Temporal Logic

Notes:Jo Anne Atlee, Dan Berry
and Richard Trefler

Fall 2012

Prescriptive vs. Descriptive Specifications

So far, the specification notations used in this class have been model-based and are said to be prescriptive. Prescriptive specifications describe how a system behaves from one input to the next. System behaviour is decomposed into states, and the specification describes for each state what input the system is ready to react to in that state and what the system's response to that input event will be.

If you want to know about longer-term system behaviour, longer than the response to a single input event, you have to examine paths through the specification, and you may have to examine several paths.

What if you wanted to specify that dialing a valid number always results in either the call being connected or a busy-tone?

What if you wanted to specify in a specification of an elevator that the elevator never moves with its doors open?

What if you wanted to specify in a traffic-light specification that if a car approaches the intersection, the light in its direction will eventually be green? If you used only a model-based notation like SDL or UML, you'd have to try to write state diagrams that covered each case in which a car approaches the intersection.

Another approach is to use a notation that is designed for expressing system-wide properties, such as propositional, predicate, or temporal logic.

So. we will be quantifying, not just over variables, but also over states of a computation, which in turn is the sequence of states that happens during run time.

Review Predicate Logic

Predicate logic is for expressing properties about fixed-valued variables

With fixed-valued variables, a logic formula is evaluated with respect to a particular assignment of values to variables. This will be in contrast to temporal and time-dependent logics, in which a formula may be evaluated over variables that change value over time.

1. Set of typed variables
   Booleans, Integers, Sets.
   If our system is object-oriented, then we may have variables for object instantiations, attributes, etc.

2. Functions on typed variables
   $+, -, *, /$ for integer variables
   $\cup, \cap$ for set variables
   $\wedge, \vee, \neg$ for boolean variables

3. Predicates
   $<, >$ for integers
   $\subset, \in$ for sets

4. Equivalence
   $=$ for comparing two values of
   the same type

5. Propositional logic connectives

   $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$

   $\text{Cond1} \rightarrow \text{Cond2} \equiv \neg\ \text{Cond1} \vee \text{Cond2}$

   IF Cond1 THEN Cond2 ELSE Cond3 $\equiv$

   $(\ \text{Cond1} \rightarrow \text{Cond2}\ ) \wedge$

   $(\ \neg\ \text{Cond1} \rightarrow \text{Cond3}\ )$

6. Quantifiers $\forall x \in T : f(x)$, $\exists x \in T : f(x)$

   $\forall x \in T : f(x)$

   for all $t \in T$: the interpretation of f with t substituted for x evaluates to $true$.

   $\exists x \in T : f(x)$

   there exists $t \in T$: the interpretation of f with t substituted for x evaluates to $true$.

Time-Dependent Logic

Time-dependent logic is for expressing time-dependent properties

Can think of variables as time-functions whose value depends on time.

Variables as time-functions

$$
\begin{aligned}
&\mathit{coin} : \mathit{time} \rightarrow \mathit{boolean} \\
&\mathit{locked} : \mathit{time} \rightarrow \mathit{boolean} \\
&\mathit{push} : \mathit{time} \rightarrow \mathit{boolean} \\
&\mathit{enter} : \mathit{time} \rightarrow \mathit{boolean} \\
&\mathit{rotating} : \mathit{time} \rightarrow \mathit{boolean} \\
&\#\mathit{entries} : \mathit{time} \rightarrow \mathit{integer} \\
&\#\mathit{coins} : \mathit{time} \rightarrow \mathit{integer}
\end{aligned}
$$

Given a variable *x*, you can ask what its value is only at some time *t*. When writing formulae, you must specify for every variable named in the formula the time that the variable's value is referenced.

$\#\mathit{coins}(0) = 0$
$\mathit{coin}(1) \rightarrow \neg \mathit{locked}(2)$

It's hard to write specifications in terms of what the variable values will be at a particular point in time. More often, one is interested in expressing the relationships between variable values. For example, anytime a coin is inserted, the barrier will be unlocked at a later time.

$$\forall\, t \in \mathit{Time} : (\mathit{coin}(t) \rightarrow \neg \mathit{locked}(t+1))$$

$$\forall\, t1 \in \mathit{Time} : (\mathit{coin}(t1) \rightarrow \exists\, t2 \in \mathit{Time} : (t2 > t1 \wedge \neg \mathit{locked}(t2)))$$

Examples

- It's always the case that the number of entries into the park is less than or equal to the number of coins received.
  $\forall\, t \in \mathit{Time} : (\#\mathit{entries}(t) \leq \#\mathit{coins}(t))$
- If a visitor pushes the turnstile and the turnstile is unlocked, then eventually the visitor (someone) will enter the park.
  $\forall\, t \in \mathit{Time} : ((\mathit{push}(t) \wedge \neg \mathit{locked}(t)) \rightarrow$
  $\quad \exists\, t1 \in \mathit{Time} : (t1 > t \wedge \mathit{enter}(t1)))$
- If a visitor pushes the turnstile when the turnstile is unlocked then the turnstile rotates until the visitor enters the park.
  $\forall\, t \in \mathit{Time} : ((\mathit{push}(t) \wedge \neg \mathit{locked}(t)) \rightarrow$
  $\quad \exists\, t1 \in \mathit{Time} : (t1 > t \wedge \mathit{enter}(t1) \wedge$
  $\quad\quad \forall\, t2 \in \mathit{Time} : (t < t2 < t1 \rightarrow \mathit{rotating}(t2))))$

Notice that we often don’t care about the values of variables at specific points in time. With the possible exception of time t=0, when we might care about the initial values of the variables. Mostly, we care about the temporal ordering of events and variable values. We want to express constraints on variable values in terms of when they change value.

