Non-classical logics

Lecture 21: Temporal Logic (Part 2)

Viorica Sofronie-Stokkermans

sofronie@uni-koblenz.de

Until now

Motivation

Models of time

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.

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.

Task: Verificaton.

Given the (formal) description of a system and of its intended behaviour, check whether the system indeed complies with this behaviour.

Linear Time Logic

Syntax

Π set of propositional variables.

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

- $\bot, \top$ are formulae;
- 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

We use an abstract model of reactive and concurrent systems.

Definition

Let Π be a finite set of propositional variables.

A Kripke structure (over Π) is a tuple $\mathcal{K} = (S, S_i, R, I)$ with

- S a non-empty set of states;
- $S_i \subseteq S$ is a set of initial states;
- $R \subseteq S \times S$ is a transition relation that is total, i.e.
  for each state $s \in S$, there is a state $s' \in S$ such that sRs';
- $I : \Pi \times S \to \{0, 1\}$ is a valuation function.

Linear Time Logic

Semantics

We use an abstract model of reactive and concurrent systems.

Definition

Let Π be a finite set of propositional variables.

A Kripke structure (over Π) is a tuple $\mathcal{K} = (S, S_i, R, I)$ with

- S a non-empty set of states;
- $S_i \subseteq S$ is a set of initial states; not very important
- $R \subseteq S \times S$ is a transition relation that is total, i.e.
  for each state $s \in S$, there is a state $s' \in S$ such that sRs';
- $I : \Pi \times S \to \{0, 1\}$ is a valuation function.

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"[a])

  Transition systems are also simply called models in what follows.

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

[a] 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 $s_d \rightarrow s_d$.

Example

Consider the following simple mutual-exclusion protocol:

```
    task body ProcA is
    begin
    loop
(0) Non_Critical_Section_A;
(1) loop [exit when Turn = 0] end loop;
(2) Critical_Section_A;
(3) Turn := 1;
    end loop;
    end ProcA;

    task body ProcB is
    begin
    loop
(0) Non_Critical_Section_B;
(1) loop [exit when Turn = 1] end loop;
(2) Critical_Section_B;
(3) Turn := 0;
    end loop;
    end ProcA;
```

Assume that the processes run asynchronously, i.e., either Process A or B makes a step, but not both. The order of executions is undetermined.

Example

$$\Pi = \{(T = i) \mid i \in \{0, 1\}\} \cup \{(X = i) \mid X \in \{A, B\}, i \in \{0, 1, 2, 3\}\}$$

$(T = i)$ means that Turn is set to i, and

$(X = i)$ means the process X is currently in Line i.

Example

We define the following Kripke structure $\mathcal{K} = (S, S_i, R, V)$:

- $S = \{0,1\} \times \{0,1,2,3\} \times \{0,1,2,3\}$
  $(t,i,j) \in S$: state in which Turn $= t$, A is at line i, B is at line j
- $S_i = \{(0,0,0),(1,0,0)\}$
- $R = R_A \cup R_B$, where
  $$R_A = \{((t,i,j),(t',i',j) \mid \quad i \in \{0,2,3\} \mid \rightarrow i' = i+1 \ (mod 4), t' = t$$
  $$t = 0, i = 1 \rightarrow i' = 2$$
  $$t = 1, i = 1 \rightarrow i' = 1$$
  $$i = 3 \rightarrow t' = 1\}$$
  and R_B is defined similarly
- $I((T = t'),(t,i,j)) = 1$ iff $t' = t$
  $I((A = i'),(t,i,j)) = 1$ iff $i' = i$
  $I((B = j'),(t,i,j)) = 1$ iff $j' = j$

Computations

Let $\mathcal{K} = (S, S_i, R, I)$ be a Kripke structure.

A computation of $\mathcal{K}$ is an infinite sequence $s_0 s_1 \ldots$ of states such that $s_0 \in S_i$ and $s_i R s_{i+1}$ for all $i \geq 0$.

Example: computation of the Kripke structure from the previous example:

$(0,0,0), (0,1,0), (0,1,1), (0,2,1), (0,3,1), (1,0,1), (1,0,2), \ldots$

Such a computation corresponds to an (asynchronous) execution of the concurrent system with Processes A and B.

Note that our formalization allows computations that are unfair, e.g., in which Process B is never executed. Such issues are not adressed on the level of Kripke structures.

