Reactive Systems: Kripke Structures and Automata

Timo Latvala

January 28, 2004

Properties of systems

- invariants: “the system never reaches a bad state”; in each reachable state P holds
  - deadlock freedom
  - mutual exclusion etc.
- safety: “a bad sequence of things does no happen”; each violation of a safety property can be observed by looking at finite “history” of the system behavior. Example: “the system should not reboot without the reset button being pressed first”.

  Invariants are a simple subclass of safety properties.
- liveness: “something good should eventually happen”; Example: X occurs infinitely often. Alternative formulation: “there is progress in the system”.

How to Specify and Check Properties?

Even simple looking systems can easily have thousands or millions of reachable states and enabled transitions. It does not make sense to graphically represent or manually explore each node and arc of so large graphs.

Invariant properties, such as deadlock freedom or the unreachability of a forbidden state, can be formulated as conditions concerning one state at a time, which a reachability analysis algorithm can easily verify when adding nodes to the reachability graph.

Formulating and verifying safety, and (especially) liveness properties (“something good should eventually happen”) call for new methods, such as *temporal logic* and *model checking*.

Kripke Structures

Temporal logics are traditionally defined in terms of *Kripke structures*. A Kripke structure is basically a graph having the reachable states of the system as nodes and state transitions of the system as edges. It also contains a labeling of the states of the system with properties that hold in each state.

To obtain a Kripke structure form the reachability graph one first needs to fix a set of atomic propositions AP, which denote the properties of individual states we are interested in. The labeling of the states (with markings) of the reachability graph is replaced with the labeling showing which atomic propositions hold in that state. (Note: This does NOT mean states with the same label should be merged!)

After this the labels (with transitions) on the arcs of the reachability graph are removed, and the result is the Kripke structure. (Note: The removal of arc labels might result in a merging of some edges.)

Definition of Kripke Structures

Definition 1 *Let AP be a non-empty set of atomic propositions. A Kripke structure is a four tuple $M = (S, s^0, R, L)$, where*

- *S is a finite set of states,*
- *$s^0 \in S$ is an initial state,*
- *$R \subseteq S \times S$ is a transition relation, for which it holds that $\forall s \in S : \exists s' \in S : (s, s') \in R$, and*
- *$L : S \to 2^{AP}$ is labeling, a function which labels each state with the atomic propositions which hold in that state.*

Kripke Structures

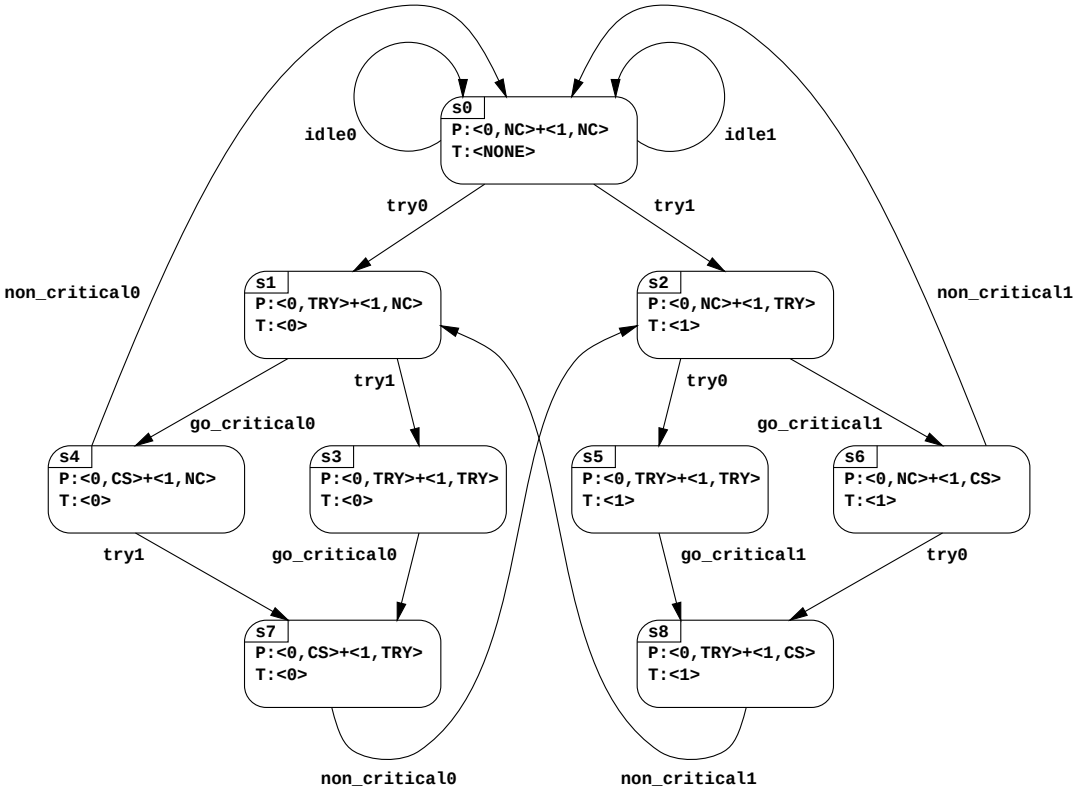
Kripke structures are the model used to give semantics (definition of when a specified property holds) for the most widely used specification languages for reactive systems, namely *temporal logics*.

