Transition Systems

Renato Neves

Universidade do Minho

HASLab
HIGH-ASSURANCE
SOFTWARE LABORATORY

Table of Contents

A Sprinkle of Linguistics

During the module we will encounter two linguistic concepts that every programmer should know:

- syntax - the rules used for determining whether a sentence is valid (in a language) or not
- semantics - the meaning of valid sentences

Example (Syntax)

The sentence/program $\mathtt{x} := \mathtt{p}\,;\mathtt{q}$ is forbidden by the syntactic rules of most programming languages

Example (Semantics)

The sentence/program $\mathtt{x} := 1$ has the meaning "writes 1 in the memory address corresponding to $\mathtt{x}$"

The need for Semantics in Formal Analysis

How can one prove that a program does what is supposed to do if its semantics (i.e. its meaning) is not established *a priori* ?

Example

What is the end result of running $x := 2\,;(x := x + 1 \parallel x := 0)$?

parallelism operator

Widely used programming languages still lack a formal semantics

Transition Systems as Semantic Providers

Transition systems are an ubiquitous mechanism for defining the semantics of programming languages —

essentially they register all steps of computations

Following tradition, we will use them to define the semantics of a simple (but powerful !) concurrent language —

and then base on this learning step to tackle Dijkstra's

Dining Philosophers Problem (*circa* 1965)

Table of Contents

Preliminaries pt. I

Recalling previous modules . . .

Definition (Functor)

A functor F sends a set X into a new set FX and a function $f : X \to Y$ into a new function $Ff : FX \to FY$ such that

$$F(\mathrm{id}) = \mathrm{id} \qquad F(g \cdot f) = Fg \cdot Ff$$

Fix a set A. The following two functors then naturally arise

- product - $X \mapsto A \times X, \;\; f \mapsto id \times f$
- exponential - $X \mapsto X^A, \;\; f \mapsto (g \mapsto f \cdot g)$

Preliminaries pt. II - the List and Powerset functors

The list functor - $X \mapsto X^*, \;\; f \mapsto \texttt{map}\ f$

applies f to every element of a given list

The powerset functor - almost like the list functor; the difference is that we do not look at the order in which elements appear and how many times they repeat. Formally,

$$X \mapsto \{A \mid A \subseteq X\}, \qquad f \mapsto (A \mapsto \{f(a) \mid a \in A\})$$

Example (Powerset on Booleans)

$\texttt{Bool} \mapsto \{\emptyset, \{\top\}, \{\bot\}, \{\top, \bot\}\}$

A (Generalised) Notion of a Transition System

Definition (Transition system)

Let F be a functor. An F-transition system is a map $X \to FX$

Some famous examples of F-transition systems

- Moore automata - $X \to A \times X^L$
- Deterministic automata - $X \to \texttt{Bool} \times X^L$
- Non-deterministic automata - $X \to \texttt{Bool} \times \underbrace{\mathrm{P}(X)}_{\text{Powerset functor}}{}^L$
- Markov chain - $X \to \underbrace{\mathrm{D}(X)}_{\text{Distribution functor}}$

Our First encounter with Coalgebra

Indeed the idea of working at the level of

Functors as Transition Types

is a very fruitful one; and which we only barely grasped (yet) —

in essence, it provides a universal theory of transition systems that can be instantiated to most kinds of transition system we will encounter in our life

Table of Contents

Calculus of Communicating Systems

Syntax

$$P, Q ::= X \mid a.P \mid \sum_{i \in I} P_i \mid P \parallel Q \mid P[f] \mid P \backslash L$$

(suited for describing communication and synchronisation protocols)

- X is a process name
- $a.P$ communicates via channel a and proceeds as P
- $\sum_{i \in I} P_i$ non-deterministic choice between processes P_i
- $P \parallel Q$ parallel composition between processes P and Q
- ...

First Steps with CCS

Some helpful conventions:

- $0 = \sum_{i \in \emptyset} P_i$ (denotes a terminating process)
- $\bar{a}$ denotes outgoing information via channel a
- τ denotes an invisible action

Some examples of processes written in CCS:

- $a.0 \parallel \bar{a}.0$ - connects two processes via channel a; information flows in one direction only
- $a.\bar{b}.0 \parallel \bar{a}.b.0$ - info. flows in one direction via a and in the inverse direction via b; the latter is used only after a is used
- $(a.\bar{b}.0 \parallel \bar{a}.b.0)\backslash\{a, b\}$ - both channels a, b are now private

First steps with CCS

Which of these expressions are valid sentences in CCS?

