Politecnico
di Milano

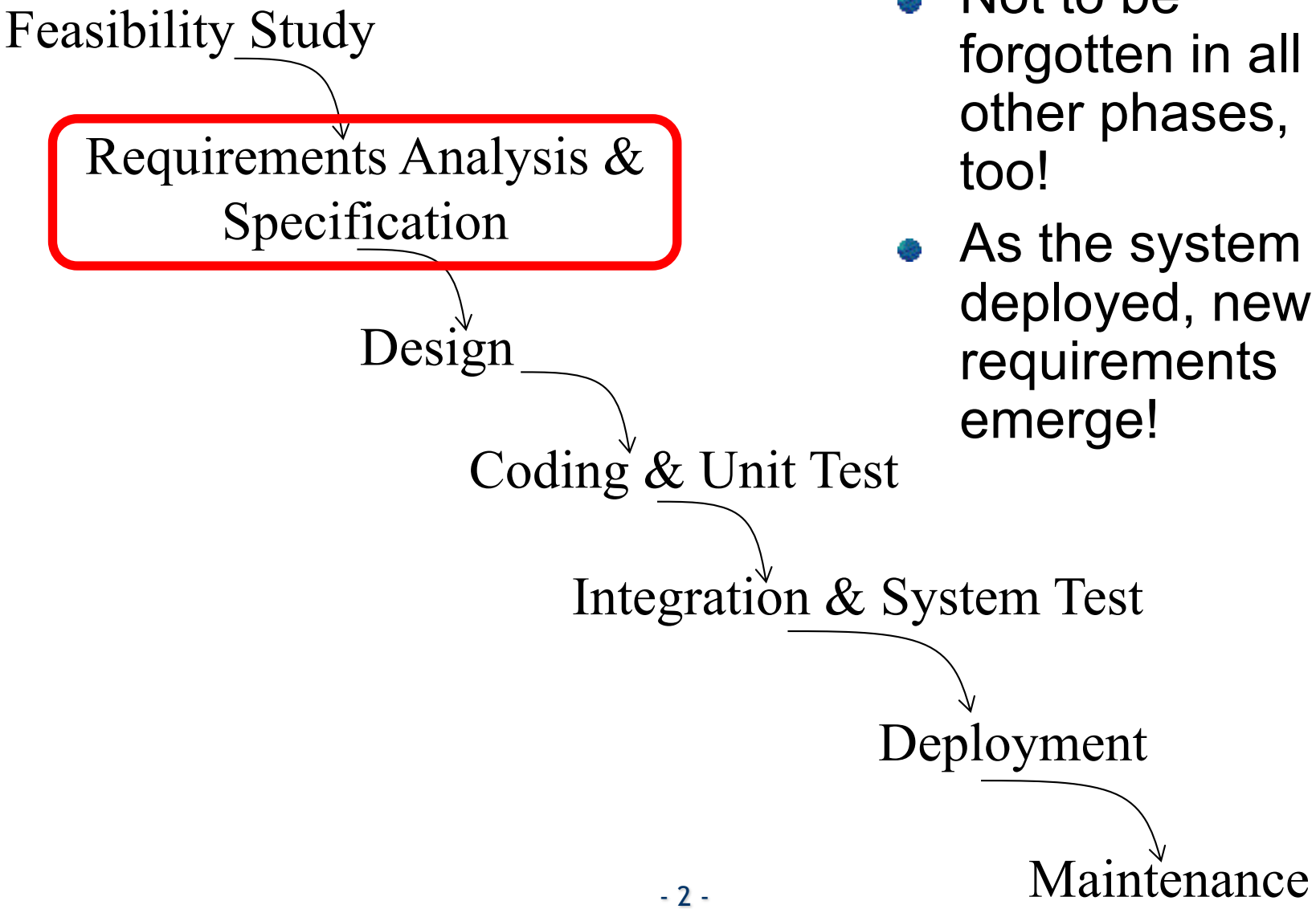
Requirements engineering (RE)

Context

Definitions

Importance and difficulties

Context: where do we find RE?



- Not to be forgotten in all other phases, too!
- As the system is deployed, new requirements emerge!

Requirements engineering: definition

- [Nuseibeh&Easterbrook ’00]
  - The primary measure of success of a software system is the degree to which it meets the purpose for which it was intended
  - Software systems requirements engineering (RE) is the process of discovering that purpose, by identifying stakeholders and their needs, and documenting these in a form that is amenable to analysis, communication, and subsequent implementation
    - Important issues
      - *Identify stakeholders*
      - *Identify their needs*
      - *Produce documentation*
      - *Analyse, communicate, implement requirements*

Requirement engineering: definition

- From Zave ‘83
  - Requirements engineering is the branch of software engineering concerned with the
    - real-world goals for,
    - functions of, and
    - constraints on
  - software systems!
  - It is also concerned with the relationship of these factors to **precise specifications of software behaviour, and to their evolution over time and across software families**

What is a requirement?

