CO519 - Theory of Computing - Logic

Part B : Modelling systems using logic

Dominic Orchard
School of Computing, University of Kent

Last updated on April 24, 2018

If you spot any errors or have suggested edits, the notes are written in LaTeX and are available on GitHub at `http://github.com/dorchard/co519-logic`. Please fork and submit a pull request with any suggested changes.

1 Modelling in logic

In this section we are going to briefly cover using propositional formula to model systems. This is a common approach in hardware design where a complete or partial model of a circuit or processor is defined in logic, against which specifications of particular properties are checked. The starting point is to work out a good way to represent/model a system as a logical formula. In this part of the course, we will consider systems modelled as simple state machines, with states and transitions between the states describing how a system can change/evolve. We will then convert this model into a propositional formula.

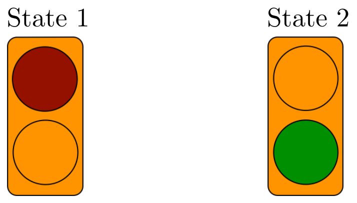
To verify a system based on its model we then need a specification of either good behaviour that we want to make sure follows from our model or of bad behaviour which we want to ensure does not follow from the model. We formulate such a specification as a propositional formula *spec*, and then prove that the following is valid:

$$\mathit{model} \rightarrow \mathit{spec}$$

The use of implication means that if the model holds then the specification must hold. Alternatively, and equivalently, we could state this as judgment $\mathit{model} \vdash \mathit{spec}$, *i.e.*, the specified behaviour follows from the model.

2 State-transition models as propositions

States Our running example will be a very simple traffic light comprising a red light and a green light, with two possible states:



Either the red light is on (left) or the green light is on (right), but never both at the same time, and there is always at least one light on. We will use two atoms r and g to represent the state of each light separately, where:

- r means the red light is on; $\neg\mathsf{r}$ therefore means the red light is off;
- g means the green light is on; $\neg\mathsf{g}$ means the green light is off.

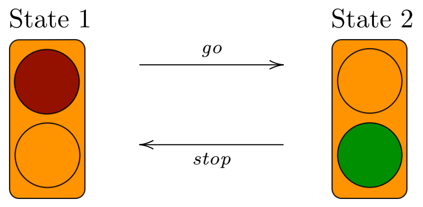
The two valid states of the system can then be modelled as two propositions:

State 1	State 2
$\mathsf{r} \wedge \neg\mathsf{g}$	$\neg\mathsf{r} \wedge \mathsf{g}$

Note that for n-propositional atoms there are 2^n possible states that can be modelled. Thus in our case, there are four possible states that we can model, two of which we want to treat as valid (the above two).

Exercise 2.1. Define two propositions, one for each of the two invalid states in the traffic light system.

State transitions So far we have modelled the states as propositions, but we want to also model the behaviour of the system in terms of the possible *transitions* between states. We can represent this with a simple diagram:



i.e., when just the red light is on it is possible to transition to a state with just the green light on, and back again.

To model state transitions we will introduce two additional atoms that model the future state of the lights in the system:

- r' for the red light being on in the *next time step*;
- g' for the green light being on in the *next time step*.

We can now express the above two transitions as implications:

$$\begin{aligned} (go): &\quad \mathsf{r} \wedge \neg\mathsf{g} \ \rightarrow\ \neg\mathsf{r}' \wedge \mathsf{g}' \\ (stop): &\quad \neg\mathsf{r} \wedge \mathsf{g} \ \rightarrow\ \mathsf{r}' \wedge \neg\mathsf{g}' \end{aligned}$$

Said another way, (go) defines that if State 1 is true now we can move to State 2 in the future (in the "next" time step of the system), and ($stop$) conversely defines that if State 2 is true now we can move to State 1 in the future.

We can now describe the full transition behaviour of the system as the conjunction of the above two formula:

$$\mathit{model} = (\mathsf{r} \wedge \neg\mathsf{g} \ \rightarrow\ \neg\mathsf{r}' \wedge \mathsf{g}') \ \wedge\ (\neg\mathsf{r} \wedge \mathsf{g} \ \rightarrow\ \mathsf{r}' \wedge \neg\mathsf{g}')$$

This provides our model of the system. You might be wondering why we don't add more to this, *e.g.*, ruling out invalid states. We will come back to this point in Section 4.

A general approach The general approach to a modelling a system in this way is to:

- Decide what to represent about the state space of the system and introduce propositional atoms for these components.
- Model state changes using "next step" atoms (usually written with an apostrophe, and called the "primed" atoms, *e.g.*, r' is read as "r prime").
- Write a propositional formula *model* using these variables which takes the conjunction of state transitions as implications.