- If a coin is inserted, the barrier will become unlocked.
- If a caller picks up the telephone handset, he will hear a dialtone.
- If I push the elevator button, the elevator will eventually arrive at my floor and open its doors.

Sometimes we care about the timing of those events

- If a train comes within 200 meters of a railroad crossing, the gate will be lowered within 10 seconds.

But for the most part, we're concerned only with the order in which events occur.

Linear Temporal Logic

Linear Temporal Logic was designed for expressing the temporal ordering of events and variable values

In temporal logic, time progresses, but the notion of exact time is abstracted away. Instead, we keep track of changes to variable values, and the order in which they occur.

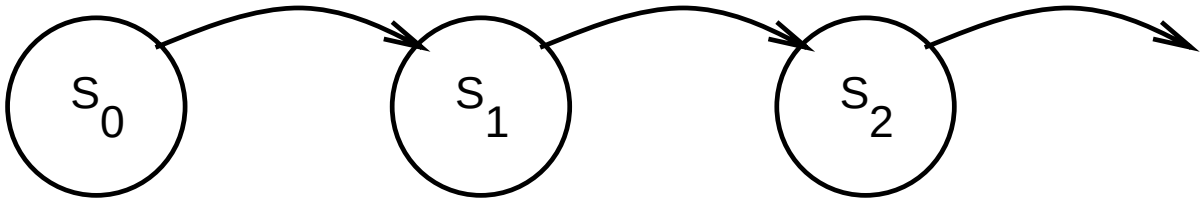
System State

The system state is an assignment of values to the model's variables.

Intuitively, the system state is a snapshot of the system's execution. In this snapshot, every variable has some value.

If we're working with an OO or UML system, then looking at a snapshot of the system, there is an explicit number of instantiated objects that are executing, each object is in exactly one state of its state diagram, and each of its attributes has some value.

This is one system state. If the system then executes an assignment statement, the value of one of its variables changes. The system enters a new system state.



There is some initial state of the system, defined by the initial values of all the variables.

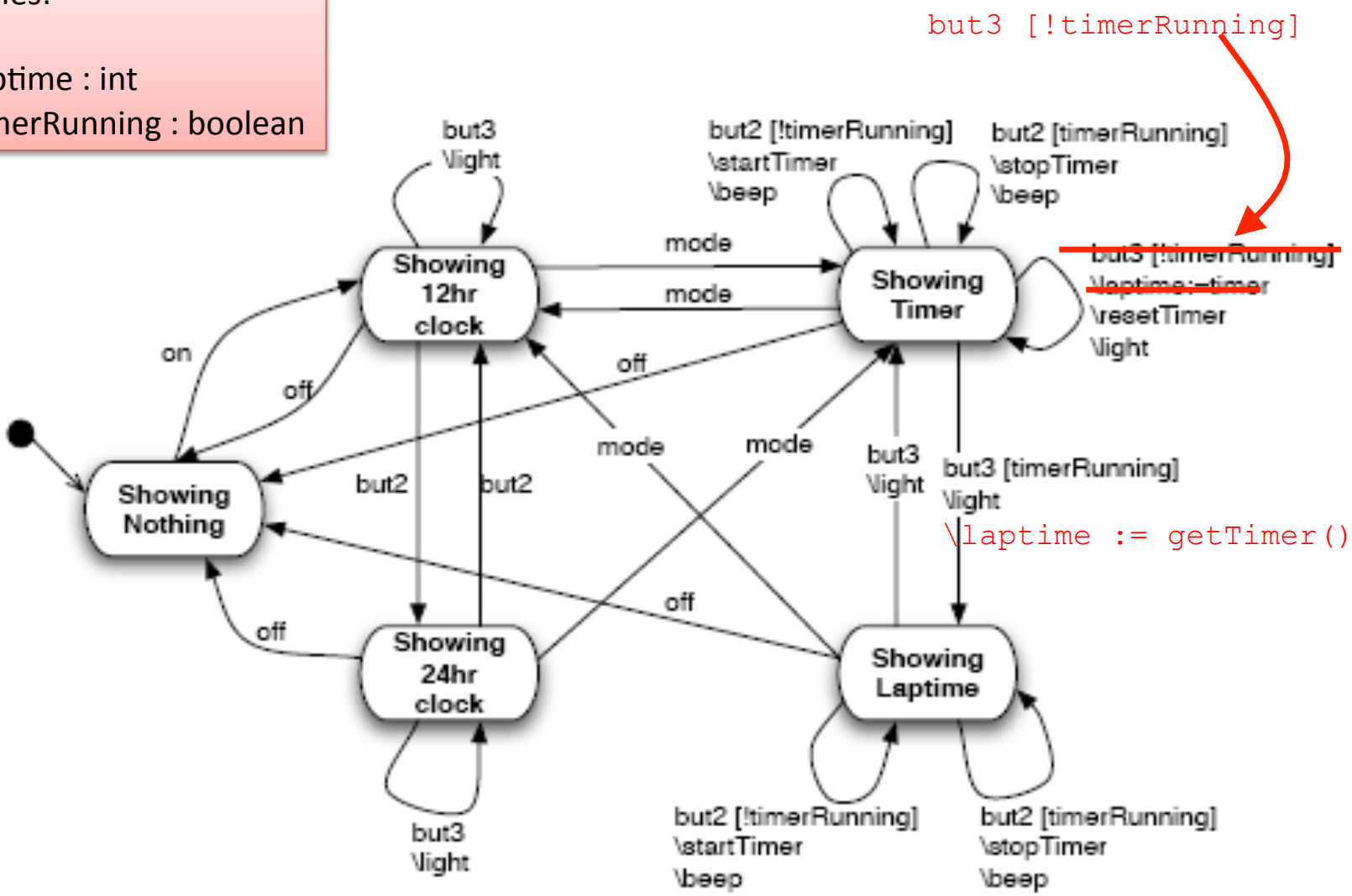
As the system executes, the values of the variables change. Each state represents a change from the previous state in the value of some variable. More than one variable can change value between two consecutive states.

