

Temporal Logic

Notes by mainly Jo Anne Atlee,
with modifications by Daniel Berry
dberry@uwaterloo.ca

Fall 2004

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 in the CU specification 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 logic and temporal logic.

Constraint Specifications

When are constraint specifications useful?

They are useful when it is desired to express

- additional constraints on a model-based specification and
- redundant, but non-obvious, system properties of specification.

These are very different uses.

In one case, the constraint specifies new behaviour not expressed in the model-based specification. In this case, the system behaviour is a conjunction of the model-based specification and the constraint.

In the second case, the constraint does not add new behaviour, but instead reiterates behaviour already specified by the model-based specification. Such specifications may be easier to read and recognize when expressed as a constraint, so the document adds them. Such specifications can be used also when reviewing the model-based specification. The reviewer can check whether or not the model satisfies the constraint.

If you use any type of constraint specifications, should indicate how you are using them.

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

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

$\text{Cond1} \rightarrow \text{Cond2} \equiv \neg \text{Cond1} \vee \text{Cond2} \equiv \text{IF Cond1 THEN}$
 Cond2 ELSE ??

$\text{IF Cond1 THEN Cond2 ELSE Cond3} \equiv$
 $(\text{Cond1} \rightarrow \text{Cond2}) \wedge$
 $(\neg \text{Cond1} \rightarrow \text{Cond3})$

6. Quantifiers $\forall x : \text{Type}(f), \exists x : \text{Type}(f)$

Let f_v^x be the formula f where all occurrences of x are substituted with value v of type Type

$\forall x : \text{Type}(f)$ – large conjunction

$$f_{v_1}^x \wedge f_{v_2}^x \wedge f_{v_3}^x \wedge f_{v_4}^x \wedge f_{v_5}^x \dots$$

$\exists x : \text{Type}(f)$ – large disjunction

$$f_{v_1}^x \vee f_{v_2}^x \vee f_{v_3}^x \vee f_{v_4}^x \vee f_{v_5}^x \dots$$

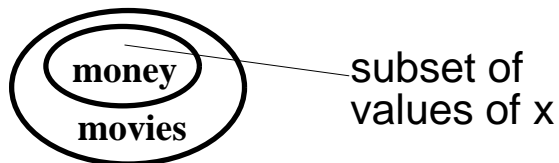
Examples:

$money : Student \rightarrow Boolean$

$movies : Student \rightarrow Boolean$

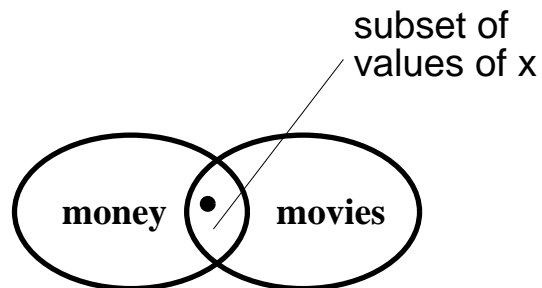
- Every student who has money will go to the movies

$$\forall s : Student (money(s) \rightarrow movies(s))$$



- Someone has money and will go to the movies

$$\exists s : Student (money(s) \wedge movies(s))$$



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

coin : time \rightarrow boolean

locked : time \rightarrow boolean

push : time \rightarrow boolean

enter : time \rightarrow boolean

rotating : time \rightarrow boolean

#entries : time \rightarrow integer

#coins : time \rightarrow integer

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.

$$\#coins(0) = 0$$

$$coin(1) \rightarrow \neg locked(2)$$

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.

$$\forall t : \textit{Time}.(\textit{coin}(t) \rightarrow \neg \textit{locked}(t + 1))$$

$$\forall t : \textit{Time}.(\textit{coin}(t) \rightarrow \\ \exists t2 : \textit{Time}.(t2 > t \wedge \neg \textit{locked}(t2)))$$

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.

$$\forall t : Time. (\#entries(t) \leq \#coins(t))$$

- If a visitor pushes the turnstile and the turnstile is unlocked, then eventually the visitor will enter the park.

$$\forall t : Time. ((push(t) \wedge \neg locked(t)) \rightarrow \\ \exists t1 : Time. (t1 > t \wedge enter(t1)))$$

- If a visitor pushes the turnstile when the turnstile is unlocked then the turnstile rotates until the visitor enters the park.

$$\forall t : Time. ((push(t) \wedge \neg locked(t)) \rightarrow \\ \exists t1 : Time. (t1 > t \wedge enter(t1) \wedge \\ \forall t2 : Time. (t < t2 < t1 \rightarrow rotating(t2))))$$

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 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 still 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.