- Examples of candidate requirements
  - “The system shall allow users to reserve taxis”
  - “The system has to provide a feedback in 5 seconds”
  - “The system should never allow non-registered users to see the list of other users willing to share a taxi”
  - “The system should be available 24/7”
  - “The system should guarantee that the reserved taxi picks the user up”
  - “The system should be implemented in Java”
  - “The search for the available taxi should be implemented in class Controller”

Functional and non-functional requirements

Types of requirements

- **Functional** requirements:
  - Describe the interactions between the system and its environment independent from implementation
  - Examples:
    - A word processor user should be able to search for strings in the text
    - “The system shall allow users to reserve taxis”
  - Are the main goals the software to be has to fulfill
- **Nonfunctional** requirements:
  - User visible aspects of the system not directly related to functional behavior
  - Examples:
    - The response time must be less than 1 second
    - The server must be available 24 hours a day
- **Constraints** (“pseudo requirements”):
  - Imposed by the client or the environment in which the system operates
    - The implementation language must be Java
    - The credit card payment system must be able to be dynamically invoked by other systems relying on it

Characteristics of nonfunctional requirements

- Constraints on how functionality has to be provided to the end user
- Independent of the application domain
- … but the application domain determines
  - Their relevance
  - Their prioritization
- Have a strong impact on the structure of the system to be
  - Example: if a system has to guarantee to be available 24 hours per day, it is likely to be thought as a (at least partially) replicated system
- Also called **Quality of Service (QoS)** attributes

Some relevant QoS characteristics

- Performance
- Reliability
- Scalability
- Capacity
- Accuracy
- Accessibility
- Availability

Externally visible properties

- Robustness
- Exception handling

How the system works in unexpected/fault conditions

- Interoperability
- ....

Systems developed with different frameworks can work together at run time

- Integrity
- Confidentiality
- ...

Security issues

Examples of bad requirements

- The system shall validate and accept credit cards and cashier's checks…high priority
- The system shall process all mouse clicks very fast to ensure users do not have to wait
- The user must have Adobe Acrobat installed

Examples of bad requirements

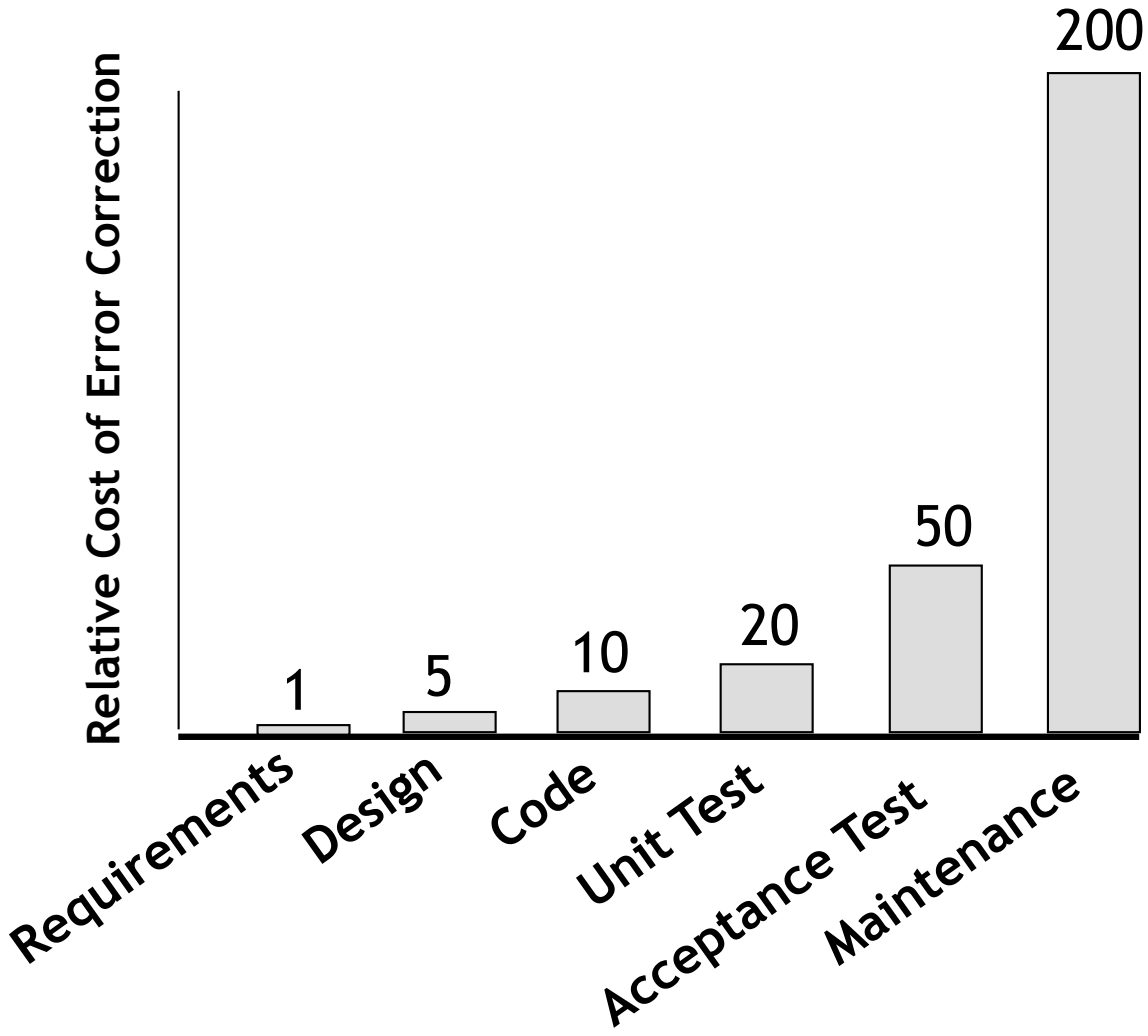
- The system shall validate and accept credit cards and cashier's checks…high priority
  - **Problem**: two requirements instead of one
  - If the credit card processing works, but the cashier's check validation does not… is this requirement pass or fail? Has to be fail, but that is misleading
  - Maybe only credit cards are high priority and cashier's checks are low priority
- The system shall process all mouse clicks very fast to ensure user's do not have to wait
  - **Problem**: this is not testable…quantify how fast is acceptable?
- The user must have Adobe Acrobat installed
  - **Problem**: this is not something our system must do
  - It could be in the constraints/assumptions or maybe operating environment sections, but it is not a functional requirement of our system

