

CITS7211 Modelling Complex Systems

Lecture Six

Modelling Concurrent Systems

*Based on material from Formal Methods
CITS4221*

Software vs Physical Systems

- As with many of the systems we have seen, concurrent software systems may be represented using interacting automata, except that rather than using spatial dimensions, the topology of the system represents communication channels between the automata.

Modeling Systems

- In this lecture we examine concurrent software as a complex system. Concurrent software exhibits many of the properties we have seen in complex physical systems, including a high degree of unpredictability, emergent behaviour and local simplicity.
- Model-checking takes a specification, or source code and examines all the possible behaviors of the model.

Model-Checking Properties

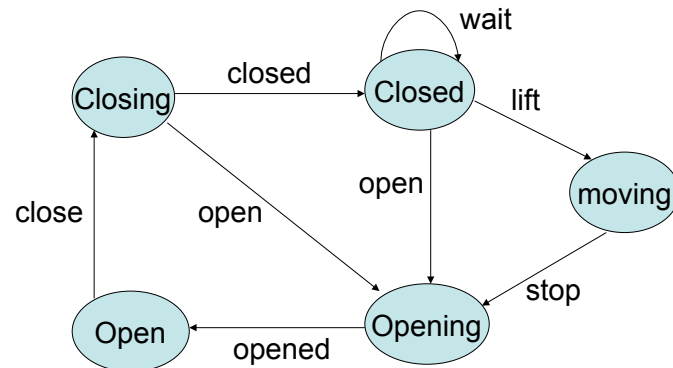
- Model-checking takes a specification and allows queries over its behavior.
- For example, given assertions P and Q, we can ask:
 - Is it possible that P is true until Q is true?
 - Is it necessary that P is true until Q is true?
 - Is it possible that Q is never true?
 - After P has been true is it essential that Q will eventually be true?

The models

Model checking uses a labeled graph to model a system where:

- the nodes are possible states of the system, labeled by boolean propositions for each assertion.
- the directed edges represent which states can be reached from other states by an operation of the system.
- This makes it easy to combine systems and model concurrency.

Example: A Lift Door



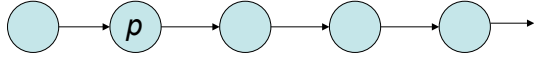
Properties

In each state we can examine which propositions are true. We also want to check how the different properties interact over time.

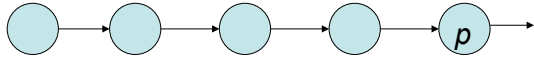
Temporal logic is used to represent these properties. Propositional logic (*and*, *or*, *not*) is extended with temporal operators: *tomorrow*, *until*, *always*, *eventually*, *for some future*, *for all futures*.

Temporal logic

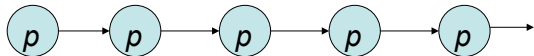
- $X p$ - at the next state p



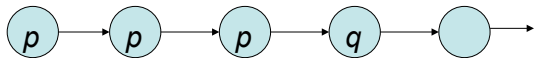
- $F p$ - (or $\langle \rangle p$) eventually p



- $G p$ - (or $[\] p$) always p

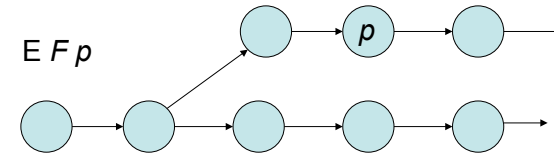


- $p U q$ - p until q

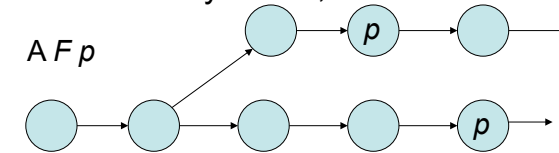


Temporal Logic

- $E \alpha$ - there exists a future such that α .

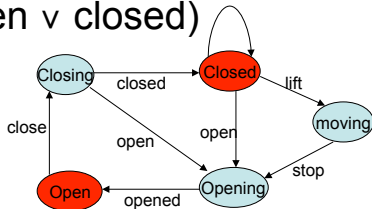


- $A \alpha$ - for every future, α .

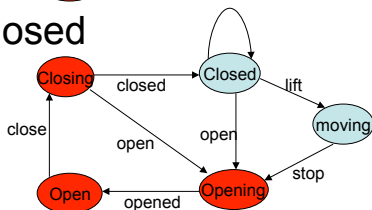


Properties in Models

• $A F (\text{open} \vee \text{closed})$



• $E G \neg \text{closed}$



The model checking problem

- Given a model (graph / xml / source code) and a specification (temporal logic) either:
 - confirm the model satisfies the specification; or
 - produce a counter-example showing how the model fails.

