Reactive and Event Based Systems
Lecture 1: Course Introduction, Process Modeling, Automata, and Formal Languages

Tijs Slaats
Hugo Lopez
Thomas Hildebrandt
Monday 22nd of November 2021

UNIVERSITY OF COPENHAGEN
DEPARTMENT OF COMPUTER SCIENCE

Overview

- **Course Introduction**
- Process Modelling
- Brief introduction to automata & formal languages

Reactive and Event Based Systems

Reactive and event-based systems follow an architectural style where components *react* to *events* generated by the environment or system itself.

Source: https://www.cmarix.com/5-effective-tips-to-build-an-engaging-and-user-focused-retail-app/

Examples: IoT data platforms, low-latency websites, mobile apps, …

Industries: Logistics, smart cities, future transportation, games, next-gen farming, …

Reactive principles:

- Responsiveness
- Flexibility
- Resiliency
- Adaptability and elasticity
- Scalability

Source: https://mitechnews.com/internet-of-things/how-big-is-iot-20-6-billion-connected-devices-by-2020/

Reactive and Event Based Systems

Course divided in three main parts:

- Week 2 - 3: *Declarative event-based processes* (Tijs Slaats)
- Week 4 - 5: *Formal models and programming languages for reactive and message based systems* (Thomas Hildebrandt)
- Week 6 - 7: *Streaming & Compliance* (Hugo A. Lopez)
- Week 8 Course conclusion

Lectures: Monday, 10:15am - 12:00pm, Kursussal 3, Universitetsparken 15 (Zoo).

Assignments: Fridays, 10:15am - 12:00pm, øv - A112, Universitetsparken 5, HCØ.

Reading: Reading for the course will be article-based, these will be announced on-the-fly.

Reactive and Event Based Systems

Assignments:

- 3 Assignments, 1 for each part.
- All contain implementation with relevant technologies
  - Reflections also important
- Mandatory! (Pass/fail, need to pass all to take the exam)
- Feedback provided by us for preparation to exam
- Groups of 2-3 students
  - Exceptions for single student groups granted on an individual basis, send us an email or message on Absalon.
- Submissions in Absalon

Reactive and Event Based Systems

Exam:

- Individual oral exam of 20 min
- Each student prepares three 8-min presentations for each of the three assignments
- One presentation to be given decided at random at the exam
- Questions on presentation as well as on the whole curriculum
- Notes and laptop allowed

Tentative schedule Part A

Week 2:

- *Lecture*: DCR Graphs, Hierarchy, Semantics
- *Assignments*: Part a: Modelling Event Patterns as DCR Graphs
- *Reading*:
  - A case for declarative process modelling: Agile development of a grant application system
  - Hierarchical Declarative Modelling with Refinement and Sub-processes

Week 3:

- *Lecture*: Process Mining, Process Conformance, Process Discovery
- *Assignments*: Part b: Implementing a conformance checker for DCR Graphs
- *Reading*:
  - The Analysis of a Real Life Declarative Process
  - DisCoveR: Accurate & Efficient Discovery of Declarative Process Models

Tentative schedule Part B

Week 4:

- *Lecture*: Formal process languages as foundation for programming languages for distributed, reactive and event based
- *Assignments*: Jolie 1
- *Reading*: TBA
  -

Week 5:

- *Lecture*:
- *Assignments*: Jolie 2
- *Reading*: TBA
  -

Tentative schedule Part C

Week 6:

- *Lecture*: Process Compliance
- *Assignments*: Part A: Model legislative policies using DCR graphs
- *Reading*: Business Process Compliance Using Reference Models of Law.

Week 7:

- *Lecture*: Online Process Mining
- *Assignments*: Part B: Conformance Checking Laws
- *Reading*: TBD

Overview

- Course Introduction
- **Process Modelling**
- Brief introduction to automata & formal languages

Processes

Process: "A series of actions or steps taken in order to achieve a particular end."[1]

Examples:

- Production of car
- Handling of an insurance claim
- Treatment for lung cancer
- Software development
- Algorithms

[1] Oxford's free English dictionary

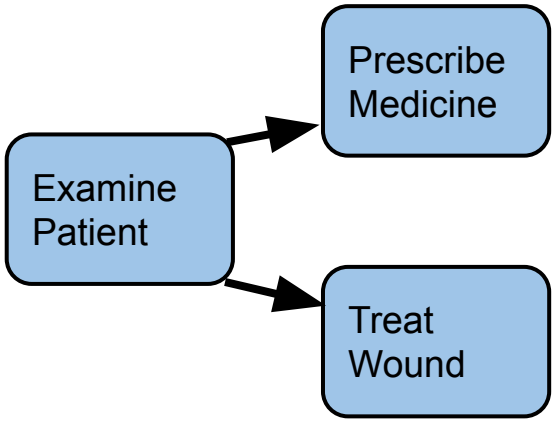
Process Modelling

How do we model a process?

