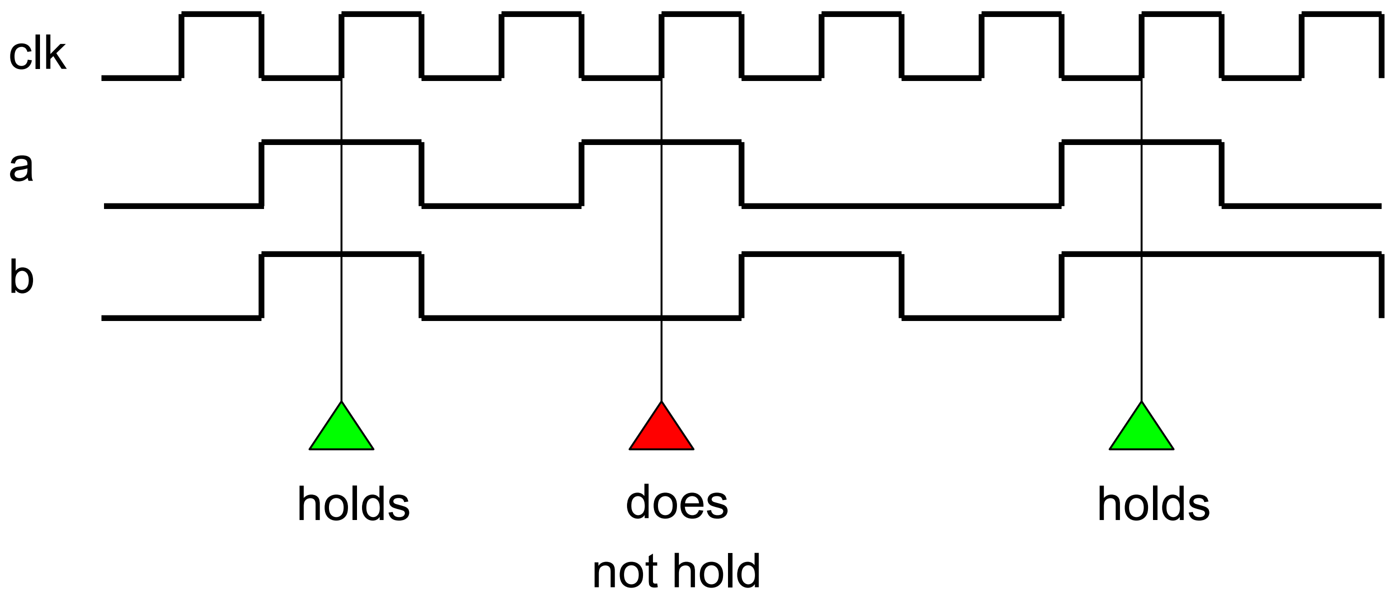
Holds and Implication

- `a` “holds” in the cycle where its value is non-zero and not unknown

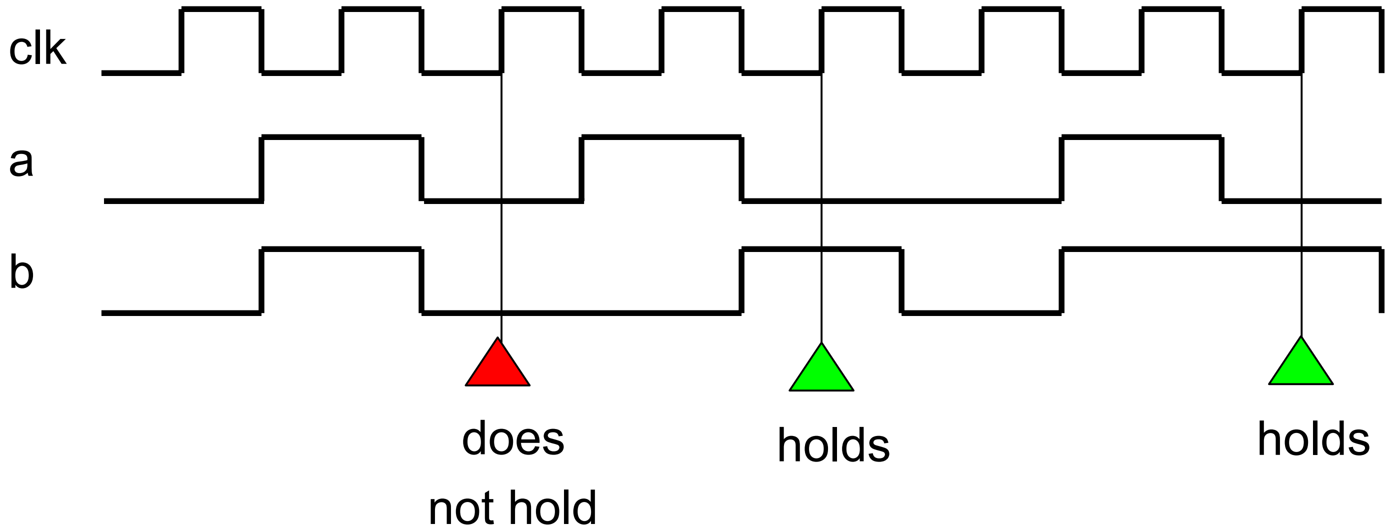
```
assert property ( @(posedge clk) a |-> b );
```