The requirements engineering process

Issues concerning RE

- Poor requirements are ubiquitous ...
  - “The system should guarantee that the reserved taxi picks the user up”: is this reasonable?
  - ”[…] requirements need to be *engineered* and have continuing review & revision” (Bell & Thayer, empirical study, 1976)
- RE is hard & critical ….
  - ”[…] hardest, most important function of SE is the iterative **extraction & refinement** of requirements" (F. Brooks, 1987)
  - Does the list in the previous slide capture all needs of the customer?
- Prohibitive cost of late correction …
  - "up to 200 x cost of early correction” (Boehm, 1981)

Cost of late correction (Boehm, 1981)



The cost of correcting an error depends on the number of subsequent decisions that are based on it

Errors made in understanding requirements have the potential for greatest cost, because many other design decisions depend on them

Impact of RE

- Survey of US software projects by Standish Group

	1994	1998
Successful	**16%**	**26%**
Challenged	**53%**	**46%**
Cancelled	**31%**	**28%**

	Successful	Challenged	Cancelled
1.	**User involvement**	**Lack of user input**	**Incomplete requirements**
2.	**Executive management support**	**Incomplete requirements**	**Lack of user input**
3.	**Clear statement of requirements**	**Changing requirements**	**Lack of resources**

What makes RE so complex ? (1)

- Broad scope
  - **Composite** systems: human organizations + physical devices + software components
  - **More than one system**: system-as-is, alternative proposals for system-to-be, system evolutions, product family
  - **Multiple abstraction levels**: high-level goals, operational details

(slide adapted from A. van Lamsweerde)

What makes RE so complex ? (2)

- **Multiple concerns**
  - functional, quality, development
  - hard and soft concerns

  → *conflicts*

- **Multiple stakeholders** with **different background**
  - clients, users, domain experts, developers, ...

  → *conflicts*

(slide adapted from A. van Lamsweerde)

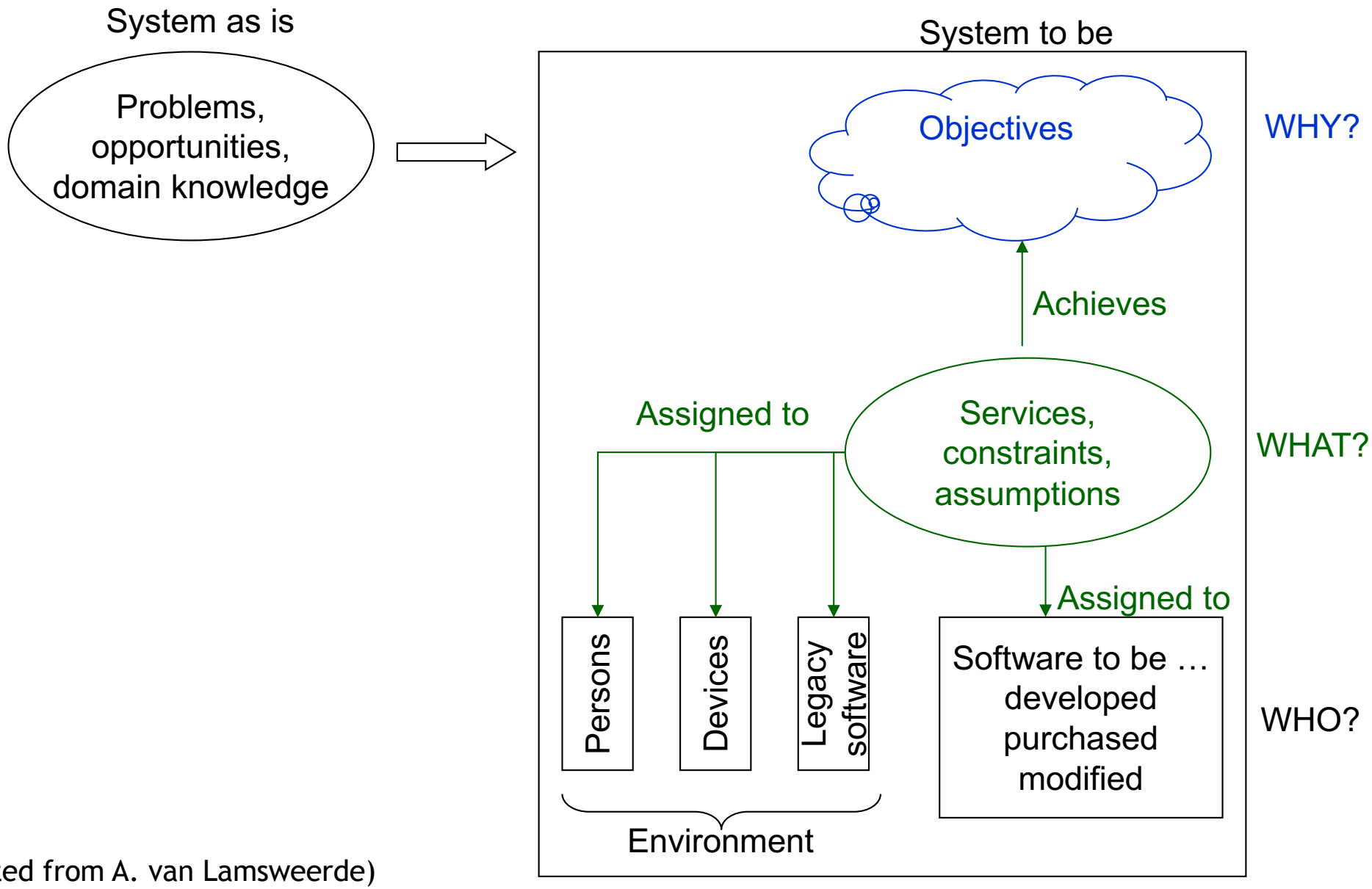
What do requirements engineers do? (1)

