

# Modelling Concurrent Systems Notes

Leon Lee

October 1, 2024

# Contents

<b>1</b>	<b>Introduction to Concurrent Systems</b>	<b>3</b>
1.1	Lecture 1 - Specification and Implementation . . . . .	3
1.2	Deadlocks . . . . .	5
1.3	Relations . . . . .	6
1.4	Reactive Systems . . . . .	6
<b>2</b>	<b>CCS</b>	<b>7</b>

# 1 Introduction to Concurrent Systems

## 1.1 Lecture 1 - Specification and Implementation

The main topics that are covered in this course:

- Formalising specifications as well as implementations of concurrent systems
- Studying the criteria for deciding whether an implementation meets a specification
- Techniques for proving whether an implementation meets a specification

Both specifications and implementations can be represented by means of **models of concurrency** such as **Labelled Transition Systems (LTSs)** or **Process Graphs**.

### Definition 1.1.1: Process Graphs and LTSs

A **process graph** is a triple  $(S, I, \rightarrow)$ , defined by the following:

- $S$  is a set of **states**
- $I \in S$  is an **initial state**
- $\rightarrow$  is a set of triples  $(s, a, t)$  with  $s, t \in S$ , and  $a$  an **action** drawn from a set **Act**

---

A **Labelled Transition System (LTS)** is a *process graph* without the initial state (but sometimes LTS is used as a synonym for process graph i.e. with the initial state)

---

Sometimes we will use process graphs with a fourth component  $\checkmark \subseteq S$  indicating the **final** states of the process: those in the system can terminate successfully

Specifications and implementations can not only be represented by LTSs or other models of concurrency, but also by **process algebraic expressions**, where complex processes are built up from constants for atomic actions using operators for *sequential*, *alternative*, and *parallel composition*.

The most popular process algebraic languages from literature are:

- **CCS**: the Calculus of Communicating Systems
- **CSP**: Communicating Sequential Processes
- **ACP**: the Algebra of Communicating Processing

We will be using ACP, focusing on the *partially synchronous parallel composition* operator

### Definition 1.1.2: ACP

The syntax of **ACP**, the **Algebra of Communicating Processes** features the operations

1.  $\epsilon$ : **Successful termination** (only in the optional extension  $ACP_\epsilon$ )
2.  $\delta$ : **Deadlock**
3.  $a$ : **Action constant** for each action  $a$
4.  $P \cdot Q$ : **Sequential Composition**
5.  $P + Q$ : **Summation, Choice, or Alternative Composition**
6.  $P \parallel Q$ : **Parallel Composition**
7.  $\partial_H(P)$ : **Restriction**, or **Encapsulation** for each set of visible actions  $H$
8.  $\tau_I(P)$ : **Abstraction** for each set of visible actions  $I$  (only in the optional extension  $ACT_\tau$ )

**Note:** There is also left and right parallel operators,  $P$

The atomic actions of ACP consist of all  $a, b, c$  etc from a given set  $A$  of visible actions, and one special action  $\tau$ , that is meant to be internal and invisible to the outside world.

For each application, a partial **communication function**  $\gamma : A \times A \rightarrow A$  is chosen that tells for each two visible actions  $a$  and  $b$  whether they synchronise (namely if  $\gamma$  is defined), and if so, what is result of their synchronisation:  $\gamma(a, b)$ . The communication function is required to be commutative and associative. The invisible action cannot take part in synchronisations.

