University of
Southampton

COMP2212 Programming Language Concepts

Labelled Transition Systems

Dr Julian Rathke

University of Southampton

Complex Semantics

- When it comes to understanding a concurrent program we must consider each of its threads and the semantics of each thread can be represented as a tree of behaviours.
- When considering whether an implementation of an particular thread meets its semantics we must have a means of considering
  - the actual behaviour of a thread (as a tree of actions) and
  - the specified semantics (as a tree of actions)
- We need a way of comparing trees of actions
- This is the topic of the next few lectures. The particular forms of trees we use are actually represented as graphs of labelled actions and are referred to as **labelled transition systems.**
- The notions of equality between concurrent systems is a rich field and forms the basis of many reasoning and verification techniques for concurrency.

Reasoning about concurrent programs

- There are some essential aspects of concurrency that make is different from sequential computation, e.g.:
  - **communication** - processes communicate with other processes either through shared memory or with message passing
  - **synchronisation** - processes must sometimes synchronise their actions to ensure atomicity
  - **nondeterminism** - what **can be observed** about a program changes from one run to the next
- Some things that we take for granted need to be rethought: e.g. how to test concurrent code?

Nondeterminism example

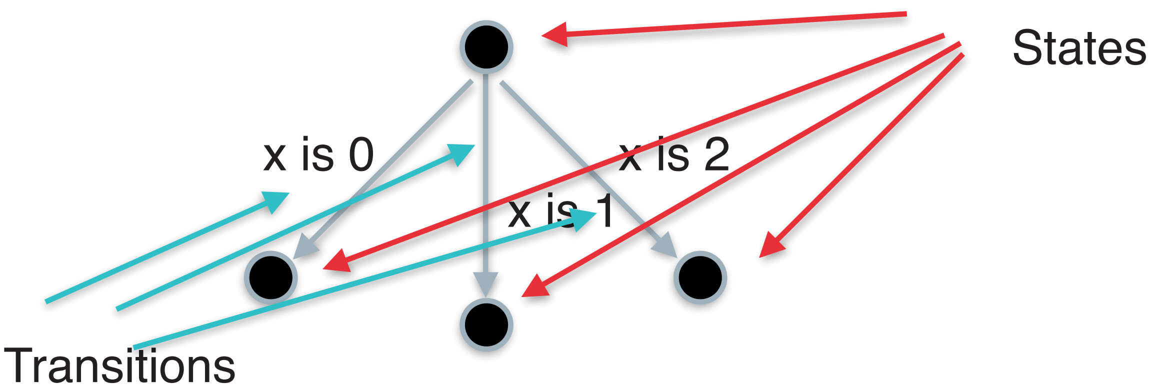
- Suppose we "run" the above three pieces of pseudo-code concurrently. What will be printed?

Thread 1	Thread 2	Thread 3
`x = 0;` `print “x is “ ++ x;`	`x = 1;`	`x = 2;`

x is "0" or "x is 1" or "x is 2"

Possibilities

- We can use a mathematical structure called a Labelled Transition System (LTS) in order to capture what can be observed about programs
  - LTS are a mathematical structure for reasoning about nondeterminism
  - the labels of transitions say “what can be observed”
  - eg.



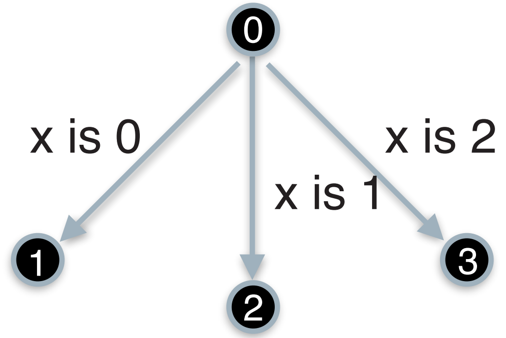
Labelled transition systems

- A labelled transition system is a mathematical structure (X, Σ, L) where
  - X is a set of states
  - Σ is an alphabet of actions
  - $L \subseteq X \times \Sigma \times X$

X is {0,1,2,3} - say

Σ is {"x is 0", "x is 1", "x is 2"}

L is { (0,"x is 0",1),
(0,"x is 1",2),
(0,"x is 2",3)
}



We write $x \xrightarrow{a} y$ to mean $(x,a,y) \in L$

LTSs and finite state automata

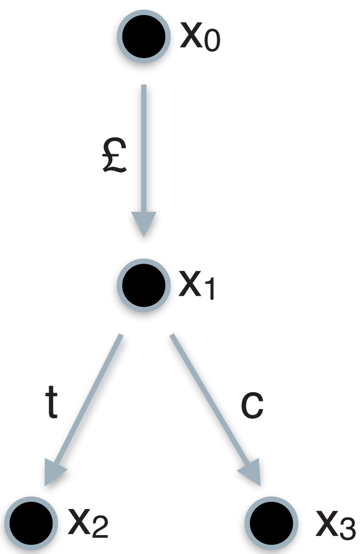
- Labelled transition systems are similar to finite state automata, which also have states and transitions, but there are important differences
  - The set of states in an LTS can be infinite: we cannot assume that our systems have only a finite number of possible states!
  - LTSs typically do not have initial and final states