- Eliciting Information
  - Project objectives, context and scope
  - Domain knowledge and requirements
- Modelling and analysis
  - Goals, objects, use cases, scenarios, ...
- Communicating requirements
  - Analysis feedback, RASD document, system prototypes, ...

What do requirements engineers do? (2)

- Negotiating and agreeing Requirements
  - Handling conflicts and risks
  - Helping in requirement selection and prioritization
- Managing and evolving Requirements
  - Managing requirements during development: backward and forward traceability
  - Managing requirements changes and their impacts

The dimensions of RE



(adapted from A. van Lamsweerde)

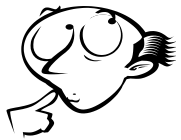
Role of requirements models

stakeholders

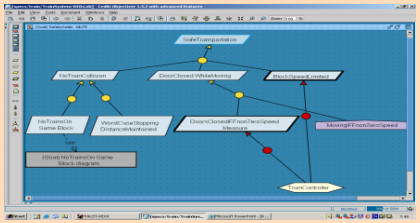
existing systems

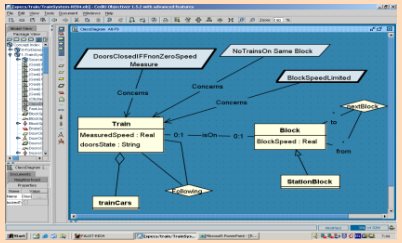
documents

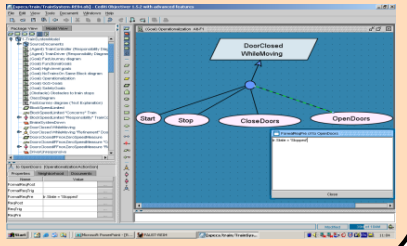
elicitation & modelling



Requirements Models

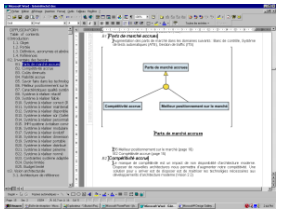






analysis & validation

generation of RE deliverables



requirements document

Understanding phenomena and requirements: the World and the Machine

The World and the Machine
(M. Jackson & P. Zave, 1995)

- Terminology
  - The **machine** = the portion of system to be developed
    typically, software-to-be + hardware
  - The **world** (a.k.a. the environment) = the portion of the real-world affected by the machine
- The purpose of the machine is always in the world

Examples

- An Ambulance Dispatching System
- A Banking Application
- A Word Processor
- ...

Example

- For every urgent call reporting an incident, an ambulance should arrive at the incident scene within 14 minutes
- For every urgent call, details about the incident are correctly encoded
- When an ambulance is mobilized, it will reach the incident location in the shortest possible time
- Accurate ambulance locations are known by GPS
- Ambulance crews correctly signal ambulance availability through mobile data terminals on board the ambulances