### Definition 1.1.3: ACP in terms of Process Graphs

Below is the **ACP** operations in terms of process graphs extended with a predicate  $\checkmark$  that signals successful termination

- $\epsilon$  is the graph with one state and no transition. This one state is the initial state, and is marked with  $\checkmark$
- $\delta$  is the graph with one state and no transitions. This one state is the initial state. It is not marked as terminating.
- $a$  is a graph with two states (and initial and a final one) and one transition between them, labelled  $a$ . The final state is marked with  $\checkmark$
- $G \cdot H$  is the process that first performs  $G$ , and upon successful termination of  $G$  proceed with  $H$ .
- $G + H$  is obtained by taking the union of copies of  $G$  and  $H$  with disjoint sets of states, and adding a fresh state **root** which will be the initial state of  $G + H$ . For each transition  $I_G \xrightarrow{a} s$  in  $G$ , where  $I_G$  denotes the initial state of  $G$ , there will be an extra transition **root**  $\xrightarrow{a} s$ , and likewise, for each transition  $I_H \xrightarrow{a} s$  in  $H$ , where  $I_H$  denotes the initial state of  $h$ , there will be an extra transition **root**  $\xrightarrow{a} s$ . **root** is labelled with  $\checkmark$  if either  $I_G$  or  $I_H$  is.
- $G||H$  is obtained by taking the Cartesian product of the states of  $G$  and  $H$ ; that is, the states of  $G||H$  are pairs  $(s, t)$  with  $s$  a state from  $G$  and  $t$  a state from  $H$ . The initial state of  $G||H$  is the pair of initial states of  $G$  and  $H$ . A state  $(s, t)$  is labelled  $\checkmark$  iff both  $s$  and  $t$  are labelled  $\checkmark$ . The transitions are
  - $(s, t) \xrightarrow{a} (s', t)$  whenever  $s \xrightarrow{a} s'$  is a transition in  $G$
  - $(s, t) \xrightarrow{a} (s, t')$  whenever  $t \xrightarrow{a} t'$  is a transition in  $H$
  - $(s, t) \xrightarrow{c} (s', t')$  whenever  $s \xrightarrow{a} s'$  is a transition in  $G$ , and  $t \xrightarrow{b} t'$  is a transition in  $H$ , and  $\gamma(a, b) = c$

Intuitively,  $G||H$  allows all possible interleavings of actions from  $G$  and actions from  $H$ . In addition, it enables actions to synchronise with their communication partners.

- $\partial_H(G)$  is just  $G$ , but with all actions in  $G$  omitted. It is used to remove the remnants of unsuccessful communication, so that the synchronisation that is enabled by parallel composition, is enforced.
- $\tau_I(G)$  is just  $G$ , but with all actions in  $I$  renamed into  $\tau$ .

These semantics are of the **denotational** kind. Here "denotational" entails that each constant denotes a process graph (up to **isomorphism**) and each ACP operator denotes an operation on process graphs (creating a new graph out of one or two argument graphs)

## 1.2 Deadlocks

Deadlock means that a process isn't "terminated" but it cannot go further  
ACP- $\epsilon$  is an alternative model where non-finishing states can also terminate

### 1.3 Relations

#### Definition 1.3.1: Equivalence Relation

A binary relation  $\sim$  on a set  $D$  is an **equivalence relation** if it is

- **Reflexive:**  $p \sim p$  for all processes  $p \in D$
- **Symmetric:** If  $p \sim q$  then  $q \sim p$
- **Transitive:** If  $p \sim q$  and  $q \sim r$  then  $p \sim r$

#### Definition 1.3.2: Equivalence Class

If  $\sim$  is an equivalence relation on  $D$ , and  $p$  is in  $D$ , then  $[p]_{\sim}$  denotes the set of all processes in  $D$  that are equivalent to  $p$ . Such a set is called an **equivalent class**

---

Equivalence relations that split a set into more sets are called **finer** or **more discriminating**, and equivalence relations that split a set into less sets are called **coarser** or **more identifying**. The finest relation is  $e$  - every element has its own class, and the coarsest relation is the universal equivalence - everything is equivalent to itself

#### Definition 1.3.3: Equivalence Relation

A binary relation on a set  $D$  is a **preorder** ( $\sqsubseteq$ ) if it is

- **Reflexive:**  $p \sqsubseteq p$  for all processes  $p \in D$
- **Transitive:** If  $p \sqsubseteq q$  and  $q \sqsubseteq r$  then  $p \sqsubseteq r$

### 1.4 Reactive Systems

## 2 CCS