Temporal Logic and Model Checking

SE 3BB4

Ryszard Janicki

Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada

Three Basic Models of Concurrency

- Algebraic or Equational: all process algebras including FSP of the textbook.
- Automata Based: Petri Nets, Asynchronous Automata, etc.
- Logic and Model Theory Based: Temporal logics (as CTL, LTL, CTL*), etc.

Verification Techniques

Formal verification techniques consist of:

- A framework of modeling systems, typically a description language of some sort
- A specification language for describing the properties to be verified
- A verification method to establish whether the description of a system satisfies the specification.

Approaches to verification can be classified as **Proof-based** and **Model-based**.

The above statements are valid for **all** systems, but they are especially important for **concurrent** systems.

Proof-based vs Model-based

- Proof-based: The *system description* is a set of formulas Γ (in a suitable logic) and the *specification* is another formula Φ. The verification method consists of trying to find a proof that $\Gamma \vdash \Phi$. This typically requires guidance and expertise from the user.
- Problem: Predicate Logic is undecidable, we will never construct a 'push button' theorem prover that could prove $P \implies Q$ for any P and Q.
- Model-based: The system is represented by a finite model $\mathcal{M}$ for an appropriate logic. The specification is again represented by a formula Φ and the verification method consists of whether a model $\mathcal{M}$ satisfies Φ. This is usually automatic, though the restriction to finite models limits the applicability.
- Problem: A model $\mathcal{M}$ may have millions of states, so an appropriate logic must be simple enough to allow efficient implementations.

Model-based approach is potentially simpler that the proof-based approach, for it is based on a single model $\mathcal{M}$ rather than a possibly infinite class of them.

Temporal Logic : Ideas

- In classical logic, formulae are evaluated within a single fixed world.
- For example, an *elementary* proposition such as “it is Monday” must be either true or false.
- Propositions are then combined using constructs such as $\wedge, \neg$, etc.
- But, most (not just computational) systems are dynamic.
- In temporal logics, evaluation takes place within a set of worlds. Thus, “it is Monday” may be satisfied in some worlds, but not in others.

Temporal Logic : Ideas (cont.)

- The set of worlds correspond to moments in time.
- How we navigate between these worlds depends on our particular view of time.
- The particular model of time is captured by a temporal accessibility relation between worlds.
- Essentially, temporal logic extends classical propositional logic with a set of temporal operators that navigate between worlds using this accessibility relation.
- To be useful for verification, an appropriate *temporal logic* must allow efficient checking algorithms. Hence it must be relatively simple.

Model Checking and Temporal Logic

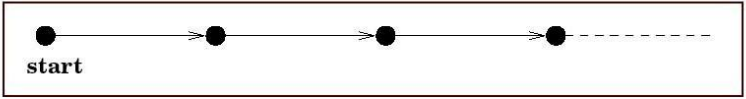
- The idea of temporal logic is that a formula is not statically true or false. Instead, the models of temporal logic contain several states and a formula can be true in some states and false in others. The *static* notion of truth is replaced by a dynamic one.
- The models $\mathcal{M}$ are *transition systems* (i.e. finite automata) and the properties Φ are formulas in temporal logic.
- To verify that a system satisfies some property we must do three things:
  1. Model the system using the description language of a model checker, arriving at a model $\mathcal{M}$.
  2. Code the property using the specification language of the model checker, resulting in a temporal logic formula Φ.
  3. Run the model checker with inputs $\mathcal{M}$ and Φ.

Model Checking and Temporal Logic

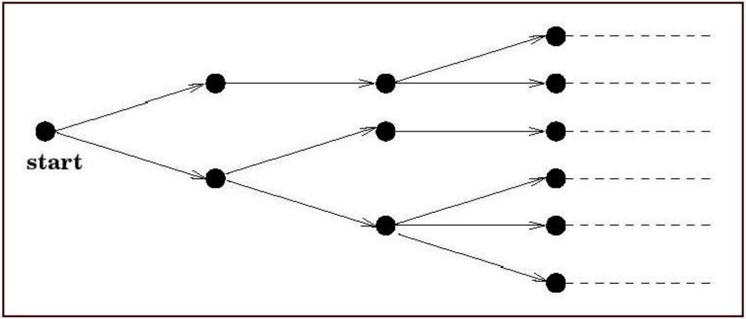
- The model checker outputs the answer "yes" if $\mathcal{M}$ satisfies Φ and "no" otherwise; in the latter case, most model checkers also produce a *trace of system behaviour which causes* this failure.
- There are many *temporal logics*, we concentrate on CTL (Computation Tree Logic) and LTL (Linear Time Logic).
- Time could be *continuous* or *discrete*, we concentrate on *discrete time*.
- $\mathcal{M}$ is not a description of an actual physical system. Models are abstractions that omit lots of real features of a physical systems. *We have similar situation in calculus, mechanics, etc., where we have straight lines, perfect circles, no friction, etc.*

Typical Models of Time

- **Linear Time**: used for Linear Temporal Logic (LTL)



- **Branching time**: used for CTL, CTL* logics, etc.



CTL (Computational Tree Logic)

- CTL is a branching-time logic, meaning that its model of time is a tree-like structure in which the future is not determined; there are different paths in the future, any one of which might be the 'actual' path that is realized.
- We work with a fixed set of atomic formulas/description ($p, q, r, \ldots$, or $p_1, p_2, \ldots$). These atoms stand for atomic descriptions of a system, like:

  the printer is busy

  there are currently no requested jobs for the printer

  the current content of register R1 is the integer 6
- The choice of atomic descriptions depends on our particular interest in a system at hand.

CTL Syntax

$\Phi ::= \bot \mid \top \mid p \mid (\neg\Phi) \mid (\Phi \wedge \Phi) \mid (\Phi \vee \Phi) \mid (\Phi \Rightarrow \Phi) \mid$
$AX\Phi \mid EX\Phi \mid A[\Phi U \Phi] \mid E[\Phi U \Phi] \mid$
$AG\Phi \mid EG\Phi \mid AF\Phi \mid EF\Phi$

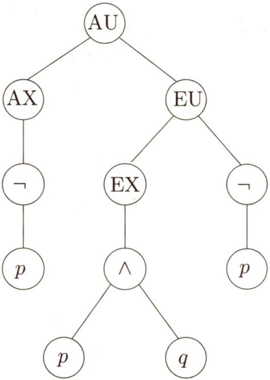
where p ranges over atomic formulas/descriptions.

- $\bot$ - false, $\top$ - true
- $\underbrace{AX, EX, AG, EG, AU, EU, AF, EF}_{\text{all pairs, each starts with either } A \text{ or } E}$ are **temporal connections**.
- A means "along All paths" (inevitably)
- E means "along at least (there Exists) one path" (possibly)
- X means "neXt state"
- F means "some Future state"
- G means "all future states (Globally)"
- U means "Until"
- X, F, G, U cannot occur without being preceded by A or E.
- every A or E must have one of X, F, G, U to accompany it.

Bindings

$$
\begin{array}{ccc}
\neg, AG, EG, AF, EF, AX & \leftarrow & \text{strongest bind} \\
\downarrow & & \\
\wedge, \vee & & \\
\downarrow & & \\
\Rightarrow, AU, EU & \leftarrow & \text{lowest bind}
\end{array}
$$

Parsing Trees



$$A[AX\neg p\ U\ E[EX(p \wedge q)\ U\ \neg p]]$$

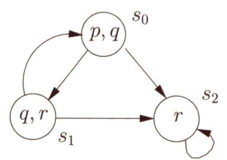
- A subformula of a CTL formula Φ is any formula Ψ whose parse tree is a subtree of Φ's parse tree.

Model

Definition

A **model** $\mathcal{M} = (S, \rightarrow, L)$ for CTL is a set of states S endowed witha transition relation $\rightarrow$ (a binary relation on S), such that every $s \in S$ has some $s' \in S$ with $s \rightarrow s'$ and a labeling function $L : S \rightarrow 2^{Atoms}$.

Example



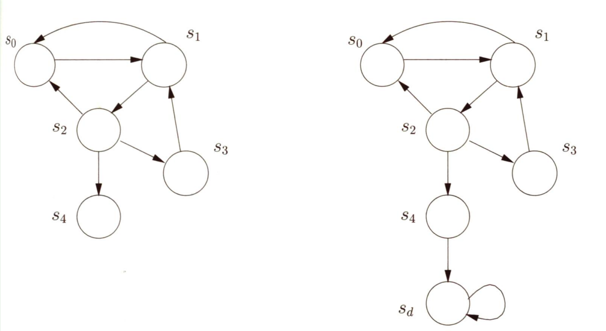
$$L(s_0) = \{p, q\},\ L(s_1) = \{q, r\},\ L(s_2) = \{r\}$$

No deadlock

Definition

"No deadlock" iff for every $s \in S$ there is at least one $s' \in S$ such that $s \to s'$.