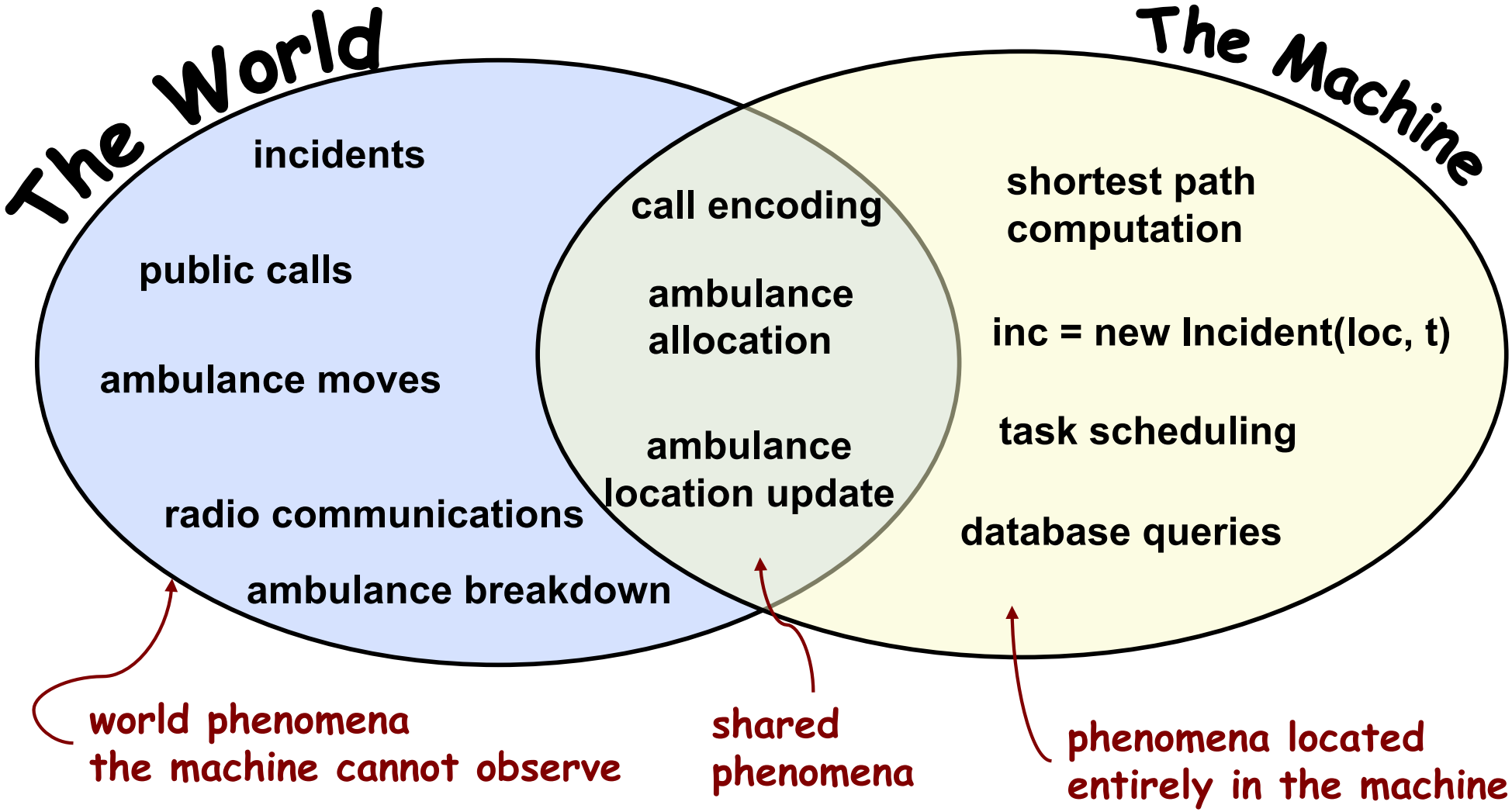
World and machine phenomena

- Requirements engineering is concerned with phenomena **occurring in the world**
  - For an ambulance dispatching system
    - the occurrences of incidents
    - the report of incidents by public calls
    - the encodings of calls details into the dispatching software
    - the allocation of an ambulance
    - the arrival of an ambulance at the incident location
- As opposed to phenomena **occurring inside the machine**
  - For the same ambulance dispatching system
    - the creation of a new object of class `Incident`
    - the update of a database entry
- **Requirements models are models of the world!**

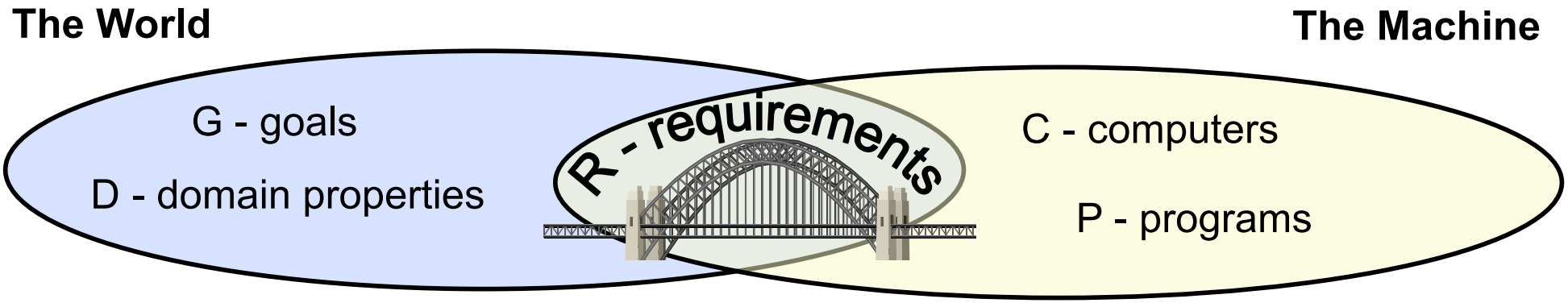
Shared phenomena

- Some world phenomena are **shared** with the machine
- Shared phenomena can be
  - **controlled** by the world and **observed** by the machine
    - e.g., the encodings of calls details into the dispatching software
  - or **controlled** by the machine and **observed** by the world
    - e.g., the allocation of an ambulance to an incident

The ambulance dispatching system



Goals, domain assumptions, and requirements



- **Goals** are **prescriptive assertions** formulated in terms of world phenomena (not necessarily shared)
- **Domain properties**/assumptions are **descriptive assertions assumed to hold** in the world
- **Requirements** are **prescriptive assertions** formulated in terms of **shared** phenomena

Goals, domain assumptions, and requirements

- **Goal:**
  - *'For every urgent call reporting an incident, an ambulance should arrive at the incident scene within 14 minutes'*
- **Domain assumptions:**
  - *For every urgent call, details about the incident are correctly encoded*
  - *When an ambulance is mobilized, it will reach the incident location in the shortest possible time*
  - *Accurate ambulances' locations are known by GPS*
  - *Ambulance crews correctly signal ambulance availability through mobile data terminals on board of ambulances*
- **Requirement:**
  - *When a call reporting a new incident is encoded, the Automated Dispatching Software should mobilize the nearest available ambulance according to information available from the ambulances' GPS and mobile data terminals*

Requirements completeness?

