### What is Temporal Logic?

- A logical formalism to describe *sequences* of any kind. We use it to describe state sequences.
- An automaton describes the actions of a system, a temporal logic formula describes some property of the set of sequences.

### The Basic Paradigm

```
System = Automaton
Specification = Temporal Logic Formula
Verification = Satisfaction
```

### The Idea of Temporal Logic

- Truth *changes* as time passes.
- We have to say *when* things are true, not just what is true.
- The logic is chosen to be as simple as possible while yet being rich enough to capture properties of interest.

### Formulas

- $\varphi$
- $\Diamond \varphi$
- $\Box \varphi$
- $\varphi \lor \psi$
- $\varphi \land \psi$
- $\varphi \Rightarrow \psi$
- True, False
- Basic formulas
- $\varphi$ and $\psi$
- $\varphi$ or $\psi$
- not $\varphi$
- $\varphi \Rightarrow \psi$
Semantics of Temporal Logic

What does it all mean?

The Temporal Semantics

- means always
- $\Diamond$ means sometime or eventually
- $O$ means next time or next step
- $U$ means until

More Formal Semantics I

Let $S$ be a sequence of states
$S[0]$ is the first state, $S[j]$ is the $j$th state
$S[j..]$ is the sequence starting from the $j$th state
$S[j..k]$ is the sequence from $j$ to $k$.

More Formal Semantics II

- $S$ sat $\varphi$ means that the sequence $S$ satisfies the formula $\varphi$
- $S$ sat True, for any $S$
- $S$ sat $\varphi$ and $\psi$ if $S$ sat $\varphi$ and $S$ sat $\psi$
- $S$ sat BF if $S[0]$ sat BF
More Formal Semantics III

- \( S \text{ sat } \varphi \) if for any \( j \), \( S[j..] \text{ sat } \varphi \)
- \( S \text{ sat } \Diamond \varphi \) if for some \( j \), \( S[j..] \text{ sat } \varphi \)
- \( S \text{ sat } \Omega \varphi \) if \( S[1..] \text{ sat } \varphi \)
- \( S \text{ sat } \varphi \cup \psi \) if for some \( j \), \( S[j..] \text{ sat } \psi \) and for any \( k < j \), \( S[k..] \text{ sat } \varphi \)

Examples I

- This machine produces the state sequence: abcabcabcabc…
- This sequence satisfies \( \square(a \Rightarrow O b) \) and \( \square(a \Rightarrow O(\Diamond a)) \)

Examples II

A simple set-reset device. It satisfies:
- \((\text{set} \Rightarrow (\Diamond \text{reset}))\)
- and also
- \((\text{error} \Rightarrow *)\)

Computation Tree Logic

Most interesting programs are not deterministic. They cannot be represented by a sequence of possible states or transitions. There has to be a tree of possible computations.

There is a logic called computation tree logic (CTL for short) which expresses this.
CTL Syntax I

- True, false
- Basic formulas p
- not φ, φ and ψ, φ or ψ, φ ⇒ ψ
- AX φ, EX φ
- AF φ, EF φ

CTL Syntax II

- AG φ, EG φ
- A[φ U ψ], E [φ U ψ]

All temporal operators come in pairs:
- A means “all paths” (inevitably)
- E means “on some path” (possibly)

CTL Syntax III

- F is the same as ◊
- G is the same as □
- X is the same as O
- U is the same as U!

Formula Patterns

- Bad thing never happens (safety):
  AG(not bad-thing)
- Good thing eventually happens (liveness):
  AF good-thing
- Bad thing could happen: EF bad-thing
A System satisfying $\text{EF}\varphi$

A System Satisfying $\text{EG}\varphi$

A System Satisfying $\text{AG}\varphi$

A System Satisfying $\text{AF}\varphi$
Example Specification Patterns

- It is possible to get to a state where *started* holds but *ready* does not hold:
  \( \text{EF(started \text{ and } \neg \text{ready})} \)
- If a request occurs then it will eventually be acknowledged:
  \( \text{AG(requested } \Rightarrow \text{AF acknowledged}) \)
- Process is enabled infinitely often on every path: \( \text{AG(AF enabled)} \)

Example Specification Patterns II

- Whatever happens, a certain process will eventually be permanently deadlocked:
  \( \text{AF(AG deadlocked)} \)
- From any state it is possible to get to a restart state: \( \text{AG(AF restart)} \)

Elevator Specification

- An upwards travelling elevator at the second floor does not change its direction when it has passengers wishing to go to the fifth floor:
  \( \text{AG(floor=2 \text{ and direction}=up \text{ and ButtonPressed5 } \Rightarrow \text{A[direction}=up \text{ U floor}= 5])} \)

Timed CTL

CTL and Temporal Logics are adequate for talking about concurrent programs but we need to talk about *real-time systems*. We do this by introducing *explicit timing constraints* into the logic. We follow the same paradigm:

Specification = TCTL formulas

System = Timed Automaton
TCTL Formulas

- Basic formulas: \( p \)
- \textbf{true}, \textbf{false}
- implication: \( \varphi \Rightarrow \psi \)
- some paths: \( \text{EF\{}=,<,>\}\{p} \)
- all paths: \( \text{AF\{}=,<,>\}\{p} \)

The Meaning of TCTL

- Now instead of state sequences we suppose that we have a \textit{function} from \( \mathbb{R} \) (time) to States.
- When we say \( \text{AF\{}=50\}\{p \) we mean that along every path there is a point \textit{50 time units from now} such that \( p \) holds.
- \( \text{AF\{}<= 50\}\{p \) means \textit{within} 50 time units \( p \) holds on all paths.

TCTL Semantics

- \( \text{AG\{}=50\}\{p \) means the same as \( \text{AF\{}=50\}\{p \)
- \( \text{AF\{<50\}\{p \) means that on all paths at \textit{some time} before 50 time units \( p \) holds, while \( \text{AG\{<50\}\{p \) means on all paths \( p \) \textit{always} holds until 50 time units have elapsed.
- Similarly we can put any relational combination in \{\} after any of AG, AF, EG, EF.

TCTL Remarks

We also have timed \textit{until} and many other operators but not all of them are allowed by the various systems such as Kronos and XUPPAAL.
Surprisingly --even in dense real-time-- model-checking is decidable though satisfiability is very undecidable.