Example



A system with a deadlock

A system without a deadlock, s_d is a "deadlock" state

Examples of CTL Formulas

- An upwards traveling elevator at the second floor does not change its direction when it has passengers wishing to go to the fifth floor:

$$AG(floor = 2 \wedge direction = up \wedge ButtonPressed5 \Rightarrow$$
$$A[direction = up \; U \; floor = 5])$$

- The elevator can remain idle on the third floor with its doors closed:

$$AG((floor = 3 \wedge idle \wedge door = closed) \Rightarrow$$
$$EG(floor = 3 \wedge idle \wedge door = closed))$$

- '$floor = 2$', '$direction = up$', $ButtonPressed5$', '$door = closed$', etc. are *names* of *atomic formulas*.

Let $\mathcal{M} = (S, \rightarrow, L)$ be a model for CTL. Given any $s \in S$, we define whether a CTL formula Φ holds in state s. We denote this by: $\mathcal{M}, s \models \Phi$

Definition (The definition of $\models$)

1. $\mathcal{M}, s \models \top$ and $\mathcal{M}, s \not\models \bot$ for all $s \in S$
2. $\mathcal{M}, s \models p$ iff $p \in L(s)$
3. $\mathcal{M}, s \models \neg\Phi$ iff $\mathcal{M}, s \not\models \Phi$
4. $\mathcal{M}, s \models \Phi_1 \wedge \Phi_2$ iff $\mathcal{M}, s \models \Phi_1$ and $\mathcal{M}, s \models \Phi_2$
5. $\mathcal{M}, s \models \Phi_1 \vee \Phi_2$ iff $\mathcal{M}, s \models \Phi_1$ or $\mathcal{M}, s \models \Phi_2$
6. $\mathcal{M}, s \models \Phi_1 \Rightarrow \Phi_2$ iff $\mathcal{M}, s \not\models \Phi_1$ or $\mathcal{M}, s \models \Phi_2$

7. $\mathcal{M}, s \models AX\Phi$ iff for all s_1 such that $s \rightarrow s_1$, we have $\mathcal{M}, s_1 \models \Phi$

   AX says: "in every next state".

8. $\mathcal{M}, s \models EX\Phi$ iff for some s_1 such that $s \rightarrow s_1$, we have $\mathcal{M}, s_1 \models \Phi$

   EX says: "in some next state".

   E is dual to A, as $\exists$ is dual to $\forall$.

9. $\mathcal{M}, s \models AG\Phi$ iff for all paths $s_1 \rightarrow s_2 \rightarrow \ldots$, where s_1 equals s, and for all s_i (including s_1) along the path, we have $\mathcal{M}, s_i \models \Phi$.

   For All computation paths beginning with s the property Φ holds Globally.

10. $\mathcal{M}, s \models EG\Phi$ iff there is a path $s_1 \rightarrow s_2 \rightarrow \ldots$, where s_1 equals s, and for all s_i along the path, we have $\mathcal{M}, s_i \models \Phi$.

    There Exists a path beginning in s such that Φ holds Globally along the path.

11. $\mathcal{M}, s \models AF\Phi$ iff for all paths $s_1 \rightarrow s_2 \rightarrow \ldots$, where s_1 equals s, there is some s_i along the path, such that $\mathcal{M}, s_i \models \Phi$.

    For All computation paths beginning with s there will be some Future state where Φ holds.

12. $\mathcal{M}, s \models EF\Phi$ iff there is a path $s_1 \rightarrow s_2 \rightarrow \ldots$, where s_1 equals s, and for some s_i along the path, we have $\mathcal{M}, s_i \models \Phi$.

    There Exists a computation path beginning in s such that Φ holds in some Future states.

13. $\mathcal{M}, s \models A[\Phi_1 \; U \; \Phi_2]$ iff for all paths $s_1 \rightarrow s_2 \rightarrow \ldots$, where s_1 equals s, that path satisfies $\Phi_1 \; U \; \Phi_2$, i.e. there is some s_i along the path, such that $\mathcal{M}, s_i \models \Phi_2$, and for each $j < i$, we have $\mathcal{M}, s_j \models \Phi_1$.

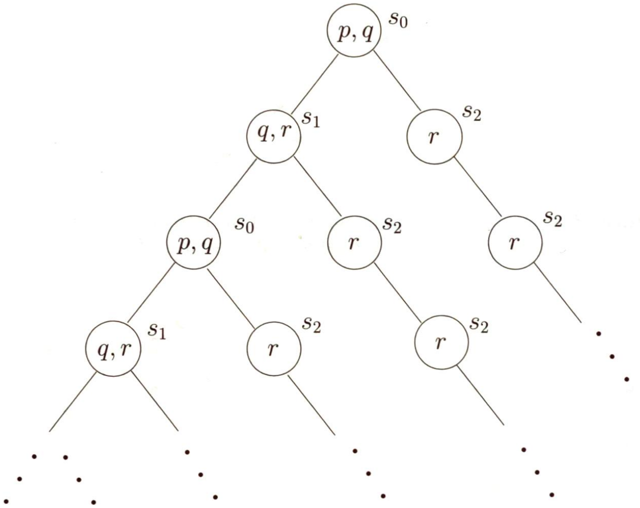
    All computation paths beginning in s satisfy that Φ_1 Until Φ_2 holds on it.

14. $\mathcal{M}, s \models E[\Phi_1 \; U \; \Phi_2]$ iff there is a path $s_1 \rightarrow s_2 \rightarrow \ldots$, where s_1 equals s, that path satisfies $\Phi_1 \; U \; \Phi_2$, as specified in (13).

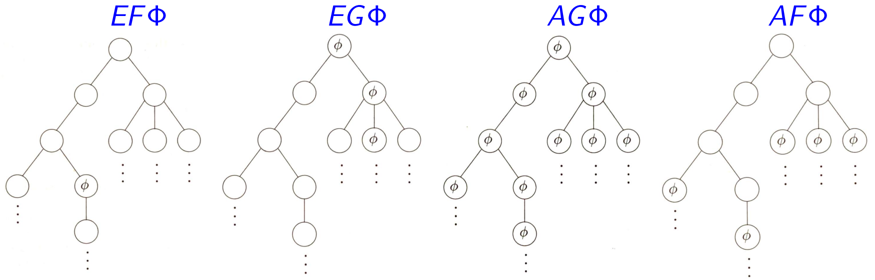
    There Exists a computation path beginning in s such that Φ_1 Until Φ_2 holds on it.

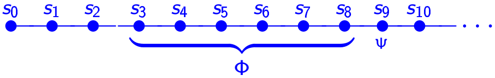
- In clauses 9-14, **the future includes the present**.

- Unwinding the system from page 14 as an infinite tree of all computation path beginning in a particular state.



Semantics: Illustrations





each of the states from s_3 to s_9 satisfies $\Phi\ U\ \Psi$

- If the given set of states is finite, then we may compute the set of all states satisfying Φ.
- If $\mathcal{M}$ is obvious, we will write $s \models \Phi$.

Some Examples for the System from Pages 14 and 21

Example

1. $\mathcal{M}, s_0 \models p \wedge q$ since $L(s_0) = \{p, q\}$
2. $\mathcal{M}, s_0 \models \neg r$ since $r \neq L(s_0)$
3. $\mathcal{M}, s_0 \models \top$ by the definition
4. $\mathcal{M}, s_0 \models EX(q \wedge r)$ since we have the leftmost computation path $s_0 \to s_1 \to s_0 \to s_1 \to \ldots$ in Figure on page 21, and $L(s_1) = \{q, r\}$
5. $\mathcal{M}, s_0 \models \neg AX(q \wedge r)$ since we have the rightmost computation path $s_0 \to s_2 \to s_2 \to s_2 \to \ldots$ in Figure on page 21, and $q \notin L(s_2)$
6. $\mathcal{M}, s_0 \models \neg EF(p \wedge r)$ since there is no computation path beginning in s_0 such that we could reach a state where $p \wedge q$ would hold.

For each $s \in S, p \in L(s) \Leftrightarrow r \notin L(s)$.