This "state" is different from "state" in a FSM or SM.

What IS the system state of a SM?

Variables:

var laptime : int
var timerRunning : boolean



state of this State Machine = an assignment of values to this 4-tuple:

(State, laptime, timerRunning, curEvent)

where State = {Sh12, Sh24, ShNT, ShTi, ShLT}, laptime = {0:00, ...}, timerRunning = {T,F}

curEvent = {on, off, but2, but3, mode}

Executions

A sequence of system states represents a particular execution of the system, and obviously, time progresses during the execution, but there is no keeping track of how long the system is in any particular state.

An execution or a computation is a sequence of system states

$\sigma = s_0, s_1, s_2, \ldots.$

Temporal Connectives

Connectives are shorthand notations that quantify over future system states — future worlds.

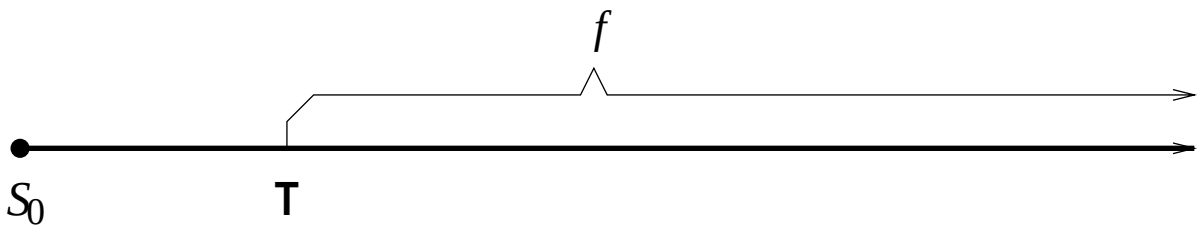
- henceforth: $\Box$
- eventually: $\Diamond$
- next state: $\bigcirc$
- until: $\mathcal{U}$
- unless: $\mathcal{W}$

Henceforth

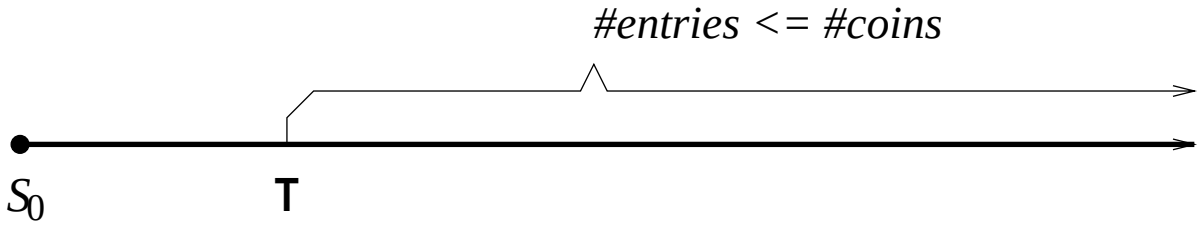
$\Box f =$

$$\begin{cases} T & \text{if } f \text{ is true in the current and} \\ & \underline{\text{all}} \text{ future system states} \\ F & \text{otherwise} \end{cases}$$

$(\sigma, j) \models \Box f \quad$ iff
for all $i \geq j : (\sigma, i) \models f$



Ex: $\Box(\#entries \leq \#coins)$



Shorthand for:

$\forall\, t \in Time.(\#entries(t) \leq \#coins(t))$, which is a shorthand for:

