Processes and Threads I

Lecture 2

Alma Rahat

CS-210: Concurrency

27 January, 2021

Questions and comments: shorturl.at/ewNO4

What did we do in the last session?

- Course introduction.
- Administration and assessments.
- Motivation.

In this lecture...

Learning outcomes.

1. To explain the basic rules of process algebra (finite state processes, FSP) for modelling concurrent problems.
2. To describe the design and implementation workflow.
3. To analyse simple problems and apply process algebra.
4. To implement simple process in Java.

Outline.

1. Abstractions and models.
2. Design and implementation workflow.
3. Introduction to Process Algebra.
4. Introduction to Labelled Transition System and its analyser.
5. Java implementation.

Abstraction

Swansea University
Prifysgol Abertawe

Engineering is based on the use of simpler, abstract models for experimentation, reasoning and exhaustive analysis.

Abstraction.

The process of removing details in the study of systems to focus attention to details of greater importance. It helps us to reason about a complex system.

Example: A car is a complex machine, but the interface (consisting of steering wheel, pedals and gear handle) is simple.
When you are teaching someone to drive you need to tell them what happens if you rotate the steering to the left, but not exactly how the machine accomplishes the manoeuvre.
The goal of your job determines what is important! If you are an automotive engineer, you might want to know the detail of how the car works.

Generalisation

Generalisation is a form of abstraction.

A concept A is a generalisation of concept B, if and only if:

- Every instance of B is also an instance of A.
- There may be instances of A that are not instances of B.

Example: An animal is a generalisation of a bird, but not all animals are birds, e.g. cats.

Use of Abstraction and Generalisation

Swansea University
Prifysgol Abertawe

Requirements To elicit critical aspects of the required system.

Design To articulate software architecture and functionalities.

Programming To generate working solutions.

Reasoning To analyse the program in an abstract domain.

Models

A model is an abstract and simplified representation of the real world.

> *All models are wrong, but some models are useful.*
> – George Box.

We will use models to gain confidence in the adequacy and validity of a proposed design through:

- Focus on abstraction of concurrency.
- Model animation to visualise behaviour.
- Automated model checking of properties (safety and progress).

Concurrent Processes

A complex system is composed of smaller activities, each represented as a sequential process. A process is the execution of a sequential program.

Key Properties.

- A process has a state: explicitly described by variables.
- A process executes (a sequence of **atomic** or indivisible) actions: to transform its state.

Think about `attributes` in a class – they tell you the state of that class's instance or object, and the actions are performed by `methods` of that class to change the state variables.

Modelling Processes

A process is modelled as a state machine, and the tools we use are:

Labelled Transition System (LTS) Graphical form: analyse, display and animate.

Finite State Process (FSP) Algebraic form: textually describe a model.

We can explore the interactions between states using the Labelled Transition System Analyser (LTSA) tool.

Design and implementation workflow

Swansea University
Prifysgol Abertawe

Step I: Deconstruct Conceptualise scenario as one or more processes with attributes (for capturing state) and a sequence of actions (or methods).

Step II: Model Three sub-steps.

1. Translate deconstruction to Process Alphabets.
2. Describe finite state machines using FSP.
3. Generate LTS and analyse with LTSA tool.

Step III: Implement Program with Java threads.

Any questions?

Scenario

Consider a simple toggle switch program: toggle down to send turn on signal, and toggle up to send turn off signal.

Step I: Deconstruct

Consider a simple toggle switch program: toggle down to send turn on signal, and toggle up to send turn off signal.

Actions (look for verbs):

- Up
- Down

State representation (look for unique states): a two-state *boolean* signal variable with initial value set to 0.

Actions are more important for the next step of modelling, but state representations would come handy in the last step of implementation.

Step II.1: Process alphabets

The alphabet of a process is the set of actions in which it can engage.

Process alphabets are implicitly defined by the actions in the process definitions.

