Hoare Logic and Model Checking

Model Checking Lecture 1 Supplement

Conrad Watt

Computer Laboratory, University of Cambridge, UK
http://www.cl.cam.ac.uk/~caw77

CST Part II – 2018/19

Lecture 1 Recap

- This course is about checking *properties* of *finite models*.
- Our *models* are expressed as ***Kripke structures***.
- Our *properties* are expressed using ***temporal logic***.
  - You have not seen a formal definition of temporal logic yet.
  - Temporal logics are less powerful than predicate logic, but are *decidable* and easy to automate.

Lecture 1 Recap - Kripke structures

- A Kripke structure is a transition system plus some labelling information.

- Formally, a Kripke structure/model is written as $M = (S, S_0, R, L)$

- S is a finite set of states.
- $S_0 \subseteq S$ is a subset of distinguished *starting states*.
- $R \subseteq S \times S$ is a transition relation over states.
- $L :: S \to \mathcal{P}(AP)$ is a labelling function.

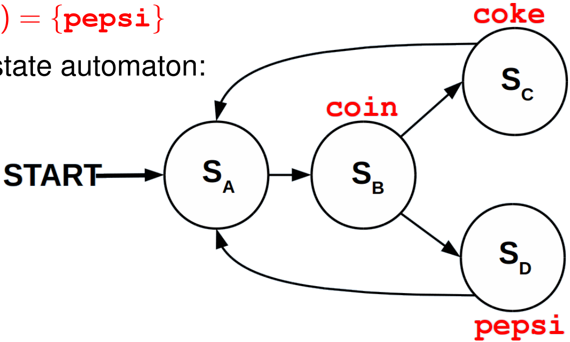
- The label function L associates each state with a set of propositional atoms (*atomic properties*).

Lecture 1 Recap - Kripke structures

Famous example - the "vending machine".

- $S \triangleq \{ \mathrm{S_A}, \mathrm{S_B}, \mathrm{S_C}, \mathrm{S_D} \}$
- $S_0 \triangleq \{ \mathrm{S_A} \}$
- $R \triangleq \{ (\mathrm{S_A}, \mathrm{S_B}), (\mathrm{S_B}, \mathrm{S_C}), (\mathrm{S_B}, \mathrm{S_D}), (\mathrm{S_C}, \mathrm{S_A}), (\mathrm{S_D}, \mathrm{S_A}) \}$
- $L(\mathrm{S_A}) = \{\}$
  $L(\mathrm{S_B}) = \{\mathtt{coin}\}$
  $L(\mathrm{S_C}) = \{\mathtt{coke}\}$
  $L(\mathrm{S_D}) = \{\mathtt{pepsi}\}$

As finite state automaton:



Lecture 1 Recap - Modelling

- Given a definition of a system, we need to work out how to represent it as a Kripke structure.
  - Then we can begin checking the truth of temporal logic properties.

Thread 1	Thread 2
`0: IF LOCK=0 THEN LOCK:=1;`	`0: IF LOCK=0 THEN LOCK:=1;`
`1: X:=1;`	`1: X:=2;`
`2: IF LOCK=1 THEN LOCK:=0;`	`2: IF LOCK=1 THEN LOCK:=0;`
`3:`	`3:`

Assumptions (always good practice to state):

- All variables 0-initialized.
- One entire line executed per step.
- Thread scheduling is non-deterministic.
- "**IF**" lines are only *scheduled* if condition is true.

Lecture 1 Example

Thread 1	Thread 2
`0:   IF LOCK=0 THEN LOCK:=1;`	`0:   IF LOCK=0 THEN LOCK:=1;`
`1:   X:=1;`	`1:   X:=2;`
`2:   IF LOCK=1 THEN LOCK:=0;`	`2:   IF LOCK=1 THEN LOCK:=0;`
`3:`	`3:`

- Need to pick (S, S_0, R, L) to *model* the program.
- Observe: two program counters, two variables - all integers.
- Individual state $(s \in S) \triangleq (pc_1, pc_2, \mathtt{LOCK}, \mathtt{X})$
- Represent state set as $S \triangleq \{0, 1, 2, 3\} \times \{0, 1, 2, 3\} \times \{0, 1\} \times \{0, 1, 2\}$
- S_0 is just the singleton set $\{\ (0, 0, 0, 0)\ \}$
- Why not just define $S \triangleq \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}$?

