Chapter 8

Labelled Transition Systems

In the previous chapters, we introduced modelling diagrams. These diagrams were all informal, in the sense, that their notions and their semantics are only defined in plain English and not in the language of mathematics. In this chapter, we introduce models of software systems that are mathematically precisely defined. We will use these models to specify software systems. Thanks to the well-defined mathematical semantics, we will be able to reason about these models using mathematical logics. The logics we will use are presented in the next chapter. We now focus on the formal models.

The important notions introduced in this chapter are:

- Labelled transition systems,
- States, paths, and executions
- Non-determinism.

8.1 Definition

The basic model for software (or hardware) systems is to consider that the system transits from one state to another depending on the action performed by the environment or by the system itself. A transition system can be visualised as graph where vertices are the states and edges represent transitions between the states. Edges can have a label. This label represents an action that triggers the state change. States can be labelled. The state labelling specifies the values of some variables in the state. We will simplify these variables to be Boolean only. Variables will therefore only be "true" or "false".

Let’s first give the formal definition of such a transition system before looking at an example:

Definition 8.1.1. A Labelled Transition System (LTS) is a tuple \((S, Act, \rightarrow, I, AP, L)\), where

- \(S\) is a set of states
- \(Act\) is a set of actions (not including the internal action \(\tau\))


CHAPTER 8. LABELLED TRANSITION SYSTEMS

• \( \rightarrow \subseteq S \times AP \times S \) is a transition relation
• \( I \subseteq S \) is a set of initial states
• \( AP \) is a set of atomic propositions
• \( L : S \rightarrow 2^{AP} \) is a labelling function

An LTS is finite if \( S, Act, \) and \( AP \) are finite.

We write \( s \xrightarrow{\alpha} s' \) for a short notation for \((s, \alpha, s') \in \rightarrow\).

To denote an internal transition – that is, a transition, the label of which is of no interest – we shall use the symbol \( \tau \).

Figure 8.1: A simple machine producing either coffee or tea.

**Example 8.1.2.** Figure 8.1 shows an example of an LTS modelling a simple beverage vending machine. The state space of this LTS is \( S = \{s_0, s_1, s_2, s_3\} \). The set of actions is \( Act = \{coin, coffee, tea\} \). The set of initial state is \( I = \{s_0\} \). For now, no atomic propositions have been defined. So, \( AP = \emptyset \).

**8.2 Execution and paths**

**8.2.1 Intuition**

Intuitively, the behaviour of an LTS can be described as follows. The LTS starts in one initial state. If a transition system has more than one initial state, an initial state is selected non-deterministically. From this selected initial state, the LTS evolves according to its transition relation. Very often, more than one transition will be possible from a given state. In that case, a transition is selected non-deterministically.

**Example 8.2.1.** Consider Figure 8.1. From the initial state, only one transition is possible. So, the LTS will always perform a coin action first and reach \( s_1 \). In \( s_1 \), there are two internal steps. The LTS will non-deterministically choose one of them and then either perform a coffee or a tea action.
8.2. EXECUTION AND PATHS

8.2.2 Executions

An execution is a possibly infinite sequence of states and actions. It has the following form:

\[ s_0 \alpha_0 s_1 \alpha_1 s_2 \alpha_2 s_2 \ldots \]

such that

\[ s_i \xrightarrow{\alpha_i} s_{i+1}, \forall i \geq 0 \]

A few properties of executions:

- executions are potentially infinite
- executions start with the initial state
- executions always end with a terminal state or are infinite

**Example 8.2.2.** Coming back to Figure 8.1, the executions of this LTS will be sequences of one or both of the following finite prefixes:

1. \( s_0 \text{ coin } s_1 \tau s_2 \text{ tea } s_0 \)
2. \( s_0 \text{ coin } s_1 \tau s_3 \text{ coffee } s_0 \)

8.2.3 Paths

A path is the projection of executions to states. A path is an infinite sequence of states:

\[ s_0 s_1 s_2 \ldots \]

such that

\[ \forall i > 0, s_i \in \text{Post}(s_{i-1}) \]

We introduce a few notations for paths:

1. \( \pi = s_0 s_1 s_2 \ldots \) denotes an infinite path.
2. \( \pi[i] \) denotes the \( i \)th element of a path.
3. \( \pi[i..] = s_0 s_1 s_2 \ldots s_i \) denotes the finite prefix up to position \( i \).
4. \( \pi[i..] = s_i s_{i+1} s_{i+2} \) denotes the suffix from position \( i \).

