

KMI/MOVE: Modelling and Verification

Lecturer: Petr Jančar

Katedra informatiky PŘF UP

The course is based on the book

Reactive Systems: Modelling, Specification and Verification

by Luca Aceto, Anna Ingólfssdottir, Kim Guldstrand Larsen, Jiří Srba
Cambridge University Press, August 2007

The authors maintain the web-page

<http://rsbook.cs.aau.dk/>

which contains a lot of useful material. (Including the slides kindly provided by Jiří Srba, which serve as a basis of presentations in our course.)

The web-page of our course is

<http://phoenix.inf.upol.cz/~jancarp/MaV/mav.htm>

Focus of the Course

- Study of mathematical models for formal description and analysis of systems (programs).
- Study of formal languages for specification of (properties of) system behaviour.
- 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 CAAL (<http://caal.cs.aau.dk>).
- Verification of a real-time algorithm in UPPAAL.

- Ask/answer questions. Be active!
- Take your own notes.
- Read the recommended literature as soon as possible after the 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).

Exercise/Project-Credit (“zápočet”) and Exam

Exercise/Project-Credit (“zápočet”):

- participating at the two miniprojects and elaborating a solid respective report,

Exam:

- Individual and oral (the questions will be specified later).
- Preparation time (star exercises).

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.

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?

Interlude: Verification of a computer program

$\{ x_1, x_2 \text{ are integers satisfying } C_1: x_1 \geq 0, x_2 > 0 \}$

Program P

```
y1 := 0; y2 := x1;  
  {  $x_1 = y_1 x_2 + y_2 \wedge 0 \leq y_2$  } ... INV  
while  $y_2 \geq x_2$  do ( $y_1 := y_1 + 1; y_2 := y_2 - x_2$ );  
 $z_1 := y_1; z_2 := y_2$ 
```

$\{ C_2: x_1 = z_1 x_2 + z_2 \wedge 0 \leq z_2 < x_2 \}$

We want to verify: $\{ C_1 \} P \{ C_2 \}$... (specification of P)

Generated verification conditions:

```
{  $C_1$  }  $y_1 := 0; y_2 := x_1$  { INV }  
{  $INV \wedge y_2 \geq x_2$  }  $y_1 := y_1 + 1; y_2 := y_2 - x_2$  { INV }  
{  $INV \wedge \neg(y_2 \geq x_2)$  }  $z_1 := y_1; z_2 := y_2$  {  $C_2$  }
```

What about:

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

Characterization of Reactive Systems

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.

Example: Peterson's protocol

Concurrent, parallel, interactive, 'nondeterministic' systems,
with ongoing behaviour ...

No input-output characterization (specification) ...

Verification of 'simple' properties ...

Peterson's protocol (to avoid critical section clash)

Process *A*:

**** noncritical region ****

flag_A := *true*

turn := *B*

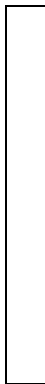
waitfor

(*flag_B* = *false* \vee *turn* = *A*)

**** critical region ****

flag_A := *false*

**** noncritical region ****



Process *B*:

**** noncritical region ****

flag_B := *true*

turn := *A*

waitfor

(*flag_A* = *false* \vee *turn* = *B*)

**** critical region ****

flag_B := *false*

**** noncritical region ****

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.

Definition

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

Interlude: 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 $(x, y) \in R$.

Some properties of relations

- 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, which means the following:
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.

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

Transitive Closure

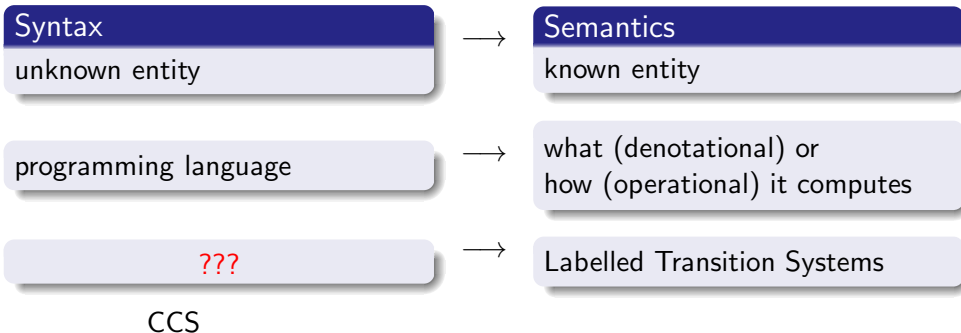
R' is the **transitive closure** of R if and only if

- 1 $R \subseteq R'$,
- 2 R' is transitive, and
- 3 R' is the smallest relation that satisfies the two conditions above.

Let $(Proc, Act, (\xrightarrow{a})_{a \in Act})$ 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?



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.

A CCS Process: Black-Box View

What is a CCS Process to its Environment?

A CCS process is a computing agent that may communicate with its environment via its interface.

Interface = Collection of **communication ports/channels**, together with an indication of whether used for input or output.

Example: A Computer Scientist

Process interface:

- **coffee** (input port)
- coin (output port)
- pub (output port)

Question: How do we describe the behaviour of the “black-box”?

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.