- Plain text:

“Attach wheels and engine to frame.”

- Drawings:



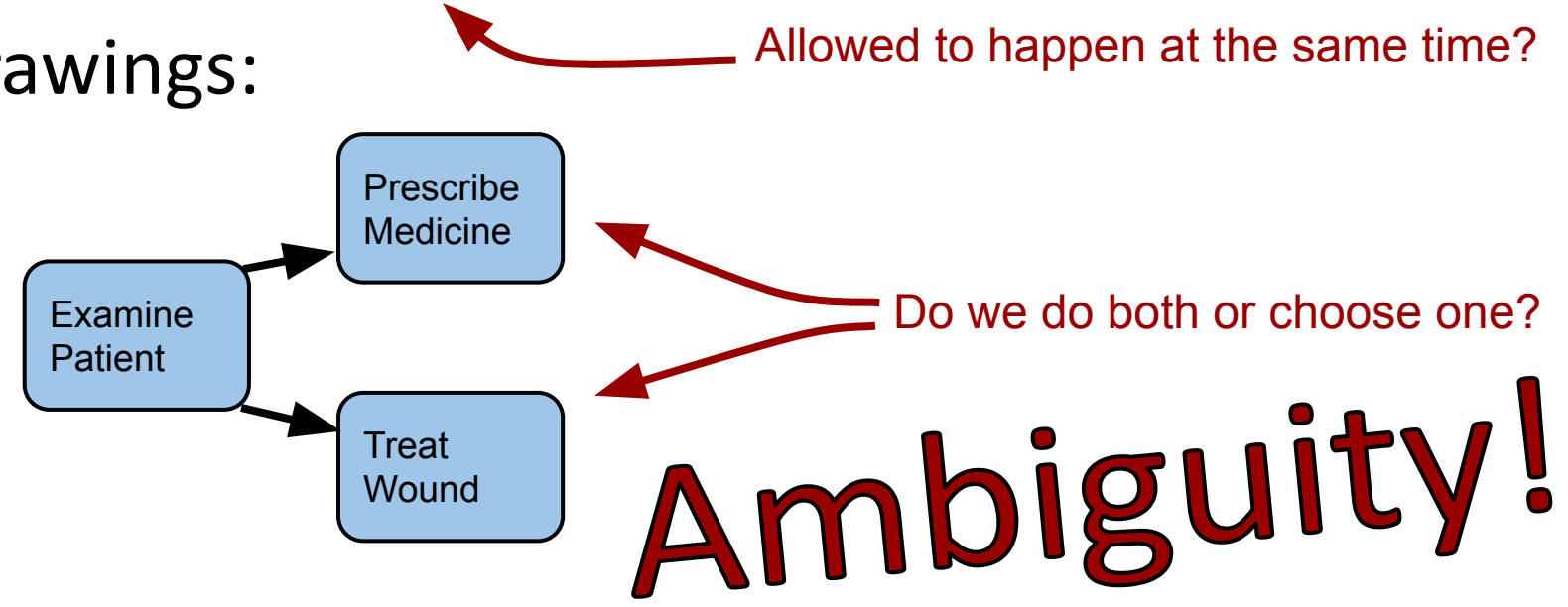
Process Modelling

How do we model a process?

- Plain text:

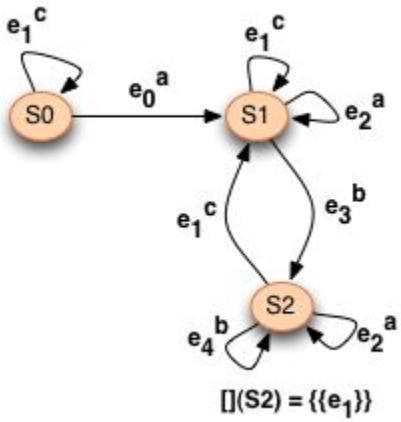
"Attach wheels and engine to frame."