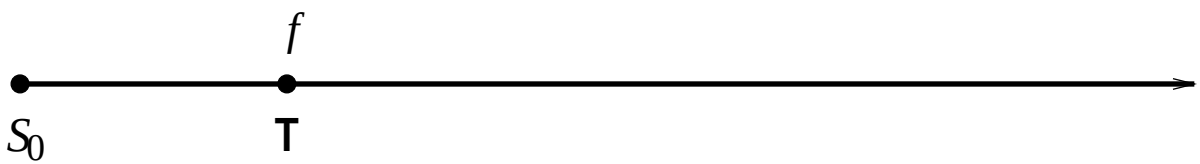
$\forall\, t \in Time.(t \geq t_0 \rightarrow (\#entries(t) \leq \#coins(t)))$

Eventually

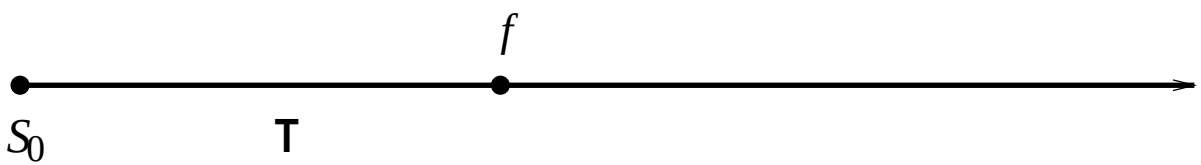
$\Diamond f =$

$$\begin{cases} T & \text{if } f \text{ is true in the current or} \\ & \underline{\text{some}} \text{ future system state} \\ F & \text{otherwise} \end{cases}$$

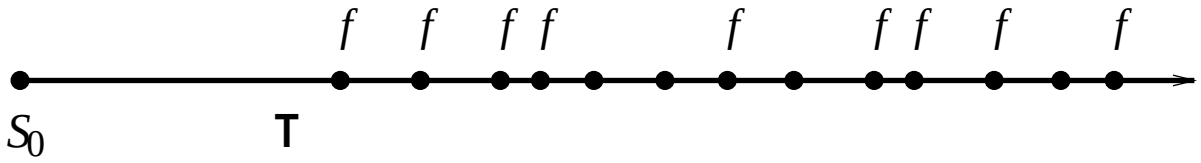
$(\sigma, j) \models \Diamond f \quad$ iff
there exists $i \geq j$ such that $(\sigma, i) \models f$
$\quad \exists i.(j \leq i \wedge (\sigma, i) \models f))$



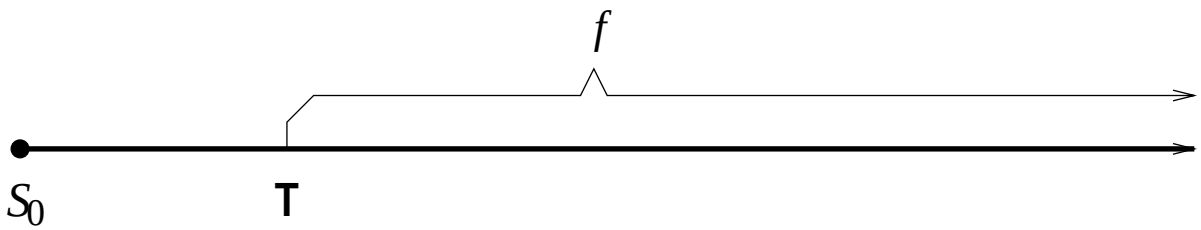
OR



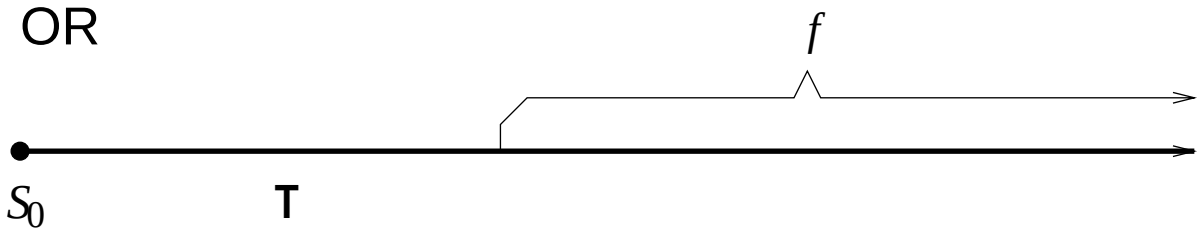
Thus, $\Box(\Diamond f)$ means that f happens infinitely often.



$\Diamond(\Box f)$ means that eventually, f is true forever.

