Hoare Logic and Model Checking

Model Checking
Lecture 7: Introduction and background

Dominic Mulligan
Based on previous slides by Alan Mycroft and Mike Gordon

Programming, Logic, and Semantics Group,
University of Cambridge

Academic year 2016–2017

Administrivia

Course website:

`www.cl.cam.ac.uk/teaching/1617/HLog+ModC/`

Contact me with questions/comments: `dpm36@cam.ac.uk`

Six lecture half course

One and a half supervisions

Book

Course mostly follows material in: “Logic in Computer Science: Modelling and Reasoning about Systems” by Huth and Ryan

Copies in library

Other interesting books:

- “Model Checking” by Clarke, Grumberg, and Peled
- “Principles of Model Checking” by Baier and Katoen

Course aims

I have three aims in this course:

1. You should be able to model simple systems in NuSMV, an LTL model checker,
2. You should be able to write the world's worst CTL model checker,
3. You should know enough to be able to learn more about the first two points above in your own time.

I have six 50 minute lectures to:

- Cover 30 years of work in model checking,
- Cover a field that has given rise to multiple Turing Awards,
- Is a great example of a fusion of theory and practice.

Model checking: an application of “formal methods”

Systems as mathematical objects

One common approach to building “better” systems is:

- Treat designs and implementations as mathematical objects
- Use mathematical methods to reason about objects
- Use mathematics to describe behaviour of objects

Systems now amenable to mathematical proof

Can establish theorems about system behaviour:

- Testing can show presence of bugs, but not absence
- Mathematical proof can establish absence of bugs

Model checking from distance

We focus on one type of formal method: **model checking**

A model checker takes as input:

- A formal model of the system to be verified,
- A property of the system to be established.

As output, a model checker gives:

- Either an assurance that the property holds of the system,
- or a counterexample execution, or trace.

Origins

Model checking originated in 1980s

Pioneers were Clarke, Emerson, Queille, and Sifakis

Three decades of non-stop development

Multiple existing implementations of model checking

Dedicated conferences and model checking competitions

Clarke, Emerson, and Sifakis won 2007 Turing Award for work

What is model checking good for?

Model checking is well suited to reasoning about:

- Concurrent and reactive systems,
- Asynchronous and synchronous circuits and hardware,
- Programs with complex control flow,
- Protocols, etc.

Often have insidious, hard to reproduce bugs

Counterexamples help:

- Refine designs and implementations,
- Aid in “intelligent debugging”

Model checking advantages (I)

Model checking is largely “push button”:

- Decidable,
- Requires little interaction from user,
- Can be used by engineers with minimal training,
- Some model checkers can understand Verilog/Java/C files.

Other advantages:

- Can be used early in the design process,
- Can be used repeatedly through implementation phase,
- Many industrial success stories.

A general tool for reasoning about systems

Can be used for reasoning about **systems**, widely construed:

- Railway interlocking mechanisms, traffic lights, etc.
- Cancer pathways, metabolic networks,
- Automated planning as model checking,
- Clinical guidelines (e.g. stroke treatment and care),
- Many other applications...

Not a panacea

But there’s also disadvantages:

- State space explosion,
- Some specification languages are unintuitive,
- Still not widely used as matter of course,
- Not well suited to programs with complex data.

Temporal properties

FOL?

Q: what language should we express system properties in?

Obvious contender: **first-order logic (FOL)**

Recall semantics of first-order logic from *Logic and Proof*:

- Fix a domain,
- Provide interpretations of function symbols and relations,
- Extend to a recursively-defined denotation for formulae, defined relative to a valuation.

Semantics works well for first-order logic

“Static” truth

Truth-value assigned by denotation function never changes

Notion of truth is **static**

Works well for mathematics: $2 + 2 = 4$ unconditionally

Natural numbers do not “evolve” through time

A linguistic puzzle

Consider the following utterance:

It is raining

How should a truth value be assigned?

Note that truth of utterance is relative to time and place:

- It may be raining now in Trondheim but is not in Cambridge,
- it may be dry now in Cambridge, but may rain tonight,
- it may never rain in Cambridge again (!?)