Model Checking Algorithms

- Model checking algorithms generally exploit graph algorithms (DFS and BFS).
- While the properties are generally simple, but the graphs are huge (billions of states), so a lot of effort is put into running algorithms efficiently over compressed data structures.
- Some examples are SMV, SPIN, UPPAAL for graph specifications as well as ZING and BOGOR for source code.

Typical properties - Safety

- The dual of reachability: $A G \neg \text{fail}$
- In the lift example we would want to show that
 $A G (\text{moving} \Rightarrow \text{closed})$
- This is trivial in terms of Graph algorithms, but model checkers often need to create the graphs on the fly.

Typical properties:Liveness

- Liveness ensures that a process never stalls (for example, in the case of deadlock).
- $A G (E X \text{ true})$ - It is always the case that the process can proceed to a next state.

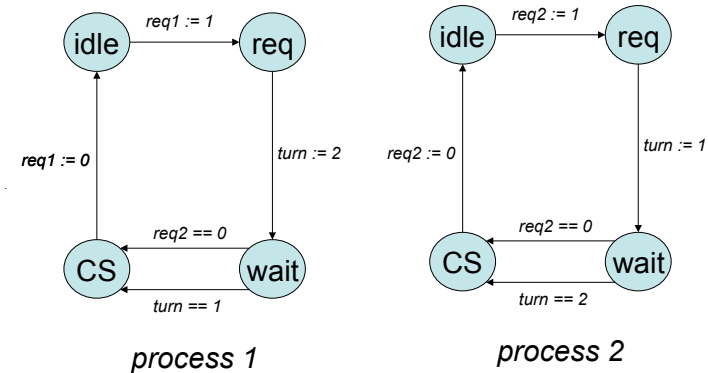
Typical properties - Fairness

- Fairness is the property that all requests should be eventually granted.
- For a model we require that visiting a state infinitely often results in all children of that state being visited.
- For the Lift Door, consider the formula
 $A(GF \text{ Closing} \Rightarrow F \text{ Closed})$

Example: Mutual exclusion protocols

- Mutual exclusion protocols are designed to ensure that two processes can access a common resource (the critical section, CS), but only ever one at a time.
- The protocols should be able to request access, and the request should always be eventually granted.

A Mutex protocol



Mutex properties

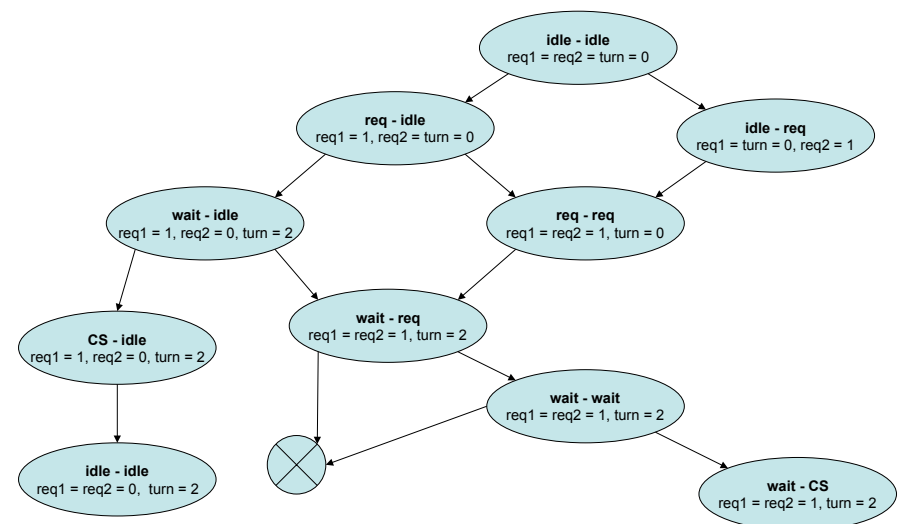
The state-space for mutex is derived by running the processes concurrently. We can then query:

- Is every request is granted:

$$A G (req(i) \Rightarrow A F CS(i))$$
 (for $i = 1, 2$)
- Is there at most one process in the critical section at any one time:

$$A G (CS(1) \Rightarrow \neg CS(2))$$

Checking Mutex



The state space explosion

- The Mutex protocol was specified using a small number of states, as well as the variables *req1*, *req2* and *turn*.
- How many states had to be explored?
- In general, small increases in the specification complexity greatly increase the state space.
- Carefully choosing which assertions to check can make model-checking much more manageable.