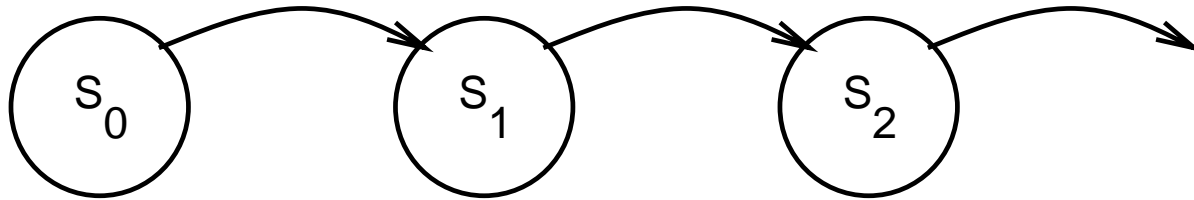
System State

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.

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

$$\sigma = s_0, s_1, s_2, \dots$$

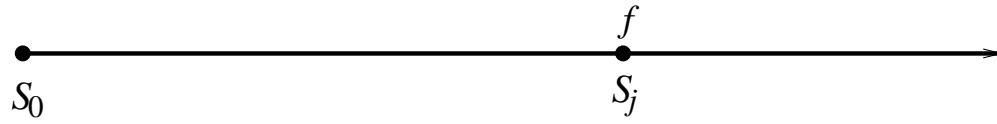
Base Formulae of LTL

In linear temporal logic (LTL), formulae are evaluated with respect to a particular execution and a particular state in that execution

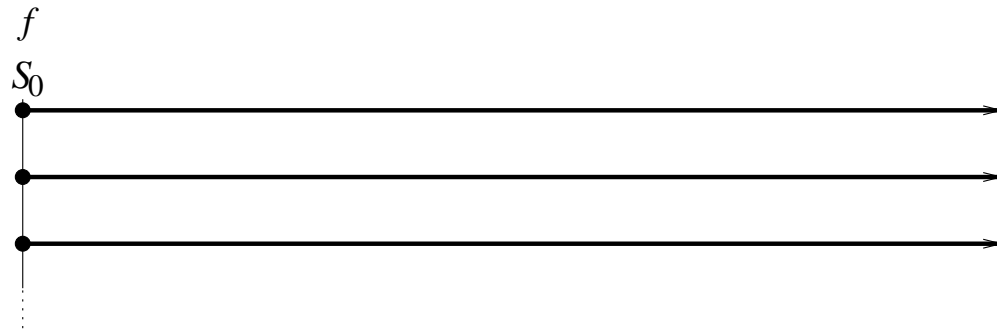
Formulae evaluated wrt a state in an execution.

$$\begin{aligned}(\sigma, j) \models f & \text{ iff } f \text{ is true in state } s_j \text{ of } \sigma \\ & \text{ iff } s_j \models f \\ \models f & \text{ iff } f \text{ is true in state } s_0 \\ & \text{ of all executions } \sigma\end{aligned}$$

$$(\sigma, j) \models f$$



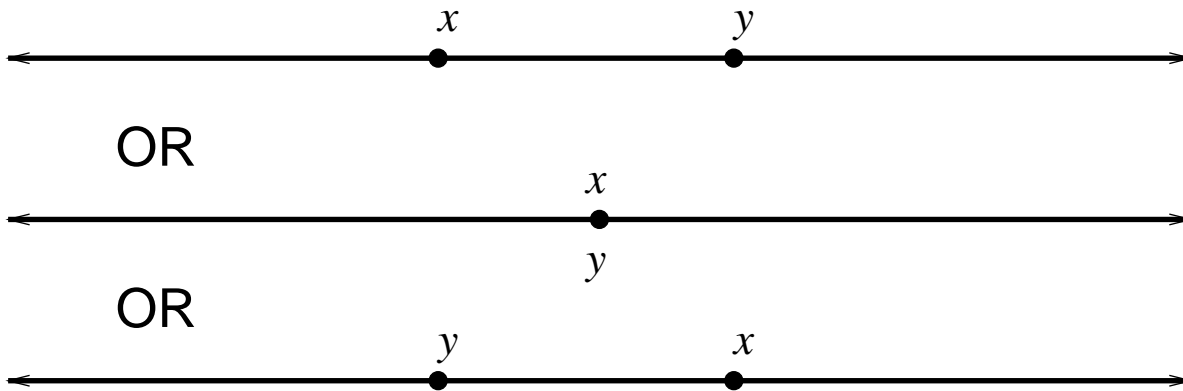
$$\models f$$



LINEAR Temporal Logic (LTL)

Time is totally ordered.

$$\forall x, y : \textit{Time}. (x < y \vee x = y \vee y < x)$$

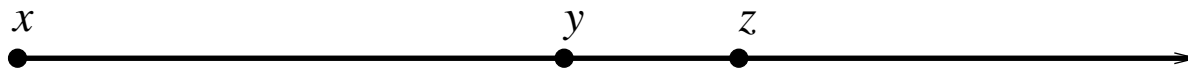


Boundedness

Time is usually bounded in the past and unbounded in the future.

$$\exists x : \textit{Time}.(\neg \exists z : \textit{Time}.(z < x))$$

$$\forall y : \textit{Time}.(\exists z : \textit{Time}.(y < z))$$



Density

$$\forall x, y : \textit{Time}.(x < y \rightarrow \exists z : \textit{Time}.(x < z < y))$$



