Modelling Concurrent Systems Notes

Leon Lee

September 17, 2024

Contents

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. ϵ: **Successful termination** (only in the optional extension ACP_ϵ)
2. δ: **Deadlock**
3. a: **Action constant** for each action a
4. $P \cdot Q$: **Sequential Composition**
5. $P + Q$: **Summation**, **Choice**, or **Alternative Composition**
6. $P||Q$: **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_τ)

The atomic actions of ACP consist of all a, b, c etc from a given set A of visible actions, and one special action τ, that is meant to be internal and invisible to the outside world.
For each application, a partial **communication function** $\gamma : A \times A \to A$ is chosen that tells for each two visible actions a and b whether they synchronise (namely if γ 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 ✓ that signals successful termination

- ϵ is the graph with one state and no transition. This one state is the initial state, and is marked with ✓
- δ is the graph with one state and no transitions. This one state is the initial state. It is not makred 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 ✓
- $G \cdot H$ 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 ✓ 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 ✓ iff both s and t are labelled ✓. 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 commmunication, 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 τ.

These semantics are of the **denotional** kind. Here ”denotional” 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)