- This property **implies** `b` must hold whenever `a` holds

- This property implies that `b` must hold in the cycle **after** `a` holds

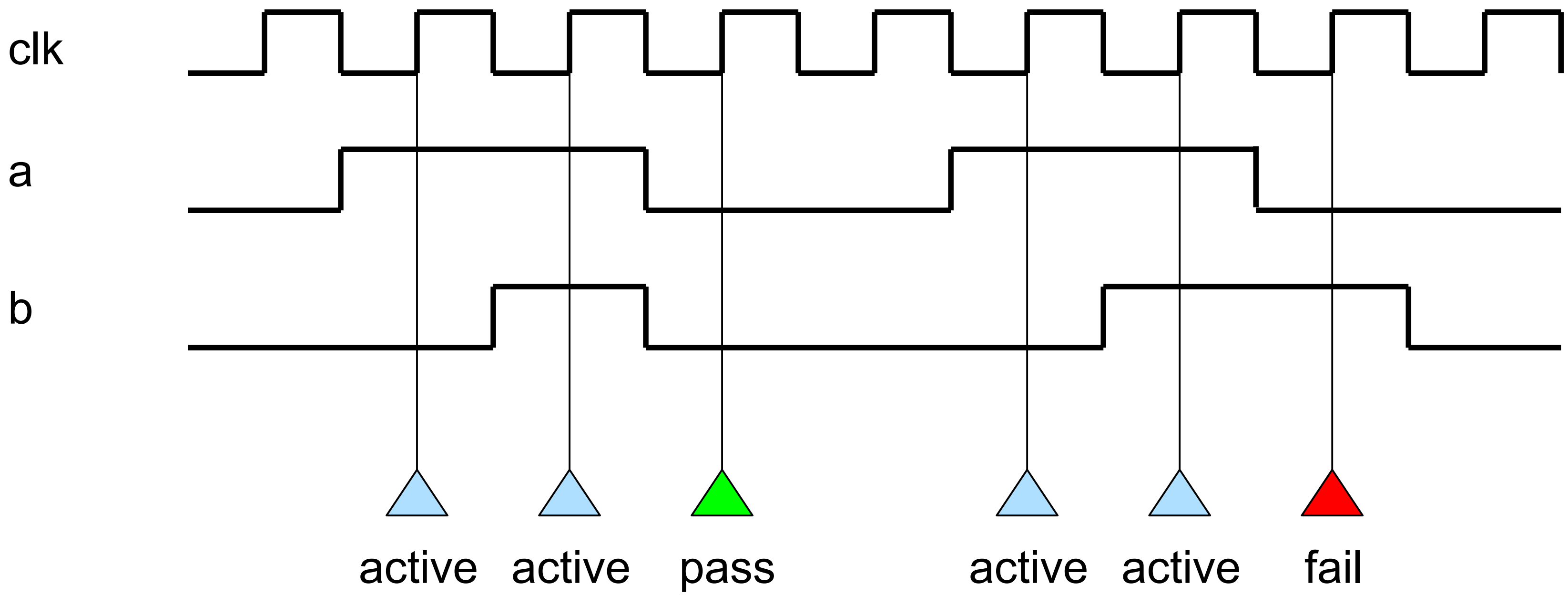
```
assert property ( @(posedge clk) a |=> b );
```