Discreteness

$\forall x, y : \textit{Time}.$

$(x < y \rightarrow \exists w : \textit{Time}.$

$(x < w \leq y \wedge \neg \exists u : \textit{Time}.$

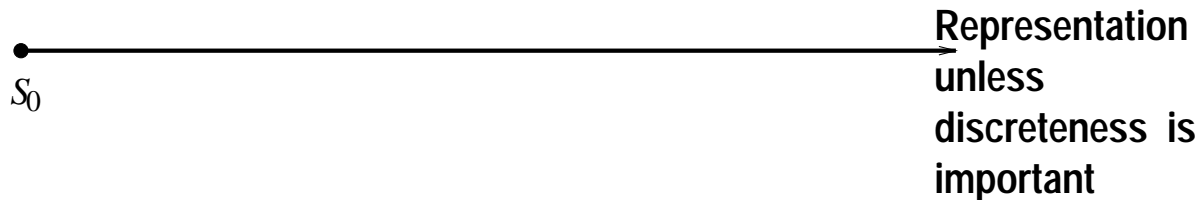
$(x < u < w))) \wedge$

$\forall x, y : \textit{Time}.$

$(x < y \rightarrow \exists z : \textit{Time}.$

$(x \leq z < y \wedge \neg \exists u : \textit{Time}.$

$(z < u < y)))$



Variables, Functions, Predicates

coin : time \rightarrow boolean

locked : time \rightarrow boolean

push : time \rightarrow boolean

enter : time \rightarrow boolean

rotating : time \rightarrow boolean

#entries : time \rightarrow integer

#coins : time \rightarrow integer

A function on *Time* is a mathematical model for a computer variable that really varies over time.

A mathematical variable is really a constant. When you say “Let $x = 5$,” and you continue to use x in the sentences thereafter, that x is 5 and it does not change until you happen to introduce another x in what is another scope.

Temporal Connectives

Connectives are shorthand notations that quantify over future system states.

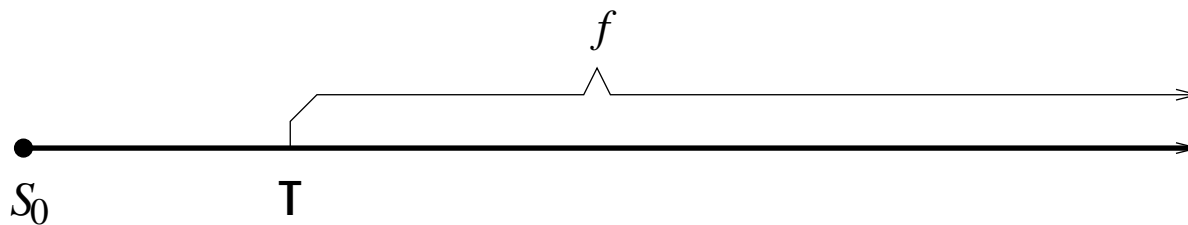
- henceforth: \square
- eventually: \diamond
- next state: \bigcirc
- until: \mathcal{U}
- unless: \mathcal{W}

Henceforth

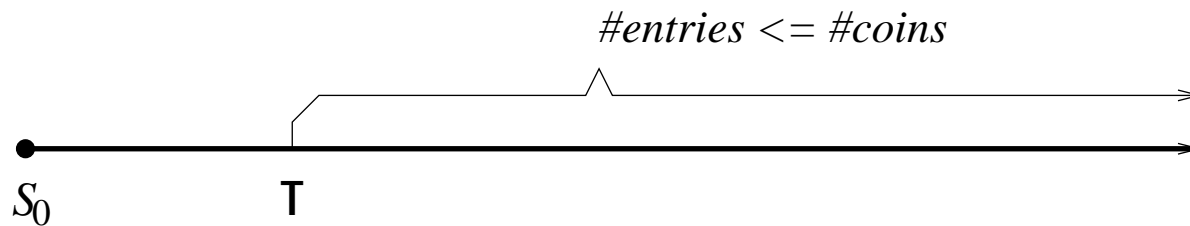
$\Box f =$

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

$$(\sigma, j) \models \Box f \quad \text{iff} \\ \forall i. (j \leq i \rightarrow (\sigma, i) \models f))$$



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



Shorthand for:

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

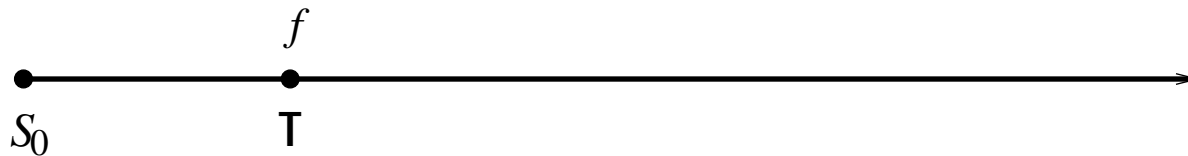
$\forall t : Time. (t \geq t_0 \rightarrow (\#entries(t) \leq \#coins(t)))$

