Introduction to the Course

Lecturer: Luca Aceto
Email: luca@ru.is or luca.aceto@gmail.com
Course web page: http://www.ru.is/faculty/luca/IMTCOURSE/
Focus of the Course

- Study of mathematical models for the formal description and analysis of programs.
- Study of formal languages for the specification of program behaviour.
- Particular focus on parallel and reactive systems.
- Verification tools and their use in the analysis of system designs.
Tentative Overview

- Transition systems and CCS.
- Strong and weak bisimilarity, bisimulation games.
- Hennessy-Milner logic and bisimulation.
- Tarski’s fixed-point theorem (possibly).
- Hennessy-Milner logic with recursively defined formulae.
- Timed automata and their semantics.
- One group project.
- More advanced topics may be covered depending on how the course develops.
Putting the theory and tools into practice!

**Two possibilities (to be taken with a pinch of salt)**

- Modelling of a solitaire game in CWB.
- Solving Rush Hour games using UPPAAL.

The project counts for 40% of the final mark for the course.
There will be lectures for three weeks.
Ask/answer questions. Be active!
Slides will be available before each lecture.
Exercises

- I will regularly post exercise sheets.
- Suggestion: Work on the exercises in groups of two-three people.
- I will post solutions to (selected) exercises for each exercise sheet.
Exam and Literature

Individual Oral Exam = Celebration!
The oral exam counts for 60% of the final mark.

Literature

- Best Reader Competition with award!
Check regularly the course web-page.

Be an active participant!

Work on the exercises.

Take your own notes.

“I hear and I forget. I see and I remember. I do and I understand.” (Confucius, 551 BC–479 BC)
Aims of the Course

Present a general theory of reactive systems and its applications. The theory supports:

- Design.
- Specification.
- Verification (possibly automatic and compositional).

Aims

1. Give the students practice in modelling parallel systems in a formal framework.
2. Give the students skills in analyzing behaviours of reactive systems.
3. Introduce algorithms and tools based on the modelling formalisms.
Aims of the Course

Present a general theory of **reactive systems** and its applications. The theory supports:

- Design.
- Specification.
- Verification (possibly automatic and compositional).

**Aims**

1. Give the students practice in modelling parallel systems in a formal framework.
2. Give the students skills in analyzing behaviours of reactive systems.
3. Introduce algorithms and tools based on the modelling formalisms.
Aims of the Course

Present a general theory of reactive systems and its applications. The theory supports:

- Design.
- Specification.
- Verification (possibly automatic and compositional).

Aims

1. Give the students practice in modelling parallel systems in a formal framework.
2. Give the students skills in analyzing behaviours of reactive systems.
3. Introduce algorithms and tools based on the modelling formalisms.
Present a general theory of reactive systems and its applications. The theory supports:

- Design.
- Specification.
- Verification (possibly automatic and compositional).

Aims

1. Give the students practice in modelling parallel systems in a formal framework.
2. Give the students skills in analyzing behaviours of reactive systems.
3. Introduce algorithms and tools based on the modelling formalisms.
Characterization of a “Classic” Program

A program transforms an input into an output.

- Denotational semantics:
  the meaning of a program is a partial function

\[ \text{states} \xrightarrow{\text{states}} \]

- Nontermination is bad!
- In case of termination, the result is unique.

Is this all we need?
Classic View

Characterization of a “Classic” Program
A program transforms an input into an output.

- Denotational semantics:
  the meaning of a program is a partial function

  \[ \text{states} \rightarrow \text{states} \]

- Nontermination is bad!
- In case of termination, the result is unique.

Is this all we need?
What about:
- Operating systems?
- Communication protocols?
- Control programs?
- Mobile phones?
- Vending machines?
Reactive systems

Characterization of a Reactive System

**Reactive System** = system that computes by reacting to stimuli from its environment.

Key Issues:
- communication and interaction
- parallelism

Nontermination is good!

The result (if any) does not have to be unique.
Characterization of a Reactive System

Reactive System = system that computes by reacting to stimuli from its environment.

Key Issues:

- communication and interaction
- parallelism

Nontermination is good!

The result (if any) does not have to be unique.
Questions

- How can we develop (design) a system that "works"?
- How do we analyze (verify) such a system?

Fact of Life

Even short parallel programs may be hard to analyze.
The Need for Theory

Conclusion

We need formal/systematic methods (tools), otherwise ...

- Intel’s Pentium-II bug in floating-point division unit
- Ariane-5 crash due to a conversion of 64-bit real to 16-bit integer
- Mars Pathfinder
- ...

Intel's Pentium-II bug in floating-point division unit
Ariane-5 crash due to a conversion of 64-bit real to 16-bit integer
Mars Pathfinder
...
### Classic vs. Reactive Computing

<table>
<thead>
<tr>
<th></th>
<th>Classic</th>
<th>Reactive/Parallel</th>
</tr>
</thead>
<tbody>
<tr>
<td>interaction</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>nontermination</td>
<td>undesirable</td>
<td>often desirable</td>
</tr>
<tr>
<td>unique result</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>semantics</td>
<td>states $\rightarrow$ states</td>
<td>?</td>
</tr>
</tbody>
</table>
How to Model Reactive Systems

Question
What is the most basic view of a reactive system (process)?
How to Model Reactive Systems

Question

What is the most basic view of a reactive system (process)?

Answer

A process performs an action and becomes another process.
A labelled transition system (LTS) is a triple

\[(\text{Proc}, \text{Act}, \{\xrightarrow{a}\mid a \in \text{Act}\})\]

where

- \(\text{Proc}\) is a set of states (or processes),
- \(\text{Act}\) is a set of labels (or actions), and
- \(\xrightarrow{a} \subseteq \text{Proc} \times \text{Proc}\) is a binary relation on states called the transition relation, for each \(a \in \text{Act}\).

We will use the infix notation \(s \xrightarrow{a} s'\) meaning that \((s, s') \in \xrightarrow{a}\).

Sometimes we distinguish an initial (or start) state.
LTSes describe process behaviour, and explicitly focus on interaction.

The Motto (after Tony Hoare and Robin Milner)

Everything is (or can be viewed as) a process!

Buffers, shared memory, Linda tuple spaces, senders, receivers, ... are all agents/processes.
Let \((Proc, Act, \{\xrightarrow{a} \mid a \in Act\})\) be an LTS.

- We extend \(\xrightarrow{a}\) to the elements of \(Act^*\).
- \(\xrightarrow{\ast} = \bigcup_{a \in Act} \xrightarrow{a}\)
- \(\xrightarrow{\ast}\) is the reflexive and transitive closure of \(\xrightarrow{}\). (Do you know what this means?)
- \(s \xrightarrow{a}\) and \(s \xrightarrow{a'}\).
- Reachable states.