

# Formal Specification and Verification

Temporal logic, part 2

19.06.2012

Viorica Sofronie-Stokkermans  
e-mail: [sofronie@uni-koblenz.de](mailto:sofronie@uni-koblenz.de)

# Temporal logic

---

Which flow of time should we use?

This depends on the application!

The main application of TL in computer science is the verification of finite-state reactive and concurrent systems.

A state is a snapshot of the system capturing the values of the variables at an instant of time.

- **Finite-state systems.**

Finite-state systems can only take finitely many states.

(Often, infinite-state systems can be abstracted into finite-state ones by grouping the states into a finite number of partitions.)

# Models of time

---

Which flow of time should we use?

This depends on the application!

The main application of TL in computer science is the verification of finite-state reactive and concurrent systems.

A state is a snapshot of the system capturing the values of the variables at an instant of time.

- **Reactive Systems.**

A reactive system interacts with the environment frequently and usually does not terminate. Its correctness is defined via these interactions.

This is in contrast to a classical algorithm that takes an input initially and then eventually terminates producing a result.

# Models of time

---

Which flow of time should we use?

This depends on the application!

The main application of TL in computer science is the verification of finite-state reactive and concurrent systems.

A state is a snapshot of the system capturing the values of the variables at an instant of time.

- **Concurrent Systems.**

Systems consisting of multiple, interacting processes. One process does not know about the internal state of the others. May be viewed as a collection of reactive systems.

# Linear Time Logic

---

## Syntax

$\Pi$  set of propositional variables.

The set of LTL (linear time logic) formulae is the smallest set such that:

- each propositional letter  $P \in \Pi$  is a formula;
- if  $F, G$  are formulae, then so are  $F \wedge G, F \vee G, \neg F$ ;
- if  $F, G$  are formulae, then so are  $\bigcirc F$  and  $F \mathcal{U} G$

**Remark:** Instead of  $\bigcirc F$  in some books also  $XF$  is used.

# Linear Time Logic

---

## Semantics

- **Transition systems**  $(S, \rightarrow, L)$   
(with the property that for every  $s \in S$  there exists  $s' \in S$  with  $s \rightarrow s'$   
i.e. no state of the system can “deadlock”<sup>a</sup>)

Transition systems are also simply called **models** in what follows.

- **Computation (execution, path)** in a model  $(S, \rightarrow, L)$   
infinite sequence of states  $\pi = s_0, s_1, s_2, \dots$  in  $S$  such that for each  
 $i \geq 0$ ,  $s_i \rightarrow s_{i+1}$ .  
We write the path as  $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \dots$

---

<sup>a</sup>This is a technical convenience, and in fact it does not represent any real restriction on the systems we can model. If a system did deadlock, we could always add an extra state  $s_d$  representing deadlock, together with new transitions  $s \rightarrow s_d$  for each  $s$  which was a deadlock in the old system, as well as  $s_d \rightarrow s_d$ .

# Linear Time Logic

---

## Semantics

Let  $TS = (S, \rightarrow, L)$  be a model and  $\pi = s_0 \rightarrow \dots$  be a path in  $TS$  ( $\pi$  represents a possible future of our system)

Whether  $\pi$  satisfies an LTL formula is defined by the satisfaction relation  $\models$  as follows:

- $\pi \models \top$
- $\pi \not\models \perp$
- $\pi \models p$  iff  $p \in L(s_0)$ ,      if  $p \in \Pi$
- $\pi \models \neg F$  iff  $\pi \not\models F$
- $\pi \models F \wedge G$  iff  $\pi \models F$  and  $\pi \models G$
- $\pi \models F \vee G$  iff  $\pi \models F$  or  $\pi \models G$
- $\pi \models \bigcirc F$  iff  $\pi^1 \models F$
- $\pi \models F \mathcal{U} G$  iff  $\exists m \geq 0$  s.t.  $\pi^m \models G$  and  $\forall k \in \{0, \dots, m-1\} : \pi^k \models F$

# Linear Time Logic

---

## Alternative way of defining the semantics:

An LTL structure  $M$  is an infinite sequence  $S_0 S_1 \dots$  with  $S_i \subseteq \Pi$  for all  $i \geq 0$ . We define satisfaction of LTL formulas in  $M$  at time points  $n \in \mathbb{N}$  as follows:

- $M, n \models p$  iff  $p \in S_n$ , if  $p \in \Pi$
- $M, n \models F \wedge G$  iff  $M, n \models F$  and  $M, n \models G$
- $M, n \models F \vee G$  iff  $M, n \models F$  or  $M, n \models G$
- $M, n \models \neg F$  iff  $M, n \not\models F$
- $M, n \models \bigcirc F$  iff  $M, n + 1 \models F$
- $M, n \models F \mathcal{U} G$  iff  $\exists m \geq n$  s.t.  $M, m \models G$  and  
 $\forall k \in \{n, \dots, m - 1\} : M, k \models F$

Note that the time flow  $(\mathbb{N}, <)$  is implicit.