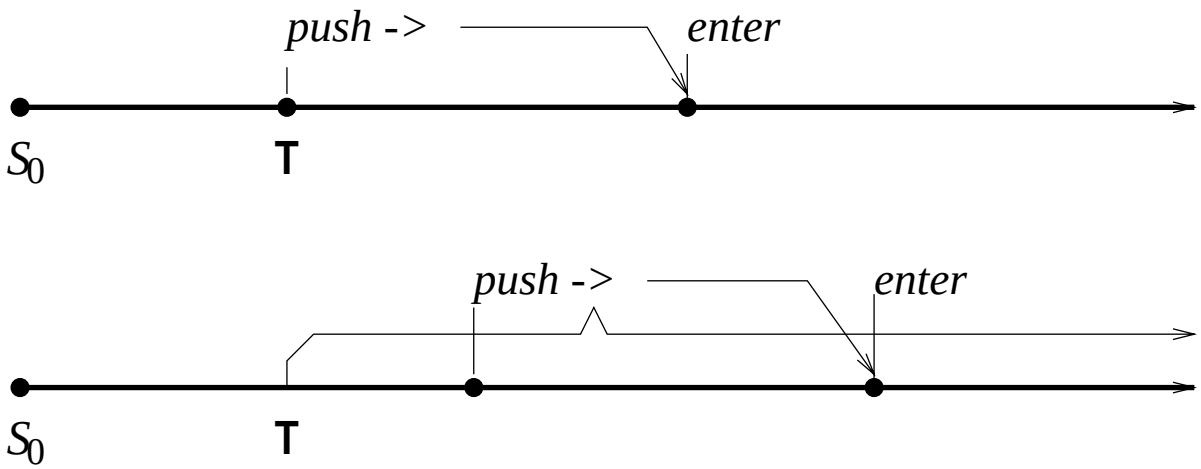


OR



Ex: $push \rightarrow \Diamond enter$

Ex: $\Box(push \rightarrow \Diamond enter)$

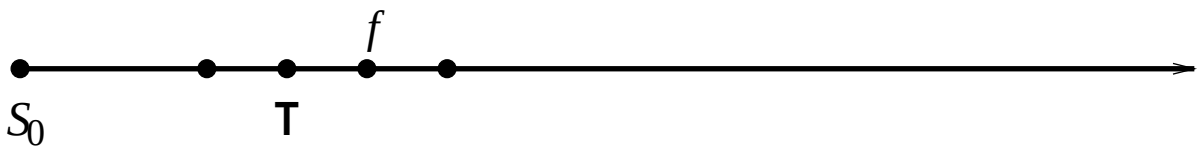


Next State

$\bigcirc f =$

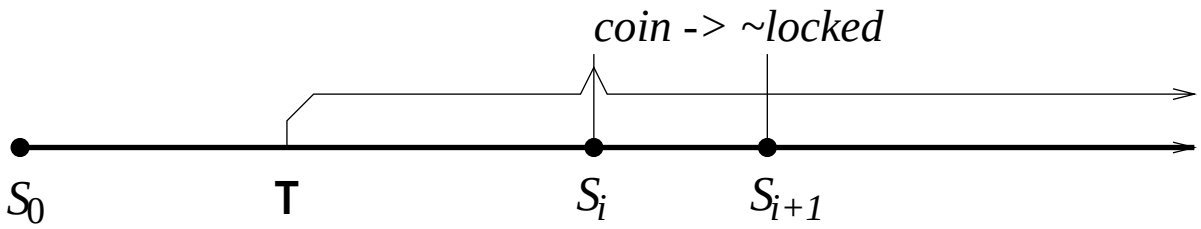
$$\begin{cases} T & \text{if } f \text{ is true in the next} \\ & \text{system state} \\ F & \text{otherwise} \end{cases}$$

$(\sigma, j) \models \bigcirc f \quad \text{iff}$
$\quad (\sigma, j+1) \models f))$



Ex: $\Box(coin \rightarrow \bigcirc\neg Locked)$

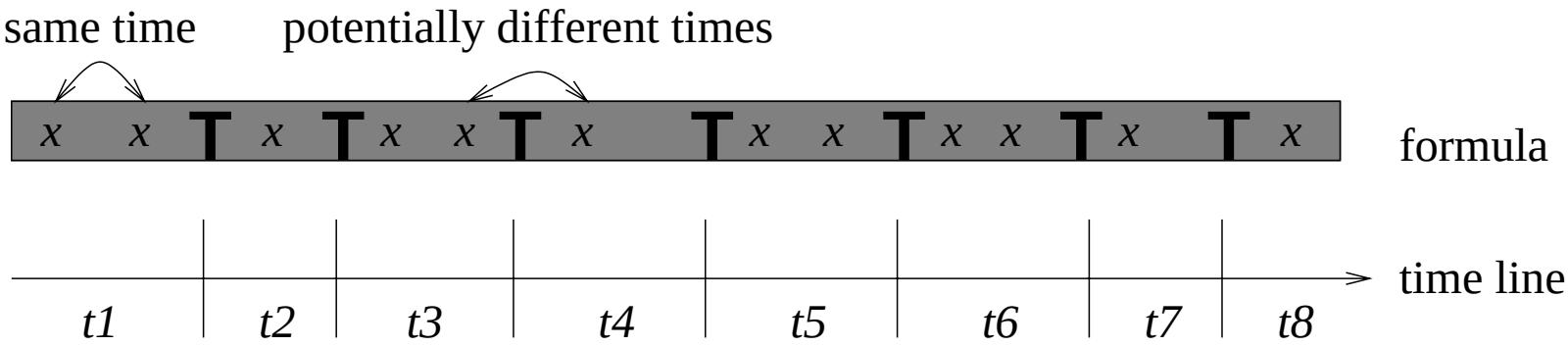
Ex: $(\Box(coin \rightarrow \bigcirc\neg Locked)) \rightarrow$
$(\Box(coin \rightarrow \Diamond\neg Locked))$



Consider a temporal logic formula that has no Boolean operator at the outermost syntactic level.