1. $a.b.P + Q$
2. $a + b$
3. $P.a$
4. $(P + Q).a$
5. $a.0 + b.0$
6. $P.Q$

CCS and Cyclic Behaviour

We now add the construct $\texttt{rec}\ X.\ P$ to the syntax of CCS – so that we can describe cyclic behaviour

Example

$\texttt{rec}\ X.\ a.b.X$ - receive communication through a and then through b; after that repeat the protocol

Example (The coffee machine and the student)

$(\texttt{rec}\ X.\ \mathit{coin}.\overline{\mathit{coffee}}.X) \mid\mid (\texttt{rec}\ Y.\ \overline{\mathit{coin}}.\mathit{coffee}.\overline{\mathit{wrk}}.Y)$

Write down a coffee machine that fails to deliver coffee sometimes

Every process P yields a transition system $X \to \mathrm{P}(X)^L$ with $P \in X$ and with the transitions prescribed by the following rules:

$$\frac{}{\alpha.P \xrightarrow{\alpha} P} \qquad \frac{P_i \xrightarrow{\alpha} Q}{\sum_{i \in I} P_i \xrightarrow{\alpha} Q}$$

$$\frac{P \xrightarrow{\alpha} P'}{P \parallel Q \xrightarrow{\alpha} P' \parallel Q} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P \parallel Q \xrightarrow{\alpha} P \parallel Q'} \qquad \frac{P \xrightarrow{\alpha} P' \qquad Q \xrightarrow{\bar{\alpha}} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}$$

$$\frac{P \xrightarrow{\alpha} P'}{P \backslash L \xrightarrow{\alpha} P' \backslash L} \ \alpha, \overline{\alpha} \notin L \qquad \frac{P[\mathtt{rec}\ X.\ P/X] \xrightarrow{\alpha} P'}{\mathtt{rec}\ X.\ P \xrightarrow{\alpha} P'}$$

Substitution of X in P by $\mathtt{rec}\ X.\ P$

Exploring CCS . . .

With the syntax and semantics of CCS now in place, we may put on our working hats and start to (formally) analyse communication and synchronisation mechanisms

Mutual Exclusion in CCS

We define three recursive processes

$$S = \texttt{rec}\, X.\ \overline{start}.finish.X \qquad \text{(the semaphore)}$$

$$P_1 = \texttt{rec}\, Y.\ start.a_1.b_1.\overline{finish}.Y \qquad \text{(process 1)}$$

$$P_2 = \texttt{rec}\, Z.\ start.a_2.b_2.\overline{finish}.Z \qquad \text{(process 2)}$$

and then write down $(S \parallel P_1 \parallel P_2) \backslash \{start, finish\}$

Question: will we ever observe a sequence of actions $x_1 \dots x_n \dots$ such that $x_i = a_1$ and $x_{i+1} = a_2$?

think of a_i as writing on a critical region and of b_i as ending this process

Dining Philosophers Problem

Two philosophers are sitting at the table in front of each other . . . thinking . . .

At some point, they will wish to eat and for that effect there are precisely two forks on the table, at their left and right-hand sides

When Philosopher 1 wishes to eat he first picks the fork on his left and then the one on his right

Philosopher 2 picks first the fork on her left and then the fork on her right

Write down this system in CCS and discover whether it is possible that both philosophers die of starvation

Table of Contents

The Quest for Observational Equivalence

Sometimes we would like to replace a program for another one whose behaviour we cannot distinguish from the original

Example

Why not replace `rec` $X.\ a.a.X$ by the simpler process `rec` $X.\ a.X$?

For such substitutions to be sound we require a formal notion of observational equivalence

Observational Equivalence Informally

Two programs are observationally equivalent if it is impossible to observe any difference in their behaviour

Here behaviour is described in terms of transition systems

... and therefore behaviour/equivalence needs to be pinned down to them

F-Transition Systems and Observational Behaviour

Every functor F induces a notion of observational behaviour

Example (Moore automata)

Every automaton $X \to A \times X$ induces a map $[\![-]\!] : X \to A^\omega$

Infinite lists over A

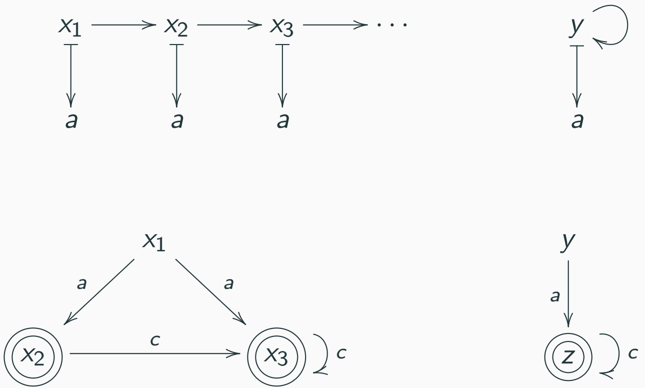
Example (Deterministic automata)

Every automaton $X \to \texttt{Bool} \times X^L$ induces $[\![-]\!] : X \times L^* \to \texttt{Bool}$

Intuitively F provides a black-box perspective to the transition system ...

states are not directly observable; only their interaction with the environment is

Question



Do x_1 and y possess the same observable behaviour in both cases?

Just Another Brief Visit to the Field of Coalgebra

The subject of systematically deriving a notion of observable behaviour from a functor goes beyond this module ...

