Temporal Logic

Carsten Lutz

Summer semester 2006 (14 lectures)

Contents

1 Introduction

1.1 Structures

The purpose of temporal logic (TL) is reasoning about time (in philosophy), and about the behaviour of systems evolving over time (in computer science). To define a TL, we need

- a *syntax*: the language for talking about time or temporal systems;
- a *semantics*: our model of time.

What model of time should we use? What is the *structure of time*? A very liberal definition:

> A *flow of time* is a pair $(T, <)$, where T is a non-empty set of *time points*, and $<$ is an irreflexive and transitive binary relation on T.

Note that antisymmetry follows (Exercise*). Depending on the intended application, we often require additional properties. One of the most fundamental decisions is whether or not time should be linear:

> $(T, <)$ is *linear* if, for all $x, y \in T$ with $x \neq y$, we have $x < y$ or $y < x$.

Important additional properties for linear flows of time:

- *Boundedness.* We have four options by combining:
  - Bounded to the past: there exists an $x \in T$ such that $x \leq y$ for all $y \in T$ (*genesis*).
  - Bounded to the future: there exists a an $x \in T$ such that $x \geq y$ for all $y \in T$ (*doomsday*).
- *Discreteness.* Existence of *direct* predecessors and successors:
  - If $x \in T$ is not genesis, then there exists a $y \in T$ such that $y < x$ and $y < z < x$ holds for *no* $z \in T$.
  - If $x \in T$ is not doomsday, then there exists a $y \in T$ such that $x < y$ and $x < z < y$ holds for *no* $z \in T$.

  It can be seen that one does not follow from the other.
- *Density.* For all $x, y \in T$ with $x < y$, there is a $z \in T$ such that $x < z < y$.
- *Dedekind completeness.* Any non-empty subset $S \subseteq T$ that has an upper bound has a *least upper bound*:
  - upper bound for S: $x \in T$ with $y \leq x$ for all $y \in S$;
  - least upper bound for S: upper bound x for S such that there is no $x' \in T$ with $x' < x$ and x' upper bound for S.

Note: The existence of least lower bounds follows (Exercise*: take a set S and let S' be all lower bounds for S. Then there exists a least upper bound for S'. Either it is in S, then we are done. Or it is before S. But then there can be no lower bound for S between S and the least upper bound for S'. Hence the least upper bound for S' is the least lower bound for S).

The following are among the most natural linear flows of time:

- The natural numbers $\mathbb{N}$ with the usual order $<$. Linear, discrete, bounded to the past, not bounded to the future. Note that other flows of time have these properties as well:

  - $T := \mathbb{N} \times \{0\} \cup \mathbb{Z} \times \{1\}$;
  - $(x, a) < (y, b)$ if (i) $a < b$ or (ii) $a = b$ and $x < y$.

  Indeed, $\mathbb{N}$ is (up to isomorphism) the unique discrete *Dedekind-complete* linear flow with a first point and no last point. NOTE: above example not Dedekind-complete. The integers $\mathbb{Z}$ can be characterized analogously.

- The rational numbers $\mathbb{Q}$. A natural dense flow of time, though with gaps (e.g. $\sqrt{2}$). The unique countable linear dense flow of time without endpoints (up to isomorphism).

- The real numbers $\mathbb{R}$. Up to isomorphism, the unique dense, Dedekind-complete flow of time without end points that is *separable*:

  there exists a countable subset $D \subseteq T$ such that, for all $x, y \in T$ with $x < y$, there is a $z \in D$ with $t < z < u$.

The alternative to linear time is *branching* time. Time can be

- branching to the future reflecting that there are many possible futures;

- branching to the past reflecting that many different histories are considered possible (due to incomplete knowledge).

Branching to the future and linear to the past is the most popular option, i.e.,

for each $x \in T$, the set $\{y \in T \mid y < x\}$ is linearly ordered by $<$.

We can identify additional properties similar to the linear case. Usually, branching time is assumed to be discrete and has a genesis.

Which flow of time should we use? This depends on the application! In the next section, we introduce the application that guides us throughout this lecture.

1.2 Reactive Systems

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

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

- *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.

- *Concurrent System.* 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.

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

Typical examples of reactive systems include microprocessors, operating systems, network protocols, aviation software, etc. The verification of such systems is an important and difficult task.

We use an abstract model of reactive and concurrent systems.