Simulation of Assertions

- A temporal property can be in one of 4 states during simulation:
  - *Inactive* (no match), *active* (so-far-so-good), *pass* or *fail*

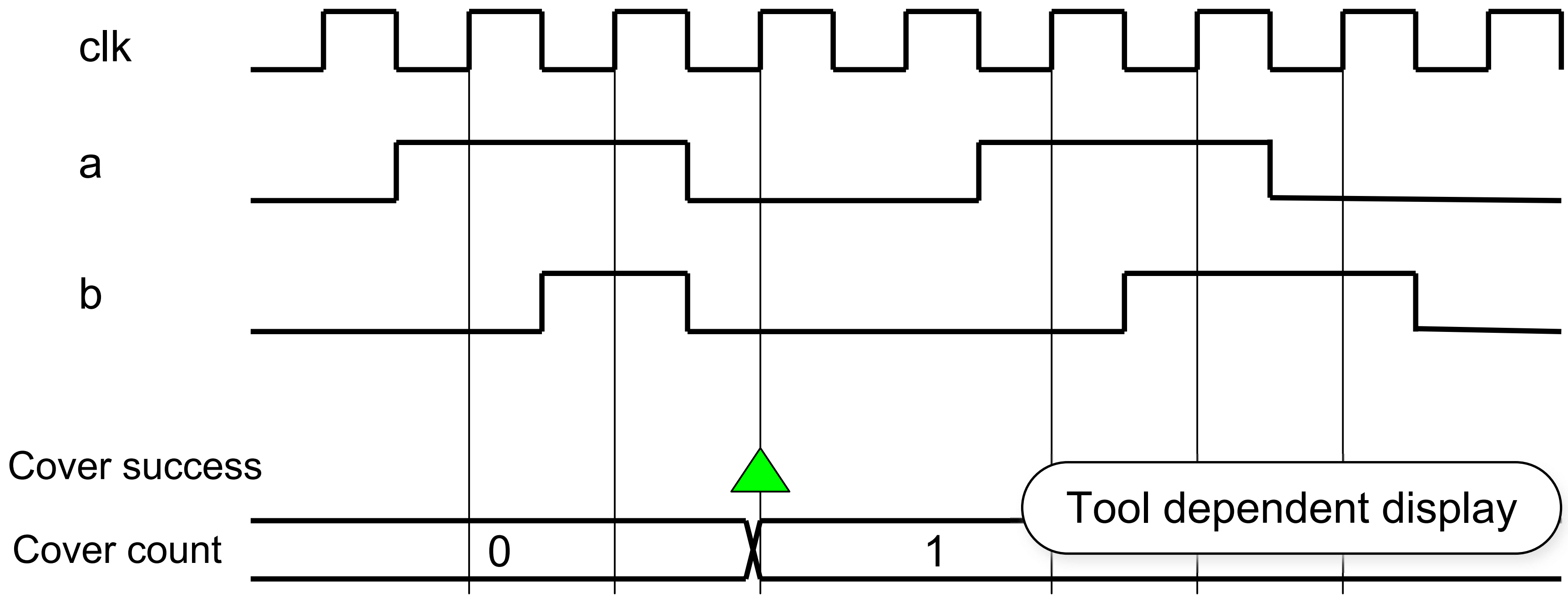
```
assert property ( @(posedge clk)
   a |=> ((a && b) ##[*0:$] !b) );
```



Simulation and Cover Property

- Simulators count the number of time the property holds
- Display information in waveforms and in a report

