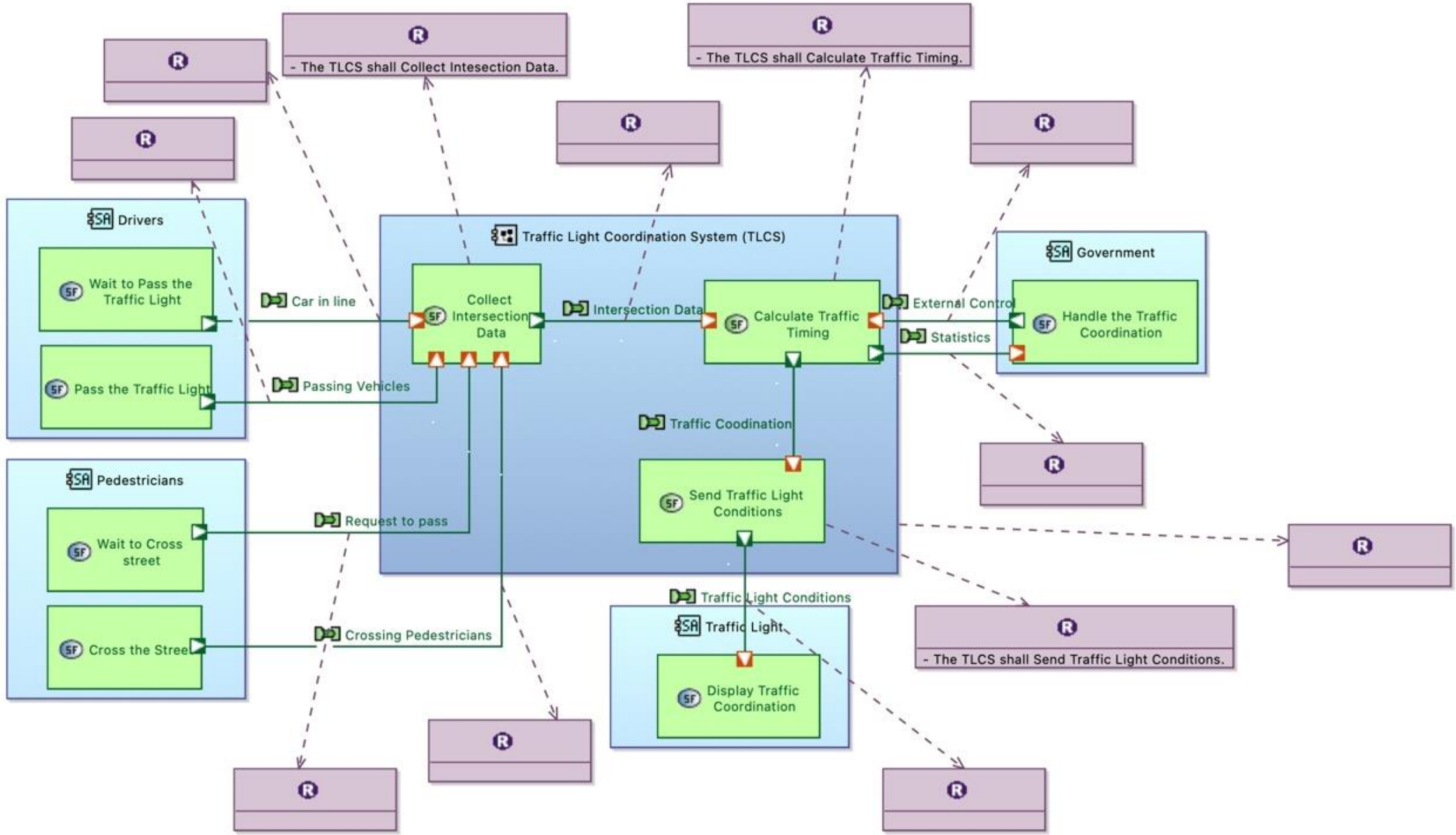
```
@startuml
actor "City Government/Operators" as GOV
actor "Pedestrians" as PED
actor "Drivers" as DRV

rectangle "Traffic Light Coordination System" {
  usecase "UC1: Reduce Congestion and Delays" as UC1
  usecase "UC2: Provide Safe Pedestrian Crossings" as UC2
  usecase "UC3: Balance Pedestrian Wait Times" as UC3
  usecase "UC4: Maintain Smooth Vehicle Flow" as UC4
}

GOV --> UC1
PED --> UC2
PED --> UC3
DRV --> UC4
@enduml
```



https://mbse-capella.org


- The TLCS shall Collect Intesection Data.
- The TLCS shall Calculate Traffic Timing.
Drivers
Wait to Pass the Traffic Light
Pass the Traffic Light
Traffic Light Coordination System (TLCS)
Collect Intersection Data
Calculate Traffic Timing
Government
Handle the Traffic Coordination
Car in line
Intersection Data
External Control
Statistics
Passing Vehicles
Traffic Coodination
Pedestricians
Wait to Cross street
Cross the Street
Send Traffic Light Conditions
Request to pass
Crossing Pedestricians
Traffic Light Conditions
Traffic Light
Display Traffic Coordination
- The TLCS shall Send Traffic Light Conditions.

DVM

REQ#ID	TITLE	STATEMENT	RATIONALE*	VER.METHOD	SUCCESS CRITERIA	TRACEABILITY*1	…

* All the rationale are related to the user history and SSM Exploration

*1 Traceability – Point which Use cases or the Architecture Model