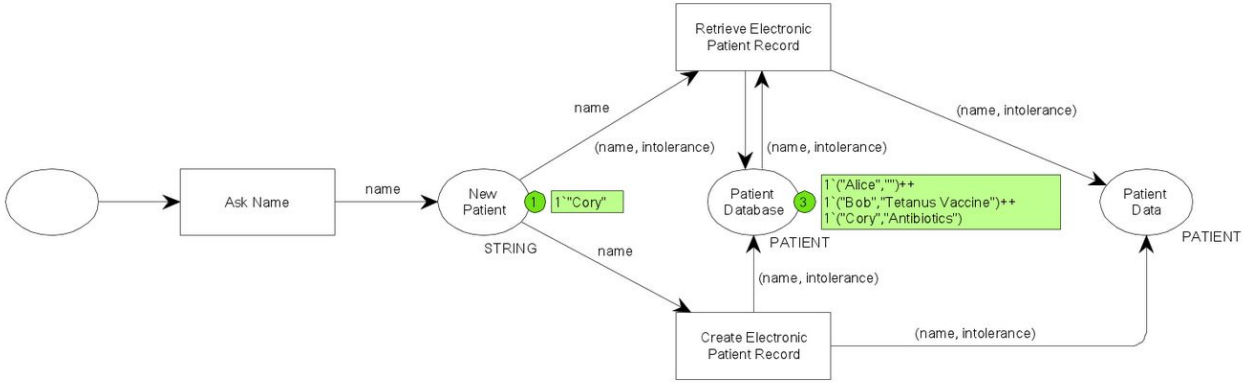
- Drawings:



Process Modelling and Computer Science

We need a well-defined language for our models: **Formal Methods**





□(Recieve Claim ⇒ ◇Evaluate Claim)
□(Approve Claim ⇒ ◇Payout Claim)
□(¬Payout Claim W Approve Claim)

Process Modelling and Computer Science

Formal models offer:

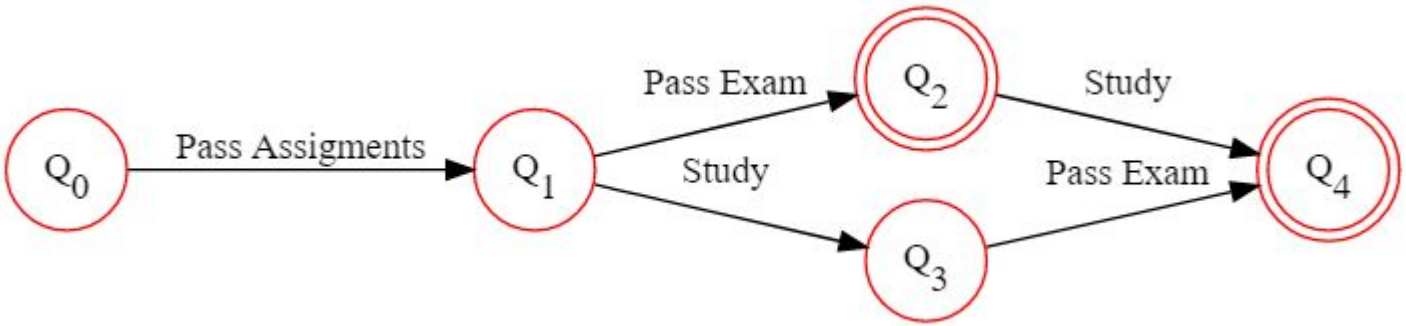
- Unambiguous semantics
- Verification
- Model checking
- Conformance checking
- Simulation
- Execution
  - Automated (fx assembly lines)
  - User guidance (fx call centers)

Overview

- Course Introduction
- Process Modelling
- **Brief introduction to automata & formal languages**

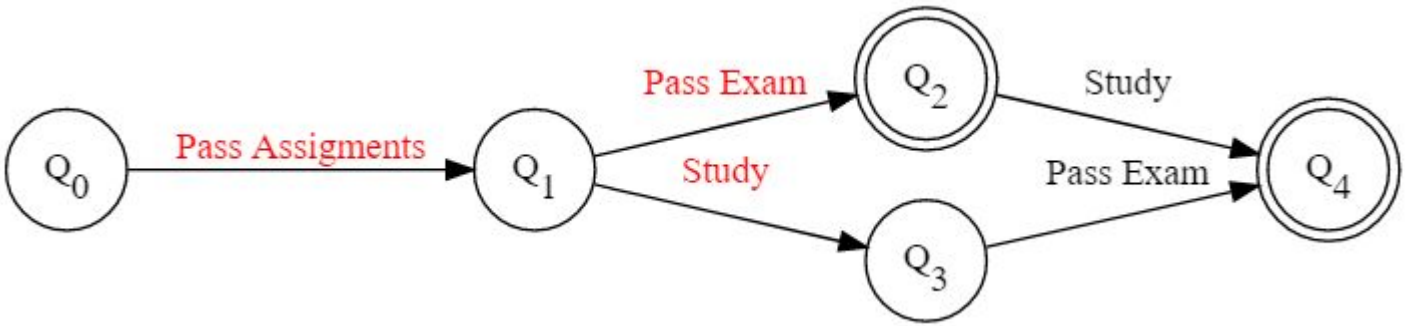
Finite State Automata

- One of many ways to formally model a process
- Consists of:
  - A finite set of states Q
  - An alphabet Σ
  - A transition function δ such that $\delta: Q \times \Sigma \rightarrow Q$
  - A *starting state* $q_0 \in Q$
  - A set of *accepting states* $F \subseteq Q$