Example (continued)

7. $\mathcal{M}, s_2 \models EGr$ since there is a computation path $s_2 \to s_2 \to s_2 \to s_2 \to \ldots$ beginning with s_2 such that r holds in all future states.
8. $\mathcal{M}, s_2 \models AGr$ since there is *only one* computation path beginning in s_2 and it satisfies r globally.
9. $\mathcal{M}, s_0 \models AFr$ since for all computation paths beginning in s_0, the system reaches a state (s_1 or s_2) such that r holds.
10. $\mathcal{M}, s_0 \models E[(p \wedge q)\ U\ r]$ since we have the rightmost computation path $s_0 \to s_2 \to s_2 \to s_2 \to \ldots$ in Figure on page 16, whose second node s_2 ($i = 1$) satisfies r, but all previous nodes (only $j = 0$, i.e. node s_0) satisfy $p \wedge q$.
11. $\mathcal{M}, s_0 \models A[p\ U\ r]$ since p holds in s_0 and r holds in any possible successor state of s_0, so $p\ U\ r$ is true for all computation paths beginning in s_0.

Practical Patterns of Specifications (1)

What kind of practically relevant properties can we check with formulas of CTL?

Suppose atomic descriptions include some words as busy, requested, ready, etc.

- It is possible to get a state where started holds but ready does not hold:

  $$EF(started \wedge \neg ready)$$

- For any state, if a request (of some resource) occurs, then it will eventually be acknowledged:

  $$AG(request \Rightarrow AF\ acknowledged)$$

- A certain process is enabled infinitely often on every computation path:

  $$AG(AF\ enabled)$$

- Whatever happens, a certain process will eventually be permanently deadlocked:

  $$AF(AG\ deadlock)$$

Practical Patterns of Specifications (2)

- From any state it is possible to get a restart state:

$$AG(EF\ restart)$$

- An upwards traveling elevator at the second floor does not change its direction when it has passengers wishing to go to the fifth floor:

$$AG(floor = 2 \wedge direction = up \wedge ButtonPressed5 \Rightarrow A[direction = up\ U\ floor = 5])$$

- The elevator can remain idle on the third floor with its doors closed:

$$AG((floor = 3 \wedge idle \wedge door = closed) \Rightarrow EG(floor = 3 \wedge idle \wedge door = closed))$$

Practical Patterns of CTL Specifications (3)

- Train doors shall always remain closed between platforms unless the train is stopped in emergency.

  We cannot specify this statement in CTL, as it should start with $\forall tr : Train, pl : Platform \ldots$ and we do not have quantifiers $\forall$ and $\exists$ in CTL!

- For train $tr75$, its doors shall always remain closed between platforms $pl2$ and $pl3$ (i.e. next platform) unless the train is stopped in emergency.

  $$AG(tr75.at.pl2 \land \neg tr75.at.pl3 \implies AG(tr75.doors = \text{`closed'}) \\ \lor\ tr75.doors = \text{`closed'}\ U\ tr75.at.pl3 \\ \lor\ (Alarm.tr75 \land \neg tr75.moving))$$

Equivalences

- Two CTL formulas Φ and Ψ are said to be **semantically equivalent** if any state in any model which satisfies one of them also satisfies the other, write then: $\Phi \equiv \Psi$.
- $$\left.\begin{array}{l} \neg AF\Phi \equiv EG\neg\Phi \\ \neg EF\Phi \equiv AG\neg\Phi \end{array}\right\} \text{ de Morgan rules}$$
- $$\neg AX\Phi \equiv EX\neg\Phi$$
- $$AF\Phi \equiv A[\top\ U\ \Phi]$$
  $$EF\Phi \equiv E[\top\ U\ \Phi]$$

Mutual Exclusion Problem

Mutual Exclusion: only one process can access a **critical section**.

Problem: to find a proper protocol and to verify our solution by checking that it has some expected properties as:

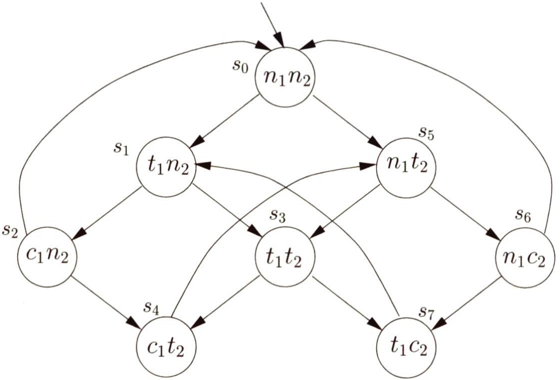
Safety: The protocol allows only one process to be in its critical section at any time.
Liveness: Whenever any process wants to enter its critical section, it will eventually be permitted to do so.

Safety: Bad things never happen.
Liveness: Good things eventually will happen.

Non-blocking: A process can always request to enter its critical section.
No strict sequencing: Processes need not enter their section in strict sequence.

Mutual Exclusion Problem: First Solution

- We assume two processes and they interleave, i.e. only one of them can make a transition at a time.
- n - non-critical state
  t - trying to enter its critical state
  c- in its critical state
- each process behaves as: $n \to t \to c \to n \to t \to c \to \ldots$



Properties of The First Solution

Safety $\Phi_1 \stackrel{def}{=} AG\neg(c_1 \wedge c_2)$

Clearly it is satisfied in every state.

Liveness $\Phi_2 \stackrel{def}{=} AG(t_1 \Rightarrow AFc_1)$

Not satisfied by s_0, since $s_0 \rightarrow s_1$ but in s_1 we have t_1 is true but AFc_1 is not, as for the path $s_1 \rightarrow s_3 \rightarrow s_7 \rightarrow s_1 \rightarrow s_3 \rightarrow s_7 \rightarrow \ldots$, c_1 is always false.

Non-blocking $\Phi_3 \stackrel{def}{=} AG(n_1 \Rightarrow EXt_1)$

Satisfied since every n_1 state has an (immediate) t_1 successor.

No strict sequencing

$\Phi_4 \stackrel{def}{=} EF(c_1 \wedge E[c_1\ U\ (\neg c_1 \wedge E[\neg c_2\ U\ c_1])])$

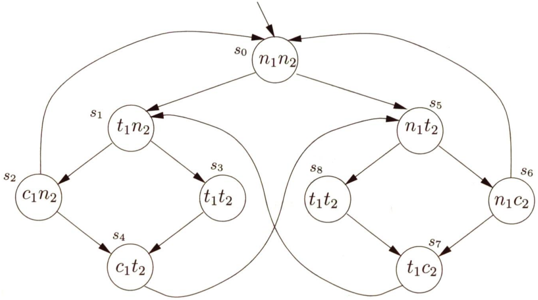
Satisfied, e.g. by the mirror path to the computation path described for liveness:

$s_5 \rightarrow s_3 \rightarrow s_4 \rightarrow s_5 \rightarrow s_3 \rightarrow s_4 \rightarrow \ldots$

Problems With Liveness

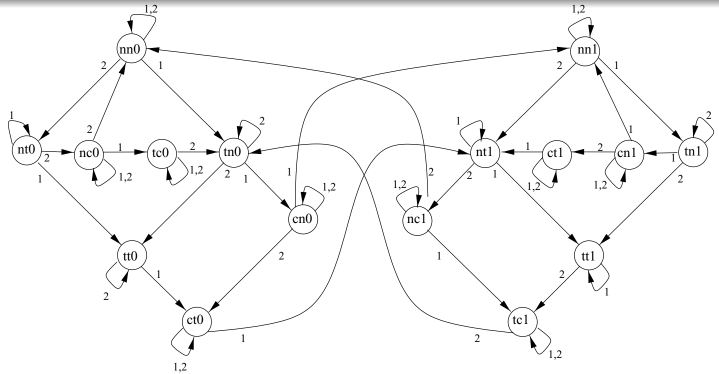
- **Reason for Liveness Failure**:
  non-determinism means that it **might** continually favour one process over another!

- The state s_3 does not distiquish between which of the processes **first** went into its trying state. We might try to split s_3 into two states.



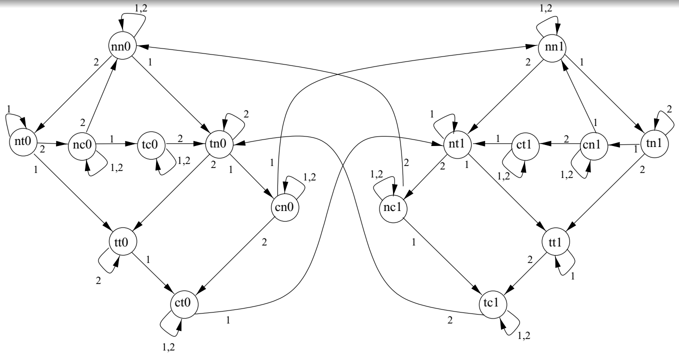
- We split the old s_3 into s_3 and s_8.
- This solution satisfies *Safety, Liveness, Non-blocking* and *No Strict Sequencing*.
- **Oversimplification**: we will move to a different state an every click of the clock! We may wish to model that a process can stay in its critical state for several ticks, but if we include an arrow from s_2 or s_6, to itself, we will again violate liveness.