As you scan the formula from left to right, each temporal operator introduces a new implied bound time variable which is at the same time or later than the previous.

In the following, the shaded rectangle is a temporal logic formula, each T is some temporal operator, each *x* is implicitly a function on time.



$t1 \leq t2$, $t2 \leq t3$, etc.

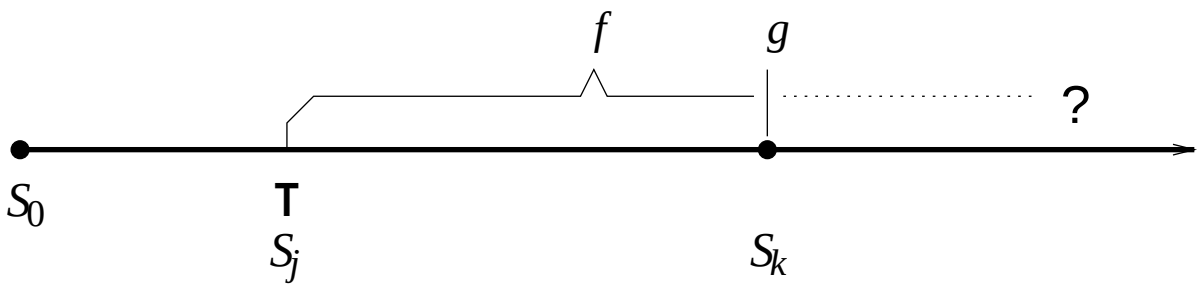
Thus, in between two consecutive temporal operators, all variables are evaluated at the same time, but variables separated by at least one temporal operator are evaluated at potentially different times.

Until

$f \mathcal{U} g =$

$$\begin{cases} T & \text{if } g \text{ is eventually true, and} \\ & f \text{ is true until then} \\ F & \text{otherwise} \end{cases}$$

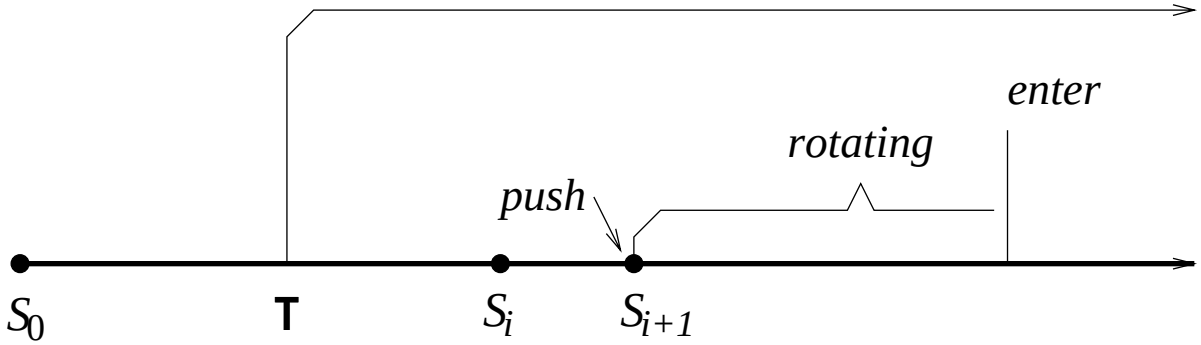
$(\sigma, j) \models f \mathcal{U} g$ iff
there exists $k \geq j$ such that $(\sigma, k) \models g$ and for all i such that $j \leq i < k$, $(\sigma, i) \models f$.



Note that $f \mathcal{U} g \rightarrow \Diamond g$

t1 t1+1 t2 ≥t1+1

Ex: $\Box(push \rightarrow \bigcirc(rotating \mathcal{U}\ enter))$



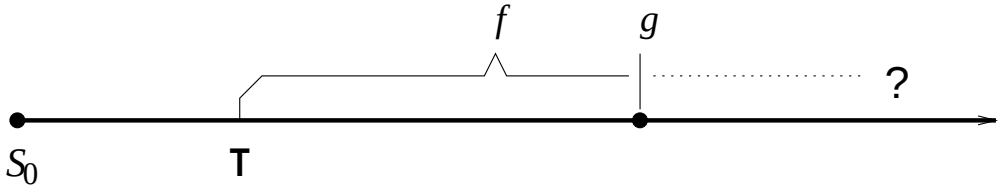
Unless

$$f \mathcal{W} g =$$

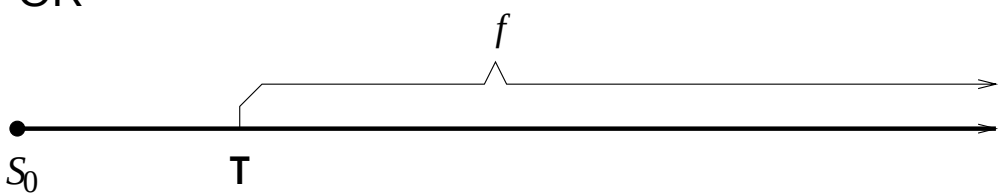
$$\begin{cases} T & \text{if } f \text{ holds indefinitely or} \\ & \text{until } g \text{ holds} \\ F & \text{otherwise} \end{cases}$$