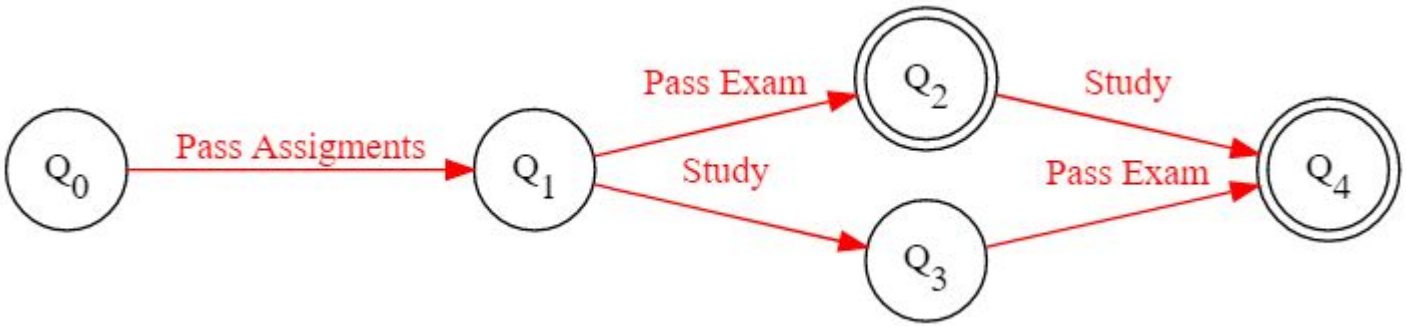
Finite State Automata

- One of many ways to formally model a process
- Consists of:
  - A finite set of states Q
  - An alphabet Σ
  - A transition function δ such that $\delta: Q \times \Sigma \rightarrow Q$
  - A *starting state* $q_0 \in Q$
  - A set of *accepting states* $F \subseteq Q$



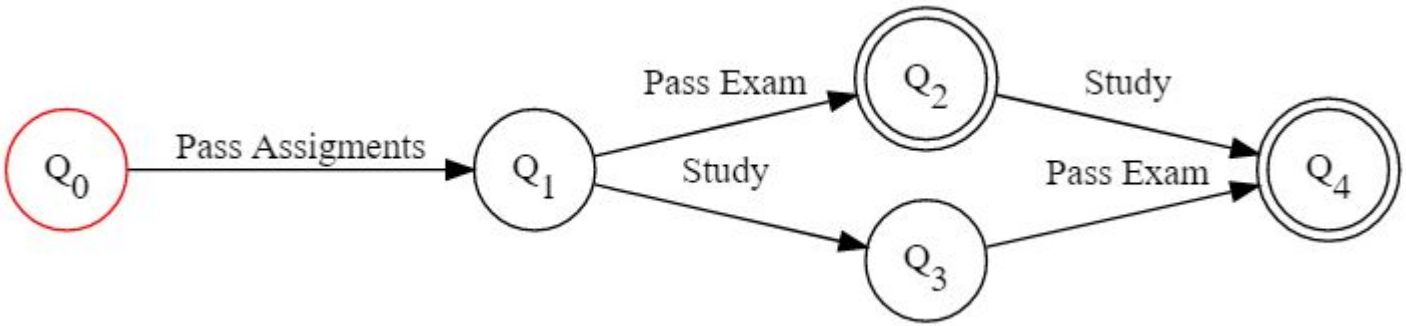
Finite State Automata

- One of many ways to formally model a process
- Consists of:
  - A finite set of states Q
  - An alphabet Σ
  - A transition function δ such that $\delta: Q \times \Sigma \rightarrow Q$
  - A *starting state* $q_0 \in Q$
  - A set of *accepting states* $F \subseteq Q$