Process:
Switch
Alphabet:
{
down,
up
}

Intermediary states (a.k.a. subprocesses): `Off` and `On`.
We will translate our deconstruction to alphabets first, then write Model in FSP code.

Step II.2: Model and process algebra

Process algebra is essentially a language to describe models with states (or subprocesses), actions and transitions. FSP is a variant of process algebra.

Basic Syntactic Rules.

- Uppercase: the name of a process, subprocess or state starts with a uppercase letter, e.g. `Switch` is a process .
- Lowercase: the name of an action starts with a lowercase letter, e.g. `up` is an action.
- Right arrow: If x is an action and P is a process then $(x->P)$ describes a process that initially engages in the action x and then behaves as exactly described by P.
- Comments: start with \\.
- Period: process definitions are terminated by '.'.
- Comma: multiple definitions – often used to define subprocesses – are seprated by ','.

Quick reference guide: `https://www.doc.ic.ac.uk/~jnm/book/ltsa/Appendix-A-2e.html`

Step II.2: Model and process algebra

```
\\FSP for a toggle switch
SWITCH = OFF,  \\process name with initial state
OFF = (down -> ON), \\description of OFF process with associated action 'down'
ON = (up -> OFF). \\description of ON process with associated action 'up'
```

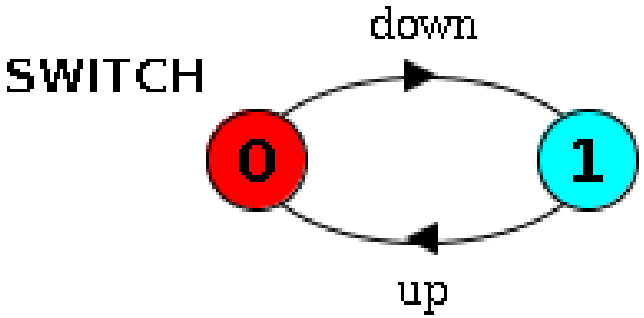
Simplification by removing local subprocesses (e.g. OFF and ON).

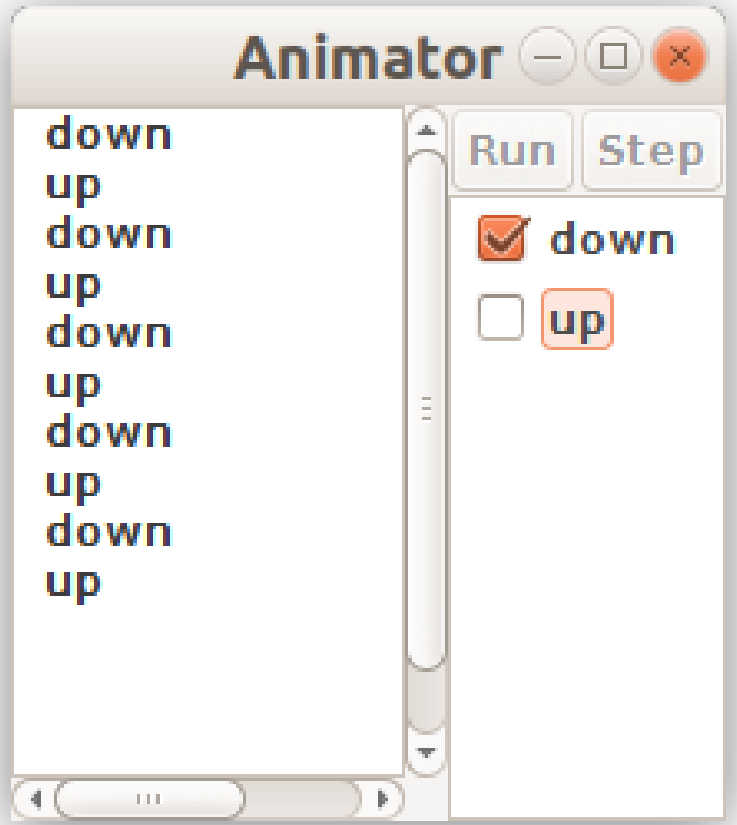
```
\\FSP for a toggle switch
SWITCH = (down -> up -> SWITCH).    \\two actions brings us back to initial state
```

Step II.3: Generate LTS and analyse

Live demonstration of LTSA tool

Step II.3: Generate LTS and analyse





- The labelled transition system (or state diagram) clearly depicts the relationships between states and actions.
- Finite states, but infinite traces.