Other examples

The lift car only moves after the doors have fully closed

After detecting a possible mid-air collision, the aircraft control software will audibly and visually alert the pilots before taking evasive action

All uses of `free` *are preceded by an accompanying use of* `malloc`

Moral: systems evolve through time

Like the weather, systems evolve through time

Time-relative properties are called **temporal properties**

Model checkers establish temporal properties of systems

“Dynamic” truth

Need a logic that can handle temporal properties

We also need an alternative notion of truth:

- What is true now need not be true in next state evolution,
- Assign truth values to claims about evolving systems.

Truth is no longer static, but has a **dynamic** flavour

Propositions hold in some **worlds** but not others

Temporal logics

Idea: use temporal logic for specifying system behaviour

Temporal logic:

- Developed by philosopher A. N. Prior in 1950s (as “tense logic”) for reasoning about time,
- Applied to verification by Pnueli (1996 Turing Award winner),
- Family of **modal logics**,
- Modalities make truth of formulae relative to time.

As modal logics, natural notion of truth is **Kripkean**

Worlds are system states, or timepoints

What is time?

Q: how should we model time?

Large design space:

- Is time continuous, or discrete?
- Does the future 'branch'? Does the past?
- Should we consider time intervals?

Temporal logics exist with all these features (and more!)

In this course:

- We focus on two temporal logics with different conceptions of time: LTL and CTL
- Both popular industrially and academically

Modelling systems

Vending machine

Consider a simple example: a vending machine

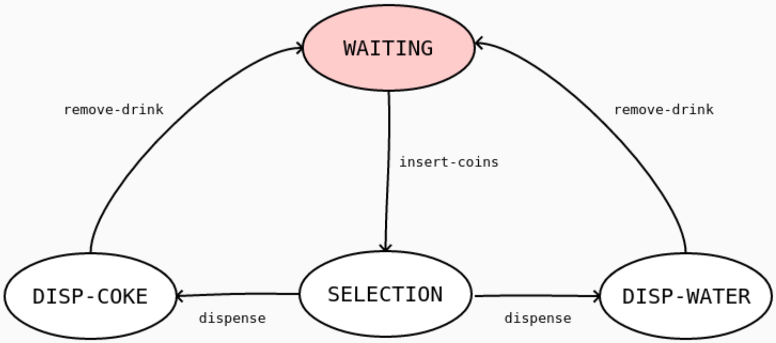
Machine can dispense coke, or water

Users insert money and make a drink selection

The machine dispenses their drink, user removes it

How can we model this “system”?

Pictorial model of vending machine



Some notes

Model is very abstract

We say nothing about e.g. internal electronics of machine

Abstractness dependent on properties we wish to establish

Some example properties

From the start state, can we get the machine to dispense water?

Is there a trace of the machine where both coke and water are dispensed?

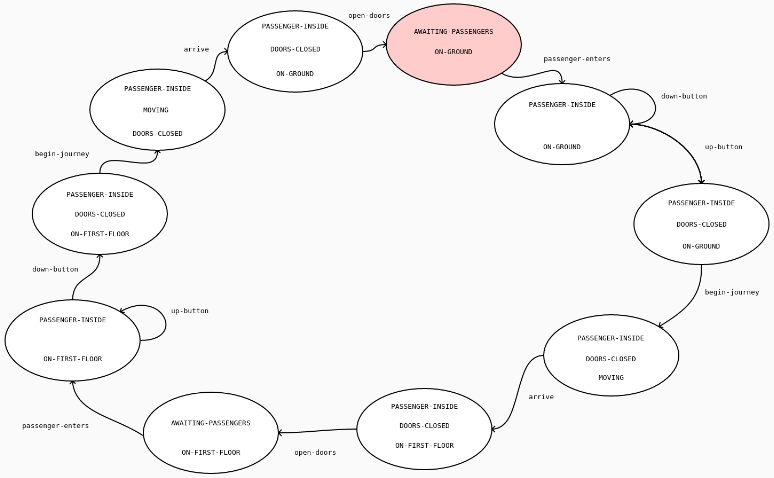
Lift

Consider an example: a lift in a two-storey building

The lift has an up and down button

It waits with its doors open for passengers

Pictorial model of lift

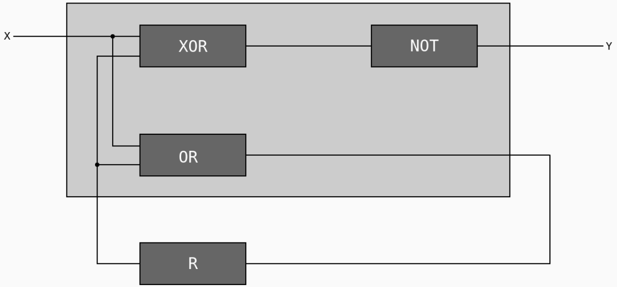


Some example properties

Can the lift ever move without a button being pressed?