# Abbreviations

---

- The future diamond

$$\Diamond\phi := \top\mathcal{U}\phi$$

Sometimes denoted also  $F\phi$

$$\pi \models \Diamond\phi \text{ iff } \exists m \geq 0 : \pi^m \models \phi$$

$$M, n \models \Diamond\phi \text{ iff } \exists m \geq n : M, m \models \phi$$

- The future box

$$\Box\phi := \neg\Diamond\neg\phi$$

Sometimes also denoted  $G\phi$

$$\pi \models \Box\phi \text{ iff } \forall m \geq 0 : \pi^m \models \phi$$

$$M, n \models \Box\phi \text{ iff } \forall m \geq n : M, m \models \phi$$

- The infinitely often operator

$$\Diamond^\infty\phi := \Box\Diamond\phi$$

$$\pi \models \Diamond^\infty\phi \text{ iff } \{m \geq 0 \mid \pi^m \models \phi\} \text{ is infinite}$$

$$M, n \models \Diamond^\infty\phi \text{ iff } \{m \geq n \mid M, m \models \phi\} \text{ is infinite}$$

- The almost everywhere operator

$$\Box^\infty\phi := \Diamond\Box\phi$$

$$\pi \models \Box^\infty\phi \text{ iff } \{m \geq 0 \mid \pi^m \not\models \phi\} \text{ is finite.}$$

$$M, n \models \Box^\infty\phi \text{ iff } \{m \geq n \mid M, m \not\models \phi\} \text{ is finite.}$$

# Abbreviations

---

- The release operator

$$\phi\mathcal{R}\psi := \neg(\neg\phi\mathcal{U}\neg\psi)$$

$$\pi \models \phi\mathcal{R}\psi \text{ iff } (\exists m \geq 0 : \pi^m \models \phi \text{ and } \forall k \leq m : \pi^k \models \psi) \text{ or } (\forall k \geq 0 : \pi^k \models \psi)$$

$$M, n \models \phi\mathcal{R}\psi \text{ iff } (\exists m \geq n : M, m \models \phi \text{ and } \forall k \leq m : M, m \models \psi) \text{ or } (\forall k \geq m : M, k \models \psi)$$

Read as

“ $\psi$  always holds unless released by  $\phi$ ” i.e.,

“ $\psi$  holds permanently up to and including the first point where  $\phi$  holds (such an  $\phi$ -point need not exist at all)”.

# Abbreviations

---

- The strict until operator:

$$FU^< G := \bigcirc(FUG)$$

$$M, n \models FU^< G \text{ iff } \exists m > n : M, m \models G \wedge \forall k \in \{n+1, \dots, m-1\}, M, k \models F$$

The difference between standard and strict until is that strict until requires  $G$  to happen in the strict future and that  $F$  needs not hold true of the current point.

# Another equivalent satisfaction relation

---

**Definition.** Let  $T = (S, \rightarrow, L)$  and  $s \in S$ .

We say that  $T, s \models \phi$  if for every computation  $\pi$  in  $T$  starting at  $s$  we have  $\pi \models \phi$ .

# Equivalence

---

We say that two LTL formulas  $F$  and  $G$  are (globally) equivalent (written  $F \equiv G$ ) if, for all transition systems  $T$  and paths  $\pi$ , we have  $\pi \models F$  iff  $\pi \models G$ .

Note that:

$$\bigcirc F \equiv \perp \vee \mathcal{U}^< F \text{ and}$$

$$F \mathcal{U} G \equiv G \vee (F \wedge \bigcirc(F \mathcal{U}^< G))$$

Thus, an equally expressive version of LTL is obtained by using  $\mathcal{U}^<$  as the only temporal operator.