Eventually

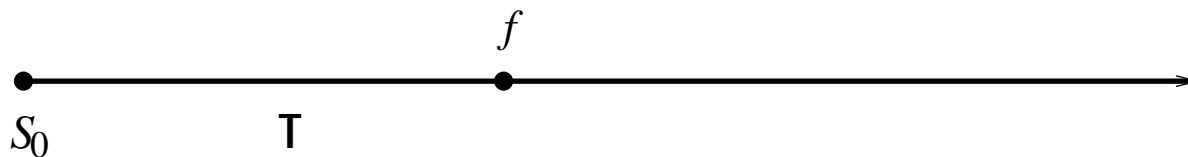
$\diamond f =$

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

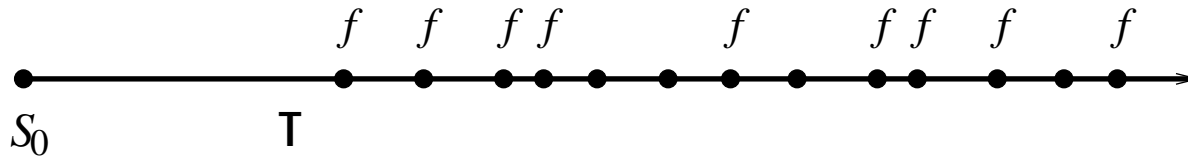
$$(\sigma, j) \models \diamond f \quad \text{iff} \\ \exists i. (j \leq i \wedge (\sigma, i) \models f))$$



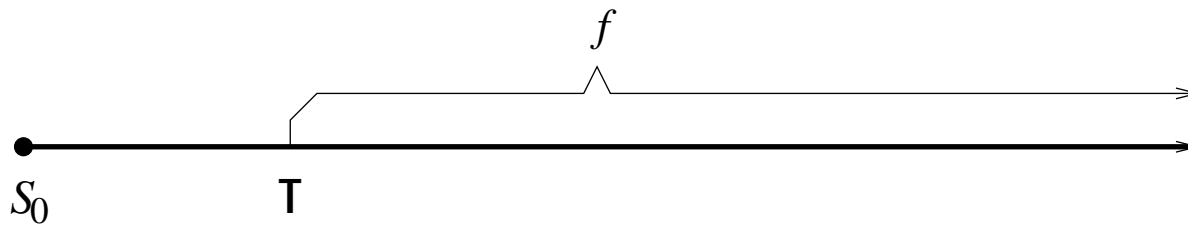
OR



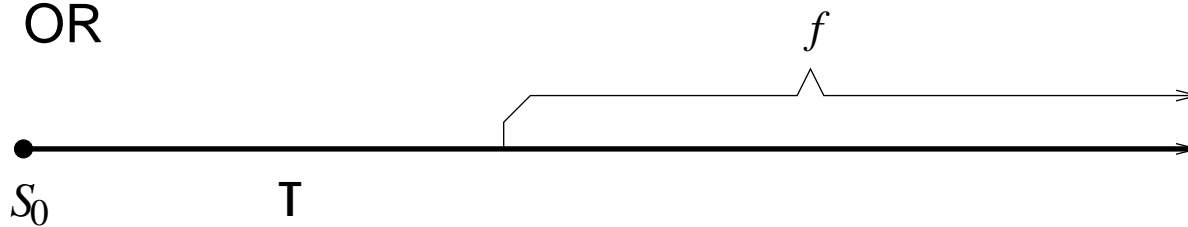
Thus, $\Box(\Diamond f)$ means that f happens infinitely often.



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

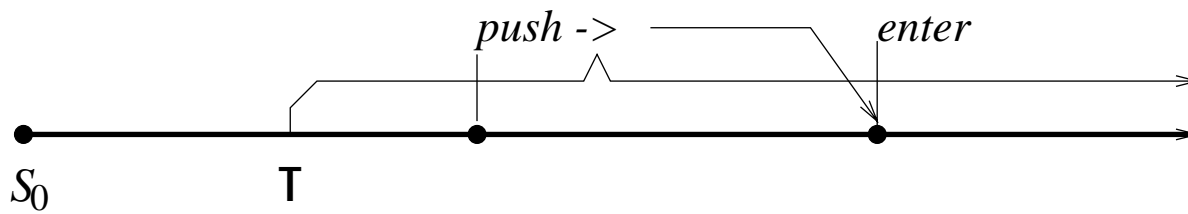
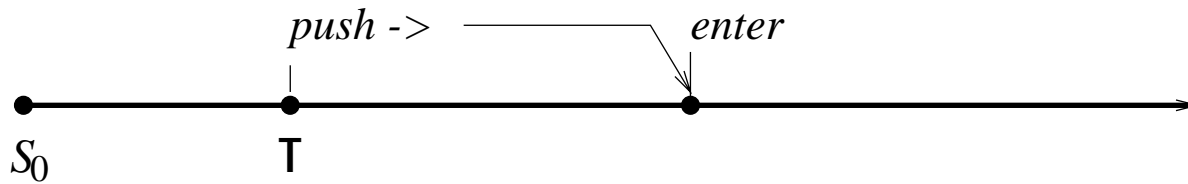