Lecture 1 Example

Thread 1	Thread 2
`0:   IF LOCK=0 THEN LOCK:=1;`	`0:   IF LOCK=0 THEN LOCK:=1;`
`1:   X:=1;`	`1:   X:=2;`
`2:   IF LOCK=1 THEN LOCK:=0;`	`2:   IF LOCK=1 THEN LOCK:=0;`
`3:`	`3:`

- Need to pick (S, S_0, R, L) to *model* the program.
- $S \triangleq \{0,1,2,3\} \times \{0,1,2,3\} \times \{0,1\} \times \{0,1,2\}$
- $S_0 \triangleq \{\ (0,0,0,0)\ \}$
- To define R, look at how states should evolve over time.
- e.g. in all states of the form $(0,0,0,-)$, either thread 1 or 2 will take the **LOCK** and advance their pc

Lecture 1 Example

Thread 1	Thread 2
0: `IF LOCK=0 THEN LOCK:=1;`	0: `IF LOCK=0 THEN LOCK:=1;`
1: `X:=1;`	1: `X:=2;`
2: `IF LOCK=1 THEN LOCK:=0;`	2: `IF LOCK=1 THEN LOCK:=0;`
3:	3:

- Need to pick (S, S_0, R, L) to *model* the program.
- $S \triangleq \{0,1,2,3\} \times \{0,1,2,3\} \times \{0,1\} \times \{0,1,2\}$
- $S_0 \triangleq \{\ (0,0,0,0)\ \}$
- Define R:

$$
\begin{aligned}
\forall pc_1\ pc_2\ lock\ x.\ & R\ (0, pc_2, 0, x) && (1, pc_2, 1, x) && \wedge \\
& R\ (1, pc_2, lock, x) && (2, pc_2, lock, 1) && \wedge \\
& R\ (2, pc_2, 1, x) && (3, pc_2, 0, x) && \wedge \\
& R\ (pc_1, 0, 0, x) && (pc_1, 1, 1, x) && \wedge \\
& R\ (pc_1, 1, lock, x) && (pc_1, 2, lock, 2) && \wedge \\
& R\ (pc_1, 2, 1, x) && (pc_1, 3, 0, x) &&
\end{aligned}
$$

Lecture 1 Example

Thread 1	Thread 2
`0:   IF LOCK=0 THEN LOCK:=1;`	`0:   IF LOCK=0 THEN LOCK:=1;`
`1:   X:=1;`	`1:   X:=2;`
`2:   IF LOCK=1 THEN LOCK:=0;`	`2:   IF LOCK=1 THEN LOCK:=0;`
`3:`	`3:`

- Need to pick (S, S_0, R, L) to *model* the program.
- What about L?
- Depends on which properties we want to prove.
- Any potential predicate on program state can be turned into a label.
- e.g. all states satisfying $(3, 3, -, -)$ could be labelled **`finished`**
- Then can express and check the temporal property "every execution of the program eventually reaches **`finished`**".
  - i.e. all lines of code in both threads are executed
  - You will later see how to state this property formally.