Kinds of nondeterminism

- **Internal** - “the machine chooses”
  - e.g. the simple code example we have examined
  - the nondeterminism is resolved by the scheduler
- **External** - “the environment chooses”
  - e.g. interactive systems such as vending machines
  - the combination of a vending machine and user can be thought of as a concurrent system

External nondeterminism - Vending machine

- The user puts in money - this is the £ action
- The machine now offers a **choice** between tea (t) and coffee (c)



$$X = \{ x_0, x_1, x_2, x_3 \}$$

$$\Sigma = \{ £, t, c \}$$

$$L = \{ (x_0, £, x_1), (x_1, t, x_2), (x_1, c, x_3) \}$$

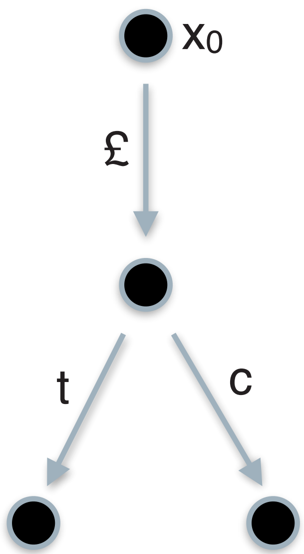
Process equivalence

University of Southampton

- Non determinism is inherent in concurrent and interactive systems
- What does it mean that a system is correct?
  - one answer: it should behave like (be **equivalent** to) some specification
  - but what should **equivalent** mean?
    - this is a surprisingly subtle question that has resulted in a lot of research over the last 40 years

First try

- Give the specification as a set of **traces**
  - a **trace** is a sequence of observations from some state
  - example:



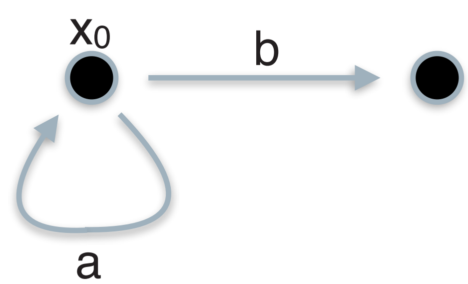
All possible traces from x_0 are
{ ε, £, £t, £c }

this is the empty trace

- Say that two states are **trace equivalent** when they have the same set of all traces possible traces.

Traces, example

- Some systems have an **infinite** set of traces



Traces from x_0 are

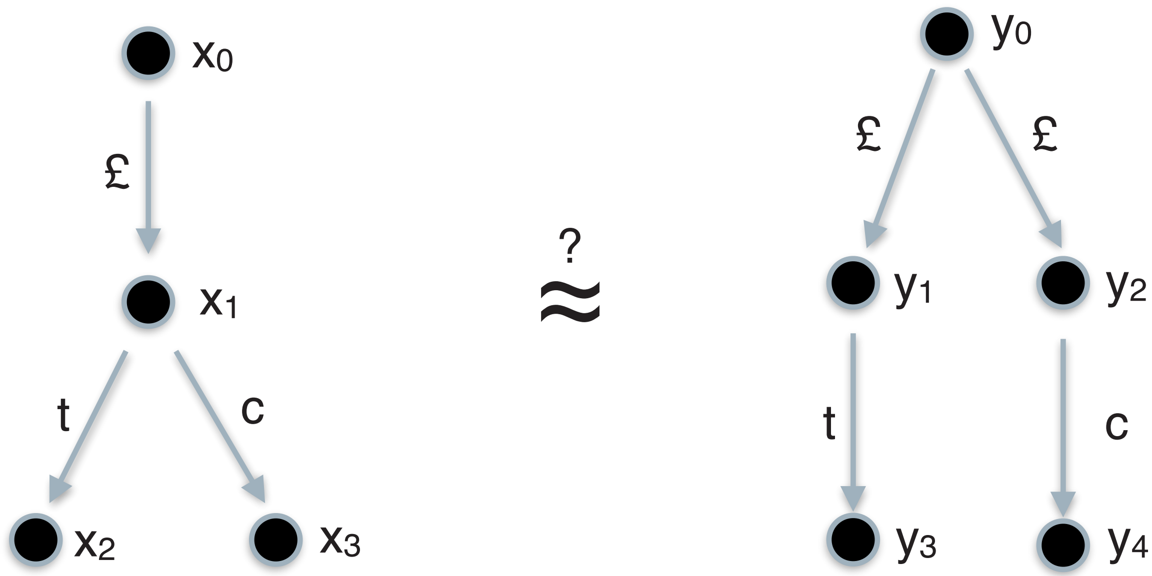
{ ε, a, b, aa, ab, aaa, aab, aaaa, aaab, aaaaa, aaaab, …. }

Indeed, empty and all the words matched by the regular expression a*b?

University of Southampton

Example: vending machines

- Should we consider these two vending machines as **equivalent**?



Are they trace equivalent?

What would be your criteria for accepting these as equivalent?

Moral

- In some cases, trace equivalence is too **coarse:** it equates too much
  - we want to distinguish the two vending machine examples
- In the next few lectures we will discuss finer ways of distinguishing between labelled transitions systems: **simulation** and **bisimulation**

Next Lecture

Simulation