Any questions?

Another example: Job process

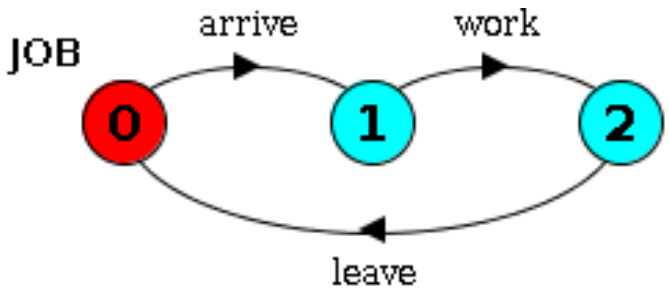
Scenario: A job program is defined as arriving at work, working, and then leaving for home.

Actions:

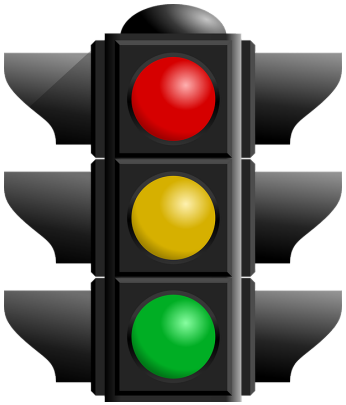
- arrive
- work
- leave

FSP:

```
\\FSP for JOB
JOB = (arrive -> work -> leave -> JOB).   \\three
actions brings us back to initial state
```



Exercise: Traffic lights

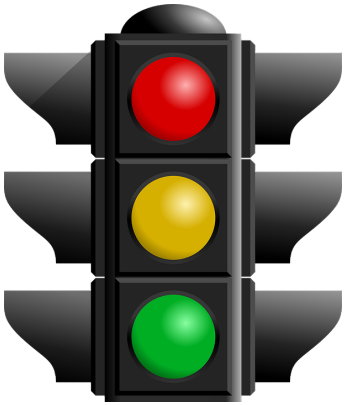


Scenario: A set of traffic lights go from red to orange to green to orange to red.

Step I: Deconstruct. What is an action in this scenario?

Please go to `www.menti.com` and enter the code 42 46 21 6.

Exercise: Traffic lights



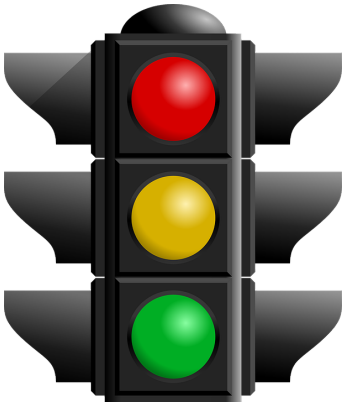
Scenario: A set of traffic lights go from red to orange to green to orange to red.

Step I: Deconstruct. What is an action in this scenario?

Please go to `www.menti.com` and enter the code 42 46 21 6.

- toRed
- toOrangeGreen
- toOrangeRed
- toGreen

Exercise: Traffic lights

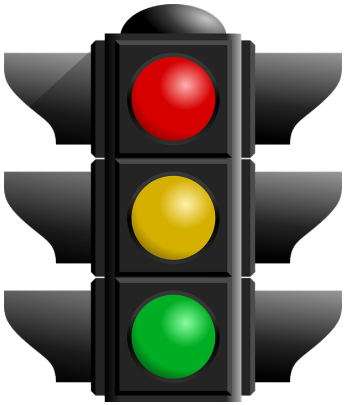


Scenario: A set of traffic lights go from red to orange to green to orange to red.