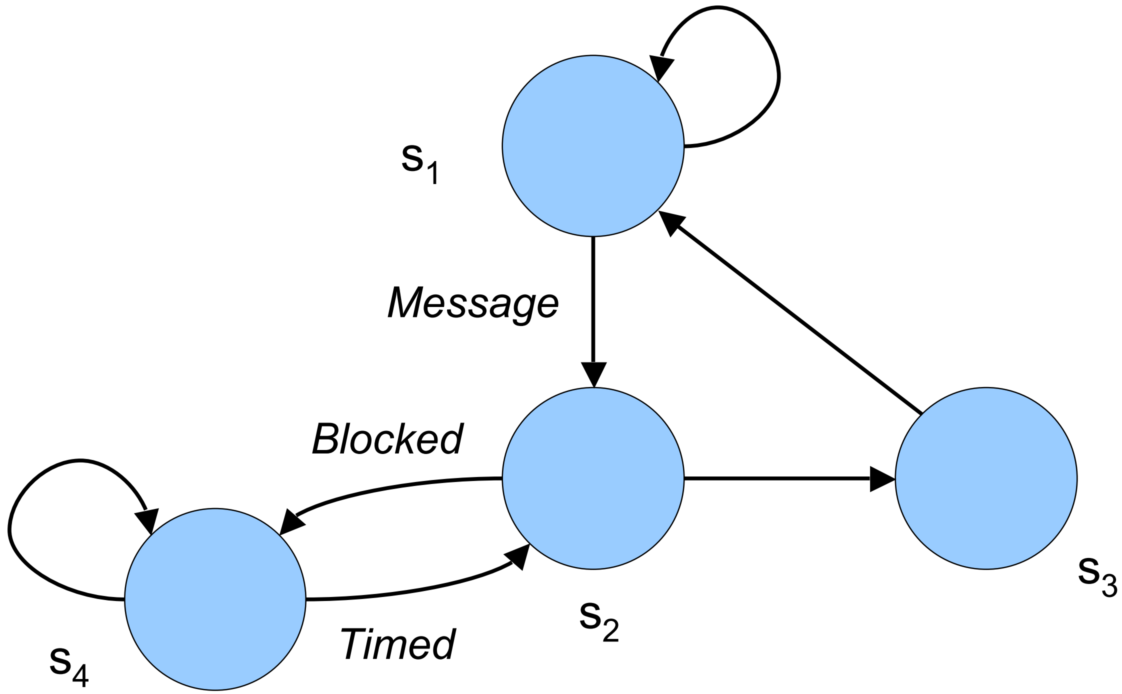
```
cover property ( @(posedge clk)
  a |=> ((a && b) ##[*0:$] !b) );
```



State Machines

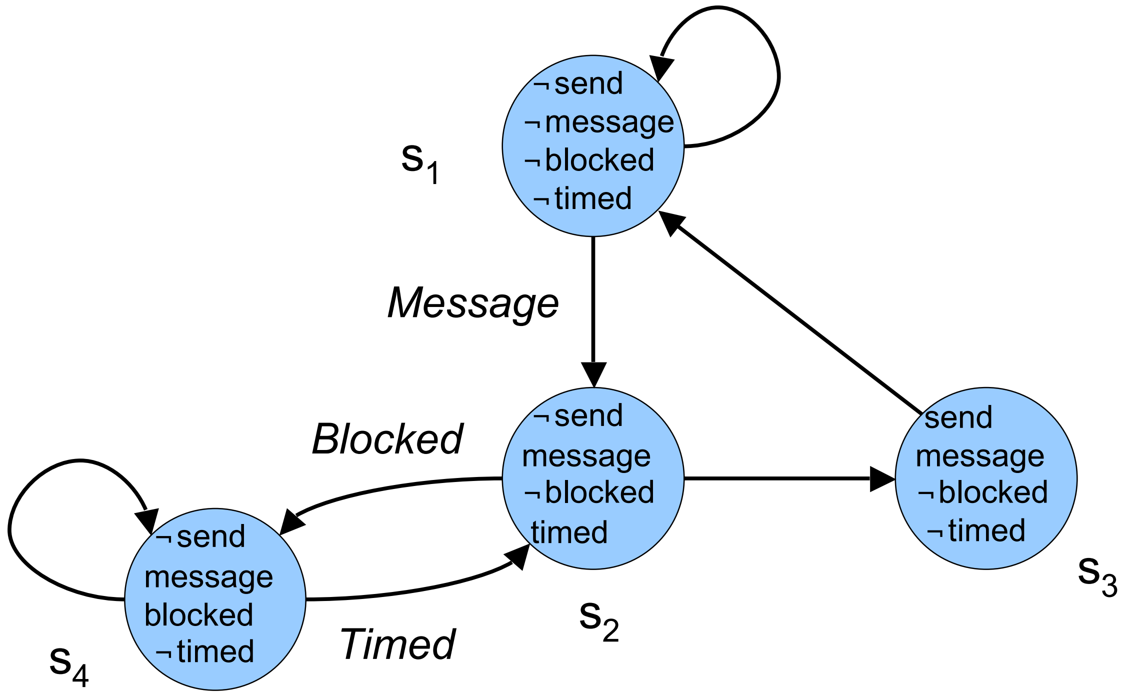
- In designing sequential systems, we (hardware designers) think in terms of *states* and *state machines*.
- We talk about the *present state* (*s*) of a system and the *next state* (*s*') of a system
- Temporal logic, model checking and so on are based on the idea that a system can be described in terms of discrete states with defined transitions between states.
- Thus, we can only verify state machines. This seems like a restriction because we often design systems in terms of datapaths and controllers (state machines). We could choose to think of the entire system as a state machine, or we could think of the datapath as (multiple) state machines.