Kripke structures can be seen as describing the behavior of the modeled system in an modeling language independent manner. Therefore, temporal logics are really modeling formalism independent. The definition of atomic propositions is the only thing that can needs to be adjusted for each formalism.

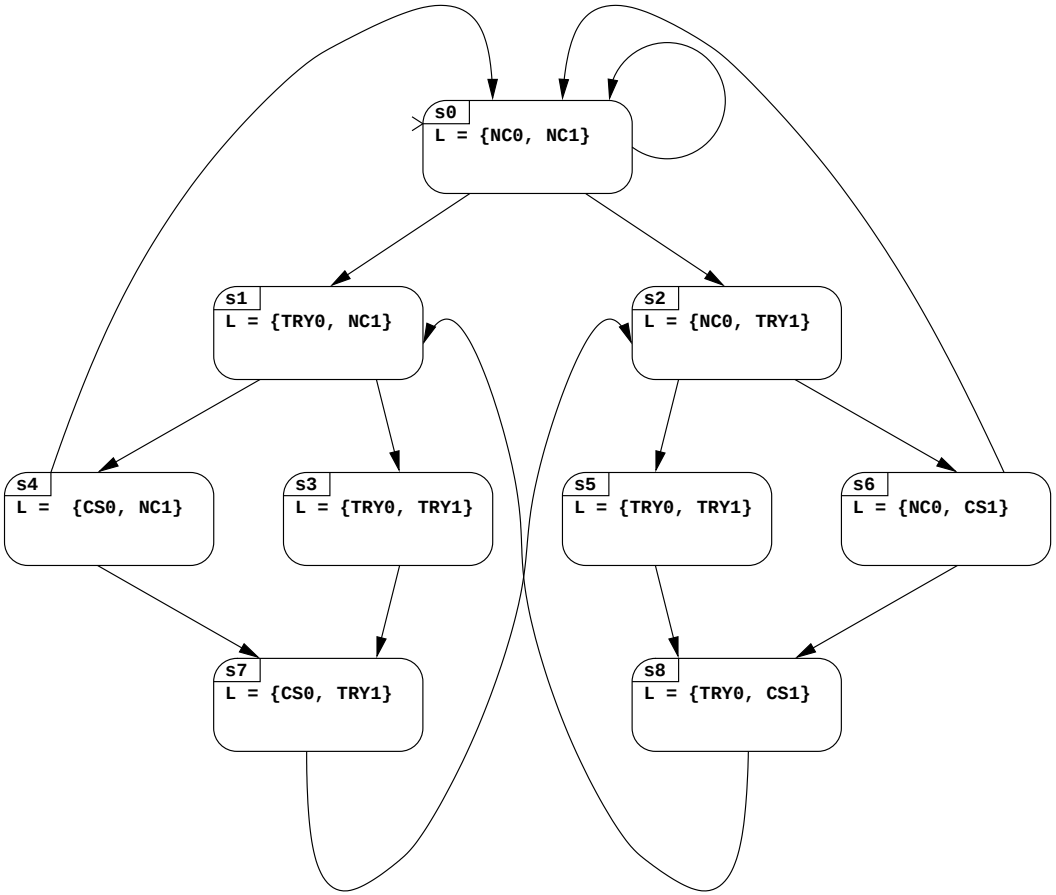
Note that in Kripke structures deadlock states are disallowed. This is for technical reasons (it simplifies the theory somewhat). Should a reachability graph contain a deadlock state, we add an edge from the deadlock state back to itself in the corresponding Kripke structure.

Example:Reachability Graph of a Mutex System

The following is a reachability graph of a (high-level) Petri net model of a Mutex algorithm.



Example: Kripke Structure of the Mutex System



Kripke Structures and Automata

As can be seen from the definition, Kripke structures have a close relationship with automata.

The changes are the following:

- labeling is on states instead of having labels on arcs,
- the labeling consists of a subset of AP instead of an element of Σ,
- there is at most one arc between any two states, and
- there is no definition of final states. (One can think of all the states being final.)

