Models of Concurrency

Rocco De Nicola

Dipartimento di Sistemi ed Informatica
Università di Firenze

Process Algebras and Concurrent Systems

Sequential Programming vs Concurrent Programming

Classical Sequential Programming

1. Denotational semantics: the meaning of a program is a partial function from states to states
2. Nontermination is bad!
3. In case of termination, the result is unique.

Concurrent - Interactive - Reactive Programming

1. Denotational semantics is very complicate due to nondeterminism
2. Nontermination might be good!
3. In case of termination, the result might not be unique.

Programming Reactive System

The classical denotational approach is not sufficient for modelling systems such as:

- Operating systems
- Communication protocols
- Mobile phones
- Vending machines

The above systems compute by reacting to stimuli from their environment and are known as Reactive Systems. Their distinguishing features are:

- Interaction (many parallel communicating processes)
- Nondeterminism (results are not necessarily unique)
- There may be no visible result (exchange of messages is used to coordinate progress)
- Nontermination is good (systems are expected to run continuously)

Analysis of Reactive Systems

Even short parallel programs may be hard to analyze, thus we need to face few questions:

1. How can we develop (design) a system that works?
2. How do we analyze (verify) such a system?

We need appropriate theories and formal methods and tools, otherwise we will experience again:

1. Intels Pentium-II bug in floating-point division unit
2. Ariane-5 crash due to a conversion of 64-bit real to 16-bit integer
3. Mars Pathfinder problems

Formal Methods for Reactive Systems

To deal with reactive systems and guarantee their correct behavior in all possible environment, we need:

1. To study mathematical models for the formal description and analysis of concurrent programs.
2. To devise formal languages for the specification of the possible behaviour of parallel and reactive systems.
3. To develop verification tools and implementation techniques underlying them.

This Course

We shall see different theories of special kind of reactive systems and their applications.

The theories aim at supporting: Design, Specification and Verification (possibly automatic and compositional) of reactive systems.

Important Questions:

- What is the most abstract view of a reactive system (process)?
- Does it capture their relevant properties?
- Is it compositional?

Our Approach

- The chosen abstraction for reactive systems is the notion of processes.
- Systems evolution is based on process transformation: A process performs an action and becomes another process.
- Everything is (or can be viewed as) a process. Buffers, shared memory, Linda tuple spaces, senders, receivers, . . . are all processes.
- Labelled Transition Systems (LTS) describe process behaviour, and permit modelling directly systems interaction.

Outline of the lectures

1. Labelled Transition Systems as Concurrency Models
2. Operators for Interaction, Nondeterminism and Concurrency
3. Process Calculi and their semantics
4. A Calculus of Communicating Systems
5. Modal and Temporal Logics
6. Tools for Systems Specification and Verification

Bibliography

- Apt K.R., Olderog E.-R., *Verification of Sequential and Concurrent Programs*, Springer-Verlag, 1997.
  I took from here the first example of these lectures.
- Fokkink Wan, *Introduction to Process Algebra*, Springer, 2000.
  A gentle introduction to ACP.
- Milner R., *Communication and Concurrency*, Prentice Hall, 1989.
  The classical book on CCS and Bisimulation.
- Roscoe A.W., *The Theory and Practice of Concurrency*, Prentice Hall, 1998.
  A good book on TCSP and the failure Model.
- Bowman H. and Gomez R., *Concurrency Theory: Calculi and Automata for Modelling Untimed and Timed Concurrent Systems*, Springer, 2006.
  A new book on concurrency theory based on LOTOS.

Bibliography ctd.

- Baeten J.C.M. and Weijland W.P., *Process Algebra*, Cambridge University Press, 1990.
  The first book on ACP and Branching Bisimulation.
- Hennessy M., *Algebraic theory of processes*, Springer-Verlag, 2001.
  A simple introduction to Algebraic, Denotational and Operational Semantics of processes based on Testing Equivalence.
- Van Glabbeek R.J., *The Linear Time - Branching Time Spectrum I*. The Semantics of Concrete, Sequential Processes*, Handbook on Process Algebras, North Holland, 2001.
  A good overview of behavioral equivalences over LTS.
- Sangiorgi D. and Walker D., *PI-Calculus: A Theory of Mobile Processes*, Cambridge University Press, 2001.
  THE book on π-calculus.
- For Klaim see: `http://music.dsi.unifi.it`

Syntax and Semantics

Syntax

Set of rules for defining ”well formed phrases”

Syntactic Domain

Set of well formed phrases

Semantic Domain

Set of known entities

Semantic Interpretation

Mapping from Syntactic Domain to Semantic Domain or Interpretation of well formed phrases in terms of known concepts

Interpreting Regular Expressions

Syntax of Regular Expressions

$E \ ::= \ 0 \mid 1 \mid a \mid E + E \mid E; E \mid E^* \ \textit{with}\ a \in A$ the set of basic actions

Syntactic Domain: *RE*

The set of terms that can be generated by the above grammar

Semantic Domain: $Pow(A^*)$

Languages: Sets of Strings over the action monoid A

Interpretation Function: $RE \longrightarrow Pow(A^*)$

Associating elements of the syntactic domain into elements of the semantic one.