... but you can always ask me about it after the lecture :-)

Definition

Fix a functor F and consider two transition systems $f : X \to FX$ and $g : Y \to FY$. Two states $x \in X$, $y \in Y$ are observationally equivalent if there exists a relation $R \subseteq X \times Y$ with $(x, y) \in R$ and there exists a transition system $b : R \to FR$ such that the diagram below commutes

$$\begin{tikzcd}
X \arrow[d, "f"'] & R \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] \arrow[d, "b"'] & Y \arrow[d, "g"] \\
FX & FR \arrow[l, "F\pi_1"] \arrow[r, "F\pi_2"'] & FY
\end{tikzcd}$$

If such is the case we write $x \sim y$

Observational Equivalence for Moore Automata

Given $\langle o_1, n_1 \rangle : X \to A \times X$ and $\langle o_2, n_2 \rangle : Y \to A \times Y$ we obtain from the previous slide that $x \sim y$ iff

- $o_1(x) = o_2(y)$
- $n_1(x) \sim n_2(y)$

Observational Equivalence for Labelled Transition Systems

Recall that we used systems of type $X \to \mathrm{P}(X)^L$ for establishing the semantics of CCS processes. This means that …

notions of observational behaviour/equivalence for such transition systems directly impact our concurrent language

Given $\overline{t_1} : X \to \mathrm{P}(X)^L$ and $\overline{t_2} : Y \to \mathrm{P}(Y)^L$, $x \sim y$ iff for all $l \in L$

- $\forall x' \in t_1(x, l).\ \exists y' \in t_2(y, l).\ x' \sim y'$
- $\forall y' \in t_2(y, l).\ \exists x' \in t_1(x, l).\ x' \sim y'$

Observational Equivalence for Labelled Transition Systems

$$\begin{tikzcd}
 & q_1 \arrow[dl, "a"'] \arrow[dr, "a"] & \\
q_2 \arrow[d, "c"] & & q_3 \arrow[d, "c"] \\
q_4 & & q_5
\end{tikzcd}
\qquad
\begin{tikzcd}
 & p_1 \arrow[d, "a"] & \\
 & p_2 \arrow[dl, "c"'] \arrow[dr, "c"] & \\
p_4 & & p_5
\end{tikzcd}$$

$$\begin{tikzcd}
 & q_1 \arrow[dl, "a"'] \arrow[dr, "a"] & \\
q_2 \arrow[d, "c"] & & q_3 \arrow[d, "\color{red}{b}"] \\
q_4 & & q_5
\end{tikzcd}
\qquad
\begin{tikzcd}
 & p_1 \arrow[d, "a"] & \\
 & p_2 \arrow[dl, "c"'] \arrow[dr, "\color{red}{b}"] & \\
p_4 & & p_5
\end{tikzcd}$$

Is Observational Equivalence a Good Notion of Equivalence?

Coinduction Principle

Two states x, y are observationally equivalent iff they produce the same observational behaviour

Definition

Consider two processes P, Q in CCS. They are equivalent (in symbols $P \sim Q$) whenever the corresponding states in the transition system are observationally equivalent

Show that

- $\texttt{rec}\ X.\ (\texttt{rec}\ Y.\ a.X \parallel b.Y) \sim (\texttt{rec}\ X.\ a.X) \parallel (\texttt{rec}\ Y.\ b.Y)$
- $(\texttt{rec}\ X.\ a.X) \parallel (\texttt{rec}\ Y.\ b.Y) \sim \texttt{rec}\ X.\ (a.X + b.X)$
- $\texttt{rec}\ X.\ (a.X + b.X) \not\sim (\texttt{rec}\ X.\ a.X) + (\texttt{rec}\ Y.\ b.Y)$

An Algorithm for Finding Observationally Equivalent States

Consider two transition systems $\overline{t_1} : X \to X^L$ and $\overline{t_2} : Y \to Y^L$

For every $\sim_k \subseteq X \times Y$ define

- $\sim_0 := X \times Y$
- $x \sim_{k+1} y$ iff for all $l \in L$:
  $\forall x' \in t_1(x,l).\ \exists y' \in t_2(y,l).\ x' \sim_k y'$;
  $\forall y' \in t_2(y,l).\ \exists x' \in t_1(x,l).\ x' \sim_k y'$

If for some $k > 0$ we obtain $\sim_k = \sim_{k+1}$ then $\sim\ := \sim_k$

Exercises

Show that

- $\texttt{rec}\ X.\ a.a.X \sim \texttt{rec}\ X.\ a.X$
- $\texttt{rec}\ X.\ (a.X + a.a.X)\ \sim\ \texttt{rec}\ X.\ a.x$
- $\texttt{rec}\ X.\ (a.X + b.X) \not\sim (\texttt{rec}X.\ a.X) + (\texttt{rec}Y.\ b.Y)$
- $P \parallel 0 \sim P$
- $P + Q \sim Q + P$
- $P \parallel Q \sim Q \parallel P$