**Example 8.2.3.** Paths of the LTS in Figure 8.1 will be sequences of one or both of the following prefixes:

1. \( s_0 s_1 s_2 s_0 \)
2. \( s_0 s_1 s_3 s_0 \)
8.3 Pre, Post, and Reachable states

Definition 8.3.1. Given a state $s$ and an action $\alpha$, the set of the successor states of $s$ after executing $\alpha$ is defined as follows:

$$Post(s, \alpha) = \{ s' \mid s \xrightarrow{\alpha} s' \}$$

So, $Post(s, \alpha)$ contains all states than can be reached from state $s$ by performing action $\alpha$.

We lift-up the notation to the set of actions:

$$Post(s) = \bigcup_{\alpha \in Act} Post(s, \alpha)$$

So, $Post(s)$ contains all states reachable from state $s$ by performing any action.

In a similar way, we can define the set of predecessor states:

Definition 8.3.2. Given a state $s$ and an action $\alpha$, the set of the predecessor states of $s$ through action $\alpha$ is defined as follows:

$$Pre(s, \alpha) = \{ s' \mid s \xleftarrow{\alpha} s' \}$$

So, $Pre(s, \alpha)$ contains all states that can reach state $s$ by performing action $\alpha$.

We can also lift-up the notation to all actions:

$$Pre(s) = \bigcup_{\alpha \in Act} Pre(s, \alpha)$$

So, $Pre(s)$ contains all states that can reach state $s$ by performing any action.

The set of reachable states contains all states reachable from the initial state by performing any action. A state $s$ is said reachable if there exists a finite execution prefix ending with $s$.

Definition 8.3.3. (Terminal State.) State $s$ in transition system $T$ is called terminal if and only if $Post(s) = \emptyset$.

Terminal states can be used to express termination of a computation. They are also a kind of deadlock that is actually not desired. For now and until written otherwise, we assume LTSs without terminal states. This means that paths and executions are always infinite.

8.4 Atomic proposition

Atomic propositions are used to represent state information, that is, information about the current state of the system. This can be for instance, the state of registers of a computer or the current value of program variables. For verification purposes, atomic propositions can help formulate requirements about an LTS.
8.5. NON-DETERMINISM

Example 8.4.1. Consider the example in Figure 8.1. A requirement about the state of this machine might be "The machine shall always have received a coin before delivering coffee". We can create an Atomic Proposition called coinInserted which is true when a coin is inserted and becomes false when a beverage is delivered. This gives the following labelling function:

- $L(s_1) = L(s_3) = L(s_3) = \{ \text{coinInserted} \}$
- $L(s_0) = \emptyset$

Remember that the labelling function only returns propositions that hold in a given state.

We have seen here how to define atomic proposition, we will see in the next chapter how to use these propositions to define properties about the system.

8.5 Non-determinism

An LTS is said deterministic if:

1. $|I| = 1$, that is, it has exactly one initial state
2. $\forall s \in S, \forall \alpha \in Act, |Post(s, \alpha)| \leq 1$, that is, for all actions and states, it has at most one successor state.

An LTS is said non-deterministic otherwise.

8.6 Conclusion

Labelled Transition Systems (LTSs) constitute a fundamental model for software and hardware systems. An execution of a transition system is an alternating sequence of states and actions that starts in an initial state. Paths are the projection of executions to states. A state is reachable if there exists a finite execution prefix starting in an initial state and ending with that state. An LTS is deterministic if it has exactly one initial state and from a state and a given action at most one state can be reached.

8.7 Exercises

Exercise 8.7.1. Consider an oven. This oven has:

- a button to start cooking;
- a button to stop cooking;
- a button to open the door;
- a sensor to detect whether the door is closed on opened;
• when the oven stops cooking, it rings a bell.

Create an LTS modelling the behaviour of the oven behaviour. Identify a set of atomic propositions such that you can express the following properties:

• The oven is never cooking with its door opened.
• The over starts cooking only after the start button is pressed.
• After starting to cook, the oven eventually stops cooking.

Express these properties. Shortly argue whether there exists a path invalidating each property.