Lecture 1 Example

Thread 1	Thread 2
`0: IF LOCK=0 THEN LOCK:=1;`	`0: IF LOCK=0 THEN LOCK:=1;`
`1: X:=1;`	`1: X:=2;`
`2: IF LOCK=1 THEN LOCK:=0;`	`2: IF LOCK=1 THEN LOCK:=0;`
`3:`	`3:`

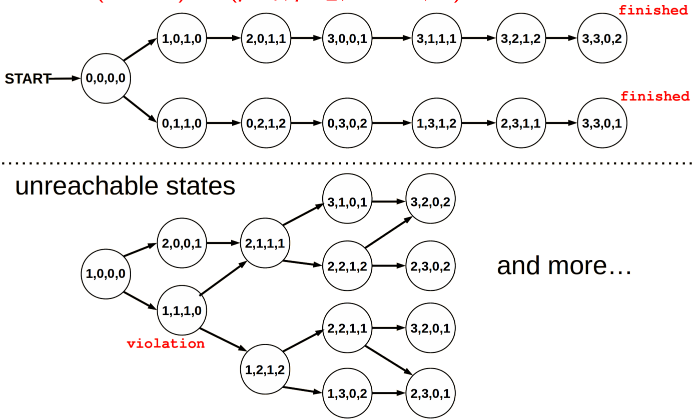
- Let's imagine we want to prove *mutual exclusion*.
- That is, both threads will never be "inside" the locked code at once.
- Can label all states satisfying $(1, 1, -, -)$ with **`violation`**.
- Then mutual exclusion can be expressed as "no execution of the program will reach a state satisfying **`violation`**".
- Equivalently, "all reachable states satisfy $\neg$**`violation`**".

Drawing Kripke Structures

Thread 1	Thread 2
`0: IF LOCK=0 THEN LOCK:=1;`	`0: IF LOCK=0 THEN LOCK:=1;`
`1: X:=1;`	`1: X:=2;`
`2: IF LOCK=1 THEN LOCK:=0;`	`2: IF LOCK=1 THEN LOCK:=0;`
`3:`	`3:`

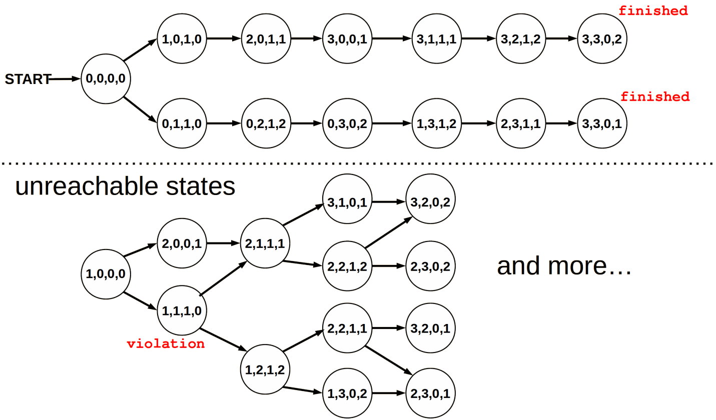
Draw Kripke structures as *finite state automata*.

Reminder: $(s \in S) \triangleq (pc_1, pc_2, \mathtt{LOCK}, \mathtt{X})$



Drawing Kripke Structures

- Convention: annotate states with atomic properties.
- Strictly speaking, should draw unreachable states too.
  - But sometimes missed out. Use your judgement.
- You will never be asked to draw something this big. (96 states!)



Lecture 1 Recap - Some Definitions

- A transition system or model is called *left-total* if no state is a “dead-end”.
- That is, every state can transition to another state.

- $R\ s_1\ s_2 \equiv (s_1, s_2) \in R$
- R^* is the transitive closure of R.

- A *path*, often written π, is a sequence of states.
- Path $R\ s\ \pi$ is true iff π is a path starting at s such that successive states in π are related by R.
- $\pi(i)$ and $\pi\ i$ are both syntax for the ith element of path π.
- $\pi{\downarrow}i$ is the suffix of π starting from its ith element.

Lecture 1 Recap - Other Slide Content

Other content from lecture 1 (less important):

- Reinterpreting slightly different formal models as Kripke models.
- Modelling systems with input (treat it as an extra component of the state).
- Historical syntax for models/interpretations.