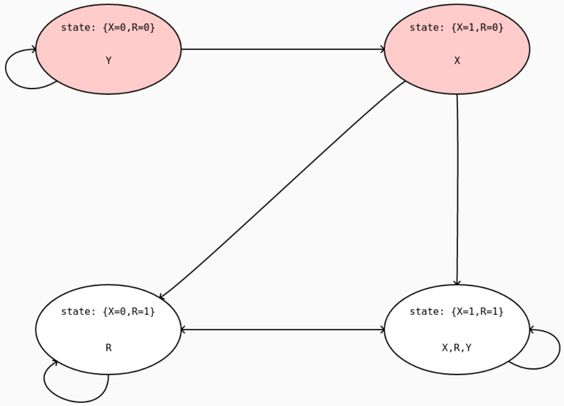
Can the lift ever keep passengers inside indefinitely?

Clocked sequential circuit



R = register, with a “memory” and initially holding 0

Pictorial model of circuit



Example properties

Output bit Y is set infinitely often

Some notes

We now have two start states: not restraining initial value of X

Representation of states now tied to system being modelled

States no longer mysterious abstract objects

A simple imperative program

```
r := x
q := 0
while y <= r do:
  r := r - y
  q := q + 1
```

Abstract representation

States modelled as tuples (representation of “state space”):

- Concrete state representation = $[0..5] \times \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}$
- State = $\langle pc, x, y, r, q \rangle$

Transitions between states follow semantics of language:

- $\langle 0, x, y, r, q \rangle \rightarrow \langle 1, x, y, x, q \rangle$
- $\langle 1, x, y, r, q \rangle \rightarrow \langle 2, x, y, r, 0 \rangle$
- If $y \leq r$ then $\langle 2, x, y, r, q \rangle \rightarrow \langle 3, x, y, r, q \rangle$
- If $r < y$ then $\langle 2, x, y, r, q \rangle \rightarrow \langle 5, x, y, r, q \rangle$
- and so on...

Example property

We will always eventually reach a point where $pc = 5$

Labelled transition systems

Transition Systems (LTS)

Examples abstracted as **transition systems (TS)**:

- A (finite) set of **states** S, with **initial states** $S_0 \subseteq S$
- A **transition relation** $\rightarrow \subseteq S \times S$

In this course **actions** are ignored (use TS rather than LTS)

We only care about transitions and states

Do not care what caused them!

Also satisfy:

- A set of **labels**, or **atomic propositions**, AP
- A **labelling function** $\mathcal{L} : S \rightarrow \mathbb{P}(AP)$

Example: handling clocked sequential circuits

Clocked sequential circuit with n inputs, m outputs, k registers

- S = $\{0,1\}^{n+k}$
- S_0 = $\{(a_1,\ldots,a_n,c_1,\ldots,c_k) \mid c_i = 0, a_j \in \{0,1\}\}$
- AP = $\{x_1,\ldots,x_n,y_1,\ldots,y_m,\ldots c_1,\ldots,c_m\}$
- $\mathcal{L}$ = $\lambda s.\ \{x_i \mid x_i = 1 \text{ at } s\} \cup \{c_i \mid c_i = 1 \text{ at } s\} \cup \{y_i \mid y_i = 1 \text{ at } s\}$
- $\rightarrow$ = derived from semantics of circuit diagram

Kripke Frames

A transition system is also known as a **Kripke Frame**

Recall from *Logic and Proof*:

Model for modal logic = Kripke Frame + labelling function
(+ satisfaction relation)

TS and labelling function is **model** of system

Temporal (modal) logics let us reason about models

Summary

- Serious software and hardware faults can and do happen
- Formal methods one way to mitigate or prevent them
- Model checking is one type of formal method
- Has advantages and disadvantages: not a silver bullet
- Can model a system using transition systems
- Can describe properties of systems using temporal logics
- Model checking: means of establishing temporal properties of models of systems