# Introduction to Model Checking

Keijo Heljanko

Department of Computer Science
University of Helsinki
keijo.heljanko@helsinki.fi

3.9-2019

# Reactive Systems

Reactive systems are a class of software and/or hardware systems which have ongoing behavior where they react to inputs provided by the environment.
(They do not terminate.)
Examples of reactive systems include:

- Mobile phones
- Data communication protocols (Internet,telephone switches)
- Traffic lights
- Elevators (lifts)
- Operating systems

# Reactive Systems vs. Algorithms

Reactive systems do not fulfill the definition of an algorithm, which says that an algorithm should:

- Terminate
- Upon termination, provide a (hopefully correct) return value.

If we want to specify the correctness of an algorithm, we usually specify it as follows:

- The algorithm should terminate on all (allowed) inputs
- On termination, the provided output should correct (with respect to a specification).

# Software failures

Software is used widely in many applications where a bug in the system can cause large damage:

- Safety critical systems: airplane control systems, medical care, train signalling systems, air traffic control, power plants, etc.
- Economically critical systems: e-commerce systems, Internet, microprocessors, etc.

# A Software Bug in Mars: Pathfinder

- In a real time operating system a high priority process waits for a lower priority process to release a lock but the lower priority process never gets runtime by the OS scheduler. Resulted in lockups of the Mars Pathfinder software that caused watchdog reboots.
- Problem could be reproduced with an identical copy on the spaceship after 18 hours. The software running on Mars was patched from the Earth.

# Finding Bugs in Reactive Systems

The principal methods for the validation of complex reactive systems are:

- Testing (using the system itself)
- Simulation (using a model of the system)
- Deductive verification (mathematical (manual) proof of correctness, in practice done with computer aided proof assistants/proof checkers)
- Model Checking ($\approx$ exhaustive testing of a model of the system)

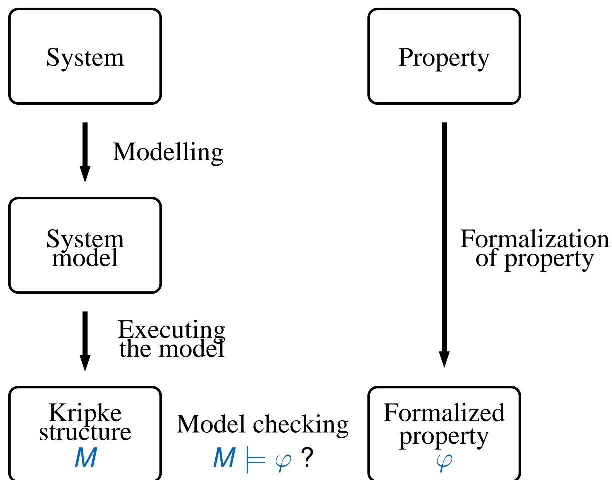
This course will focus on the model checking approach.

# Model Checking

In model checking every execution of the model of the system is simulated obtaining a Kripke structure $M$ describing all its behaviors. $M$ is then checked against a system property $\varphi$:

- Yes: The system functions according to the specified property (denoted $M \models \varphi$).
  The symbol $\models$ is pronounced "models",
  hence the term model checking.
- No: The system is incorrect (denoted $M \not\models \varphi$), a counterexample is returned: an execution of the system which does not satisfy the property.

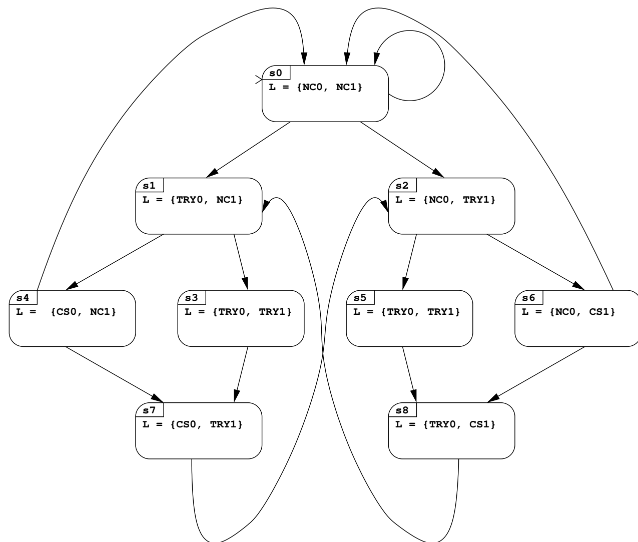
# Models and Properties



# Kripke Structures

Kripke structure is a fully modelling language independent way of representing the behavior of parallel and distributed system. Kripke structures are graphs which describe all the possible executions of the system, where all internal state information has been hidden, except for some interesting atomic propositions.

# Example: Mutex - Kripke structure



# Kripke structure

Kripke structure is a directed graph, where:

- The states of the graph are all possible reachable states of the system.
- There is an arc from state $s$ to state $s'$ if and only if ( iff from now on) it is possible to move with an atomic action from state $s$ to the state $s'$.
- The valuation $L$ of each state contains exactly those atomic propositions which hold in that state.

# Formal Definition

## Definition

Let $AP$ be a finite set of atomic propositions. 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 the initial state (marked with a wedge),
- $R \subseteq S \times S$ is the transition relation, ($(s, s') \in R$ is drawn as an arc from $s$ to $s'$), and
- $L : S \to 2^{AP}$ is a valuation, i.e. a function which maps each state to those atomic propositions which hold in that state.