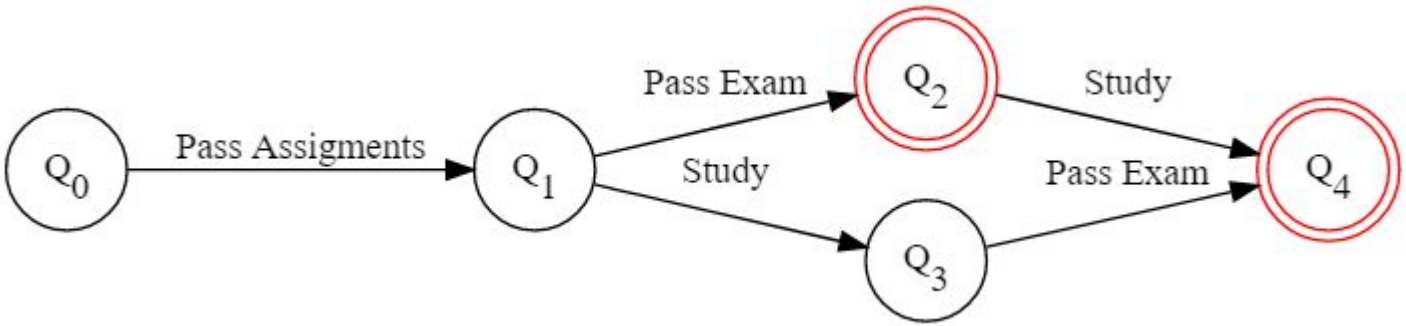
Finite State Automata

- One of many ways to formally model a process
- Consists of:
  - A finite set of states Q
  - An alphabet Σ
  - A transition function δ such that $\delta: Q \times \Sigma \rightarrow Q$
  - A *starting state* $q_0 \in Q$
  - A set of *accepting states* $F \subseteq Q$



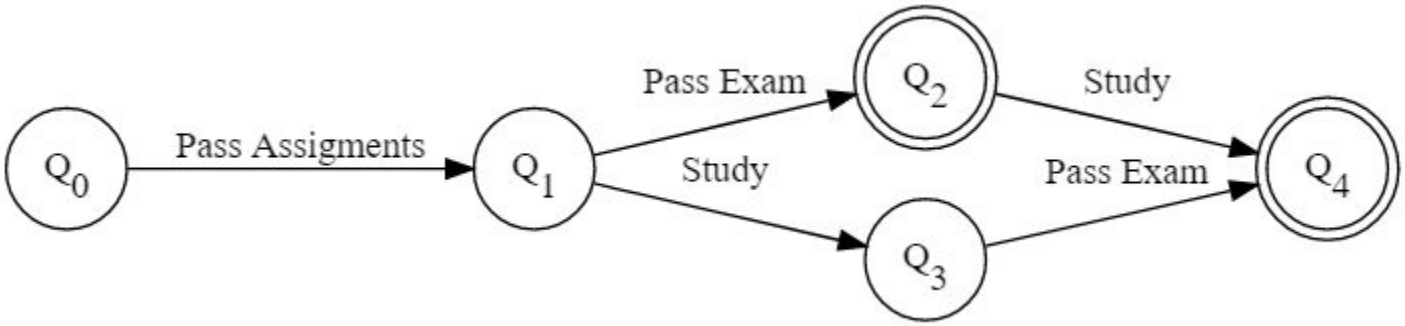
Finite State Automata

- One of many ways to formally model a process
- Consists of:
  - A finite set of states Q
  - An alphabet Σ
  - A transition function δ such that $\delta: Q \times \Sigma \rightarrow Q$
  - A *starting state* $q_0 \in Q$
  - A set of *accepting states* $F \subseteq Q$



Formal languages

Runs / executions:
The different ways we can execute the process,
consisting of the intermediate states and steps taken.

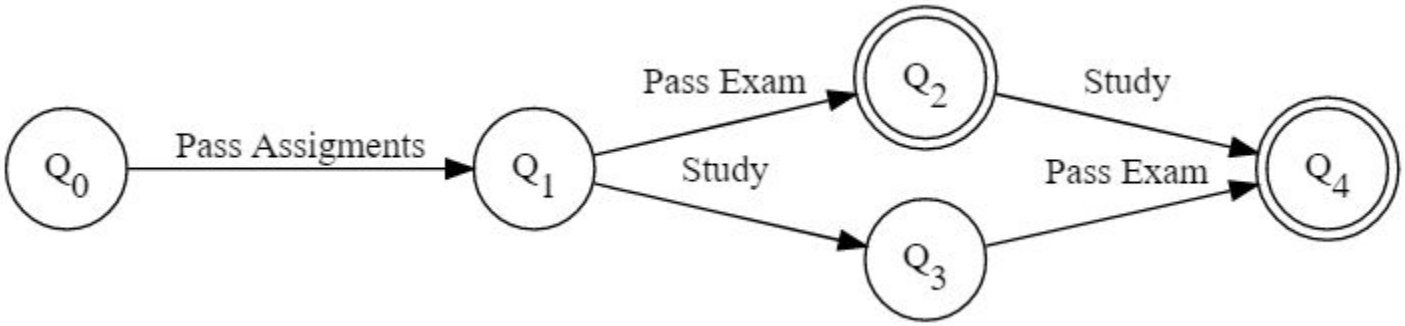


Runs / executions:

<u>Question</u>: What are the runs of this model?

