Today

• Introduction: Real-Time Systems

• Overview: content (and non-content) of the lecture

• Formalia: dates/times, exercises, exam admission

• Literature

Introduction

What is a Real-Time System?

Other example: Gas Burner

- gas valve
- flame sensor
- ignition

- Leakage is practically unavoidable:
  - for ignition, first open valve
  - then ignite the available gas
  - ignition may fail...

- Leakage is safety critical:
  Igniting large amounts of leaked gas may lead to a dangerous explosion.

When is a Gas Burner a Real-Time System?
No, Really, What is a Real-Time System?

The examples have in common that it matters when in time the output for a given input (sequence) takes place. For instance:

- "fire" 300 ms after "crash",
- within any interval of at least 60 s, leakage (≡ = have the gas valve open without a flame) amounts to at most 5% of the time.

Note: quantitative (here) vs. qualitative notions of time (untimed).

Often: There is a physical environment, which has a notion of time, and which evolves while our controller is computing.

(Contrast): vending machine for soft-drinks:

- If the customer is really thirsty, she'll wait.
- Neither the usage of a really fast or a really slow contemporary controller causes a violation of (timing) requirements.

(Real) Contrast: transformational systems, such as computing π.

Other Definitions

[Douglass, 1999]

- "A real-time system is one that has performance deadlines on its computations and actions."

Distinguish:

- "Hard deadlines: performance requirements that absolutely must be met each and every event or time mark." (Latedata can be bad data.)
- "Soft deadlines: for instance about average response times." (Latedata is still good.)

Design Goal:

- A timely system, i.e. one meeting its performance requirements.

Note: performance can in general be any unit of quantities:

- (discrete) number of steps or processor instructions,
- (discrete or continuous) number of seconds,
- etc.

Definitions: Reactive vs. Real-Time vs. Hybrid Systems

- Reactive Systems interact with their environment by reacting to inputs from the environment with certain outputs.
- A Real-Time System is a reactive system which, for certain inputs, has to compute the corresponding outputs within given time bounds.
- A Hybrid System is a real-time system consisting of continuous and discrete components. The continuous components are time-dependent (!) physical variables ranging over a continuous valueset.

A system is called Safety Critical if and only if a malfunction can cause loss of goods, money, or even life.

Reactive Systems

Real-Time Systems

Hybrid Systems

The Problem: Constructing Safety-Critical RT Systems

Reactivesystems can be partitioned into:

- plant
- sensors
- actuators
- controller

"In constructing a real-time system, the aim is to control a physically existing environment, the plant, in such a way that the controlled plant satisfies all desired (timing) requirements."

The design of safety critical (reactive) systems requires a high degree of precision: We want—at best—to be sure that a design meets its requirements.

Real-time systems are often safety-critical.

The lecture presents approaches for the precise development of real-time systems based on formal, mathematical methods.

Constructing Safety-Critical RT Systems: Examples

Controller

- crash
- fire

"When a crash is detected at time \( t \), fire the airbag at \( t + 300 \text{ ms} \pm \epsilon \)."

A controller program is easy:

```c
while (true) do
  poll_sensors();
  if (crash) tmr.start(300ms);
  if (tmr.elapsed()) fire := 1;
  update_actuators();
od
```

And likely to be believed to be correct.

Constructing Safety-Critical RT Systems: Examples

More complicated:

- additional features.

More complicated:

- distributed implementation.

Sens

Controller

Act
Leakage is safety-critical: Igniting large amounts of leaked gas may lead to a dangerous explosion.

Controller program for ignition:

```
while (!flame) do
  open_valve(); wait(t); ignite();
od
```

Is it correct? (Here: Is it avoiding dangerous explosions?)

Prerequisites:
- plant
- sensors
- actuators
- controller

To design a controller that meets its requirements, we need:
- a formal model of behaviour in (quantitative) time,
- a language to concisely, conveniently specify requirements on behaviour,
- a language to specify behaviour of controllers,
- a notion of "meet" and a methodology to verify "meeting".

Then we can devise a methodology to get from requirements to a (correct) implementation — here: following [Olderog and Dierks, 2008].

**Sketch of the Methodology: Gas Burner Example**

- **Requirements**
  - At most 5% of any at least 60 s long interval amounts to leakage.
  - Reflective Design
    - Time intervals with leakage last at most 1 s.
    - After each leakage, wait 30 s before opening valve again.
  - Constructive Design
    - PLC Automaton
      - (open valve for 0.5 s; ignite; if no flame after 0.1 s close valve)
  - Implementation
    - IEC 61131-3 program

**Content Overview**

- Introduction
  - First-order Logic
  - Duration Calculus (DC)
  - Semantical Correctness: Proof with DC
  - DC Decidability
  - DC Implementables
    - PLC Automata
  - Timed Automata (TA), Uppaal
  - Networks of Timed Automata
  - Region/Zone Abstraction
  - Extended Timed Automata
  - Undecidability Results

- Automatic Verification

- Recap

- Tying It All Together
Recallover-simplified airbag controller:

```c
while (true) do
    poll_sensors();
    if (crash) tmr.start(300ms); 
    if (tmr.elapsed()) fire := 1;
    update_actuators();
od
```

• The execution of `poll_sensors()` and `update_actuators()` also take time! (And we have to consider it!)