Step I: Deconstruct. what would be an appropriate intermediary state?

Please go to `www.menti.com` and enter the code 42 46 21 6.

Exercise: Traffic lights



Scenario: A set of traffic lights go from red to orange to green to orange to red.

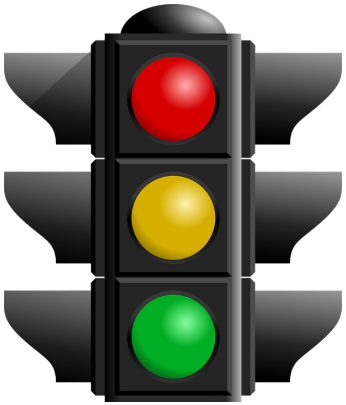
Step I: Deconstruct. what would be an appropriate intermediary state?

Please go to `www.menti.com` and enter the code 42 46 21 6.

- Red
- OrangeGreen
- OrangeRed
- Green

A three bit binary variable may do the job. Initial state might be Red.

Exercise: Traffic lights



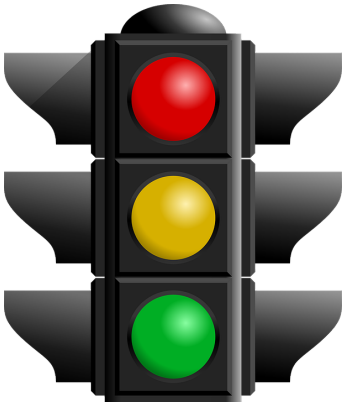
Scenario: A set of traffic lights go from red to orange to green to orange to red.

Step II: Process alphabets, FSP and LTSA
Process: TrafficLight
Actions:

- toRed
- toOrangeGreen
- toOrangeRed
- toGreen

Please go to `www.menti.com` and enter the code 92 12 15 7.

Exercise: Traffic lights

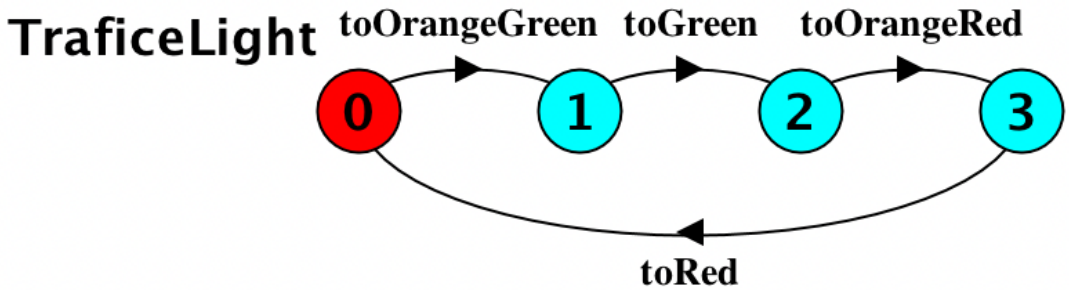


Scenario: A set of traffic lights go from red to orange to green to orange to red.

FSP:

```
TraficeLight = Red,
Red = (toOrangeGreen -> OrangeGreen),
OrangeGreen = (toGreen -> Green),
Green = (toOrangeRed -> OrangeRed),
OrangeRed = (toRed -> Red).
```

LTS:



Any questions?

Homework

Swansea University
Prifysgol Abertawe

Scenario: A day program consists of waking up, studying, and going back to sleep.

Follow the design and implementation workflow on this problem.

Summary

- When we discuss a **process** the context is a model, and when we discuss a **thread** the context is implementation.
- Finite State Process (FSP) is a type of process algebra that helps us describe processes.
- Labelled Transition System (LTS) is a graphical version of the FSP, and allows us to examine the system interactions.
- LTSA tool can be used to effectively visualise a LTS.
- Design and implementation workflow:
  1. Deconstruct
  2. Model
  3. Implement
- A process is implemented in Java using the `Thread` class.