Formal languages

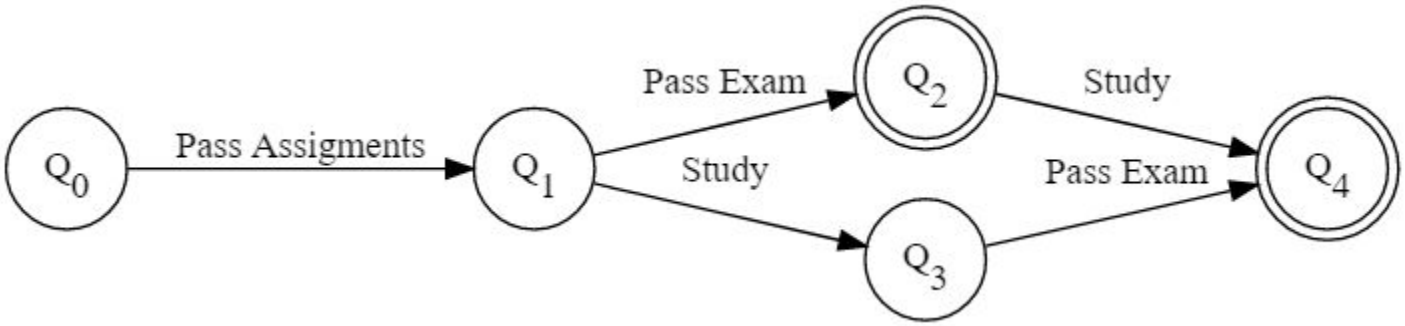
Runs / executions:
The different ways we can execute the process,
consisting of the intermediate states and steps taken.



Runs / executions:
Q0
Q0 —PA→ Q1
Q0 —PA→ Q1 —PE→ Q2
Q0 —PA→ Q1 —S→ Q3
Q0 —PA→ Q1 —PE→ Q2 —S→ Q4
Q0 —PA→ Q1 —S→ Q3 —PE→ Q4

Formal languages

Accepting runs / executions:
The different ways we can execute the process,
consisting of the intermediate states and steps taken,
which *satisfy the accepting condition*.

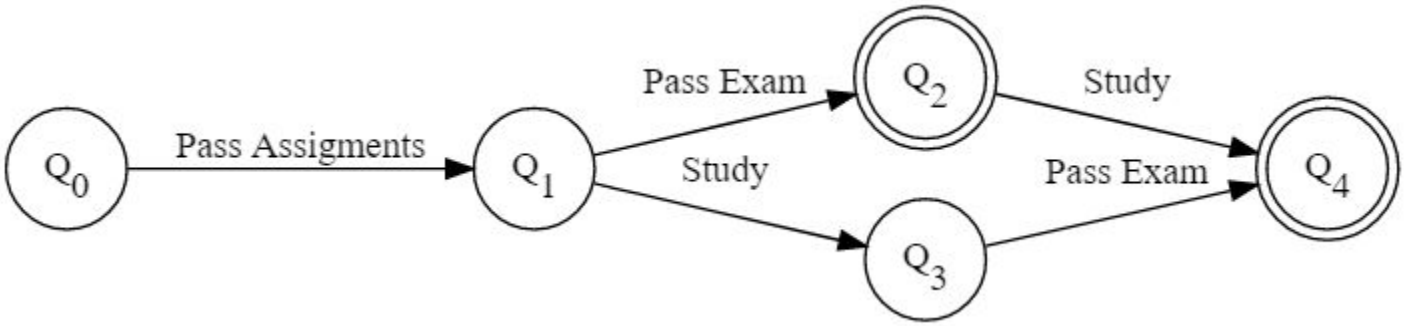


Accepting runs / executions:

<u>Question</u>: What are the accepting runs of this model?

Formal languages

Accepting runs / executions:
The different ways we can execute the process,
consisting of the intermediate states and steps taken,
which *satisfy the accepting condition*.