Example

Interesting properties that can be verified in Example 2 include the following:

- **Mutual exclusion:** can A and B be at Line (2) at the same time?

  (holds)

- **Guaranteed accessibility:** if process $X \in \{A, B\}$ is at Line (2), is it guaranteed that it will eventually reach Line (3)?

  (holds, but only in computations that execute both Process A and Process B infinitely often)

Later, we will express such properties as temporal logic formulas.

Computation trees

Kripke structures can be non-deterministic, i.e., for an $s \in S$, the set $\{s' | sRs'\}$ can have arbitrary cardinality.

Thus, in general there is more than a single computation.

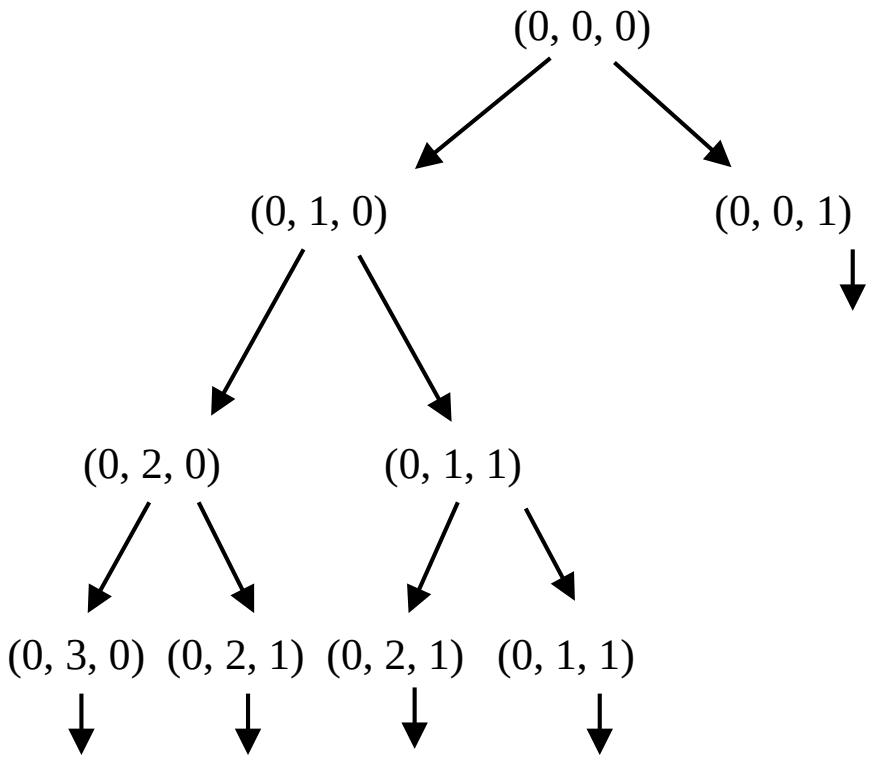
Instead of considering single computations in isolation, we can arrange all of them in a computation tree.

Informally, for $s \in S_i$, the (infinite) computation tree $T(\mathcal{K}, s)$ of $\mathcal{K}$ at $s \in S$ is inductively constructed as follows:

- use s as the root node;
- for each leaf s' of the tree, add successors $\{t \in S | s'Rt\}$.

Computation trees

The computation tree of the Kripke structure from the previous example starting at state (0, 0, 0) is:



Linear Time Logic

Syntax

Π set of propositional variables.

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

- $\bot, \top$ are formulae;
- 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

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

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

- $\pi \models \top$
- $\pi \not\models \bot$
- $\pi \models p$ 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 $\forall k \in \{0, \ldots, m-1\} : \pi^k \models F$

Linear Time Logic

Alternative way of defining the semantics:

An LTL structure M is an infinite sequence $S_0 S_1 \ldots$ 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 $\forall k \in \{n, \ldots, m-1\} : M, k \models F$

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

Abbreviations

- The future diamond

  $\Diamond\phi := \top\mathcal{U}\phi$ Sometimes denoted also $F\phi$

  $\pi \models \Diamond\phi$ iff $\exists m \geq 0 : \pi^m \models \phi$ $M, n \models \Diamond\phi$ iff $\exists m \geq n : M, m \models \phi$

- The future box

  $\Box\phi := \neg\Diamond\neg\phi$ Sometimes also denoted $G\phi$

  $\pi \models \Box\phi$ iff $\forall m{\geq}0 : \pi^m \models \phi$ $M, n \models \Box\phi$ iff $\forall m{\geq}n : M, m \models \phi$

- The infinitely often operator

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

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

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

- The almost everywhere operator

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

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

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

Abbreviations

- The release operator
  $\phi \mathcal{R} \psi := \neg(\neg\phi \mathcal{U} \neg\psi)$

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

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

  Read as
  "ψ always holds unless released by ϕ" i.e.,
  "ψ holds permanently up to and including the first point where ϕ holds (such an ϕ-point need not exist at all)".