- The requirements R are **complete** if
  1. **R** ensures satisfaction of the goals **G** in the context of the domain properties **D**

     R and D ⊨ G

     Analogy with program correctness: a Program P running on a particular Computer C is correct if it satisfies the Requirements R

     P and C ⊨ R
  2. **G** adequately capture all the stakeholders' needs
  3. **D** represents valid properties/assumptions about the world

Politecnico
di Milano

What can go wrong when defining G, R, D?

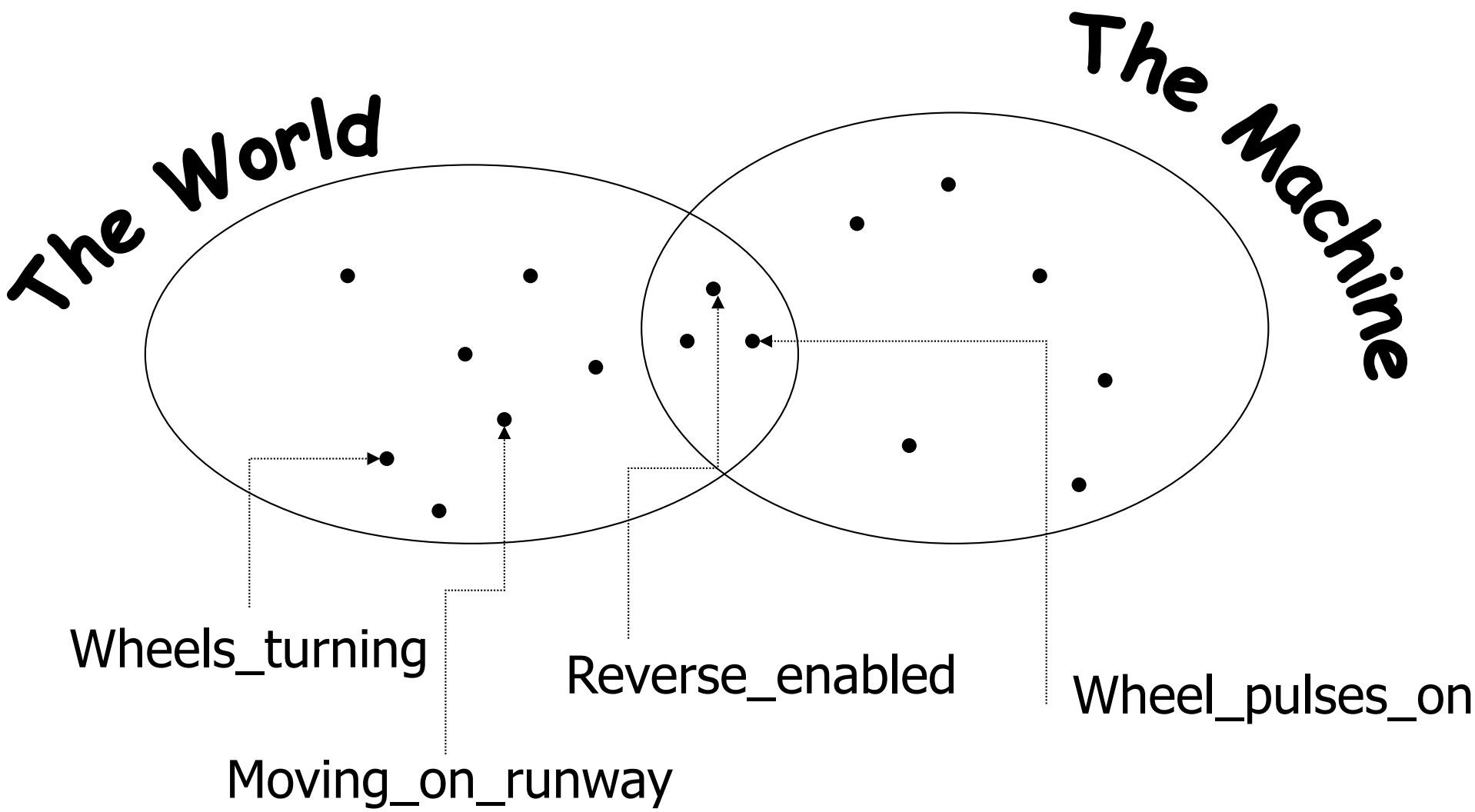
The A320 example

Example – Airbus A320

https://aviation-safety.net/database/record.php?id=19930914-2

- A Lufthansa Airbus on a flight from Frankfurt landed at Warsaw Airport in bad weather (rain and wind)
- On landing, the aircraft's software-controlled braking system did not deploy when activated by the flight crew and it was about 9 seconds before the braking system activated
- There was insufficient runway remaining to stop the plane and the aircraft ran into a grass embankment
- Two people were killed and 54 injured
- Several causes:
  - Human errors
  - Software errors (braking control system)