Denotational Semantics of Regular Expressions

Regular Expression have a denotational semantics that associates to each expression the language (i.e. the set of strings) generated by it.

$$
\begin{aligned}
\mathcal{L}[\![0]\!] &= \{\,\} \\
\mathcal{L}[\![1]\!] &= \{\varepsilon\} \\
\mathcal{L}[\![a]\!] &= \{a\} \quad \text{(with } a \in A\text{)} \\
\mathcal{L}[\![e + f]\!] &= \mathcal{L}[\![e]\!] \cup \mathcal{L}[\![f]\!] \\
\mathcal{L}[\![e \cdot f]\!] &= \mathcal{L}[\![e]\!] \cdot \mathcal{L}[\![f]\!] \\
\mathcal{L}[\![e^*]\!] &= (\mathcal{L}[\![e]\!])^*
\end{aligned}
$$

Operational Semantics for Concurrent Processes

Systems behaviour is described by associating to each program a behaviour represented as a transition graph.

Two main models (or variants thereof) have been used:

Kripke Structures

State Labelled Graphs: States are labelled with the properties that are considered relevant (e.g. the value of - the relation between - some variables)

Labelled Transition Systems

Transition Labelled Graph: Transition between states are labelled the action that induces the transition from one state to another.

In this lectures, we shall mainly rely on **Labelled Transition Systems** and actions will play an important role

Finite State Automata

Definition

A *finite state automaton* M is a 5-tuple

$M = (Q, A, \rightarrow, q_0, F)$ *where*

- Q is a finite set of states
- A is the alphabet
- $\rightarrow \subseteq Q \times (A \cup \{\varepsilon\}) \times Q$ is the transition relation
- $q_0 \in Q$ is a special state called initial state,
- $F \subseteq Q$ is the set of (final states)

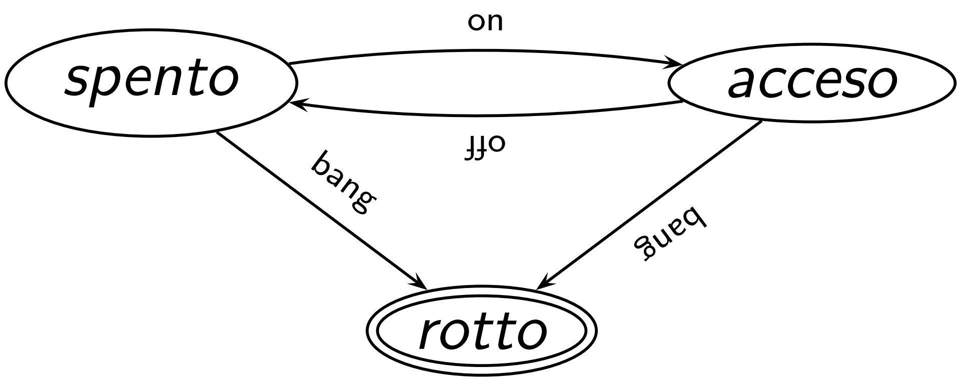


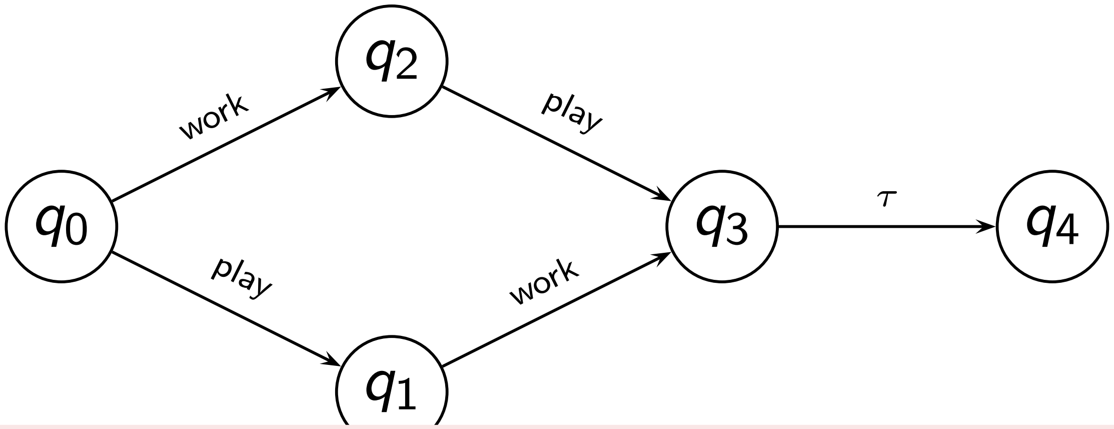
Figure: Finite state automaton

Labelled Transition Systems

Definition

A Labelled Transition System S is a 4-tuple $S = (Q, A, \rightarrow, q_0)$ *where* :