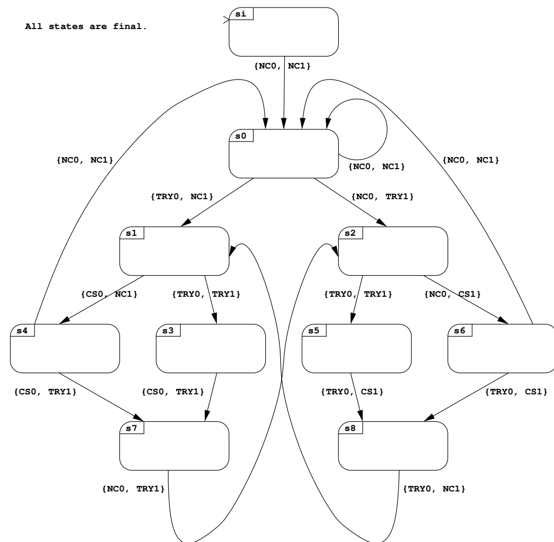
# Kripke Structures and Automata

Kripke structures have a close relationship with finite state automata (FSA):
The changes are the following:

- labelling is on states instead of having labels on arcs,
- alphabet $\Sigma$ consists of the subsets of $AP$,
- there is at most one arc between any two states, and
- there is no definition of final states.
  (All the states are final.)

It is easy to derive a FSA out of a Kripke structure.

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



# Model Checking - Ingredients

- A way of modelling the system conveniently - modelling language
- A way of describing all the behaviors of the system model in a modelling language independent way - Kripke structure
- A way of specifying properties - assertions, automata, regular expressions, temporal logics
- An algorithm to check whether the property holds for the system - model checker

# Benefits of Model Checking

- In principle automated: Given a system model and a property, the model checking algorithm is fully automatic
- Counterexamples are valuable for debugging
- Already the process of modelling catches a large percentage of the bugs: rapid prototyping of concurrency related features

# Drawbacks of Model Checking

- State explosion problem: Capacity limits of model checkers often exceeded
- Manual modelling often needed:
  - Model checker used might not support all features of the implementation language
  - Abstraction needed to overcome capacity problems
- Reverse engineering of existing already implemented systems to obtain models is time consuming and often futile

# Model Checking in the Industry

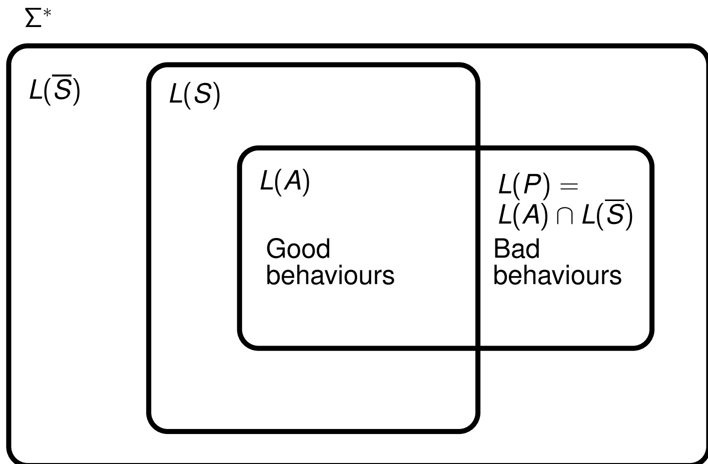
- Microprocessor design: All major microprocessor manufacturers use model checking methods as a part of their design process
- Design of Data-Communications Protocol Software: Model checkers have been used as rapid prototyping systems for validating new data-communications protocols under standardization.
- Critical Software: NASA space program is model checking code used by the space program.
- Operating Systems: Microsoft is using model checking to verify the correct functioning of new Windows device drivers.

# Automata Theoretic Approach

A short theory of model checking using automata

- Assume you have a finite state automaton (FSA) of the behavior of the system $A$
  (see automaton $A_M$ obtained from the Kripke structure $M$ as an example)
- Assume the specified property is also specified with an FSA $S$
- Now the system fulfils the specification, if the language of the system is contained in the language of the specification:
  i.e., it holds that $L(A) \subseteq L(S)$

# Language Inclusions



# Automata Theoretic Approach (cnt.)

- We need to generate the product automaton: $P = A \cap \overline{S}$, where $\overline{S}$ is an automaton which accepts the complement language of $L(S)$

# Automata Theoretic Approach (cnt.)

- If $L(P) = \emptyset$, i.e., $P$ does not accept any word, then the property holds and thus the system is correct
- Otherwise, there is some run of $P$ which violates the specification, and we can generate a counterexample execution of the system from it (more on this later)

# State Explosion from Intersection

- Note, however, that even if $A_1, A_2, A_3, A_4$ have $k$ states each, the automaton $A'_4 = A_1 \cap A_2 \cap A_3 \cap A_4$ (sometimes alternatively called the synchronous product and denoted $A'_4 = A_1 \times A_2 \times A_3 \times A_4$) can have $k^4$ states, and thus in the general $A'_i$ will have $k^i$ states.
- Therefore even if a single use of $\cap$ is polynomial, repeated applications often will result in a state explosion problem.
- In fact, the use of $\times$ as demonstrated above could in principle be used to compose the behavior of a parallel system from its components.