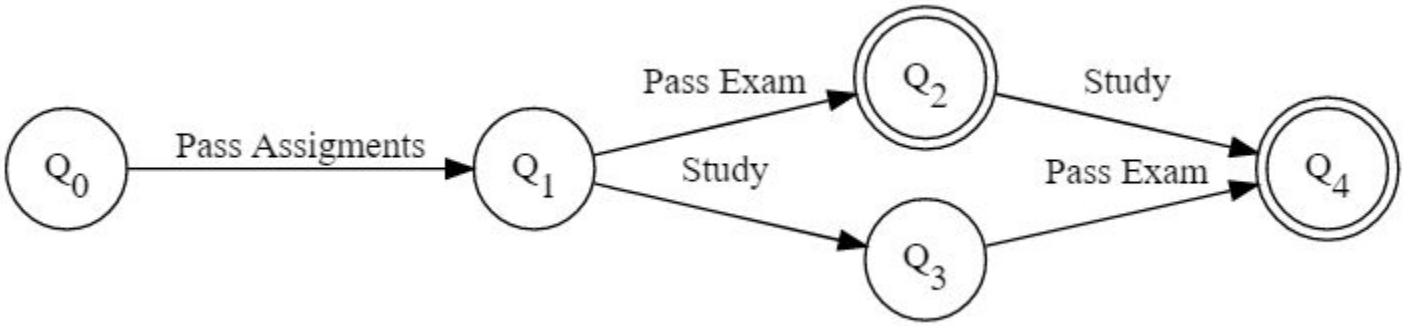


Accepting runs / executions:

Q0 —PA→ Q1 —PE→ Q2
Q0 —PA→ Q1 —PE→ Q2 —S→ Q4
Q0 —PA→ Q1 —S→ Q3 —PE→ Q4

Formal languages

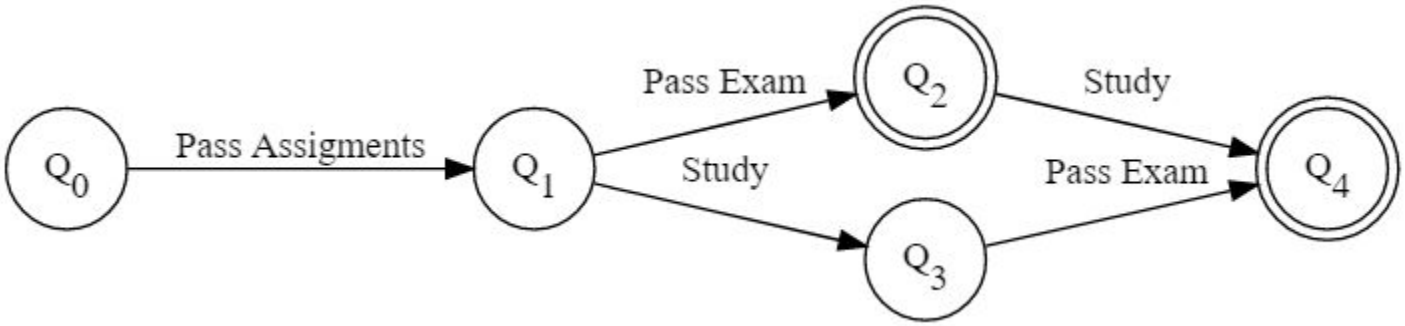
Traces:
The possible sequences of activities that the process can exhibit.



Traces:
<>
<PA>
<PA, PE>
<PA, PE, S>
<PA, S>
<PA, S, PE>

Formal languages

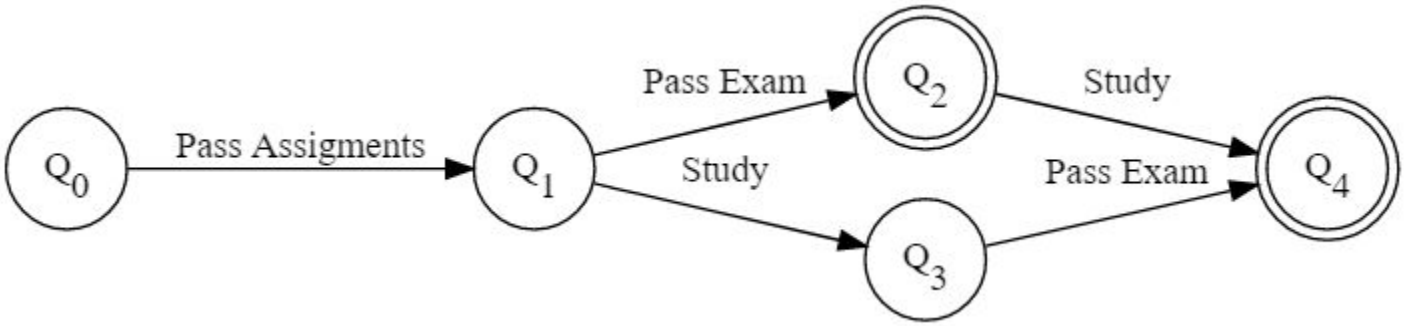
Accepting traces:
The possible sequences of activities that the process can exhibit which correspond to an accepting run of the process.