Definition 1. Let PL be a finite set of *propositional letters*. A *Kripke structure* over PL is a tuple $K = (S, S_I, R, V)$ with

- S a non-empty set of *states*;

- $S_I \subseteq S$ is a set of *initial states*;

- $R \subseteq S \times S$ a transition relation that is total, i.e., for each state $s \in S$, there is a state $s' \in S$ such that sRs';

- $V : S \to 2^{\mathsf{PL}}$ a *valuation function.*

$\triangle$

Example 2. *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.*

Define a set of propositional letters

$$\mathsf{PL} = \{(T = i) \mid i \in \{0, 1\}\} \cup \{(X = i) \mid X \in \{A, B\} \textit{ and } i \in \{0, 1, 2, 3\}\}.$$

Intuitively, $(T = i)$ *means that* Turn *is set to* i*, and* $X = i$ *means the process* X *is currently in Line* i*. Define a Kripke structure* $K = (S, S_I, R, V)$ *as follows:*

- $S = \{0, 1\} \times \{0, 1, 2, 3\} \times \{0, 1, 2, 3\}$.

  Intuitively, in a tuple (t, p_A, p_B)*,* t *is the value of* Turn *and* p_A/p_B *is the line that Process A/B is currently executing*

- $S_I = \{(0, 0, 0), (1, 0, 0)\}$;

- $R = R_A \cup R_B$*, where*

  $R_A = \{(t, p_A, p_B), (t', p'_A, p'_B) \mid p_B = p'_B$ *and*

  1. $p_A \in \{0, 2, 3\} \implies p'_A = p_A + 1 \mod 4$ *and* $t' = t$
  2. $t = 0 \wedge p_A = 1 \implies p'_A = 2$
  3. $t = 1 \wedge p_A = 1 \implies p'_A = 1$
  4. $p_A = 3 \implies t' = 1\ \}$

  and R_B *is defined dually.*

- $V(t, p_A, p_B) = \{(T = t), (A = p_A), (B = p_B)\}$*, for all* $(t, p_A, p_B) \in S$.

Let $K = (S, S_I, R, V)$ be a Kripke structure. A *computation* of K is an infinite sequence $s_0 s_1 \cdots$ of states such that $s_0 \in S_I$ and $s_i R s_{i+1}$ for all $i \geq 0$. An example computation of the Kripke structure from Example 2 is

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

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.

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

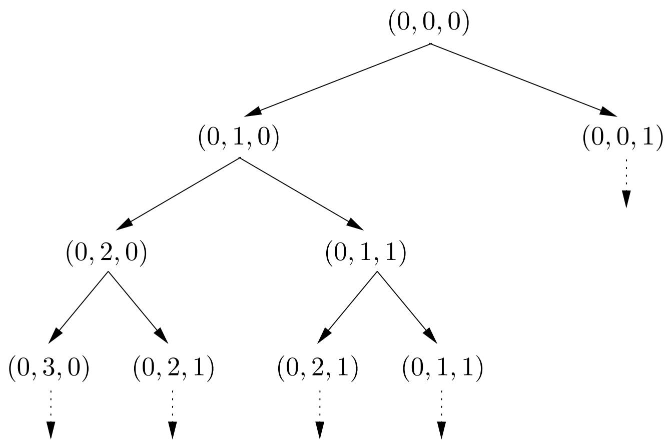
- Mutual exclusion: can A and B be at Line (2) at the same time? (holds)
- Guaranteed accessibility: if process $X \in \{A, B\}$ is at Line (2), is it guaranteed that it will eventually reach Line (3)? (holds, but only in computations that execute both Process A and Process B infinitely often)

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

Kripke structures can be non-deterministic, i.e., for an $s \in S$, the set $\{s' \mid sRs'\}$ 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* $T(K, s)$ of 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 $\{t \in S \mid s'aRt\}$.

The computation tree of the Kripke structure from Example 2 starting at state $(0,0,0)$ is:



To verify properties, we can consider single computations or the whole tree. This leads to two natural types of temporal structures:

1. computations are isomorphic to the linear time flow $(\mathbb{N}, <)$;

2. computation trees can be viewed as branching time flows that are infinite trees;

This gives rise to two different families of temporal logics.

2 Linear Time Logic

2.1 Syntax and Semantics

Definition 3. The set of *LTL (linear time logic) formulas* is the smallest set such that

- each propositional letter $p \in \mathsf{PL}$ is a formula;
- if φ and ψ are formulas, then so are $\varphi \wedge \psi$ and $\neg\varphi$;
- if φ and ψ are formulas, then so are $\bigcirc\varphi$ and $\varphi \mathcal{U} \psi$.