OR



Ex: $push \rightarrow \diamond enter$

Ex: $\square(push \rightarrow \diamond enter)$

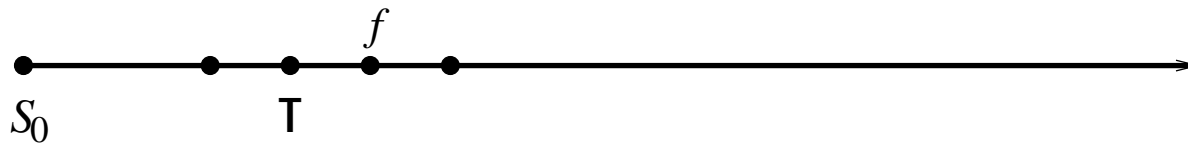


Next State

$\bigcirc f =$

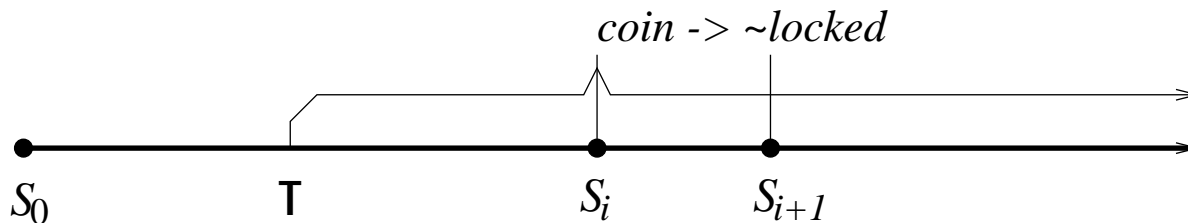
$$\begin{cases} T & \text{if } f \text{ is true in the } \underline{\text{next}} \\ & \text{system state} \\ F & \text{otherwise} \end{cases}$$

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



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

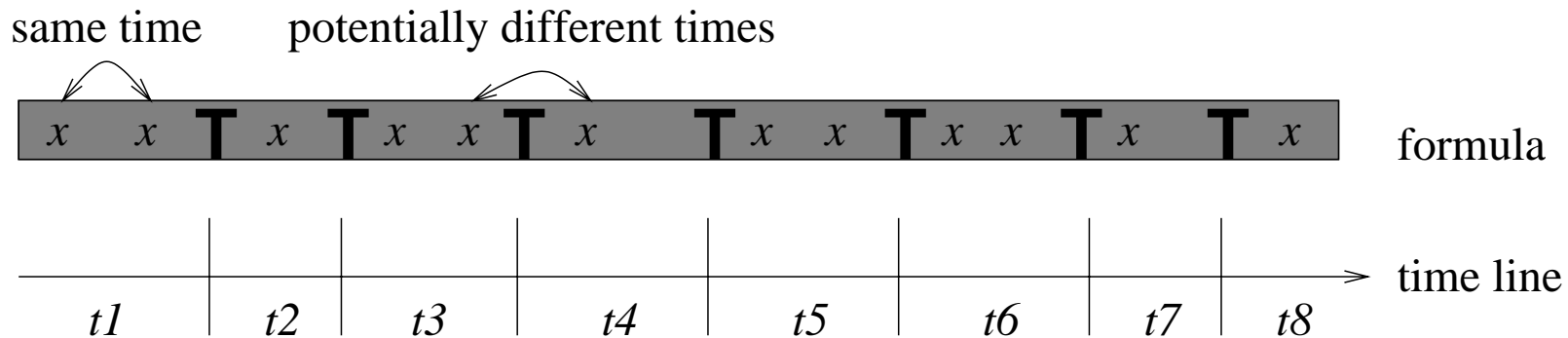
Ex: $(\Box(\textit{coin} \rightarrow \bigcirc \neg \textit{Locked})) \rightarrow$
 $(\Box(\textit{coin} \rightarrow \Diamond \neg \textit{Locked}))$



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.



$t1 \leq t2, t2 \leq t3, \text{ etc.}$

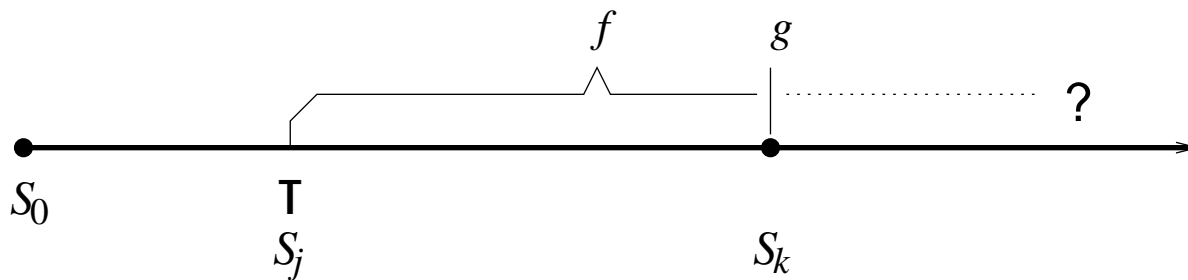
Thus, in between two consecutive temporal operators, all variables are evaluated at the same time, but variables separated by at least one temporal operator are evaluated at potentially different times.