Accepting traces:
<PA, PE>
<PA, PE, S>
<PA, S, PE>

Formal languages

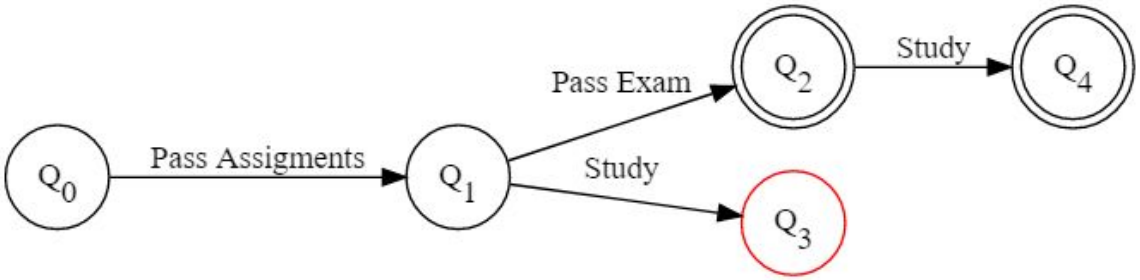
Language:
The set of all accepting traces.



Language:
{<PA, PE>,
<PA, PE, S>,
<PA, S, PE>}

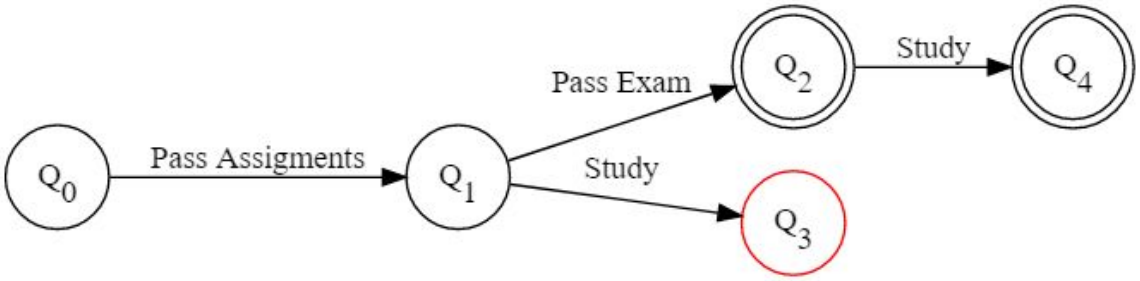
Deadlock & livelock

Deadlock:
A state in a system or process from which we can do *no further actions*, but which is *not accepting*.



Deadlock & livelock

Deadlock:
A state in a system or process from which we can do *no further actions*, but which is *not accepting*.

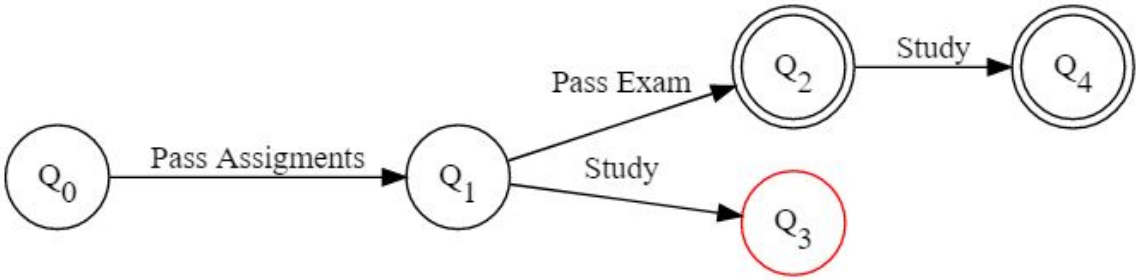


Livelock:
A state or set of states in a system or process, in which we *can still do actions*, but from which *no accepting run is possible*.

Question: How can we get from the model above to a model in livelock with one basic change?

Deadlock & livelock

Deadlock:
A state in a system or process from which we can do *no further actions*, but which is *not accepting*.



Livelock:
A state or set of states in a system or process, in which we *can still do actions*, but from which *no accepting run is possible*.