Mutual Exclusion: Third Solution (with some tricks)



- *ct*0 means process 1 is in **c**ritical section, process 2 is **t**rying and the Boolean variable *turn* = **0**.
- The variable *turn* indicate which process will get into critical section, 0 indicates process 1, 1 indicate process 2. It is also often represented by two predicates *turn* = 1 and *turn* = 2.
- The labels on the transitions denote the process which makes the move. The label 1, 2 means that either process could make that move. The labels are redundant but increase readability.

Tool: *FAIRNESS* ϕ



- We want to express that finite sequences $nn0 \rightarrow nn0 \rightarrow \ldots \rightarrow nn0$, or $ct0 \rightarrow \ldots \rightarrow ct0$ are valid, but **not** infinite versions for some of them.
- The tool *FAIRNESS* ϕ, available in most model checkers, allows ignoring any path along with ϕ is not satisfied infinitely often.
- While *FAIRNESS* ϕ and *Fair Choice* discussed for FSPs have similar roots, they are different tools and concepts!
- To guarantee that no process will use a critical section infinitely, we have to invoke *FAIRNESS* $\neg c$ (or something semantically similar).

Role of *turn*

- Because the boolean variable turn has been explicitly introduced to distinguish between states s_3 and s_8 of figure from page 33, we now distinguish between certain states (for example, $ct0$ and $ct1$) which were identical before.
- However, these states are not distinguished if you look just at the transitions from them.
- Therefore, they satisfy the same CTL (or LTL) formulas which don't mention turn.
- Those states are distinguished only by the way they can arise.

Role of *turn*

- We have eliminated an over-simplification made in the model of page 33. Recall that we assumed the system would move to a different state on every tick of the clock (there were no transitions from a state to itself).
- In figure from pages 34 and 35 , we allow transitions from each state to itself, representing that a process was chosen for execution and did some private computation, but did not move in or out of its critical section.
- Of course, by doing this we have introduced paths in which one process gets stuck in its critical section, whence the need to invoke a fairness constraint to eliminate such paths.

Model Checking Algorithms

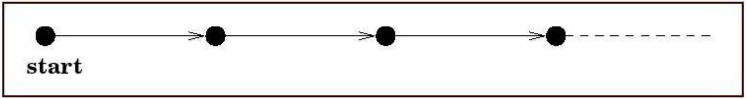
- Humans $\rightarrow$ unwinding, infinite trees
- Computers $\rightarrow$ must use transition system as it needs to check on *finite* data structures.
- How one can consider $\mathcal{M}, s_0 \overset{?}{\models} \Phi$ as a computational problem?
    1. Input: $\mathcal{M}, \phi, s_0$ Output: 'yes' or 'no'
    2. Input: $\mathcal{M}, \phi$ Output: all states s such that $\mathcal{M}, s \models \Phi$.
- One may show that (1) $\Leftrightarrow$ (2)
- The most efficient algorithms use fixed points approach, and can handle millions of states and long formulas.

Refinement, Filters, Fairness

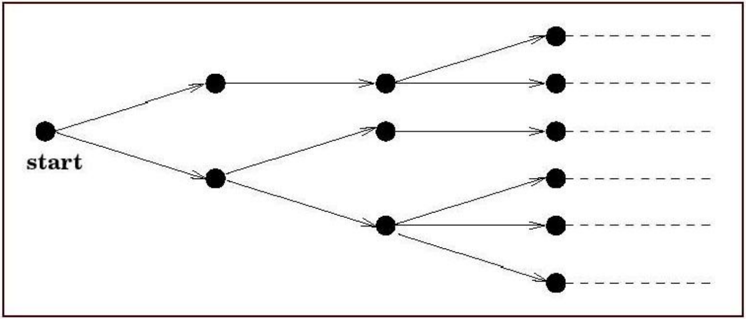
- $\mathcal{M}, s_0 \models \Phi$ might fail because $\mathcal{M}$ may contain behaviour that is unrealistic, or guaranteed not to occur in the actual system being analyzed.
- Refine $\mathcal{M}$, or
- stick to the original model and impose a **filter** on the model check:
  instead $\mathcal{M}, s_0 \models \Phi$ verify $\mathcal{M}, s_0 \models (\Psi \Rightarrow \Phi)$, where Ψ encodes the refinement of our model expressed as a specification.
- Unfortunately, not all refinements of models for CTL model checking can be done in this way.
- Simple Fairness: Φ is true infinitely often.
- Fairness: If Ψ is true infinitely often, then Φ is also true infinitely often.

Typical Models of Time

- **Linear Time**: used for Linear Temporal Logic (LTL)



- **Branching time**: used for CTL, CTL* logics, etc.



LTL Syntax

$\Phi ::= \bot \mid \top \mid p \mid (\neg\Phi) \mid (\Phi \wedge \Phi) \mid (\Phi \vee \Phi) \mid (\Phi \Rightarrow \Phi) \mid$
$(G\Phi) \mid (F\Phi) \mid (X\Phi) \mid (\Phi\ U\ \Phi) \mid (\Phi\ W\ \Phi) \mid (\Phi\ R\ \Phi)$

where p ranges over atomic formulas/descriptions.

- $\bot$ - false, $\top$ - true
- $G\Phi, F\Phi, X\Phi, \Phi\ U\ \Phi, \Phi\ W\ \Phi, \Phi\ R\ \Phi$ are **temporal connections**.
- X means "neXt moment in time"
- F means "some Future moments"
- G means "all future moments (Globally)"
- U means "Until"
- W means "Weak-until"
- R means "Release"
- An LTL formula is evaluated on a path, or a set of paths.
- A set of paths satisfies Φ if every path in the set satisfies Φ.
- Consider the path $\pi \stackrel{\text{df}}{=} s_1 \to s_2 \to \ldots$.
  We write π^i for the suffix starting at s_i, i.e. π^i is $s_i \to s_{i+1} \to s_{i+2} \to \ldots$.

Semantics

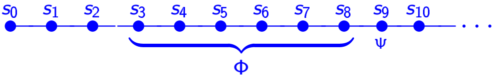
Let $\mathcal{M} = (S, \rightarrow, L)$ be a model as for CTL. We define when a path $\pi = s_1 \rightarrow s_2 \rightarrow \ldots$ satisfies an LTL formula as follows.

Definition (The definition of $\models$)

1. $\pi \models \top$
2. $\pi \not\models \bot$
3. $\pi \models p$ iff $p \in L(s_1)$ This means that atoms are evaluated in the first state along the path in consideration.
4. $\pi \models \neg\Phi$ iff $\pi \not\models \Phi$
5. $\pi \models \Phi_1 \wedge \Phi_2$ iff $\pi \models \Phi_1$ and $\pi \models \Phi_2$
6. $\pi \models \Phi_1 \vee \Phi_2$ iff $\pi \models \Phi_1$ or $\pi \models \Phi_2$
7. $\pi \models \Phi_1 \Rightarrow \Phi_2$ iff $\pi \not\models \Phi_1$ or $\pi \models \Phi_2$
8. $\pi \models X\Phi$ iff $\pi^2 \models \Phi$
9. $\pi \models G\Phi$ iff for all $i \geq 1, \pi^i \models \Phi$
10. $\pi \models F\Phi$ iff there is some $i \geq 1$ such that $\pi^i \models \Phi$

10. $\pi \models \Phi\ U\ \Psi$ iff there is some $i \geq 1$ such that $\pi^i \models \Psi$ and for all $j = 1, \ldots, i-1$ we have $\pi^j \models \Phi$
11. $\pi \models \Phi\ W\ \Psi$ iff either there is some $i \geq 1$ such that $\pi^i \models \Psi$ and for all $j = 1, \ldots, i-1$ we have $\pi^j \models \Phi$; or for all $k \geq 1$ we have $\pi^k \models \Phi$
12. $\pi \models \Phi\ R\ \Psi$ iff either there is some $i \geq 1$ such that $\pi^i \models \Phi$ and for all $j = 1, \ldots, i$ we have $\pi^j \models \Psi$' or for all $k \geq 1$ we have $\pi^k \models \Psi$

- The meaning of $\Phi\ U\ \Psi$ is similar to that in CTL, i.e.



each of the states from s_3 to s_9 satisfies $\Phi\ U\ \Psi$

- Weak-until is just like U, except that $\phi\ W\ \Psi$ does not require that Ψ is eventually satisfied along the path in question, which is required by by $\Phi\ U\ \Psi$.
- Release R is dual to U; that is $\Phi\ R\ \Psi$ is equivalent to $\neg(\neg\Phi\ U\ \neg\Psi)$

Practical Patterns of LTL Specifications (1)

What kind of practically relevant properties can we check with formulas of LTL?
Suppose atomic descriptions include some words as busy, requested, ready, etc.

