Semantics and Verification 2006

Lecture 1

Lecturer (1-8):	Jiri Srba	B2-203, `srba@cs.aau.dk`
Lecturer (9-15):	Kim G. Larsen	B1-209, `kgl@cs.aau.dk`
Assistants:	Bjørn Haagensen	B2-205, `bh@cs.aau.dk`
	Jacob I. Rasmussen	B1-205, `illum@cs.aau.dk`

Focus of the Course

- 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 automata and their semantics.
- Binary decision diagrams and their use in verification.
- Two mini projects.

Mini Projects

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

Lectures

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

Tutorials

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

Exam

- Individual and oral.
- Preparation time (star exercises).
- Pensum dispensation.

Literature

- On-line literature.
- Compendiums 2006 (195 kr).
- **Help us to proof-read the book, please!**
  "Reactive Systems: Modelling, Specification and Verification"

Hints

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

Aims of the Course

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.

Classical View

Characterization of a Classical Program
Program transforms an input into an output.

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

$$states \hookrightarrow states$$

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

Is this all we need?

Reactive systems

What about:

- Operating systems?
- Communication protocols?
- Control programs?
- Mobile phones?
- Vending machines?

Reactive systems

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.

Analysis of Reactive Systems

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 a 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
- ...

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.

Labelled Transition System

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) $(a \| b)$

Binary Relations

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 $x, y, z \in A$

Closures

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 $R' \subseteq R''$.

Closures

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 $R' \subseteq R''$.

Closures

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 $R' \subseteq R''$.

Labelled Transition Systems – Notation

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		Semantics
unknown entity	$\longrightarrow$	known entity
programming language	$\longrightarrow$	what (denotational) or how (operational) it computes
???	$\longrightarrow$	Labelled Transition Systems
CCS		

Calculus of Communicating Systems

CCS

Process algebra called "Calculus of Communicating Systems".

Insight of Robin Milner (1989)

Concurrent (parallel) processes have an algebraic structure.

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

Process Algebra

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 ∥ x:=2); x:=x+2; (x:=x-1 ∥ 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.