Finite State Machine



- $S = \{s_1, s_2, s_3, s_4\}$
  - If v_1 is the state variable indicating that the FSM is in state s_1, then
    - $v_1' = v_3 \vee (v_1 \wedge \neg \mathit{Message})$
    - and so on

Finite State Machine



- Four atomic propositions:
  - send, message, blocked, timed

Properties

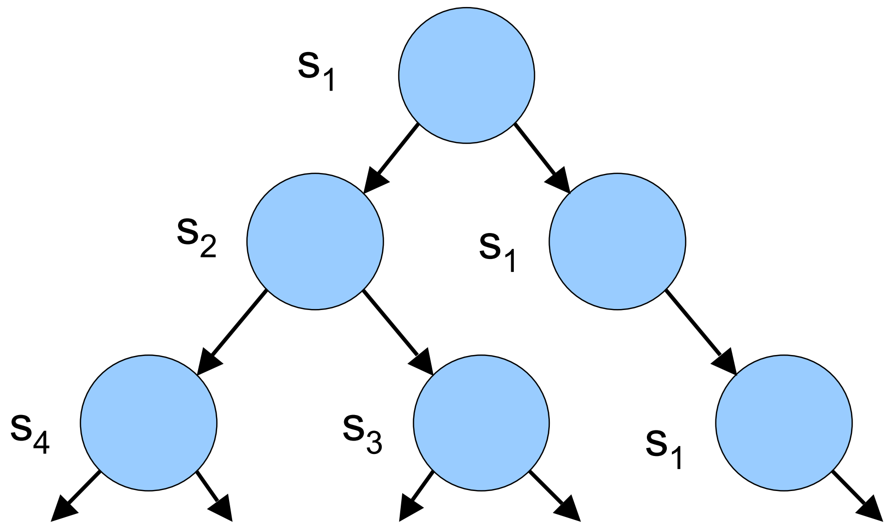
- Recall that a property is some fact about a system
- For example, in the communications interface, a property might be:

  "If we have a message, it will always be sent"

- We want to prove that this property is true. To do that we need some way of expressing that property in a mathematical way.
- There are two important classes of properties:
  - Safety – "bad things never happen"
  - Liveness – "good things happen eventually"