From Kripke Structure to Finite State Automaton

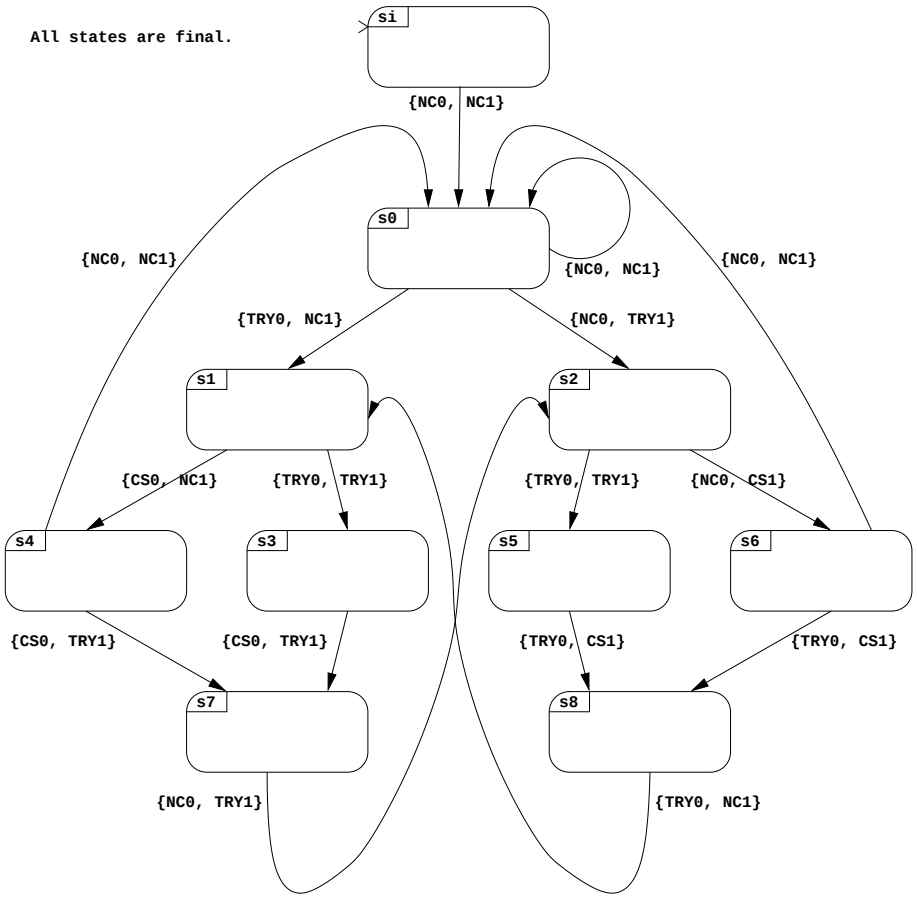
An often used trick in model checking is to actually use an automaton directly derived from the Kripke structure. We define an automaton $\mathcal{A}_M$, which accepts exactly the (finite) sequences of valuations in a *path* through the Kripke structure.

From Kripke Structure to Finite State Automaton

Let $M = (S, s^0, R, L)$ be a Kripke structure over a set of atomic propositions AP. Define an automaton $\mathcal{A}_M = (\Sigma, S_M, S_M^0, \Delta_M, F_M)$, where

- $\Sigma = 2^{AP}$,
- $S_M = S \cup \{s^i\}$,
- $S_M^0 = \{s^i\}$,
- For all $s, s' \in S_M, a \in \Sigma$: $(s, a, s') \in \Delta_M$ iff
  $L(s') = a$ and $(((s, s') \in R)$ or $(s = s^i$ and $s' = s^0))$; and
- $F_M = S_M$.

Example: The Automaton $\mathcal{A}_M$



Model Checking of Safety Properties with Automata

An example safety property for a path in the Kripke structure is the following:

Spec: A (finite) path in the Kripke structure satisfies *Spec* iff it does not contain a state having both *CR0* and *CR1* holding at the same time.

It is easy to give a specification automaton $\mathcal{S}$ which accepts all sequences of $\mathcal{A}_M$ corresponding to paths which satisfy this property. (A two-state deterministic automaton will suffice.)

In this case the complement of the specification $\neg Spec$ is the following:

A (finite) path in the Kripke structure satisfies $\neg Spec$ iff it contains a state in which both $CR0$ and $CR1$ hold at the same time.

It can be checked by an automaton $\overline{S}$ (also a simple two-state deterministic automaton).

Now all paths through the Kripke structure satisfy $Spec$ iff there is no path which satisfies $\neg Spec$.

Implementing Safety Model Checking

To implement this, we can use automata theory to obtain the product automaton $\mathcal{P} = \mathcal{A}_M \times \overline{\mathcal{S}}$, and check that indeed $L(\mathcal{P}) = \emptyset$. (By observing that no accepting state of $\mathcal{P}$ is reachable from its initial states.)

Thus in our running example the safety property $Spec$ holds for all paths through the Kripke structure M.

Why not Use Automata as the Specification Language?

For even slightly more complicated specifications expressing the specification directly as an automaton can be too complicated. This is one of the reasons temporal logics are more widely used as a specification formalisms than automata.

It is expensive to complement automata. Thus even if the specification can be easily expressed as an automaton, its complement can be too big to handle.

Liveness properties cannot be handled by finite state automata on finite strings. Thus to handle liveness, we have to change our definition of automata. This brings new problems we have not encountered so far.

Automata are, however, one of the main implementation techniques in implementing model checking algorithms for more expressive temporal logics, a subject which we will discuss next.