Until

$f \mathcal{U} g =$

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

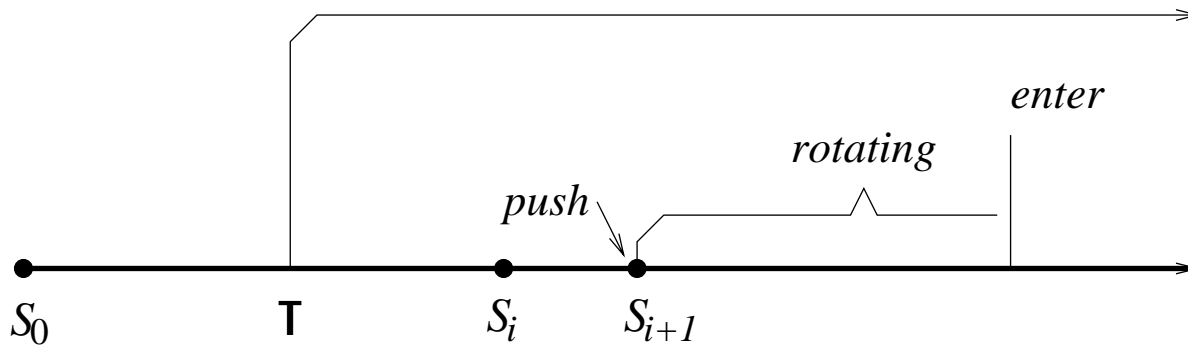
$$\begin{aligned} (\sigma, j) \models f \mathcal{U} g & \text{ iff} \\ & \exists k. (k \geq j \wedge ((\sigma, k) \models g) \wedge \\ & \forall i. (j \leq i < k \rightarrow (\sigma, i) \models f)) \end{aligned}$$



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

$\frac{t_1}{\text{-----}} \quad \frac{t_{1+1}}{\text{-----}} \quad \frac{t_2 \geq t_{1+1}}{\text{-----}}$

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



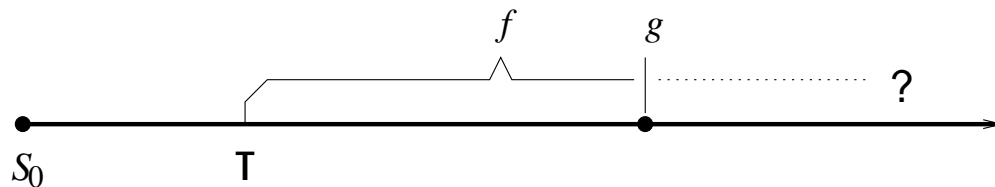
Unless

$f \mathcal{W} g =$

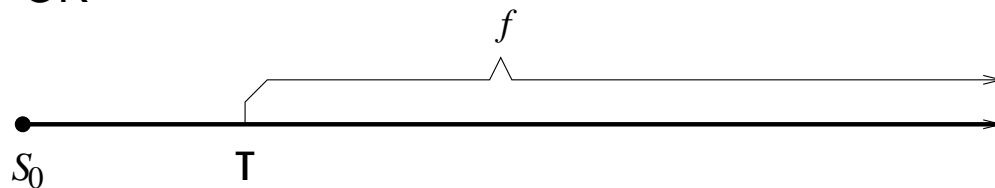
$$\begin{cases} T & \text{if } f \text{ holds indefinitely or} \\ & \text{until } g \text{ holds} \\ F & \text{otherwise} \end{cases}$$