Abbreviations

- The strict until operator:
  $F\mathcal{U}^{<}G := \bigcirc(F\mathcal{U}G)$
  $M, n \models F\mathcal{U}^{<}G$ iff $\exists m > n : M, m \models G \wedge \forall k \in \{n+1, ..., m-1\}, M, k \models F$

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 $T = (S, \to, L)$ and $s \in S$.

We say that $T, s \models \phi$ if for every computation π 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 systens T and paths π, we have $\pi \models F$ iff $\pi \models G$.

Note that:

$\bigcirc F \equiv \bot \mathcal{U}^{<} F$ 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)

Temporal Properties

A temporal property is a set of LTL structures
(those on which the property is true).

Thus, a temporal property P can be defined using an LTL formula F:

$$P = \{M | M, 0 \models F\}.$$

When given a Kripke structure $\mathcal{K}$ representing a reactive system and an LTL formula F representing a temporal property,

$\mathcal{K}$ satisfies F if $M, 0 \models F$ for all traces M of $\mathcal{K}$.

In this case, we write $\mathcal{K} \models F$.

Typical properties of reactive systems that need to be checked during verification are safety properties, liveness properties, and fairness properties.

Safety properties

Intuitively, a safety property asserts that "nothing bad happens"

general form: Condition $\rightarrow \Box F_{\text{Safe}}$

Examples of safety properties:

- **Mutual Exclusion.** For the example:

$$\Box(\neg((A = 2) \wedge (B = 2)))$$

- **Freedom from Deadlocks:** At any time, some process should be enabled:

$$\Box(enabled_1 \vee \cdots \vee enabled_k)$$

- **Partial Correctness:** If F is satisfied when the program starts, then G will be satisfied if the program reaches a distinguished state:

$$F \rightarrow \Box(\text{Dist} \rightarrow G)$$

where Dist $\in \Pi$ marks the distinguished state.

Liveness properties

Intuitively, a liveness property asserts that "something good will happen"

Examples of liveness properties:

- **Guaranteed Accessibility.** For the example:

$$\Box(A = 1 \rightarrow \Diamond(A = 2)) \wedge \Box(B = 1 \rightarrow \Diamond(B = 2))$$

- **Responsiveness:** If a request is issued, it will eventually be granted:

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

- **Total Correctness:** If F is satisfied when the program starts, then the program terminates in a distinguished state where G is satisfied:

$$\phi \rightarrow \Diamond(\text{Dist} \wedge G)$$

  Note that, in contrast, partial correctness is a safety property.

Fairness properties

When modelling concurrent systems, it is usually important to make some fairness assumptions. Assume that there are k processes, that $\text{enabled}_i \in \Pi$ is true in a state s if process $\#i$ is enabled in s for execution, and that executed_i is true in a state s if process $\#i$ has been executed to reach s.

Examples of fairness properties

- **Unconditional Fairness:** Every process is executed infinitely often:

$$\bigwedge_{1 \leq i \leq k} \Diamond^{\infty} \text{executed}_i$$

  Unconditional fairness is appropriate when processes can (and should!) be executed and any time. This is not always the case.

Fairness properties

When modelling concurrent systems, it is usually important to make some fairness assumptions. Assume that there are k processes, that $\text{enabled}_i \in \Pi$ is true in a state s if process $\#i$ is enabled in s for execution, and that executed_i is true in a state s if process $\#i$ has been executed to reach s.

Examples of fairness properties

- **Strong Fairness:** Every process enabled infinitely often is executed infinitely often:

$$\bigwedge_{1 \leq i \leq k} (\Diamond^{\infty} \text{enabled}_i \rightarrow \Diamond^{\infty} \text{executed}_i)$$

  Processes enabled only finitely often need not be guaranteed to be executed: they eventually and forever retract being enabled.