Example – Airbus A320 Braking Logic



Goal, domain assumptions, and requirement

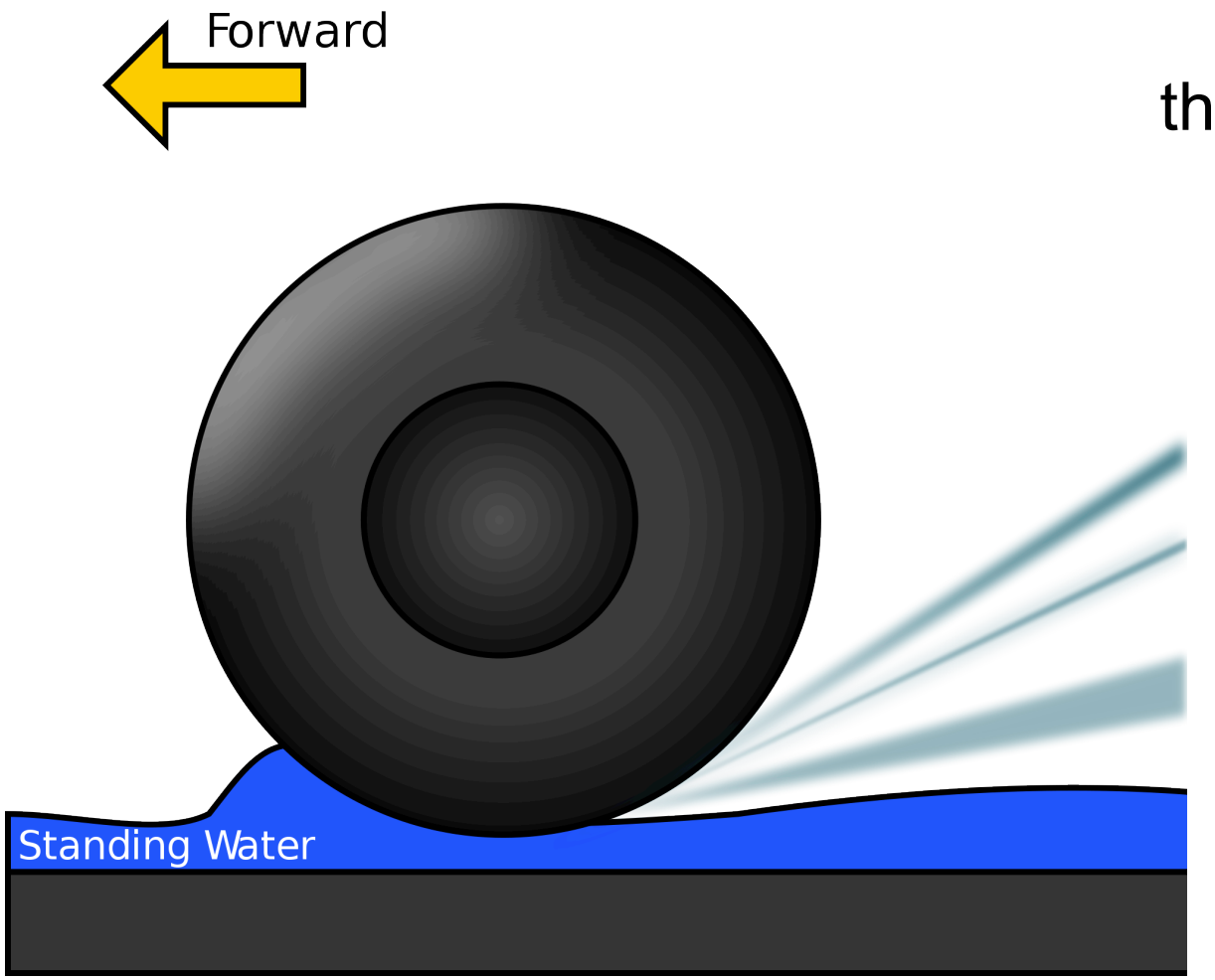
- Goal G:
  - “Reverse thrust shall be enabled if and only if the aircraft is moving on the runway”
- Domain Assumptions D:
  - Wheel pulses on if and only if wheels turning
  - Wheels turning if and only if moving on runway
- Requirements R:
  - Reverse thrust enabled if and only if wheel pulses on
- Verification: R and D $\models$ G applies!

Correctness arguments

- Goal
  - Reverse_enabled $\Leftrightarrow$ Moving_on_runway
- Domain properties
  - Wheel_pulses_on $\Leftrightarrow$ Wheels_turning
  - Wheels_turning $\Leftrightarrow$ Moving_on_runway
- Requirements
  - Reverse_enabled $\Leftrightarrow$ Wheels_pulses_on
- can prove that $R \wedge D \models G$

- … but D are not valid assumptions!!!
- **Invalid domain assumptions -> Warsaw accident**

Ever heard of hydroplaning?



Domain assumptions D do not apply to reality, the correctness argument is bogus!

Incorrect domain assumptions lead to disasters!

Politecnico
di Milano

How to derive requirements from goals: An example

The turnstile example

(M. Jackson, P. Zave, "Deriving Specifications from Requirements: An Example", Proceedings of ICSE 95, 1995)

The turnstile control system

Goals

G1: At any time entries should never exceed accumulated payments (for simplicity, assume 1 coin for 1 entrance)

G2: Those who pay are not prevented from entering (by the "machine")

- They are optative descriptions
- Both are said to be **safety** properties
  - They state that nothing bad will ever occur

Domain analysis (1/2)

- Identify the relevant environment phenomena
  - here: **events**
- "Relevant" with respect to the goals
  - i.e., control people entrance

Domain analysis (2/2)

- **Enter**: puts turnstile back in home position
- **Coin**: coin inserted
- **Push**: frees turnstile
- **Lock**: } electric signals
- **Unlock**: }

environment controlled

machine controlled

shared phenomena

Domain assumptions

- They are real world properties
- They do not depend on the machine

D1: Push and Enter alternate, starting with Push
  - one cannot enter without first pushing
  - one cannot push until the previous visitor entered

D2: Push always leads to Enter
  - enforced by a hydraulics system

D3: If Locked, Push cannot occur

Deriving requirements from goals

- Must find the **constraints on shared phenomena** to be enforced by the machine to achieve the goals
  - G1: At any time, entries should never exceed accumulated payments
  - G1 can be enforced by controlling either entries or coins
    - the machine cannot compel Coin events
    - it can prevent Enter events

How to constrain Enter events?