- It is impossible to get a state where started holds but ready does not hold:
  $$G\neg(started \land \neg ready)$$
- For any state, if a request (of some resource) occurs, then it will eventually be acknowledged:
  $$G(requested \Rightarrow F\ acknowledged)$$
- A certain process is enabled infinitely often on every computation path:
  $$GF\ enabled$$
- On all path, a certain process will eventually be permanently deadlocked:
  $$FG\ deadlock$$

- If the process is enabled infinitely often, then it runs infinitely often

$$GF\ enabled \Rightarrow GF\ running$$

- An upwards traveling elevator at the second floor does not change its direction when it has passengers wishing to go to the fifth floor:

$$G(floor = 2 \wedge direction = up \wedge ButtonPressed5 \Rightarrow (direction = up\ U\ floor = 5))$$

Practical Patterns of LTL Specifications (3)

- Train doors shall always remain closed between platforms unless the train is stopped in emergency.

  We cannot specify this statement in LTL, as it should start with $\forall tr : Train, pl : Platform \ldots$ and we do not have quantifiers $\forall$ and $\exists$ in neither LTL nor CTL!

- For train $tr75$, its doors shall always remain closed between platforms $pl2$ and $pl3$ (i.e. next platform) unless the train is stopped in emergency.

$$
\begin{aligned}
& G(tr75.at.pl2 \wedge \neg tr75.at.pl3 \implies G(tr75.doors = \text{'closed'}) \\
& \quad \vee \; tr75.doors = \text{'closed'} \; U \; tr75.at.pl3 \\
& \quad \vee \; (Alarm.tr75 \wedge \neg tr75.moving))
\end{aligned}
$$

Impossible LTL Specifications

There are some things which are **not** possible to say in LTL, however. One big class of such things are statements which assert the existence of a path, such as these ones:

- For any state it is *possible* to get a **restart** state (i.e., there is a path from all states to a state satisfying **restart**).
- The lift *can* remain idle on the third floor with its doors closed (i.e., from the state in which it is on the third floor, there is a path along it stays there).

LTL cannot express these because it cannot directly assert the existence of path. CTL has operators for quantifying over paths, and **can** express these properties.

Equivalences

- $\neg G\ \Phi \equiv F\ \neg\Phi \qquad \neg F\ \Phi \equiv G\ \neg\Phi \qquad \neg X\ \Phi \equiv X\ \neg\Phi$
- $\neg(\Phi\ U\ \Psi) \equiv \neg\Phi\ R\ \neg\Psi \qquad \neg(\Phi\ R\ \Psi) \equiv \neg\Phi\ U\ \neg\Psi$
- $F\ (\Phi \vee \Psi) \equiv F\ \Phi \vee F\ \Psi$
- $G\ (\Phi \wedge \Psi) \equiv G\ \Phi \wedge G\ \Psi$
- $F\ \Phi \equiv \top\ U\ \Phi \qquad G\ \Phi \equiv \bot\ R\ \Phi$
- $\Phi\ U\ \Psi \equiv \Phi\ W\ \Psi \wedge F\ \Psi$
- $\Phi\ W\ \Psi \equiv \Phi\ U\ \Psi \vee G\ \Phi$
- $\Phi\ W\ \Psi \equiv \Psi\ R\ (\Phi \vee \Psi)$
- $\Phi\ R\ \Psi \equiv \Psi\ W\ (\Phi \wedge \Psi)$

Mutual Exclusion and LTL

Safety: $\Phi_1 \stackrel{def}{=} G\neg(c_1 \wedge c_2)$

Liveness: $\Phi_2 \stackrel{def}{=} G(t_1 \Rightarrow F\ c_1)$

Non-blocking: Let's just consider process 1. We would like to express the property as: for every state satisfying n_1, there is a successor satisfying t_1. Unfortunately, this existence quantifier on paths ('there is a successor satisfying...) cannot be expressed in LTL (it can in CTL).

No strict sequencing: We might consider this as saying: there is a path with two distinct states satisfying c_1 such that no state in between has that property. However, we cannot express 'there exists a path', so let us consider the complement formula instead. The complement says that all path having a c_1 period that ends cannot have a further c_1 until a c_2 state occurs, i.e.

$\Psi_3 \stackrel{def}{=} G(c_1 \Rightarrow c_1\ W(\neg c_1 \wedge \neg c_1\ W\ c_2))$. We have to show that Ψ_3 does not hold!

The analysis is very similar to that of CTL, liveness does not hold for the first solution but it does for the second one.

CTL vs LTL

- It is possible to get a state where started holds but ready does not hold:
  CTL: $EF(started \wedge \neg ready)$
  LTL: $G\neg(started \wedge \neg ready)$
- For any state, if a request (of some resource) occurs, then it will eventually be acknowledged:
  CTL: $AG(requested \Rightarrow AF\ acknowledged)$
  LTL: $G(requested \Rightarrow F\ acknowledged)$
- A certain process is enabled infinitely often on every computation path:
  CTL: $AG(AF\ enabled)$
  LTL: $GF\ enabled$
- Whatever happens, a certain process will eventually be permanently deadlocked:
  CTL: $AF(AG\ deadlock)$
  LTL: $FG\ deadlock$

CTL* Logic

It allows nested modalities and boolean connectives before applying the path quantifiers E and A.

- $A[(p\ U\ r) \vee (q\ U\ r)]$: along all paths, either p is true until r, or q is true until r.
  $\not\equiv A[(p \vee q)\ U\ r]$
  It can be expressed in CTL, but it is not easy.
- $A[X\ p \vee X\ X\ p]$: along all paths, p is true in the next state, or the next but one.
  $\not\equiv AX\ p \vee AXAX\ p$
  It **cannot** be expressed in CTL.
- $E[GF\ p]$: there is a path along which p is infinitely often true.
  $\not\equiv EGEF\ p$
  It **cannot** be expressed in CTL.

The syntax of CTL* involves two classes of formulas:

- **state formulas**, which are evaluated in states:

  $$\Phi ::= \bot \mid \top \mid p \mid (\neg\Phi) \mid (\Phi \wedge \Phi) \mid (\Phi \vee \Phi) \mid (\Phi \Rightarrow \Phi) \mid A[\alpha] \mid E[\alpha]$$

  where p is any atomic formula and α is any path formula.

- **path formulas**, which are evaluated along paths:

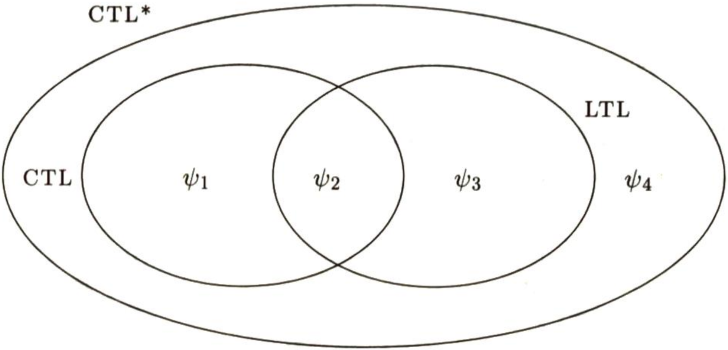
  $$\alpha ::= \Phi \mid (\neg\alpha) \mid (\alpha \wedge \alpha) \mid (\alpha \vee \alpha) \mid (\alpha \Rightarrow \alpha) \mid (\alpha\ U\ \alpha) \mid (G\ \alpha) \mid (F\ \alpha) \mid (X\ \alpha)$$

  where Φ is any state formula.

LTL, CTL vs CTL*

- LTL is a subset of CTL*.
  Although the syntax of LTL does not include A, E, the semantic viewpoint of LTL is that we consider all path. Therefore, the LTL formula α is equivalent to the CTL* formula $A[\alpha]$.
- CTL is a subset of CTL*.
  CTL is a fragment of CTL* in which we restrict the form of path formulas to:
  $$\alpha ::= (\Phi\ U\ \Phi) \mid (G\ \Phi) \mid (F\ \Phi) \mid (X\ \Phi)$$

LTL, CTL vs CTL*



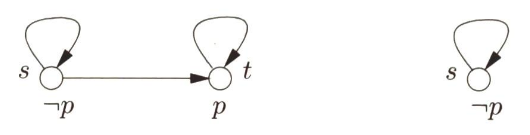
In CTL but not in LTL

$\Psi_1 \stackrel{def}{=} AGEF\ p$

Wherever we got to, we can always get back to a state in which p is true. Useful in finding deadlock in protocols.