- Q is a set of states
- A is a finite set of actions
- $\rightarrow \subseteq Q \times A \times Q$ is a ternary relation called transition relation it is often written $q \xrightarrow{a} q'$ instead of $(q, a, q') \in \rightarrow$
- $q_0 \in Q$ is a special state called initial state.


$$\begin{tikzcd}
 & q_2 \arrow[dr, "play"] & & \\
q_0 \arrow[ur, "work"] \arrow[dr, "play"'] & & q_3 \arrow[r, "\tau"] & q_4 \\
 & q_1 \arrow[ur, "work"'] & &
\end{tikzcd}$$

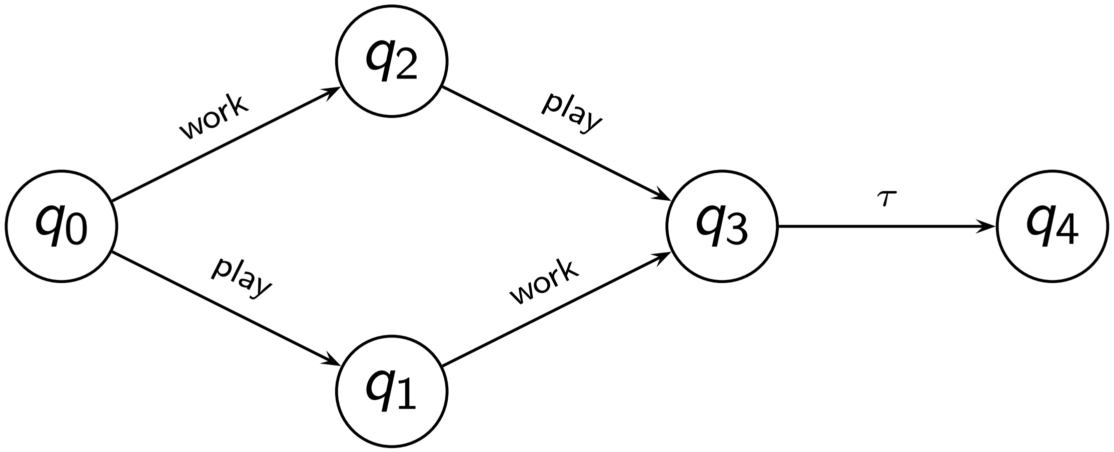
If initial states are not relevant (or known) LTSs are triples $(Q, A, \rightarrow)$ . . .

A Simple Example

Example (Bill-Ben)

$S = (Q, A, \rightarrow)$ where:

- $Q = \{\ q_0,\ q_1,\ q_2,\ q_3,\ q_4\ \}$
- $A = \{\ play,\ work,\ \tau\ \}$
- $\rightarrow =$
  $\{(q_0, play, q_1),\ (q_0, work, q_2),\ (q_1, work, q_3),\ (q_2, play, q_3),\ (q_3, \tau, q_4)\}$



Internal and External Actions

An elementary action of a system represents the atomic (non-interruptible) abstract step of a computation that is performed by a system to move from one state to the other.

Actions represent various activities of concurrent systems:

1. Sending a message
2. Receiving a message
3. Updating values
4. Synchronizing with other processes
5. ...

We have two main types of atomic actions:

- Visible Actions
- Internal Actions

Why operators for describing systems

How can we describe very large automata or LTSs?

As a table?

Rows and columns are labelled by states, entries are either empty or marked with a set of actions.

As a listing of triples?

$\rightarrow = \{(q_0, a, q_1),\ (q_0, a, q_2),\ (q_1, b, q_3),\ (q_1, c, q_4),\ (q_2, \tau, q_3),\ (q_2, \tau, q_4)\}.$

As a more compact listing of triples?

$\rightarrow = \big\{(q_0, a, \{q_1, q_2\}),\ (q_1, b, q_3),\ (q_1, c, q_4),\ (q_2, \tau, \{q_3, q_4\})\big\}.$

As XML?

`<lts><ar><st>q0</st><lab>a</lab><st>q1</st></ar>...</lts>`.

Why operators for describing systems - ctd

Linguistic aspects are important!

The previous solutions are ok for machines . . . not for humans.

Are prefix and sum operators sufficient?