• Maybe in lecture: How to determine the WCET of, for instance, C code. (A science of its own.)

Scheduling

• Recallover-simplified airbag controller:

Controller

Actuator

• Not in lecture: Specialised methods to determine...

• ...whether the bus provides sufficient bandwidth.

• ...whether the Real-Time OS controlling CPU's controller schedules the airbag control code in time.

• ...how to distribute tasks over multiple CPUs.

• etc. (Also a science of its own.)
Formalia: Exercises and Tutorials

Recall: exercises online on Thursday before lecture, regular turn in on corresponding tutorial day until 10:00 local time.

Should working in groups of max. 3, clearly give names on submission.

Please submit electronically by mail to me (cf. homepage); LaTeX styles on homepage; paper submissions are tolerated.

Didactical aim:
- Deal more extensively with notions from lecture (easy)
- Explore corner cases or alternatives (medium)
- Evaluate/appreciate approaches (difficult)
- Additional difficulty: imprecise/unclear tasks—by intention

True aim: most complicated rating system ever, namely two ratings:
- Good-will: "reasonable solution with knowledge before tutorial"
- Evil/Exam: "reasonable solution with knowledge after tutorial"

10% bonus for early submission.

Exam Admission:
50% of the maximum possible non-bonus good-will points in total are sufficient for admission to exam.

Exam Form: (oral or written) not yet decided.

Mid-term Evaluation:
We will have a mid-term evaluation, but we're always interested in comments/hints/proposals concerning form or content.

That is, students are asked to evaluate lecture, lecturer, and tutor ...

State Variables (or Observables)
We assume that the real-time systems we consider is characterised by a finite set of state variables (or observables) \( \text{obs}_1, \ldots, \text{obs}_n \) each equipped with a domain \( D(\text{obs}_i) \), \( 1 \leq i \leq n \).

Example:
- Gas Burner
  - \( G: \text{Time} \rightarrow \{0, 1\} \)
  - \( F: \text{Time} \rightarrow \{0, 1\} \)
  - \( I: \text{Time} \rightarrow \{0, 1\} \)
  - \( H: \text{Time} \rightarrow \{0, 1\} \)

Real-Time Behaviour, More Formally...

Formalia: Evaluation

Speaking of grading and examination ...

Formalia: Questions?
System Evolution over Time

One possible evolution (or behaviour) of the considered system over time is represented as a function \( \pi : \text{Time} \rightarrow D(\text{obs}_1) \times \cdots \times D(\text{obs}_n) \).

If (and only if) observable \( \text{obs}_i \) has value \( d_i \in D(\text{obs}_i) \) at time \( t \in \text{Time} \), \( 1 \leq i \leq n \), we set:

\[
\pi(t) = (d_1, \ldots, d_n).
\]

For convenience, we use \( \text{obs}_i : \text{Time} \rightarrow D(\text{obs}_i) \) to denote the projection of \( \pi \) onto the \( i \)-th component.

There are two main choices for the time domain \( \text{Time} \):

- Discrete time: \( \text{Time} = \mathbb{N}_0 \), the set of natural numbers.
- Continuous or ordered time: \( \text{Time} = \mathbb{R}_+ \), the set of non-negative real numbers.

Throughout the lecture, we shall use the continuous time model and consider discrete time as a special case.

Because:

- Plant models usually live in continuous time,
- We avoid too early introduction of hardware considerations,
- An interesting view: continuous-time is a well-suited abstraction from the discrete-time realms induced by clock cycles etc.

Example: Gas Burner

One possible evolution of the considered system over time is represented as function \( \pi : \text{Time} \rightarrow D(\text{obs}_1) \times \cdots \times D(\text{obs}_n) \).

If (and only if) observable \( \text{obs}_i \) has value \( d_i \in D(\text{obs}_i) \) at time \( t \in \text{Time} \),\( 1 \leq i \leq n \), we set:

\[
\pi(t) = (d_1, \ldots, d_n).
\]

For convenience, we use \( \text{obs}_i : \text{Time} \rightarrow D(\text{obs}_i) \) to denote the projection of \( \pi \) onto the \( i \)-th component.

References