Proof.

Let Φ be an LTL formula such that $A[\Phi]$ is allegedly equivalent to $AGEF\ p$.



Since $\mathcal{M}, s \models AGEF\ p$, we have $\mathcal{M}, s \models A[\Phi]$. The paths from s in $\mathcal{M}'$ are a subset of those from s in $\mathcal{M}$, so we have $\mathcal{M}', s \models A[\Phi]$. Yet, it is **not** the case that $\mathcal{M}', s \models AGEF\ p$, a contradiction. □

In CTL and in LTL

$\Psi_2 \stackrel{def}{=} AG(p \Rightarrow AF\ q)$ in CTL

$\Psi_2 \stackrel{def}{=} G(p \Rightarrow F\ q)$ in LTL

any p is eventually followed by a q.

In LTL but not in CTL

$\Psi_3 \stackrel{def}{=} A[GF\ p \Rightarrow F\ q]$ in CTL*

$\Psi_3 \stackrel{def}{=} GF\ p \Rightarrow F\ q$ in LTL

there are infinitely may p along the path, then there is an occurrence of q.

Application: many fairness constraints are of the form "infinitely often requested implies eventually acknowledged"

In CTL* but neither in CTL nor in LTL

$\Psi_4 \stackrel{def}{=} E[GF\ p]$

there is a path with infinitely many p.

Weak-until in CTL: Motivation

$s \models A[p \ U \ q] \iff$ along all paths from s, q is true somewhere along the path and p is true from the present state until the state win which q is true.

$\Downarrow$

a path in which p is permanently true and q never true does not satisfy $p \ U \ q$.

- Sometimes our intuition about "until" suggest that we should accept paths in which q never holds, provided p is permanently true.
- The indicator light stays on until the elevator arrives.
- If elevator never arrives, the light stays on permanently and this is OK.

Weak-until in CTL, LTL and CTL*

- In LTL and CTL*:

$$p\ W\ q \equiv (p\ U\ q) \vee G\ p$$

- In CTL:

$$E[p\ W\ q] \equiv E[p\ U\ q] \vee EG\ p$$
$$A[p\ W\ q] \equiv A[p\ U\ q] \vee AG\ p$$

Linear Temporal Logic Again

Linear Temporal Logic (LTL): Intuitions

- Consider the simple Linear Temporal Logic (LTL) where the accessibility relation is isomorphic to the Natural Numbers.
- Typical temporal operators used are:
  - $\bigcirc\varphi$ - φ is true in the **next** moment in time (*Next*)
  - $\Box\varphi$ - φ is true in **all** future moments (*Always*)
  - $\Diamond\varphi$ - φ is true in **some** future moments (*Sometimes*)
  - $\varphi \mathbf{U} \psi$ - φ is true **until** ψ is true (*Until*)
- Other operators are: $\wedge, \vee, \neg, \implies, \iff$, i.e. *propositional operators* with standard semantics
- **Examples:**

$$\Box((\neg passport \vee \neg ticket) \implies \bigcirc \neg board\_flight)$$

$$\Box(requested \implies \Diamond received)$$

$$\Box(received \implies \bigcirc processed)$$

$$\Box(processed \implies \Diamond\Box done)$$

From the above we should be able to infer that it is not the case that the system continually re-sends a request, but never sees it completed ($\Box\neg done$); i.e. the statement

$$\Box requested \wedge \Box\neg done$$

Temporal Semantics

- A system history in LTL is an infinite temporal sequence of system states.
- Time is isomorphic to the set Nat of natural numbers, and a history H is defined as a function:

  $$H : Nat \rightarrow States$$
- The function H assigns to every time point i in Nat, the system state at that time point $H(i)$.
- To define the LTL semantics more precisely, we write

  $$(H, i) \models \varphi$$

  to express that the LTL formula φ is satisfied by history H at time i.

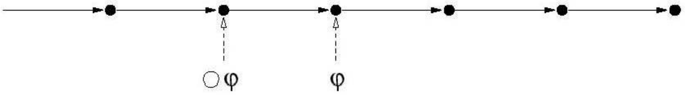
Semantics: Atomic proposition

- In LTL we have only boolean values, and boolean variables are interpreted as *atomic propositions*. Each state $s \in States$ has a set of atomic propositions assigned to it. The set of all atomic proposition is often denoted by Σ (*alphabet*).
- We do **not** have any other types in LTL, we do not have natural numbers, only boolean values.
- We have to simulate numbers by boolean variables. This can be done for finite sets of numbers. For example if $x \in \{1,2,3,4,5\}$, we can simulate by **five** boolean variables x_is_1, x_is_2, x_is_3, x_is_4 and x_is_5.
- **We define that an atomic proposition p is true at a time point "i", as follows:**

  $(H,i) \models p$ **iff** p is assigned to the state $H(i)$

$$(H,i) \models \bigcirc\varphi \textbf{ iff } (H,i+1) \models \varphi$$

- This operator provides a constraint on the next moment in time.



- **Examples:**

$$(sad \wedge \neg rich) \implies \bigcirc sad$$
$$((x=0) \wedge add3) \implies \bigcirc(x=3)$$

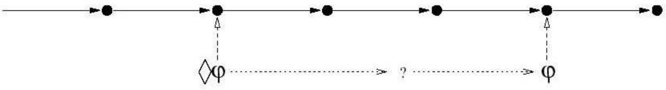
In the above '$x=0$', $x=3$' and '$add3$' are boolean variables (*atomic propositions*)

Temporal Operators: ‘sometime’

$$(H,i) \models \Diamond\varphi \textbf{ iff } \exists j. j \geq i \wedge (H,j) \models \varphi$$

$$(H,i) \models \Diamond\varphi \textbf{ iff } \text{for some } j \geq i : (H,j) \models \varphi$$

- While we can be sure that φ *will* be true either now or in the future, we can not be sure exactly *when* it will be true.



- **Examples:**

  $(\neg resigned \wedge sad) \implies \Diamond famous$

  $sad \implies \Diamond happy$

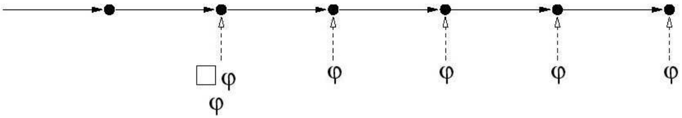
  $send \implies \Diamond receive$

Temporal Operators: 'always'

$$(H, i) \models \Box\varphi \textbf{ iff } \forall j . j \geq i \wedge (H, j) \models \varphi$$

$$(H, i) \models \Box\varphi \textbf{ iff } \text{for all } j \geq i : (H, j) \models \varphi$$

- This can represent invariant properties.



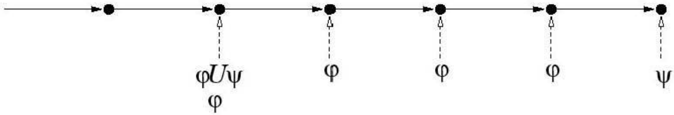
- **Examples:**

  $\mathit{lottery\text{-}win} \implies \Box \mathit{rich}$

Temporal Operators: ‘until’

$(H, i) \models \varphi \mathbf{U} \psi$ **iff** $(\exists j . j \geq i \wedge (H, j) \models \psi) \wedge$
$(\forall k . i \leq k < j \implies (H, k) \models \varphi)$

$(H, i) \models \varphi \mathbf{U} \psi$ **iff** there exists$j \geq i$ such that $(H, j) \models \psi$, and
for every $k, i \leq k < j \implies (H, k) \models \varphi$



- **Examples:**

$$start\_lecture \implies talk\mathbf{U}end\_lecture$$
$$born \implies alive\mathbf{U}dead$$
$$request \implies reply\mathbf{U}acknowledgment$$

Equivalences in LTL

- $\neg\Box\varphi = \Diamond\neg\varphi$
- $\Diamond\varphi = \mathit{true}\mathbf{U}\varphi$
- $\Diamond(\varphi \vee \psi) \equiv \Diamond\varphi \vee \Diamond\psi$
- $\Box(\varphi \wedge \psi) \equiv \Box\varphi \wedge \Box\psi$
- $\neg \bigcirc \varphi \equiv \bigcirc\neg\varphi$
- $\neg(\varphi\mathbf{U}\psi) \equiv (\neg\psi\mathbf{U}(\neg\varphi \wedge \neg\psi)) \vee \Box\neg\psi$

Temporal Logic in Computer Science

- Temporal logic was originally developed in order to represent tense in natural language.
- Within Computer Science, it has achieved a significant role in the formal specification and verification of concurrent reactive systems.
- Much of this popularity has been achieved as a number of useful concepts can be formally, and concisely, specified using temporal logics, e.g.
  - safety properties
  - liveness properties
  - fairness properties
- Temporal logic allows to use very powerful *model checking* tools as for example SPIN (for LTL).