Fairness properties

When modelling concurrent systems, it is usually important to make some fairness assumptions. Assume that there are k processes, that $\text{enabled}_i \in \Pi$ is true in a state s if process $\#i$ is enabled in s for execution, and that executed_i is true in a state s if process $\#i$ has been executed to reach s.

Examples of fairness properties

- **Weak Fairness:** Every process enabled almost everywhere is executed infinitely often.

$$\bigwedge_{1 \leq i \leq k} (\Box^{\infty} \text{enabled}_i \rightarrow \Diamond^{\infty} \text{executed}_i)$$

This means that a process cannot be enabled constantly in an infinite interval without being executed in this interval.

Satisfiability

An LTL formula F is satisfiable if there exists an LTL structure M and $n \in \mathbb{N}$ such that $M, n \models F$.

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

When using LTL for verification, we are usually interested in whether a formula holds at point 0 of an LTL structure.

Lemma. Every satisfiable LTL formula F has a model M with $M, 0 \models F$.

Proof (Sketch)
Let $M, n \models F$, and let M' be the model obtained from M by dropping all time points $0, \ldots, n-1$. Thus, time point n in M is time point 0 in M'.

It is easy to prove by induction on the structure of G that, for all LTL formulas G and $i \geq 0$, we have $M', i \models G$ iff $M, n+i \models G$.

It follows that $M', 0 \models F$.

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 Kripke structure $\mathcal{K} = (S, S_i, R, I)$ and an LTL formula F, check whether $\mathcal{K} \models F$.

Recall: this is the case if all traces M of $\mathcal{K}$ satisfy $M, 0 \models F$.

Model checking

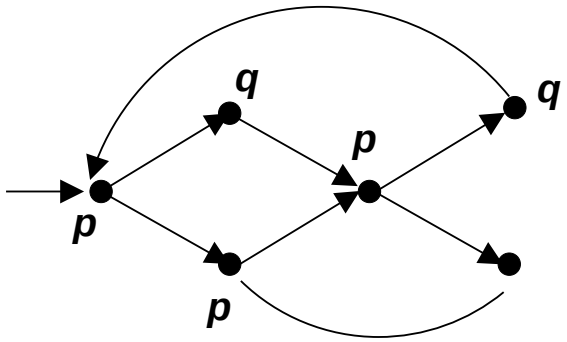
The LTL model checking problem is as follows: given a Kripke structure $\mathcal{K} = (S, S_i, R, I)$ and an LTL formula F, check whether $\mathcal{K} \models F$.

Recall: this is the case if all traces M of $\mathcal{K}$ satisfy $M, 0 \models F$.

Example:

The following Kripke structure satisfies $\Box(q \rightarrow \bigcirc \bigcirc \bigcirc p)$.

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



Model checking

The LTL model checking problem is as follows: given a Kripke structure $\mathcal{K} = (S, S_i, R, I)$ and an LTL formula F, check whether $\mathcal{K} \models F$.

Recall: this is the case if all traces M of $\mathcal{K}$ satisfy $M, 0 \models F$.

One can use the automata on infinite words used for checking satisfiability to obtain an elegant approach to model checking.

Lemma If $\mathcal{K} = (S, S_i, R, I)$ is a Kripke structure then there exists an automaton $\mathcal{A}_\mathcal{K}$ (of size linear in the size of $\mathcal{K}$) such that $L(\mathcal{A}_\mathcal{K})$ is the set of traces of $\mathcal{K}$.

Model checking

This suggests an algorithm for checking whether $\mathcal{K} \models F$:

- construct a Büchi automaton A such that $L(A) = L(\mathcal{A}_{\mathcal{K}}) \cap L(\mathcal{A}_{\neg F})$
- perform an emptyness test.

Connection to First-Order Logic

Another characterization of temporal properties that can be expressed in LTL is obtained by relating LTL to the monadic first-order theory of the natural numbers.

Let $FO^{<}$ denote the following first-order language:

- no function symbols and constants;
- binary predicate symbols: "suc" for successor, an order predicate $<$, and equality;
- countably infinite supply of unary predicates.

Connection to First-Order Logic

We may interpret formulas of $FO^<$ on LTL structures:

- quantification is over $\mathbb{N}$,
- the binary predicates are interpreted in the obvious way, and
- the unary predicates are identified with propositional variables.

Connection to First-Order Logic

We write $\phi(x_1, ..., x_n)$ to indicate that the variables in the $FO^<$ formula ϕ are $x_1, ..., x_n$.

