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).

Famous example - the “vending machine”.

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

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.

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

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

Need to pick \((S, S_0, R, L)\) to model the program.

Observe: two program counters, two variables - all integers.

Individual state \((s \in S) \triangleq (pc_1, pc_2, \text{LOCK}, x)\)

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 \(S \triangleq \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}\)?

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 \(R\):
\[
\forall pc_1 pc_2 \text{lock } x. R \left(0, pc_2, 0, x \right) \quad \left(1, pc_2, 1, x \right) \quad \left(2, pc_2, 0, x \right) \quad \left(3, pc_2, 0, x \right)
\]
\[
\forall pc_1, pc_2, \text{lock } x. R \left(1, pc_2, \text{lock}, x \right) \quad \left(2, pc_2, \text{lock}, 1 \right)
\]
\[
\forall pc_1, pc_2, x. R \left(2, 0, pc_2, 1, x \right) \quad \left(3, pc_2, 0, x \right)
\]
\[
\forall pc, pc_1, pc_2, x. R \left(pc, pc_1, pc_2, \text{lock}, 1 \right)
\]
\[
\forall pc, pc_1, pc_2, x. R \left(pc, pc_1, \text{lock}, x \right) \quad \left(pc_1, pc_2, 2, \text{lock}, 2 \right)
\]
\[
\forall pc, pc_1, pc_2, x. R \left(pc, pc_1, 2, \text{lock}, x \right) \quad \left(pc_1, 3, 0, x \right)
\]
Lecture 1 Example

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

Thread 2
0: IF LOCK=0 THEN LOCK:=1;
1: X:=2;
2: IF LOCK=1 THEN LOCK:=0;
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.

Drawing Kripke Structures

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

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

Draw Kripke structures as finite state automata.
Reminder: \((s \in S) \triangleq (pc_1, pc_2, LOCK, X)\)

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 Example

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

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!)
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 \( \pi \), is a sequence of states.

Path \( R \ s \ \pi \) is true iff \( \pi \) is a path starting at \( s \) such that successive states in \( \pi \) are related by \( R \).

\( \pi(i) \) and \( \pi\ i \) are both syntax for the \( i \)th element of path \( \pi \).

\( \pi\downarrow i \) is the suffix of \( \pi \) starting from its \( i \)th element.

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.