3 Defining specifications as propositions

Consider the following property which we might want to check for our system:

> *If we are in a valid state and change state, then our new state is also valid.*

We can abstract the notion of a valid state with the following meta-level operation (you can think of this as a function from the two propositions to a proposition):

$$\mathsf{valid\text{-}state}(r, g) = (r \wedge \neg g) \vee (\neg r \wedge g)$$

From this, we can then capture our specification as the proposition:

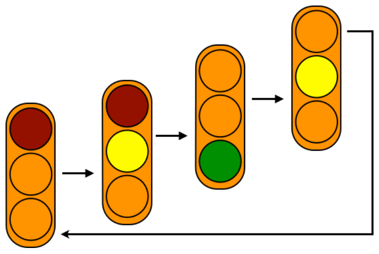
$$\mathit{specification} = \mathsf{valid\text{-}state}(\mathsf{r}, \mathsf{g}) \rightarrow \mathsf{valid\text{-}state}(\mathsf{r'}, \mathsf{g'})$$

i.e., a valid state now implies a valid state in the next time step. To verify that our system (based on its model) satisfies this property, we then need to prove that the following is true:

$$\begin{aligned} &\mathit{model} \rightarrow \mathit{specification} \\ \equiv\ &((\mathsf{r} \wedge \neg\mathsf{g} \ \rightarrow\ \neg\mathsf{r}' \wedge \mathsf{g}') \wedge (\neg\mathsf{r} \wedge \mathsf{g} \ \rightarrow\ \mathsf{r}' \wedge \neg\mathsf{g}')) \rightarrow (\mathsf{valid\text{-}state}(\mathsf{r}, \mathsf{g}) \rightarrow \mathsf{valid\text{-}state}(\mathsf{r'}, \mathsf{g'})) \\ \equiv\ &((\mathsf{r} \wedge \neg\mathsf{g} \ \rightarrow\ \neg\mathsf{r}' \wedge \mathsf{g}') \wedge (\neg\mathsf{r} \wedge \mathsf{g} \ \rightarrow\ \mathsf{r}' \wedge \neg\mathsf{g}')) \rightarrow (((\mathsf{r} \wedge \neg\mathsf{g}) \vee (\neg\mathsf{r} \wedge \mathsf{g})) \rightarrow ((\mathsf{r'} \wedge \neg\mathsf{g'}) \vee (\neg\mathsf{r'} \wedge \mathsf{g'}))) \end{aligned}$$

This is quite a big proposition so we might not want to prove it by hand. Instead, in the next part of the course we are going to use an algorithmic technique for proving that this holds (Part C, specifically **Example 5** in the notes). We will later see that this does indeed hold, and thus the system (as described by the model) satisfies our specification.

Exercise 3.1. Write down a model for a more realistic traffic light, *i.e.*, that can be described by the following states and transitions:



4 When is a model a good model?

All models are wrong but some are useful (Box, 1978)

Indeed, a model is necessarily "wrong" in the sense that a model abstracts some of the details of a real system; we are not capturing every aspect of the system, such as: how are the transitions triggered? or, what happens if power is cut to traffic light? Some models, though eliding details, are useful in the sense that we can detect when the system does not behave according to our specification or we can verify that it does behave according to our specification. We just need enough detail in the model to capture the things we want to prove.

Our running example has the model:

$$\mathit{model} = (\mathsf{r} \land \neg\mathsf{g} \;\rightarrow\; \neg\mathsf{r}' \land \mathsf{g}') \land (\neg\mathsf{r} \land \mathsf{g} \;\rightarrow\; \mathsf{r}' \land \neg\mathsf{g}')$$

We might add to this model the explicit exclusion of invalid states. Let's define the condition of the invalid states via the meta-level function:

$$\mathsf{invalid\text{-}state}(r, g) = (\neg r \land \neg g) \lor (r \land g)$$

Then we could define an alternate model as:

$$\mathit{model2} = \mathit{model} \land \neg\mathsf{invalid\text{-}state}(\mathsf{r}, \mathsf{g})$$

Thus, the new model is the old model and that the current state is not one of the invalid states.

In the case of the proving our specification from Section 3 (that valid states transition to valid states) we do not need this additional detail in the model. But we might want to have this more restrictive model if we want to prove, for example, that we can never be in an invalid state, regardless of what state we started from. This can be captured by the new specification:

$$\mathit{specification2} = \mathsf{invalid\text{-}state}(\mathsf{r}, \mathsf{g}) \lor \mathsf{invalid\text{-}state}(\mathsf{r}', \mathsf{g}')$$

and proving that the following proposition is valid:

$$\mathit{model2} \rightarrow \neg\mathit{specification2}$$

Note we are using a negative property here: we show that the new model implies that the bad behaviour is not possible.

For the old model, a similar proposition capturing the same idea is valid:

$$\mathit{model} \rightarrow \neg \mathit{specification2}$$

However, the following proposition is also valid!!!

$$\mathit{model} \rightarrow \mathit{specification2}$$

Why? If the right-hand side of the implication is true (we have an invalid state) the left-hand side is still true (the premise of each transition implication is false, therefore the implications are trivially true); the original model never excludes invalid states on their own, only as the result of a transition from a valid state. Thus, the original model was not a good model for checking the *specification2* property, but it was sufficient for *specification*.

Creating a rich enough model is up to you, and requires some care and thought about the domain and what properties are of interest.