

# Semantics and Verification 2008

## Lecture 1

Lecturer (1-8): Jiri Srba      srba@cs.aau.dk

Lecturer (9-15): Kim G. Larsen      kgl@cs.aau.dk

Assistant: Bjørn Haagenen      bh@cs.aau.dk

- Study of mathematical models for the formal description and analysis of programs.
- Particular focus on parallel and reactive systems.
- Verification tools and implementation techniques underlying them.

# Overview of the Course

- Transition systems and CCS.
- Strong and weak bisimilarity, bisimulation games.
- Hennessy-Milner logic and bisimulation.
- Tarski's fixed-point theorem.
- Hennessy-Milner logic with recursively defined formulae.
- Timed CCS.
- Timed automata and their semantics.
- Binary decision diagrams and their use in verification.
- Two mini projects.

- Verification of a communication protocol in CWB.
- Verification of a real-time algorithm in UPPAAL.
  
- Pensum dispensation.

- Ask questions.
- Take your own notes.
- Read the recommended literature as soon as possible after the lecture.

- Regularly before each lecture.
- Supervised peer learning.
- Work in groups of 2 or 3 people.
- **Print out the exercise list**, bring literature and your notes.
- Feedback from teaching assistant on your request.
- **Star exercises (\*)** (part of the exam).

- Individual and oral.
- Preparation time (solving one selected star exercise).
- Pensum dispensation.

- Book “Reactive Systems: Modelling, Specification and Verification” by L. Aceto, A. Ingólfssdóttir, K.G. Larsen and J. Srba. Available in the local bookshop at Fredrik Bajersvej 7B.
- On-line literature.



- Check regularly the course web-page.
- **Anonymous feedback form** on the course web-page.
- Attend and actively participate during tutorials.
- Take your own notes.

Present a general theory of reactive systems and its applications.

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

Program transforms an input into an output.

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

$$states \mapsto states$$

- **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?

## Characterization of a Reactive System

**Reactive System** is a 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.

## 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
- ...

# Classical vs. Reactive Computing

	Classical	Reactive/Parallel
interaction	no	yes
nontermination	undesirable	often desirable
unique result	yes	no
semantics	$states \hookrightarrow states$	?



# How to Model Reactive Systems

## Question

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

## Answer

A process performs an action and becomes another process.

## Definition

A **labelled transition system** (LTS) is a triple  $(Proc, Act, \{\xrightarrow{a} \mid a \in Act\})$  where

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

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

Sometimes we distinguish the **initial** (or **start**) state.

# Sequencing, Nondeterminism and Parallelism

LTS explicitly focuses on **interaction**.

LTS can also describe:

- sequencing  $(a; b)$
- choice (nondeterminism)  $(a + b)$
- limited notion of parallelism (by using interleaving)  $(a \parallel b)$

## Definition

A binary relation  $R$  on a set  $A$  is a subset of  $A \times A$ .

$$R \subseteq A \times A$$

Sometimes we write  $x R y$  instead of  $(x, y) \in R$ .

## Properties

- $R$  is **reflexive** if  $(x, x) \in R$  for all  $x \in A$
- $R$  is **symmetric** if  $(x, y) \in R$  implies that  $(y, x) \in R$  for all  $x, y \in A$
- $R$  is **transitive** if  $(x, y) \in R$  and  $(y, z) \in R$  implies that  $(x, z) \in R$  for all  $x, y, z \in A$

Let  $R$ ,  $R'$  and  $R''$  be binary relations on a set  $A$ .

## Reflexive Closure

$R'$  is the **reflexive closure** of  $R$  if and only if

- 1  $R \subseteq R'$ ,
- 2  $R'$  is reflexive, and
- 3  $R'$  is the *smallest* relation that satisfies the two conditions above, i.e., for any relation  $R''$ :  
if  $R \subseteq R''$  and  $R''$  is reflexive, then  $R' \subseteq R''$ .

Let  $R$ ,  $R'$  and  $R''$  be binary relations on a set  $A$ .

## Symmetric Closure

$R'$  is the **symmetric closure** of  $R$  if and only if

- 1  $R \subseteq R'$ ,
- 2  $R'$  is symmetric, and
- 3  $R'$  is the *smallest* relation that satisfies the two conditions above, i.e., for any relation  $R''$ :  
if  $R \subseteq R''$  and  $R''$  is symmetric, then  $R' \subseteq R''$ .

Let  $R$ ,  $R'$  and  $R''$  be binary relations on a set  $A$ .

## Transitive Closure

$R'$  is the **transitive closure** of  $R$  if and only if

- 1  $R \subseteq R'$ ,
- 2  $R'$  is transitive, and
- 3  $R'$  is the *smallest* relation that satisfies the two conditions above, i.e., for any relation  $R''$ :  
if  $R \subseteq R''$  and  $R''$  is transitive, then  $R' \subseteq R''$ .

Let  $(Proc, Act, \{\xrightarrow{a} \mid a \in Act\})$  be an LTS.

- we extend  $\xrightarrow{a}$  to the elements of  $Act^*$
- $\longrightarrow = \bigcup_{a \in Act} \xrightarrow{a}$
- $\longrightarrow^*$  is the reflexive and transitive closure of  $\longrightarrow$
- $s \xrightarrow{a}$  and  $s \not\xrightarrow{a}$
- reachable states



# How to Describe LTS?

Syntax

unknown entity



Semantics

known entity

programming language



what (denotational) or  
how (operational) it computes

???



Labelled Transition Systems

CCS

## CCS

Process algebra called “Calculus of Communicating Systems”.

## Insight of Robin Milner (1989)

Concurrent (parallel) processes have an algebraic structure.

$$\boxed{P_1} \text{ op } \boxed{P_2} \Rightarrow \boxed{P_1 \text{ op } P_2}$$

## Basic Principle

- 1 Define a few **atomic processes** (modelling the simplest process behaviour).
- 2 Define compositionally **new operations** (building more complex process behaviour from simple ones).

## Example

- 1 atomic instruction: assignment (e.g.  $x:=2$  and  $x:=x+2$ )
- 2 new operators:
  - sequential composition ( $P_1; P_2$ )
  - parallel composition ( $P_1 \parallel P_2$ )

Now e.g.  $(x:=1 \parallel x:=2); x:=x+2; (x:=x-1 \parallel x:=x+5)$  is a process.

# CCS Basics (Sequential Fragment)

- *Nil* (or 0) process (the only atomic process)
- action prefixing ( $a.P$ )
- names and recursive definitions ( $\stackrel{\text{def}}{=}$ )
- nondeterministic choice ( $+$ )

## This is Enough to Describe Sequential Processes

Any finite LTS can be (up to isomorphism) described by using the operations above.