This cannot be done with the standard until

# Equivalence

---

Some useful equivalences (exercise: prove them):

$$\neg \bigcirc F \equiv \bigcirc \neg F$$

(self-duality of next)

$$\diamond \diamond F \equiv \diamond F$$

(idempotency of diamond)

$$\bigcirc \diamond F \equiv \diamond \bigcirc F$$

(commutation of next with Diamond)

$$\diamond \diamond^\infty F \equiv \diamond^\infty F \equiv \diamond^\infty \diamond F$$

(absorption of diamonds by infinitely often “)

$$F \mathcal{U} G \equiv \neg(\neg F \mathcal{R} \neg G)$$

(until and release are duals)

$$F \mathcal{U} G \equiv G \vee (F \wedge \bigcirc(F \mathcal{U} G))$$

(unfolding of until)

$$F \mathcal{R} G \equiv (F \wedge G) \vee (G \wedge \bigcirc(F \mathcal{R} G))$$

(unfolding of release)

# Satisfiability

---

An LTL formula  $F$  is satisfiable if there exists a transition system  $M$  and a path  $\pi$  such that  $\pi \models F$ .

Such a structure is called a model of  $F$ .

In verification, satisfiability can be used to detect contradictory properties, i.e., properties that are satisfied by no computation of any reactive system.

**Example:** The following property is contradictory (unsatisfiable):

$$p \wedge \Box(p \rightarrow \bigcirc p) \wedge \Diamond \neg p$$

# Satisfiability

---

LTL satisfiability can be decided using automata on infinite words (Büchi automata).



# Model checking

---

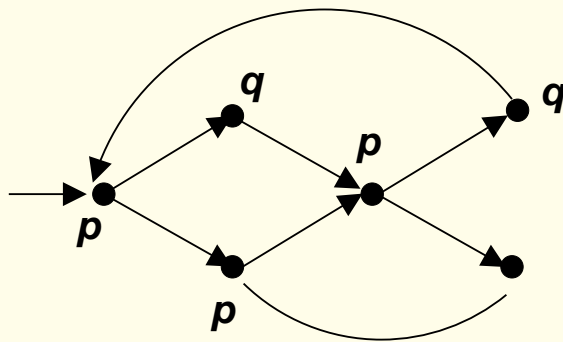
The LTL model checking problem is as follows: given a transition system  $T = (S, \rightarrow, L)$  and an LTL formula  $F$ , check whether  $T \models F$ .

Recall: this is the case if all computations  $\pi$  of  $T$  satisfy  $\pi \models F$ .

## Example:

The following transition system satisfies  $\Box(q \rightarrow \bigcirc \bigcirc \bigcirc p)$ .

It does not satisfy  $\Box(p \rightarrow p\mathcal{U}q)$ .



# Branching Time Logic: CTL

---

When doing model checking, we effectively use LTL in a branching time environment:

Every state in a transition system that has more than a single successor gives rise to a “branching” in time.

This is reflected by the fact that usually, a transition system has more than a single computation.

Branching time logics allow us to explicitly talk about such branches in time.

# CTL: Syntax

---

The class of computational tree logic (CTL) formulas is the smallest set such that

- $\top, \perp$  and each propositional variable  $P \in \Pi$  are formulae;
- if  $F, G$  are formulae, then so are  $F \wedge G, F \vee G, \neg F$ ;
- if  $F, G$  are formulae, then so are  
     $A \bigcirc F$  and  $E \bigcirc F$ ,  
     $A(F \mathcal{U} G)$  and  $E(F \mathcal{U} G)$ .

The symbols  $A$  and  $E$  are called path quantifiers.

# Abbreviations

---

Apart from the Boolean abbreviations, we use:

$A \diamond F$  for  $A(\top \mathcal{U} F)$

$E \diamond F$  for  $E(\top \mathcal{U} F)$

$A \Box F$  for  $\neg E \diamond \neg F$

$E \Box F$  for  $\neg A \diamond \neg F$

Note that formulas such as  $E(\Box q \wedge \Diamond p)$  are not CTL formulas.