For an $FO^<$ formula $\phi(x_1, ..., x_n)$, an LTL structure M, and $n_1, ..., n_k \in \mathbb{N}$, we write $M \models \phi[n_1, ..., n_k]$ if ϕ is true in M with variable x_i bound to value n_i, for $1 \leq i \leq k$.

Examples:

- For $\phi(x_1, x_2) = \neg p(x_1) \wedge p(x_2) \wedge \forall x_3.(x_1 < x_3 \rightarrow \neg q(x_3))$, we have $\emptyset\{p\} \ldots \{p\} \ldots \models \phi[0, 1]$.
- The following formula $\phi(x)$ expresses that there exists a future point that agrees with the current point (identified by the free variable) on the unary predicates $p_1, ..., p_n$:

$$\phi(x) = \exists y(x < y \wedge \bigwedge_{1 \leq i \leq n} (p_i(x) \leftrightarrow p_i(y)))$$

Connection to First-Order Logic

We say that an $FO^<$ formula $\phi(x)$ with exactly one free variable is equivalent to an LTL formula F if for all LTL models M and $n \in \mathbb{N}$ we have

$$M, n \models F \quad \text{iff} \quad M \models \phi[n].$$

Theorem: For every LTL formula F, there exists an equivalent $FO^<$ formula.

Proof The following translation $\mu : F_{LTL} \to FO^<$ takes LTL formulas F to equivalent $FO^<$ formulae:

$\mu(\top) = \top$; $\mu(\bot) = \bot$; $\mu(p)(x) = p(x)$ for every propositional variable p

$\mu(\neg F)(x) = \neg\mu(F)(x)$

$\mu(F \wedge G)(x) = \mu(F)(x) \wedge \mu(G)(x)$

$\mu(\bigcirc F)(x) = \exists y(suc(x, y) \wedge \mu(F)(y))$

$\mu(F \mathcal{U} G)(x) = \exists y(x < y \wedge \mu(G)(y) \wedge \forall z(x \leq z < y \to \mu(F)(z)))$

In the last two cases, variables y and z are newly introduced for every translation step.

Connection to First-Order Logic

What about the converse?

In general, are there $FO^<$ formulas $\phi(x)$ for which there is no equivalent LTL formula?

Connection to First-Order Logic

What about the converse?

In general, are there $FO^<$ formulas $\phi(x)$ for which there is no equivalent LTL formula?

Obviously there are: the formula $\exists y(y < x)$ states that there exists a previous time point – which cannot be expressed using only the future operators of LTL.

When we want to compare FO¡ with LTL, we should extend the latter with past-time temporal operators $\bigcirc^-$ and $\mathcal{S}$.

$M, n \models \bigcirc^- F$ iff $n > 0$ and $M, n-1 \models F$

$M, n \models F\mathcal{S}G$ iff $\exists m \leq n : M, m \models G$ and $M, k \models F$ for all $k \in \{m+1, ..., n\}$

Connection to First-Order Logic

This variant of LTL is called LTL with past operators (LTLP).

Connection to First-Order Logic

This variant of LTL is called LTL with past operators (LTLP).

Theorem (Kamp) For every $FO^<$ formula with one free variable, there exists an equivalent LTLP formula.

Proof. Out of the scope of this lecture.

Branching Time Logic: CTL

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

> Every state in a Kripke structure/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 Kripke structure/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

- $\top, \bot$ 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

In terms of Kripke structures $\mathcal{K} = (S, S_i, R, I)$/transition systems $T = (S, \rightarrow, L)$.

CTL: Semantics

Let $T = (S, \to, L)$ 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 $s \to t$
$(T, s) \models A \bigcirc F$	iff	$(T, t) \models F$ for all $t \in S$ with $s \to t$
$(T, s) \models A(F \mathcal{U} G)$	iff	for all computations $\pi = s_0 s_1 \ldots$ 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 $k < m$
$(T, s) \models E(F \mathcal{U} G)$	iff	there exists a computation $\pi = s_0 s_1 \ldots$ 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.
- $A\Box(\text{enabled}_1 \wedge \ldots \text{enabled}_k)$
  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

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).