$$f \mathcal{W} g \quad \text{iff} \quad (f \mathcal{U} g) \vee (\Box f)$$

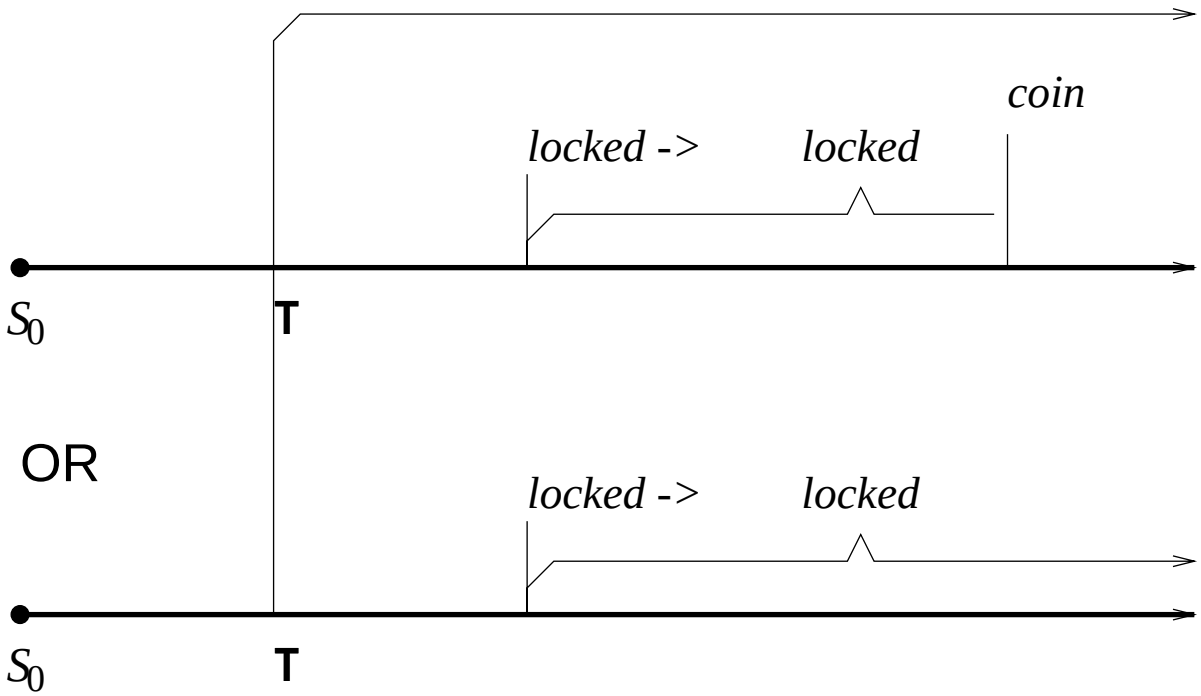
Unless is like Until, without the guarantee that g might happen.