$\triangle$

We use the following Boolean abbreviations:

$$
\begin{array}{rcl}
\varphi \vee \psi & \text{for} & \neg(\neg\varphi \wedge \neg\psi) \\
\varphi \rightarrow \psi & \text{for} & \neg\varphi \vee \psi \\
\varphi \leftrightarrow \psi & \text{for} & (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi) \\
\top & \text{for} & p \vee \neg p \text{ for some fixed } p \in \mathsf{PL} \\
\bot & \text{for} & \neg\top
\end{array}
$$

Definition 4. An *LTL* structure M is an infinite sequence $S_0 S_1 \cdots$ with $S_i \in 2^{\mathsf{PL}}$ for all $i \geq 0$. We define satisfaction of LTL formulas in M at time points $n \in \mathbb{N}$ as follows:

$$
\begin{array}{rcl}
M, n \models p & \text{iff} & p \in S_n, \text{ for all } p \in \mathsf{PL} \\
M, n \models \varphi \wedge \psi & \text{iff} & M, n \models \varphi \text{ and } M, n \models \psi \\
M, n \models \neg\varphi & \text{iff} & M, n \models \varphi \text{ is } \textit{not} \text{ the case} \\
M, n \models \bigcirc\varphi & \text{iff} & M, n+1 \models \varphi \\
M, n \models \varphi \mathcal{U} \psi & \text{iff} & \exists m \geq n : M, m \models \psi \wedge \forall k \in \{n, \ldots, m-1\} : M, k \models \varphi
\end{array}
$$

$\triangle$

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

The connection between Kripke structures and LTL structures is as follows: every computation of a Kripke structure gives rise to an LTL structure. To see this, let $K = (S, S_I, R, V)$ be a Kripke structure. A computation $s_0, s_1, \ldots$ of K induces an LTL structure $V(s_0)V(s_1)\cdots$. Such an LTL structure is called a *trace* of K.

We introduce a number of useful temporal abbreviations:

- The *future diamond*

  $\Diamond\varphi := \top\,\mathcal{U}\,\varphi$, i.e, $M, n \models \Diamond\varphi$ iff $\exists m \geq n : M, m \models \varphi$.

- The *future box*

  $\Box\varphi := \neg\Diamond\neg\varphi$, i.e, $M, n \models \Box\varphi$ iff $\forall m \geq n \rightarrow M, m \models \varphi$.

- The *infinitely often* operator

  $\Diamond^{\infty}\varphi := \Box\Diamond\varphi$, i.e, $M, n \models \Diamond^{\infty}\varphi$ iff $\{m \geq n \mid M, m \models \varphi\}$ is infinite.

- The *almost everywhere* operator

  $\Box^{\infty}\varphi := \Diamond\Box\varphi$, i.e, $M, n \models \Diamond^{\infty}\varphi$ iff $\{m \geq n \mid M, m \not\models \varphi\}$ is finite.

- The *release* operator

  $\varphi\,\mathcal{R}\,\psi := \neg(\neg\varphi\,\mathcal{U}\,\neg\psi)$, i.e,

  $$M, n \models \varphi\,\mathcal{R}\,\psi \text{ iff } \forall m \geq n : \text{if } M, k \not\models \varphi \text{ for all } k < m, \text{ then } M, m \models \psi$$

  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 a φ-point need not exist at all.

- The *strict* until operator:

  $\varphi\,\mathcal{U}^{<}\psi := \bigcirc(\varphi\,\mathcal{U}\,\psi)$, i.e.,

  $$M, n \models \varphi\,\mathcal{U}^{<}\psi \text{ iff } \exists m > n : M, m \models \psi \wedge \forall k \in \{n+1, \ldots, m-1\} : M, k \models \varphi$$

  We say that two LTL formulas φ and ψ are *(globally) equivalent* (written $\varphi \equiv \psi$) if, for all LTL structures M and $i \geq 0$, we have $M, i \models \varphi$ iff $M, i \models \psi$.

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

  Note that $\bigcirc\varphi \equiv \bot\,\mathcal{U}^{<}\varphi$ and $\varphi\,\mathcal{U}\,\psi \equiv \psi \vee (\varphi \wedge \bigcirc(\varphi\,\mathcal{U}^{<}\psi))$. 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

Some useful equivalences that will be useful later on (exercise: prove them):

$$
\begin{array}{rcll}
\neg\bigcirc\varphi & \equiv & \bigcirc\neg\varphi & \text{(self-duality of next)} \\
\Diamond\Diamond\varphi & \equiv & \Diamond\varphi & \text{(idempotency of diamond)} \\
\bigcirc\Diamond\varphi & \equiv & \Diamond\bigcirc\varphi & \text{(commutation of next with Diamond)} \\
\Diamond\Diamond^{\infty}\varphi & \equiv & \Diamond^{\infty}\varphi \;\equiv\; \Diamond^{\infty}\Diamond\varphi & \text{(absorption of diamonds by “inf.often”)} \\
\varphi\,\mathcal{U}\,\psi & \equiv & \neg(\neg\varphi\,\mathcal{R}\,\neg\psi) & \text{(until and release are duals)} \\
\varphi\,\mathcal{U}\,\psi & \equiv & \psi \vee (\varphi \wedge \bigcirc(\varphi\,\mathcal{U}\,\psi)) & \text{(unfolding of until)} \\
\varphi\,\mathcal{R}\,\psi & \equiv & (\varphi \wedge \psi) \vee (\psi \wedge \bigcirc(\varphi\,\mathcal{R}\,\psi)) & \text{(unfolding of release)}
\end{array}
$$