Computation Tree

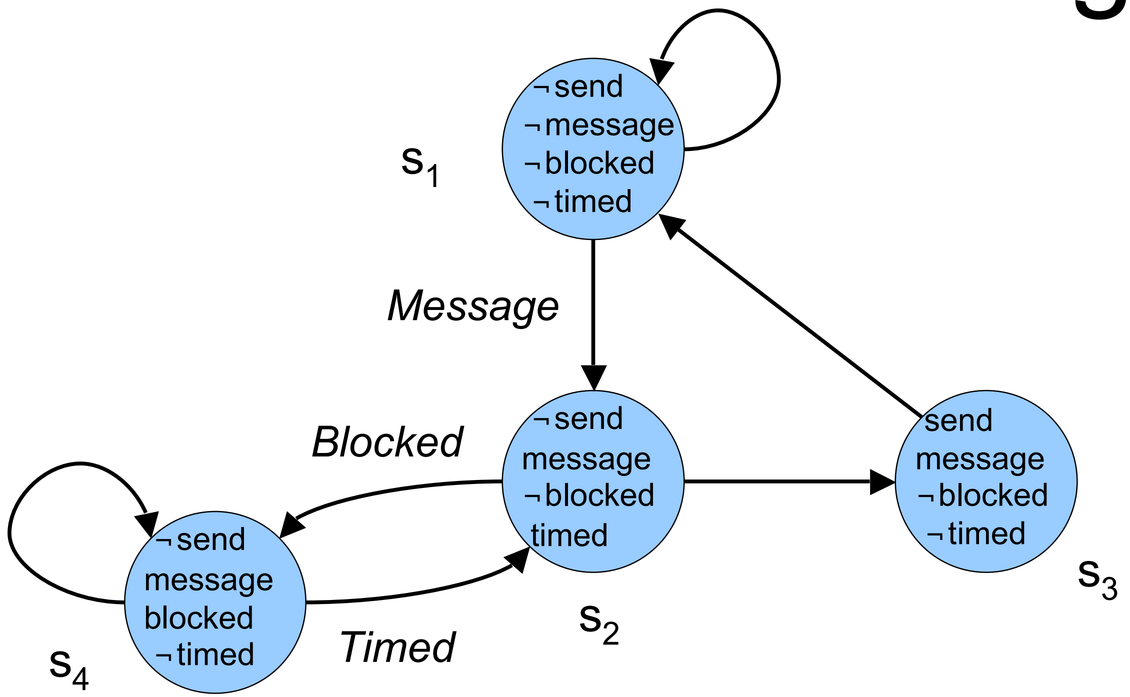
- *Computation Tree Logic* (CTL*)
  - CTL* formulas describe properties of *computation trees*
  - A computation tree is formed by choosing an initial state and unwinding the structure as an infinite tree:



Temporal Logic

- CTL* formulas are expressed in terms of *path quantifiers*:
  - **A** – "for all computation paths"
  - **E** – "there exists a computation path"
- and *temporal operators*:
  - **X** – "next time (state)"
  - **F** – "eventually" or "in the future"
  - **G** – "always" or "globally"
  - **U** – "until"
  - **R** – "release"
- CTL is a subset of CTL*, in which each each temporal operator is immediately preceded by a path quantifier, (for example, **AX**, "for all paths, at the next time")
  - Example CTL formula:
  - **EF** (send $\wedge\neg$message) – "there exists a path, where, eventually, send holds (is true) but message does not hold"

Model Checking



- Let us assume we wish to check some property of this system, for example:
  - "If we have a message, it will always be sent"
- In CTL, this is expressed as:
  - **AG**(message → **AF** send)
  - "for all paths, globally, the proposition message implies that for all paths, eventually, the proposition send holds"

Model Checking

- It is helpful to rewrite this formula as:
  - $\neg$**EF**(message $\wedge$ **EG** $\neg$send)

  "there does not exist a path such that eventually message holds and there exists a path globally where send does not hold"

- The states where message is true are:
  - S(message) = $\{s_2, s_4\}$
- We now need to find S(**EG** ¬send), in other words, does there exist a path over which, globally, send does not hold? We can see that:
  - S(**EG** ¬send) = $\{s_1, s_2, s_4\}$
- Thus:
  - S(message ∧ **EG** ¬send) = $\{s_2, s_4\}$
- Now, from which states does there exist a path that eventually (**EF**) leads to these states. Again, we can see that all states can lead to these states:
  - **EF**(message ∧ **EG** ¬send) = $\{s_1, s_2, s_3, s_4\}$
  - or, ¬**EF**(message ∧ **EG** ¬send) = $\varnothing$

Fairness

- What does this mean?
  - The proposition fails. We attempted to check whether the arrival of a message would *always* result in the message being sent, and using temporal logic proved that this is not true.
  - Intuitively, the system can stay *forever* in states s_2 and s_4. Each time we try to send the message, it is blocked by some other activity in the system.
- Is this likely?
  - Intuitively, no. "Forever" is a long time – we would expect that eventually the message would not be blocked.
- In temporal logic, we have the concept of *fairness*.
  - We add fairness constraints, F, to the FSM model:

$$M = (S, R, L, F)$$

Model Checking with Fairness

- In the message sending example, a fairness constraint might be:
  - message $\wedge$ send $\wedge \neg$blocked
- Any infinite paths in which that fairness constraint does not hold are therefore excluded.
  - The infinite sequence s_2, s_4, s_2, s_4, … is excluded because the constraint does not hold in any state in that path
- Thus:
  - S(**EG** ¬send) = ∅
  - S(message ∧ **EG** ¬send) = ∅
- So:
  - **EF**(message ∧ **EG** ¬send) = ∅
  - ¬**EF**(message ∧ **EG** ¬send) = $\{s_1, s_2, s_3, s_4\}$
  - **AG**(message → **AF** send) = $\{s_1, s_2, s_3, s_4\}$
- By excluding unfair paths, the system is proved to work correctly.

