Formal Methods - 02_Modeling_Transistion_Systems

Francesco Penasa

February 21, 2020

lect 02 material 2020/02/20:
HANDOUTS : `http://disi.unitn.it/~rseba/DIDATTICA/fm2020/02_TRANSITION_SYSTEMS_HANDOUT.pdf`
SLIDES : `http://disi.unitn.it/~rseba/DIDATTICA/fm2020/02_TRANSITION_SYSTEMS_SLIDES.pdf`

1 Transistion Systems as Kripke Models

Kripke models are used to describe reactive systems:

1. nonterminating systems with **infinite** behaviours
2. **dynamic evolution** of modeled systems;
3. **state:** snapshot of the system (value of vars, program counter, etc.)
4. **realistic representation**, the system can be simulated and validated before the implementation

Kripke model: formal definition

$$< S, I, R, AP, L >$$

1. $S =$ a **finite** set of states
2. $I =$ initial states
3. $R =$ set of **directed** transisitions rules $R = S\dot{S}$
4. $AP =$ **observable variables** (boolean variables)
5. $L =$ labeling function $L : S =\rightarrow 2^{AP}$; tells us what is the value of the boolean variables for each states
6. R is Total, all states have an outgoing link, we don't consider sink states.

In Kripke structures the value of every variable is always assigned in each state. Each state identifies univocally the values of the atomic propositions which hold there.

TODO INSERT IMAGE GRAPH

$$N, T, C == observable\_variables$$

1. N = not in critical section and not trying to acquire the lock
2. turn = who has the rigth to enter the critical section
3. arc = step = when one process makes a move

A **path** is an **infinite** sequence of states

Path in a Kripke Model A **path** in a Kripke model M is an infinite sequence of states. It must start from an initial state and use existing transition rules.

$$\pi = s_0, s_1, s_2, \ldots \in S^\omega$$

such that $s_0 \in I$ and all other $s_i \in R$
Reachable if there is a path from the initial state to the target state.

1.1 Composing Kripke Models

Complex Models are obtained by composition of smaller ones. Both compositions are associative.

1.1.1 Asynchronous Composition

At each time istant, one component is selected to perform a transition (for instance communication protocols). The composition is obtained as all the possible interleaving possibilities between two models, **no contemporary moves**. The time is continous and nothing can happen in the same istant. **We must be really careful about the shared variables**. The main bounds are about the initial model and the respect to the **shared**observable variable.

$$M_1 || M_2 || M_3$$

1.1.2 Synchronous Composition

Components evolve in parallel, fixed time slots in which all the component perform a trasition (for instance a sequential hardware circuit). The boundaries are the same as the asynchronous state with the only difference about the transition rules.

$$M_1 \times M_2 \times M_3$$

2 Languages for Transition Systems

We only reason in therms of evolution of variables, we don't care or need the graph representation. We care about the possible moves. A Kripke model is usually presented in a structured language.

Each component is presented by specifying

1. state variables (AP, S and the map function L)
2. initial values for state variables (initial state)
3. instruction (transitions)

2.1 SMV (NoSMV and NoXMV)

No sequence, no implicit order (no program counter), we write the relations.

1. booleans in many forms
2. declarations of state variable; assignment for the initial state; assignment for the transition relation
3. allows both synchronous and asynchronous composition (more sync)

Big systems are represented in modular parts

2.2 PROMELA

Sequential execution

1. C-like
2. set of processes that interacts with **shared variables** and **communication channels**
3. allows both synchronous and asynchronous composition (more async)

2.3 SDL

For infinite state machine and usage of continuos time.

1. booleans in many forms
2. allows TIME representations
3. represents states, message I/O, conditions, clock operations, subroutines
4. allows both synchronous and asynchronous composition (async)

3 Properties of Transition Systems

3.1 Safety Properties

Bad events never happens

1. Deadlock freedom
2. Mutual Exclusion

Can be refuted by a *finite* behaviour

3.2 Liveness properties

Something desiderable will eventually happen (sooner or later)

1. Starvation Freedom

Can be refuted by infinite behaviour: presented typically as a loop.

3.3 Fairness

Something desirable will happen infinitely often. Whenever a subroutine takes control, it will always return it.
Can be refuted by infinite behaviour: presented typically as a loop.

4 Questions

1. Is **turn** an observable variable?