They are ok to describe small finite systems:

- $p = a.b.(c + d)$
- $q = a.(b.c + b.d)$
- $r = a.b.c + a.c.d$

But additional operators are needed

- to design systems in a structured way (e.g. $p|q$)
- to model systems interaction
- to abstract from details
- to represent infinite systems

Operational Semantics

To each process, built using the above mentioned operators, an LTS is associated by relying on structural induction to define the meaning of each operator.

Inference Systems

An inference system is a set of inference rule of the form

$$\frac{p_1, \cdots, p_n}{q}$$

Transition Rules

For each operator *op*, we have a number of rules of the form below, where $\{i_1, \cdots, i_m\} \subseteq \{1, \cdots, n\}$.

$$\frac{E_{i_1} \xrightarrow{\alpha_1} E'_{i_1} \quad \cdots \quad E_{i_m} \xrightarrow{\alpha_m} E'_{i_m}}{op(E_1, \cdots, E_n) \xrightarrow{\alpha} op(E'_1, \cdots, E'_n)}$$

The Elegance of Operational Semantics

Automata as terms

Few SOS rules define all the automata that can ever be specified with the chosen operators. Given any term, the rules are used the derive the corresponding automaton. The set of rules is fixed once and for all.

Structural induction

The interaction of complex systems is defined in terms of the behavior of their components.

A remark

The LTS is the least one satisfying the inference rules.

Rule induction

A property is true for the whole LTS if whenever it holds for the premises of each rule, it holds also for the conclusion.

Presentations of Labelled Transition Systems

Process Algebra as denotations of LTS

- LTS are represented by terms of process algebras.
- Terms are interpreted via operational semantics as LTS.

Process Algebra Basic Principles

1. Define a few elementary (atomic) processes modelling the simplest process behaviour;
2. Define appropriate composition operations to build more complex process behaviour from (existing) simpler ones.

Regular Expressions as Process Algebras

Syntax of Regular Expressions

$E \; ::= \; 0 \mid 1 \mid a \mid E + E \mid E ; E \mid E^*$ *with* $a \in A$ *and* $-$ *below* $-$ $\mu \in A \cup \{\varepsilon\}$

Operational Semantics of Regular Expressions

(Tic) $$\frac{}{1 \xrightarrow{\varepsilon} 1}$$

(Atom) $$\frac{}{a \xrightarrow{a} 1}$$

(Sum$_1$) $$\frac{e \xrightarrow{\mu} e'}{e + f \xrightarrow{\mu} e'}$$

(Sum$_2$) $$\frac{f \xrightarrow{\mu} f'}{e + f \xrightarrow{\mu} f'}$$

(Seq$_1$) $$\frac{e \xrightarrow{\mu} e'}{e; f \xrightarrow{\mu} e'; f}$$

(Seq$_2$) $$\frac{e \xrightarrow{\varepsilon} 1}{e; f \xrightarrow{\varepsilon} f}$$

(Star$_1$) $$\frac{}{e^* \xrightarrow{\varepsilon} 1}$$

(Star$_2$) $$\frac{e \xrightarrow{\mu} e'}{e^* \xrightarrow{\mu} e'; e^*}$$

Operators for Concurrency and Process Algebras

Operators for Processes Modelling

Processes are composed via a number of basic operators

1. Basic Processes
2. Action Prefixing
3. Sequentialization
4. Choice
5. Parallel Composition
6. Abstraction
7. Infinite Behaviours

A few examples for Regular Expressions

$(a+b)^* \xrightarrow{a} 1; (a+b)^*$

$$\dfrac{\dfrac{\dfrac{}{a \xrightarrow{a} 1}\,(Atom)}{a+b \xrightarrow{a} 1}\,(Sum_1)}{(a+b)^* \xrightarrow{a} 1; (a+b)^*}\,(Star_2)$$

$1; (a+b)^* \xrightarrow{\varepsilon} (a+b)^*$

$$\dfrac{\dfrac{}{1 \xrightarrow{\varepsilon} 1}\,(Tic)}{1; (a+b)^* \xrightarrow{\varepsilon} (a+b)^*}\,(Seq_2)$$

Another Example On Regular Expressions

$(a^* + b^*)^* \xrightarrow{b} 1; b^*; (a^* + b^*)^*$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{b \xrightarrow{b} 1}\ (Atom)}{b^* \xrightarrow{b} 1; b^*}\ (Star_2)}{a^* + b^* \xrightarrow{b} 1; b^*}\ (Sum_2)}{(a^* + b^*)^* \xrightarrow{b} 1; b^*; (a^* + b^*)^*}\ (Star_2)$$

A remark

$(a^* + b^*)^* \xrightarrow{c} 1$ would not contradict any rule, but it cannot be in the least LTS, because it cannot be inferred by using the rules we presented earlier.