# CTL: Semantics

---

Let  $T = (S, \rightarrow, L)$  be a transition system. We define satisfaction of CTL formulas in  $T$  at states  $s \in S$  as follows:

$(T, s) \models p$	iff	$p \in L(s)$
$(T, s) \models \neg F$	iff	$(T, s) \models F$ is not the case
$(T, s) \models F \wedge G$	iff	$(T, s) \models F$ and $(T, s) \models G$
$(T, s) \models F \vee G$	iff	$(T, s) \models F$ or $(T, s) \models G$
$(T, s) \models E \bigcirc F$	iff	$(T, t) \models F$ for some $t \in S$ with $s \rightarrow t$
$(T, s) \models A \bigcirc F$	iff	$(T, t) \models F$ for all $t \in S$ with $s \rightarrow t$
$(T, s) \models A(FUG)$	iff	for all computations $\pi = s_0 s_1 \dots$ of $T$ with $s_0 = s$ , there is an $m \geq 0$ such that $(T, s_m) \models G$ and $(T, s_k) \models F$ for all $k < m$
$(T, s) \models E(FUG)$	iff	there exists a computation $\pi = s_0 s_1 \dots$ of $T$ with $s_0 = s$ , such that there is an $m \geq 0$ such that $(T, s_m) \models G$ and $(T, s_k) \models F$ for all $k < m$

# Example of formulae in CTL

---

- $E\Diamond((A = 2) \wedge (B = 2))$

It is possible to reach a state where both processes are in the critical section.

- $A\Box(\text{enabled}_1 \wedge \dots \wedge \text{enabled}_k)$

freedom from deadlocks (a safety property);

- $A\Box(\text{req} \rightarrow A\Diamond\text{grant})$

every request will eventually be acknowledged (a liveness property);

- $A\Box(A\Diamond\text{enabled}_i)$

process  $i$  is enabled infinitely often on every computation path (unconditional fairness)

- $A\Box(E\Diamond\text{Restart})$

from every state it is possible to get to a restart state

# Equivalence

---

We say that two CTL formulas  $F$  and  $G$  are (globally) equivalent (written  $F \equiv G$ )

if, for all CTL structures  $T = (S, \rightarrow, L)$  and  $s \in S$ , we have

$$T, s \models F \text{ iff } T, s \models G.$$

# Equivalence

---

We say that two CTL formulas  $F$  and  $G$  are (globally) equivalent (written  $F \equiv G$ )

if, for all CTL structures  $T = (S, \rightarrow, L)$  and  $s \in S$ , we have

$$T, s \models F \text{ iff } T, s \models G.$$

## Examples:

$$\neg A \Diamond F \equiv E \Box \neg F$$

$$\neg E \Diamond F \equiv A \Box \neg F$$

$$\neg A \bigcirc F \equiv E \bigcirc \neg F$$

$$A \Diamond F \equiv A[\top \mathcal{U} F]$$

$$E \Diamond F \equiv E[\top \mathcal{U} F]$$



# CTL

---

Why is CTL called a tree logic?

Intuitively, it can talk about branching paths (which exists in a tree), but not about joining path (which do not exist in a tree).

# CTL\*

---

CTL\* is a logic which combines the expressive powers of LTL and CTL, by dropping the CTL constraint that every temporal operator ( $\bigcirc, \mathcal{U}, \Box, \Diamond$ ) has to be associated with a unique path quantifier ( $A, E$ ).

# CTL vs LTL

---

We want to compare the expressive power of LTL and CTL.

To do this, we give a branching time reading to LTL formulas that is inspired by our interpretation of LTL formulas in model checking:

we view LTL formulas as implicitly universally quantified.

(in LTL we consider all paths)

LTL formula  $F \mapsto CTL^*$  formula  $AF$

CTL is also a subset of  $CTL^*$ , since it is the fragment of  $CTL^*$  in which path quantifiers can only be applied to formulae starting with  $\bigcirc, \mathcal{U}, \square, \diamond$ .

# CTL vs LTL

---

**Definition.** We call two CTL<sup>\*</sup> formulas  $F$  and  $G$  equivalent if, for all transition systems  $T$  and states  $s$  of  $T$ , we have  $(T, s) \models F$  iff  $(T, s) \models G$ .

Some (but not all) LTL formulas can be converted into CTL formulas by adding an  $A$  to each temporal operator.

**Theorem.** There exists formulae in LTL which cannot be expressed in CTL and vice-versa.

- In CTL but not in LTL:  $A\Box E\Diamond F$

This expresses: wherever we have got to, we can always get to a state in which  $F$  is true.

This is also useful, e.g., in finding deadlocks in protocols.

- In LTL but not in CTL:  $A[\Box\Diamond p \rightarrow \Diamond q]$

“If there are infinitely many  $p$  along the path, then there is an occurrence of  $q$ .”