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

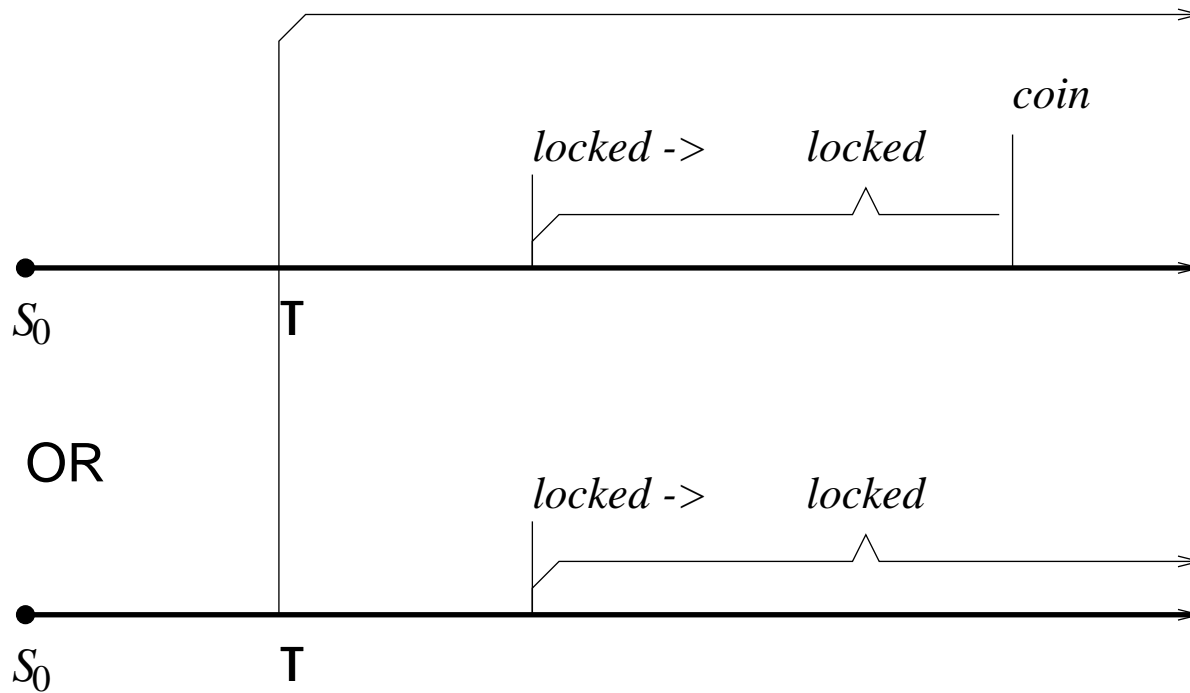
Unless is like Until, without the guarantee that g might happen.



OR



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



More on Variables

Recall:

A function on *Time* is a mathematical model for a computer variable that really varies over time.

A mathematical variable is really a constant. When you say “Let $x = 5$,” and you continue to use x in the sentences thereafter, that x is 5 and it does not change until you happen to introduce another x in what is another scope.

Temporal logic does its magic only on true variables, i.e., functions on *Time*.

Thus,

$\forall t : \textit{Time}.(t > t_0 \rightarrow x(t) > 5)$ is a longhand for $\Box x > 5$

and

$\exists t : \textit{Time}.(t > t_0 \wedge x(t) > 5)$ is a longhand for $\Diamond x > 5$

Note that a mathematical variable that is not on *Time* is a constant, not only in fact, but also to a temporal logic formula.

Thus, if $y = 5$, then

$\Box x > 5$ is the same as $\Box x > y$,

and this is a shorthand for

$\forall t : \textit{Time}.x(t) > y$

Also tautologies and theorems are true independent of the temporal operator. Thus,

$$T \rightarrow \Box T \text{ and}$$

$$T \rightarrow \Diamond T,$$

but something that is true henceforth is not necessarily a tautology or theorem.

Thus, the theorem $\sum_{i=1}^n i = \frac{n \times (n+1)}{2}$ holds independently of time, and

$$\Box \sum_{i=1}^n i = \frac{n \times (n+1)}{2}, \text{ and}$$

$$\Diamond \sum_{i=1}^n i = \frac{n \times (n+1)}{2}.$$

It is usually incorrect to write $\Box z(t)$.

It is incorrect to have (t) after z after a temporal operator such as \Box and \Diamond *unless* z has two time parameters, i.e.,

$z : \textit{Time} \times \textit{Time}$,

in which case, one of the *Times* is implicit in the temporal logic and the other is left explicit.

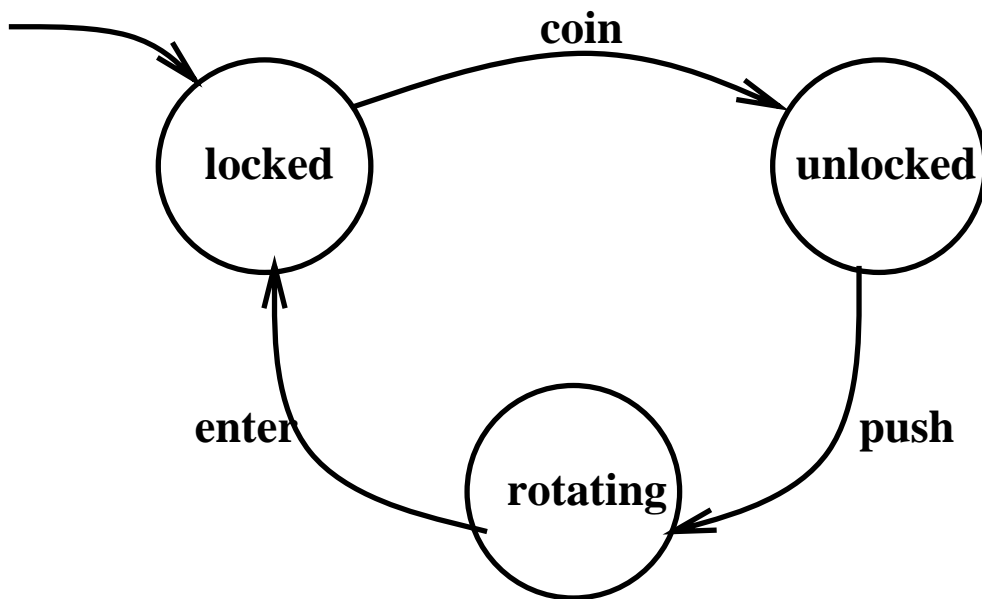
In other words, $\Box z(t)$ is a shorthand for