D1: Push and Enter alternate (starting with Push)

- From D1 we derive (*):

(*) At any time t, if e Enter and p Push events were observed, then $p-1 \leq e \leq p$

- We get (**) by strengthening G1 via (*), by referring to shared phenomena:

(**) At any time t, if p Push and c Coin events were observed, then $p \leq c$

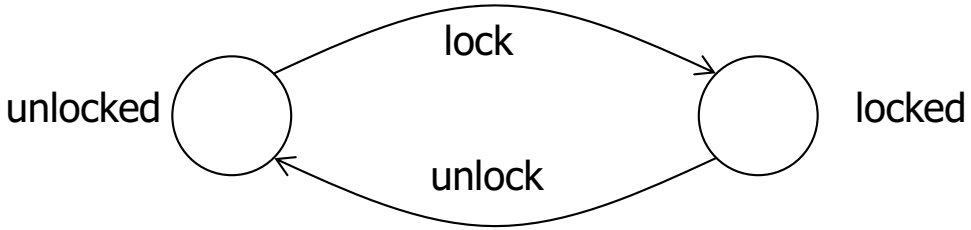
G1: At any time entries should never exceed accumulated payments ($e \leq c$)

if this can be enforced, then G1 holds!

How to ensure p ≤ c?

- When p=c, must prevent further Push until Coin event occurs, … but how can the machine prevent Push?

R1: Impose that Lock and Unlock alternate (initially locked)



R2:

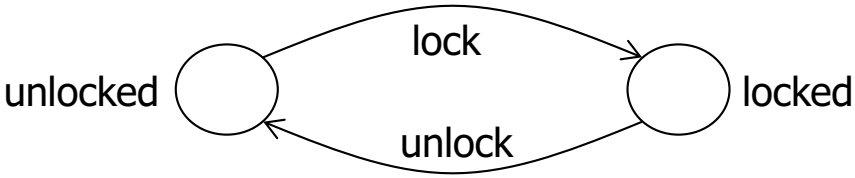
i. If locked and p=c, the machine must not unlock the turnstile, and

ii. If unlocked and p=c, the machine must perform a Lock in time to prevent further Push

Proof that G1 holds

G1: At any time entries should never exceed accumulated payments

- Need to prove that if p=c, no further p can occur if c does not change
- Two possible cases



locked (i.e., l=u+1)

one cannot push (according to R1, R2.i, and D3)

unlocked (i.e. l=u)

immediate lock prevents push (according to R1, R2.ii, and D3)

R1: Impose that Lock and Unlock alternate (initially locked)

D3: If Locked, Push cannot occur

R2.i: If locked and p=c, the machine must not unlock

R2.ii: If unlocked and p=c, the machine must perform a Lock in time to prevent further Push

Deriving requirements from goals

G2: Those who pay are not prevented from entering (by the "machine")

- May be transformed into a further requirement:

R3: i) If unlocked and p<c, the machine does not Lock as long as p<c

ii) If locked and p<c, the machine must perform an Unlock

Proof of G2

G2: Those who pay are not prevented from entering (by the "machine")

R3.i: If unlocked and p<c, the machine does not Lock as long as there is credit

- From R3.i, a new p can occur, hence an e (from D2 and D3)

R3.ii: If locked and p<c, the machine must perform Unlock event

- From R3.ii, we enter case R3.i immediately

D3: If Locked, Push cannot occur

D2: Push always leads to Enter

A warning about terminology

- RE is a relatively young domain, there's no consensus on terminology yet, in particular about what is a requirement
- In Jackson's work
  - what we call a goal is called a requirement
  - what we call a requirement is called a specification
- Other people (e.g., van Lamsweerde) also use the terms 'System Requirements' & 'Software Requirements'

Observation

- The boundary between the World & the Machine is generally not given at the start of a development project
- The purpose of a RE activity is
  - to identify the real goals of the project
  - to explore alternative ways to satisfy the goals, through
    - alternative pairs (Req, Dom) such that Req and Dom $\models$ G
    - alternative interfaces between the world and the machine
  - to evaluate the strengths and risks of each alternative, in order to select the most appropriate one

Exercise

- Consider the following requirements engineering problem for a pay-per-view system:
- The goal G can be expressed as follows:
  - “Only users who have paid for the service can access application SV (streaming video)”
- We can assume the following valid domain properties D (it is a domain property because it is guaranteed by an existing access control system):
  - “After submitting a payment, personal data and e-mail address, a user receives a password which authorizes access to SV”
- Assume the following requirement R for the “software-to-be”:
  - “Access to SV shall only be granted after a user types an authorized password”

Questions for you

1. Can you formalize such reasoning?
2. What kind of reasoning should you do to prove the correctness of the requirements?
3. What could possibly be wrong in this?