This is an interesting thing to be able to say; for example, many fairness constraints are of the form “infinitely often requested implies eventually acknowledged”.

# Model Checking

---

The CTL model checking problem is as follows:

Given a transition system  $T = (S, \rightarrow, L)$  and a CTL formula  $F$ , check whether  $T$  satisfies  $F$ , i.e., whether  $(T, s) \models F$  for all  $s \in S$ .

# Model Checking

---

The CTL model checking problem is as follows:

Given a transition system  $T = (S, \rightarrow, L)$  and a CTL formula  $F$ , check whether  $T$  satisfies  $F$ , i.e., whether  $(T, s) \models F$  for all  $s \in S$ .

## Method (Idea)

- (1) Arrange all subformulas  $F_i$  of  $F$  in a sequence  $F_0, \dots, F_k$  in ascending order w.r.t. formula length: for  $1 \leq i < j \leq k$ ,  $F_i$  is not longer than  $F_j$ ;
- (2) For all subformulas  $F_i$  of  $F$ , compute the set

$$\text{sat}(F_i) := \{s \in S \mid (T, s) \models F_i\}$$

in this order (from shorter to longer formulae);

- (3) Check whether  $S \subseteq \text{sat}(F)$ .

# Model Checking

---

How to compute  $\text{sat}(F_i)$

- $p \in \Pi \mapsto \text{sat}(p) = \{s \mid L(p, s) = 1\}$
- $\text{sat}(F_i \wedge F_j) = \text{sat}(F_i) \cap \text{sat}(F_j)$
- $\text{sat}(\neg F_i) = S \setminus \text{sat}(F_i)$
- $\text{sat}(E \bigcirc F_i) = \{s \mid \exists t \in S : (s \rightarrow t) \wedge t \in \text{sat}(F_i)\}$
- $\text{sat}(A \bigcirc F_i) = \{s \mid \forall t \in S : (s \rightarrow t) \wedge t \in \text{sat}(F_i)\}$
- $\text{sat}(E(F_i \mathcal{U} F_j))$  and  $\text{sat}(A(F_i \mathcal{U} F_j))$  are computed with the following procedures:

# Model Checking

---

$$F = E(F_i \mathcal{U} F_j)$$

```
sat(F) := T := sat(F_j)
while T != {} do
  choose s in T
  T := T \ {s}
  for all t in S with t -> s do
    if t in sat(F_i) and t not in sat(F) then
      sat(F) := sat(F) U {t}
      T := T U {t}
```

$$F = A(F_i \mathcal{U} F_j)$$

```
sat(F) := T := sat(F_j)
while T != {} do
  choose s in T
  T := T \ {s}
  for all t in S with t -> s do
    flag = 1
    for all t' in S with t -> t' do
      if t' not in sat(F) then flag := 0
    if t in sat(F_i) and t not in sat(F) and flag = 1 then
      sat(F) := sat(F) U {t}
      T := T U {t}
```



# Model Checking

---

**Theorem.**  $(T, s) \models F$  iff  $s \in \text{sat}(F)$ .

**Consequence.** CTL model checking is decidable.

Concerning the complexity, we observe the following: if  $F$  is of length  $n$ , then at most  $n$  sets  $\text{sat}(F_i)$  need to be computed. How complex is it to compute each such set?

- $F$  is a propositional letter or of the form  $F_1 \wedge F_2$  or  $\neg F_1$ :  $O(|S|)$  steps needed;
- $F$  is of the form  $E \bigcirc F_i$  or  $E(F_1 \mathcal{U} F_2)$ :  $O(|S| + |\rightarrow|)$  steps needed  
the maximum cardinality of the initial set  $\text{sat}(F_j)$  is  $|S|$ , and, in the forall loop, each edge from  $\rightarrow$  is “touched” at most once (in all iterations of the while);
- $F$  is of the form  $A(F_1 \mathcal{U} F_2)$ :  $O(|S| + |\rightarrow|^2)$  steps needed  
the maximum cardinality of the initial set  $\text{sat}(F_j)$  is  $|S|$ , the outer forall loop touches each edge from  $\rightarrow$  at most once, and the inner forall loop touches each edge at most once for each step done by the outer forall loop.

There exist more efficient algorithms (complexity  $|F| \cdot O(|S| + |\rightarrow|)$ ).