Liveness and Fairness

- The property

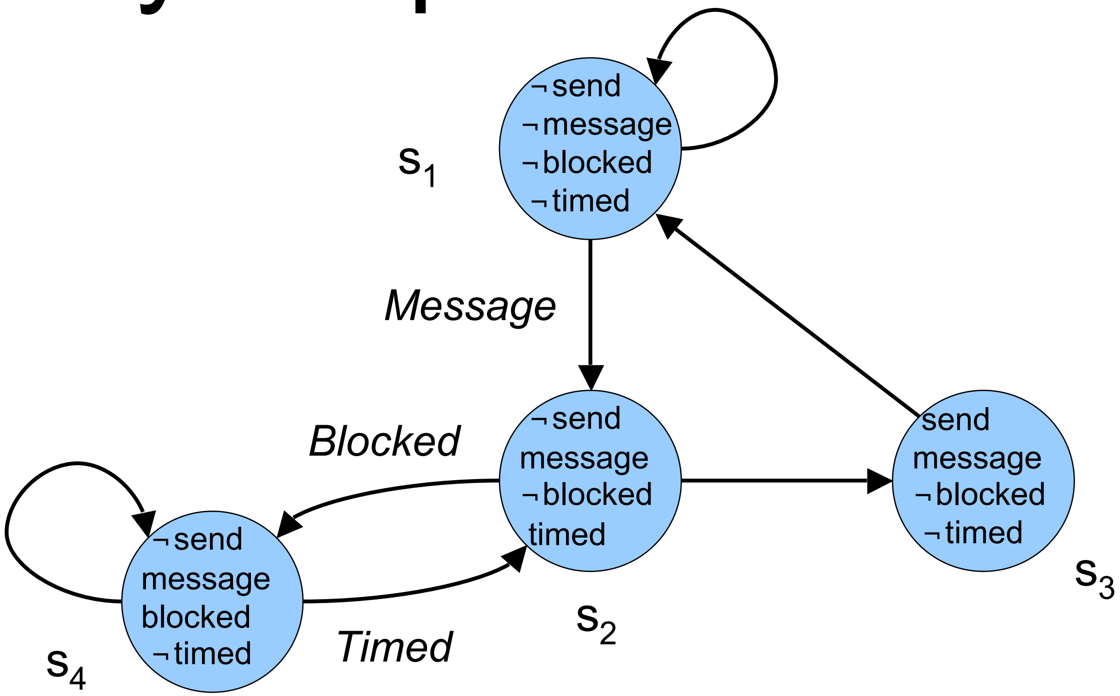
  "If we have a message, it will always be sent"

  is a *Liveness* property ("good things happen eventually")
- This is expressed in CTL, as we have seen, as
  - **AG**(message → **AF** send)
- In CTL, all operators must have a path quantifier (**E** or **A**)
- In CTL*, any combinations are allowed; the property can be written as:
  - **AG**(message → **F** send)
- CTL* is the more general, but it is harder to write model checkers.
- CTL <u>cannot</u> express *Fairness*; the Liveness property in CTL* is:
  - **A**((**GF** message) → send)

Linear Time Logic (LTL)

- CTL can't express fairness
- CTL* is too complicated
- Linear Time Logic – no path quantifiers (**A** or **E**)
- In LTL, the liveness property, including fairness, is:
  - (**GF** message) → send)
- LTL runs in linear time (CTL runs in branching time)
- LTL assumes that the trace (the sequence of states) is a subset of the traces defined by the LTL property
- If the trace is not in the set
  - It violates the property
  - It is a counter-example
- BUT, LTL cannot express branching!
- So, use both CTL and LTL model checkers.

Safety Properties and Invariants



- A safety property of the communications interface is that
  - "send and blocked" do not occur at the same time
- In CTL: P= $\mathbf{AG}(\neg\text{send} \vee \neg\text{blocked})$
- We can see that this property holds in all states, provided that the initial state is one of the four shown
- This is known as an *Invariant*