Safety Properties

- Safety: "something bad will not happen"
- Typical examples:

  $\Box\neg(reactor\_temp > 1000)$

  $\Box\neg((x = 0) \wedge \bigcirc\bigcirc\bigcirc(y = z/x))$

  In the above '$x = 0$' and '$y = z/x$' are boolean variables (*atomic propositions*)

  and so on...
- Usually: $\Box\neg \ldots$.

Liveness Properties

- Liveness: "something good will happen"
- Typical examples:

  $\Diamond rich$

  $\Diamond(x > 5)$

  $\Box(start \implies \Diamond terminate)$

  $\Box(Trying \implies \Diamond Critical$

  and so on...
- Usually: $\Diamond \ldots$

Fairness Properties

- Often only really useful when scheduling processes, responding to messages, etc.
- Strong Fairness: “if something is attempted/requested infinitely often, then it will be successful/allocated infinitely often”
- Typical example:

$$\Box\Diamond \mathit{ready} \implies \Box\Diamond \mathit{run}$$

Sample Statements

- An upwards traveling elevator at the second floor does not change its direction when it has passengers wishing to go to the fifth floor:

$$\Box(\mathit{floor} = 2 \wedge \mathit{direction} = \mathit{up} \wedge \mathit{ButtonPressed5} \Rightarrow (\mathit{direction} = \mathit{up}\ \mathbf{U}\ \mathit{floor} = 5))$$

- A certain process is enabled infinitely often on every computation path:

$$\Box\Diamond\ \mathit{enabled}$$

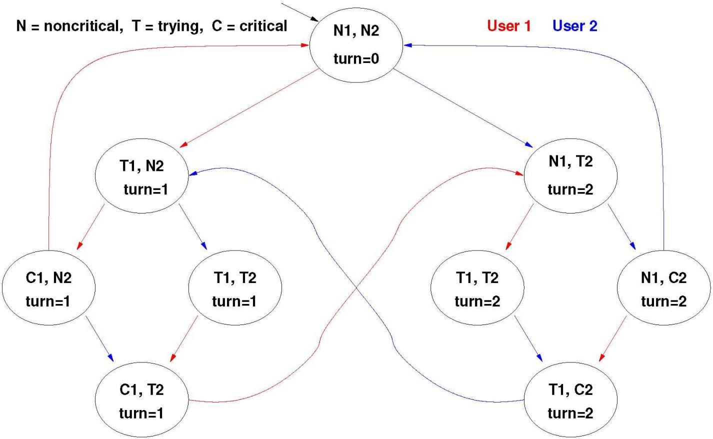
- On all path, a certain process will eventually be permanently deadlocked:

$$\Diamond\Box\ \mathit{deadlock}$$

LTL: How it works

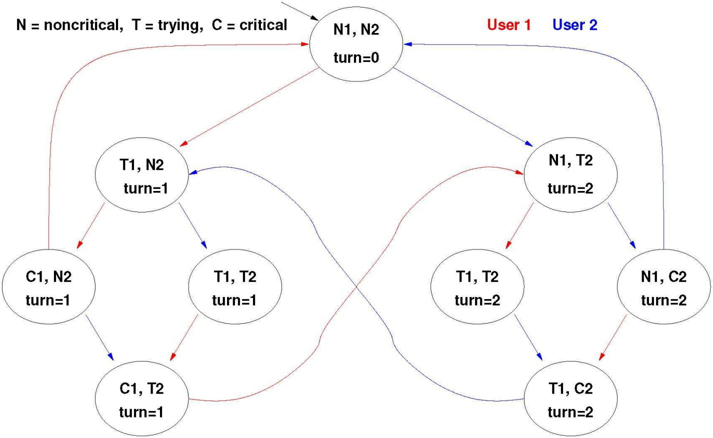
- Full modeling with temporal logic consists of two steps:
  1. provide a temporal formula ϕ that describes desired properties
  2. provide a temporal model of the system $\mathcal{M}$ and show that ϕ satisfies it, i.e. prove $\mathcal{M} \models \phi$
- For requirements, we usually stop with providing a temporal formula ϕ that describes desired properties
- There are many software supports for Linear Temporal Logic.
- The most known are:
  1. SPIN that allows us to verify if a given LTL formula is satisfied is a given model
  2. a model is a kind of finite state automaton, called Kripke Structure and can be defined using the language PROMELA
- Some examples will follow

Example 1: mutual exclusion (safety)



$$\mathcal{KM} \models \Box\neg(C_1 \wedge C_2)\ ?$$

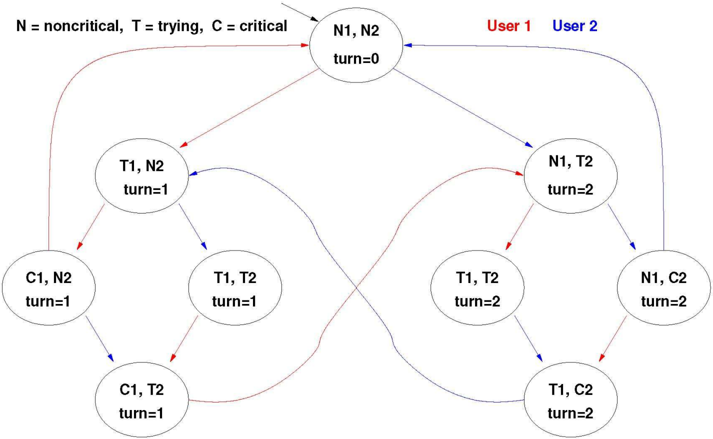
Example 1: mutual exclusion (safety)



$$\mathcal{KM} \models \Box\neg(C_1 \wedge C_2) \text{ ?}$$

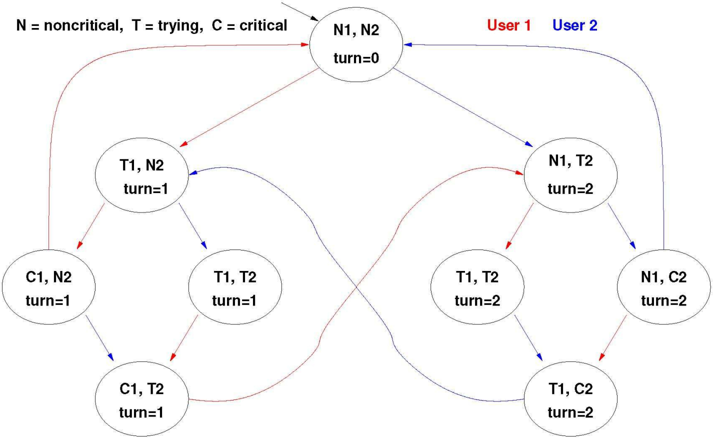
YES: There is no reachable state in which $(C_1 \wedge C_2)$ holds!

Example 2: mutual exclusion (liveness)



$$\mathcal{KM} \models \Diamond C_1 \ ?$$

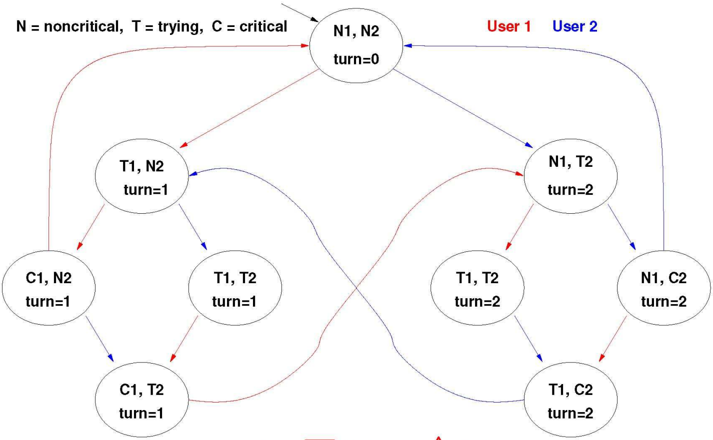
Example 2: mutual exclusion (liveness)



$$\mathcal{KM} \models \Diamond C_1 \ ?$$

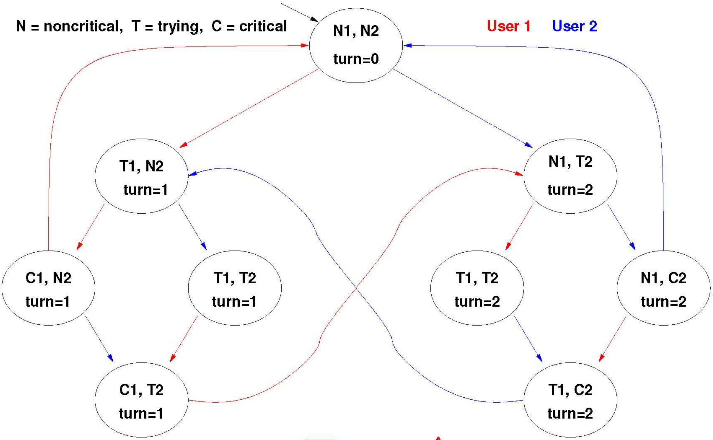
NO: the blue cyclic path is a counterexample!