OR

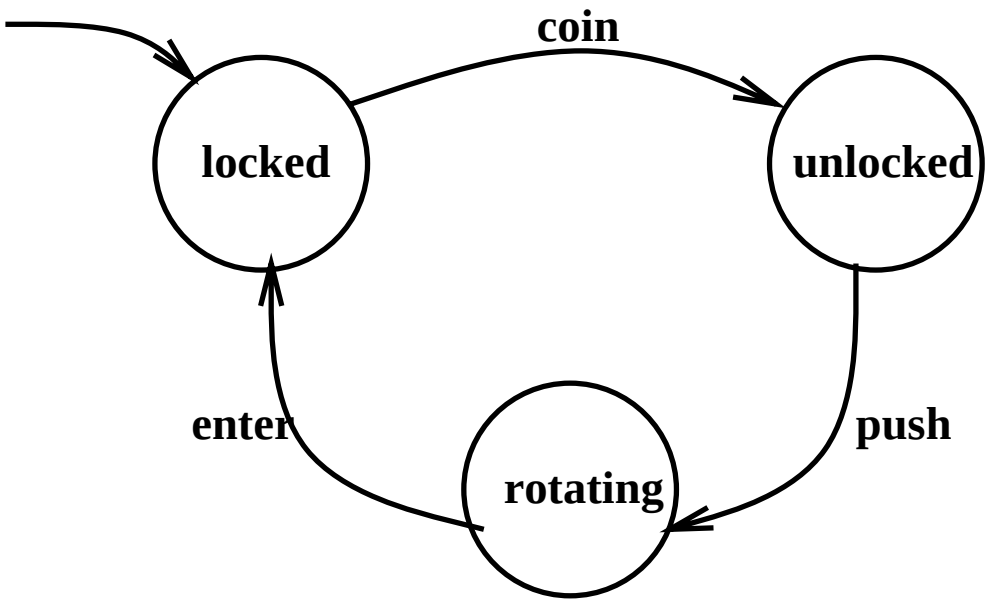


Ex: $\Box(locked \rightarrow (locked\, \mathcal{W}\, coin))$



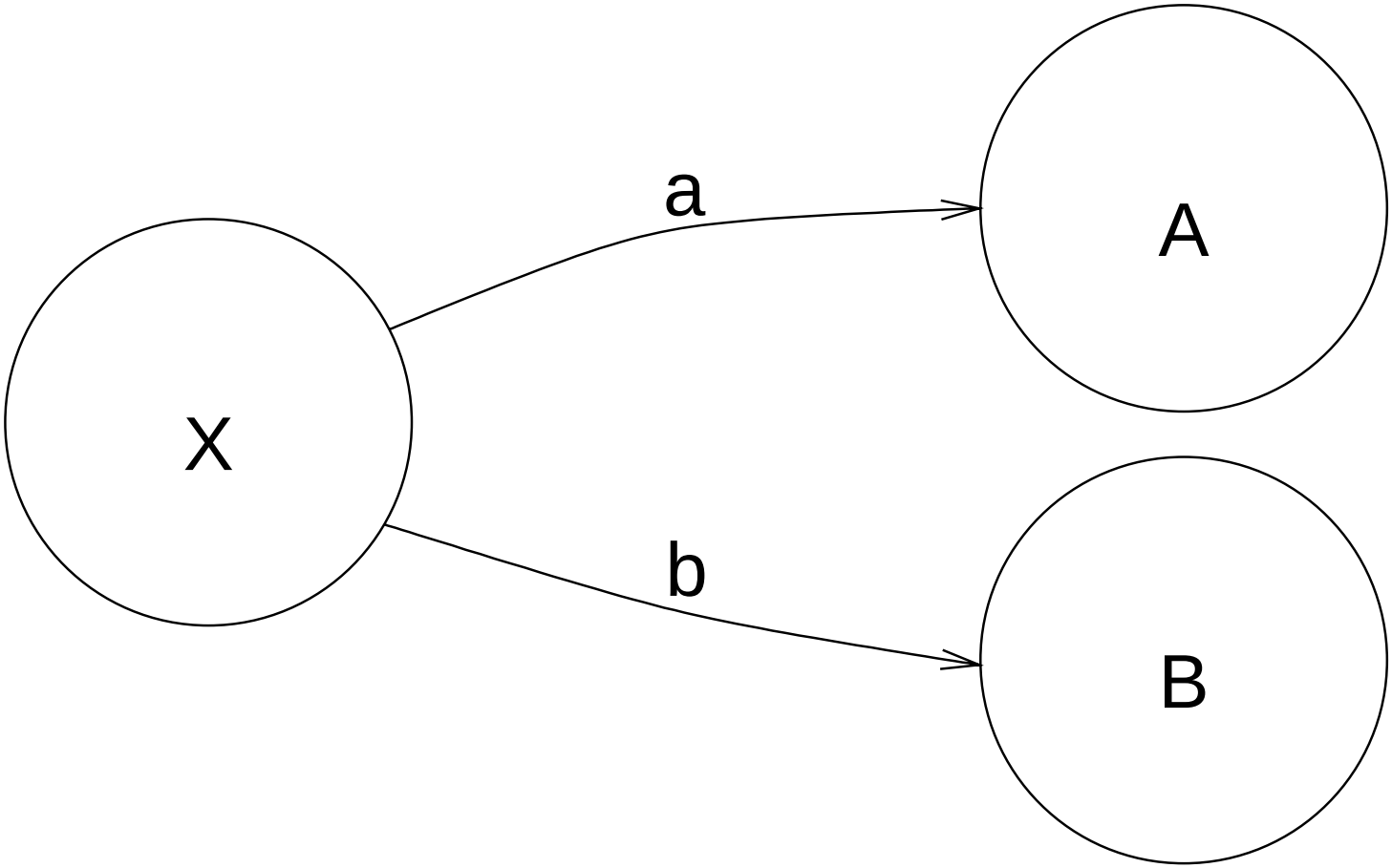
Describing Behaviour of Finite State Machines

You can use these connectives to describe the behaviour of a finite state machine.



$\Box(locked \rightarrow (locked\ \mathcal{W}\ coin))$
$\Box((locked \wedge coin) \rightarrow \bigcirc(unlocked))$
$\Box(unlocked \rightarrow (unlocked\ \mathcal{W}\ push))$
$\Box((unlocked \wedge push) \rightarrow \bigcirc(rotating))$
$\Box(rotating \rightarrow (rotating\ \mathcal{U}\ enter))$
$\Box((rotating \wedge enter) \rightarrow \bigcirc(locked))$

Note: $unlocked \not\equiv \neg locked$ and $\neg locked \equiv rotating \vee unlocked$



$\Box(X \rightarrow (X\ \mathcal{W}\ (a \vee b)))$
$\Box((X \wedge a) \rightarrow \bigcirc(A))$
$\Box((X \wedge b) \rightarrow \bigcirc(B))$