Let $\mathcal{K} = (S, S_i, R, I)$ be a Kripke structure with $S_i = \{s_0\}$. We define a tree-shaped Kripke structure $T(\mathcal{K}) = (S', S_i, R', I')$ as follows:

- S' is the set of all finite computations of $\mathcal{K}$, i.e., $S' = \{s_0 \dots s_k \mid s_i R s_{i+1} \text{ for all } i < k\}$;
- $R' = \{(p, p') \in S' \times S' \mid p = qs, p' = ps' \text{ for some } s, s' \in S \text{ with } sRs'\}$;
- $I(P, p) = I(s)$ if $p = p's$ for some $p' \in \{\epsilon\} \cup S'$ and $s \in S$.

$T(\mathcal{K})$ is called the unravelling of $\mathcal{K}$.

Observe that $T(\mathcal{K})$ has no leaves because $\mathcal{K}$ is total.

CTL

CTL formulas cannot distinguish between a state in a Kripke structure and the corresponding states in the tree-shaped unravelling.

Lemma Let $\mathcal{K}$ be a Kripke structure, s a state of $\mathcal{K}$, $p = s_0 \dots s_k$ a state of $T(\mathcal{K})$ such that $s_k = s$, and F a CTL formula.

Then $(\mathcal{K}, s) \models F$ iff $(T(\mathcal{K}), p) \models F$.

Proof. By induction on the structure of F.

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.

Definition. We call two LTL or CTL formulas F and G equivalent if, for all Kripke structures $\mathcal{K}$ and states s of $\mathcal{K}$, we have

$$(\mathcal{K}, s) \models F \text{ iff } (\mathcal{K}, s) \models G.$$

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.

Definition. We call two LTL or CTL formulas F and G equivalent if, for all Kripke structures $\mathcal{K}$ and states s of $\mathcal{K}$, we have

$$(\mathcal{K}, s) \models F \text{ iff } (\mathcal{K}, s) \models G.$$

Theorem.

- For the CTL formula $E\Diamond p$ there is no equivalent LTL formula.
- For the LTL formula $\Diamond\Box p$ there is no equivalent CTL formula.

Model Checking

The CTL model checking problem is as follows:

> Given a Kripke structure $\mathcal{K} = (S, S_i, R, I)$ and a CTL formula F, check whether K satifies F, i.e., whether $(\mathcal{K}, s) \models F$ for all $s \in S_i$.

Model Checking

The CTL model checking problem is as follows:

> Given a Kripke structure $\mathcal{K} = (S, S_i, R, I)$ and a CTL formula F, check whether K satifies F, i.e., whether $(\mathcal{K}, s) \models F$ for all $s \in S_i$.

Method (Idea)

(1) Arrange all subformulas F_i of F in a sequence $F_0, \ldots F_k$ 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

$$sat(F_i) := \{s \in S | (\mathcal{K}, s) \models F_i\}$$

in this order (from shorter to longer formulae);

(3) Check whether $S_i \subseteq sat(F)$.

Model Checking

How to compute $sat(F_i)$

- $p \in \Pi \mapsto sat(p) = \{s \mid I(p, s) = 1\}$
- $sat(F_i \wedge F_j) = sat(F_i) \cap sat(F_j)$
- $sat(\neg F_i) = S \backslash sat(F_i)$
- $sat(E \bigcirc F_i) = \{s \mid \exists t \in S : sRt \wedge t \in sat(F_i)\}$
- $sat(A \bigcirc F_i) = \{s \mid \forall t \in S : sRt \wedge t \in sat(F_i)\}$
- $sat(E(F_i \mathcal{U} F_j))$ and $sat(A(F_i \mathcal{U} F_j)$ 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 tRs 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 tRs do
     flag = 1
     for all t' in S with tRt' 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. $(\mathcal{K}, s) \models F$ iff $s \in sat(F)$.

Consequence. CTL model checking is decidable.

Concerning the complexity, we observe the following: if F is of length n, then at most n sets $sat(F_i)$ 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 $E(F_1 \mathcal{U} F_2)$: $O(|S| + |R|)$ steps needed

  the maximum cardinality of the initial set $sat(F_j)$ is $|S|$, and, in the forall loop, each edge from R is "touched" at most once (in all iterations of the while);
- F is of the form $A(F_1 \mathcal{U} F_2) : O(|S| + |R|^2)$ steps needed

  the maximum cardinality of the initial set $sat(F_j)$ is $|S|$, the outer forall loop touches each edge from R 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 $|F| \cdot O(|S| + |R|)$).