Example 3: mutual exclusion (liveness)



$$\mathcal{KM} \models \Box(T_1 \Rightarrow \Diamond C_1) \ ?$$

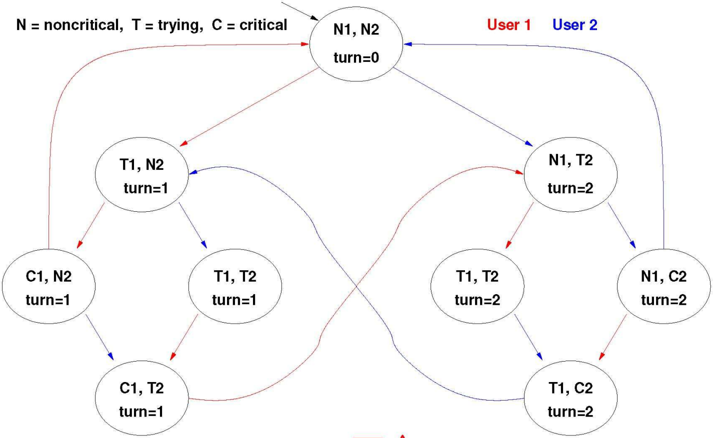
Example 3: mutual exclusion (liveness)



$$\mathcal{KM} \models \Box(T_1 \Rightarrow \Diamond C_1) \ ?$$

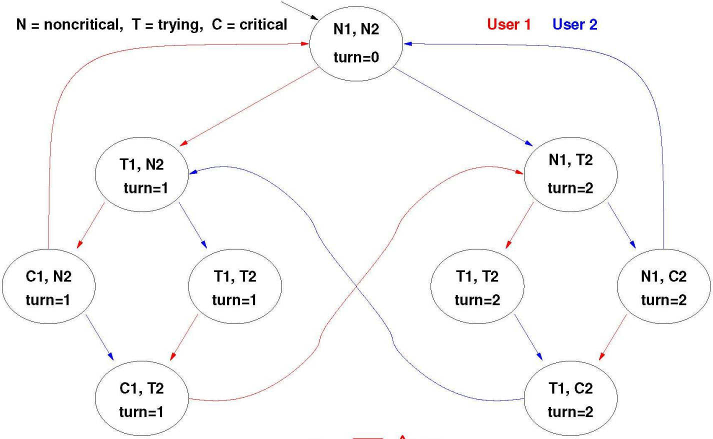
YES: in every path if T_1 holds afterwards C_1 holds!

Example 4: mutual exclusion (fairness)



$$\mathcal{KM} \models \Box \Diamond C_1 \ ?$$

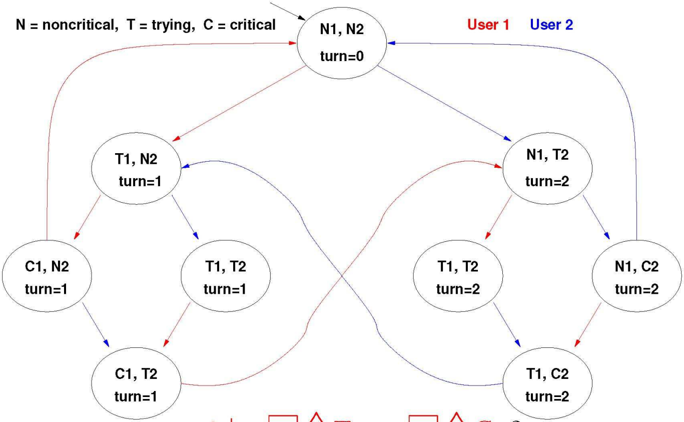
Example 4: mutual exclusion (fairness)



$$\mathcal{KM} \models \Box\Diamond C_1 \; ?$$

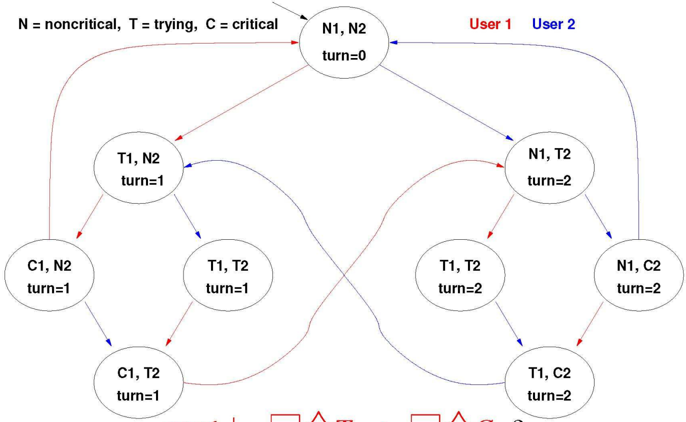
NO: the blue cyclic path is a counterexample!

Example 4: mutual exclusion (strong fairness)



$$\mathcal{KM} \models \Box\Diamond T_1 \Rightarrow \Box\Diamond C_1 \ ?$$

Example 4: mutual exclusion (strong fairness)



$$\mathcal{KM} \models \Box\Diamond T_1 \Rightarrow \Box\Diamond C_1 \text{ ?}$$

YES: every path which visits T_1 infinitely often also visits C_1 infinitely often!

LTL: Final Comments

- LTL was designed as a tool for proving properties of huge systems, in range of millions of states.
- Efficient using of LTL requires some good software support as for instance SPIN, while classical predicate calculus is mainly used by human beings.
- As oppose to predicate calculus that is useful for both describing and proving properties of systems, LTL is usually used for proving properties.

“Finite/Experimental” Induction

- How the induction method works in research?
- One analysis a problem, solves several concrete cases, and then on the basis of knowledge, intuition and plain gut feelings, some formula is proposed.
- Then the induction can be used to prove that a given formula works in all cases.
- How finding new laws works in physics, chemistry, biology etc.?
- On the basis of some past experiments, theoretical analysis, intuition and just plain gut feelings a new law is formulated.
- A serious of focused experiments is conducted.
- If all experiments confirm (up to some experiments errors) a proposed law, the law is considered true.

- You have most likely already heard about the problem of 'proving program properties', it was probably discussed on 'formal methods' and/or 'advanced discrete mathematics' courses. One popular apprach is based on *Hoare Logic*.
- In some cases such method can be fully automatized.
- But for many important cases, it cannot, at least right now.
- Sorting property *SP* for a given array $A[1..n]$ can be defined as $SP: \ \forall i,j \in \mathbb{N}.\ i < j \implies A[i] \leq A[j]$.
- Let *Sort* be a sorting procedure written in some appropriate language/precise pseudo-code.
- A 'dream' solution is to a system/program that takes *Sort* and *SP* as inputs, a user click a button and, after some maybe long time, the system answers 'YES' or 'NO', and provides some explanation (trace or reasoning) in the second case.
- Unfortunately, to my knowledge, such a system/program does not exists yet.

- Define $SP(n)$ as $SP(n): \ \forall i,j \leq n.\ i < j \implies A[i] \leq A[j]$, for example $SP(73): \ \forall i,j \leq 73.\ i < j \implies A[i] \leq A[j]$.
- **There are** several systems that can deal with the input *Sort* and $SP(n)$, for any *concrete* n, say $SP(73)$.
- Note that for example $SP(4) \equiv A[1] \leq A[2] \leq A[3] \leq A[4]$, so the problem is reduced to *propositional logic* (with presumable long formulas).
- Suppose that you run *Sort* with $SP(100), SP(150), SP(200), \ldots, SP(1000)$ and in all cases the answer was 'YES'.
- Can you conclude that *Sort* is correct, so it will work for any n?

- Consider model checking and the mutual exclusion problem.
- The case for $n = 2$ was analysed in class (textbook), suppose you have implemented the solution (for example, using popular system SPIN) for any concrete n.
- Suppose the solution was correct for $n = 10, 20, \ldots, 100$.
- Can you conclude that the solution works for any n?
- In model checking we usually start with the simplest case and then extend the model for more complex cases, until the computational capacity of a platform is reached.
- I call this approach *finite* or *experimental induction*.