$\forall t_1 : \textit{Time}.(t_1 > t_0 \rightarrow z(t_1, t))$

Describing Behaviour of Finite State Machines

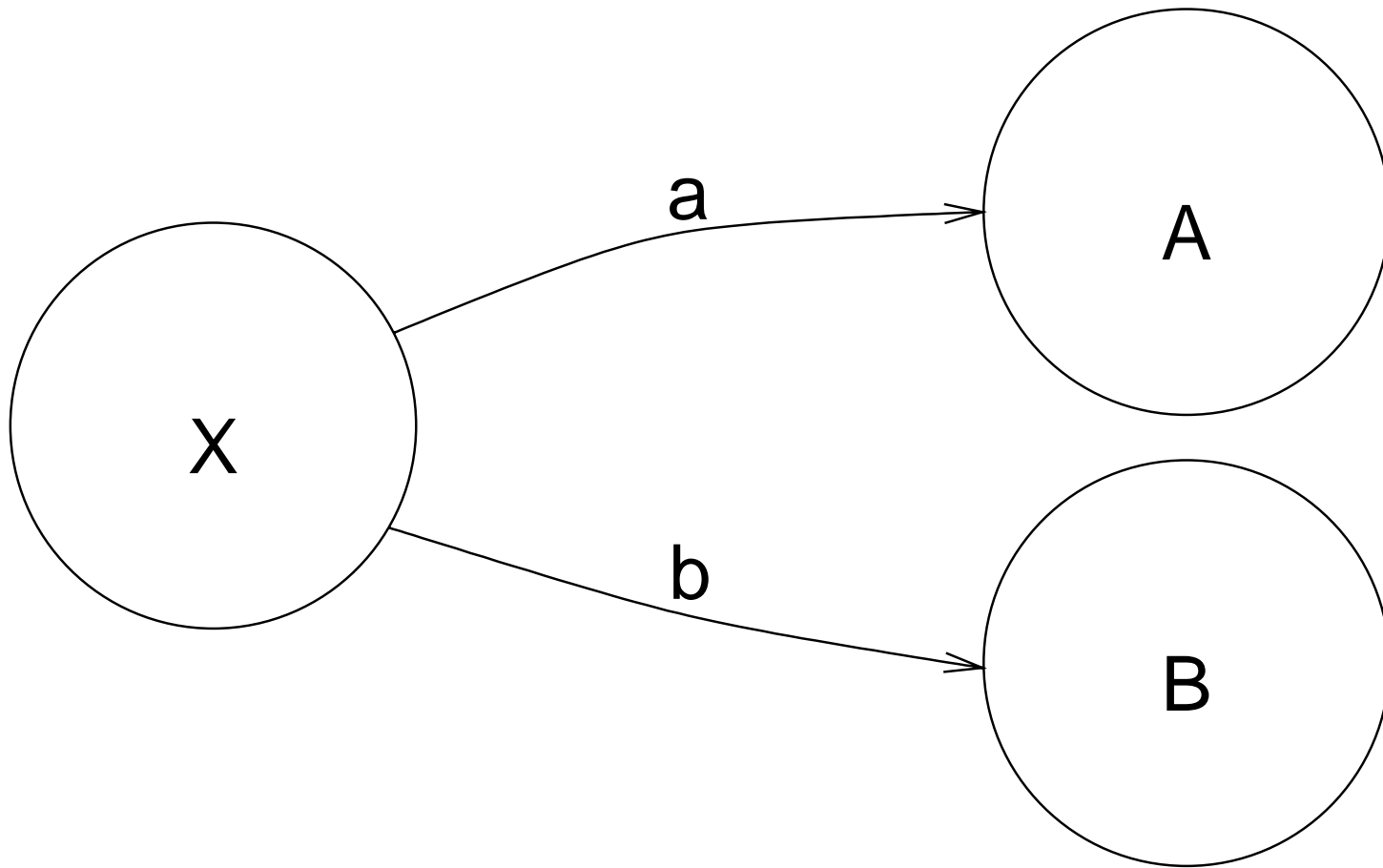
You can use these connectives to describe the behaviour of a finite state machine.

Assume that the machine is in only one state at a time.



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

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



$\square(X \rightarrow (X \mathcal{W} (a \vee b)))$

$\square((X \wedge a) \rightarrow \bigcirc(A))$

$\square((X \wedge b) \rightarrow \bigcirc(B))$