2.2 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 φ:

$$P = \{M \mid M, 0 \models \varphi\}.$$

When given a Kripke structure K representing a reactive system and an LTL formula φ representing a temporal property, K *satisfies* φ if $M, 0 \models \varphi$ for all traces M of K. In this case, we write $K \models \varphi$.

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

Safety

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

- *Mutual Exclusion.* E.g. in Example 2,

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

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

  $$\Box(\mathsf{enabled}_1 \vee \cdots \vee \mathsf{enabled}_k)$$

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

  $$\varphi \to \Box(\mathsf{dist} \to \psi)$$

  where $\mathsf{dist} \in \mathsf{PL}$ marks the distinguished state.

Liveness

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

- *Guaranteed Accessibility.* E.g. in Example 2:

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

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

  $$\Box(\mathsf{req} \to \Diamond\mathsf{grant})$$

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

  $$\varphi \to \Diamond(\mathsf{dist} \wedge \psi)$$

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

Fairness

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

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

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

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

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

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

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

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

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

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

Usually, only the *fair* computations of a system are relevant because fairness is ensured by some scheduling mechanism. Thus, if

- K is a system in which $\mathsf{enabled}_i$ and $\mathsf{executed}_i$ are set appropriately,
- φ describes the fairness condition and
- ψ the property to be verified (e.g. a safety or liveness property),

then we are interested in whether $K \models \varphi \rightarrow \psi$.

2.3 Satisfiability

An LTL formula φ is *satisfiable* if there exists an LTL structure M such that $M, n \models \varphi$ for some $n \in \mathbb{N}$. Such a structure is called a *model of* φ. In verification, satisfiability can be used to detect contradictory properties, i.e., properties that are satisfied by *no* computation of *any* reactive system. For example, the following property is contradictory:

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

Exercise (*): prove that this LTL formula is unsatisfiable.

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

Lemma 5. *Every satisfiable LTL formula φ has a model M with $M, 0 \models \varphi$.*

Proof. (sketch) Let $M, n \models \varphi$, 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 ψ that, for all LTL formulas ψ and $i \geq 0$, we have $M', i \models \psi$ iff $M, n+i \models \psi$. It follows that $M', 0 \models \varphi$. ❏

LTL satisfiability can be decided using automata on infinite words.

Definition 6. A (non-deterministic) Büchi automaton is a tuple $\mathcal{A} = (Q, \Sigma, I, \Delta, F_1, \ldots, F_k)$, $k \geq 0$, with

- Q a finite set of *states*;
- Σ a finite *alphabet*;
- $I \subseteq Q$ a set of *initial states*;
- $\Delta \subseteq Q \times \Sigma \times Q$ a *transition relation*;
- $F_i \subseteq Q$ a set of *accepting states*, for $1 \leq i \leq k$.

Let $w = a_0 a_1 \cdots \in \Sigma^\omega$. A *run* of $\mathcal{A}$ on w is a word $q_0 q_1 \cdots \in Q^\omega$ such that

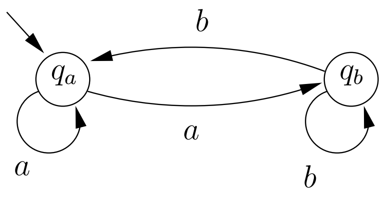
- $q_0 \in I$;
- $(q_i, w_i, q_{i+1}) \in \Delta$ for all $i \geq 0$.

A run $q_0, q_1, \cdots$ is *accepting* if the set $\{i \mid q_i \in F_j\}$ is infinite for all j with $1 \leq j \leq k$. The ω-language accepted by $\mathcal{A}$ is defined as

$$L(\mathcal{A}) := \{w \in \Sigma^\omega \mid \text{there is an accepting run of } \mathcal{A} \text{ on } w\}.$$

△

Example 7. *Let $\Sigma = \{a, b\}$. Then the following automaton accepts the language of those $w \in \Sigma^\omega$ that contain both a and b an infinite number of times, where $F_1 = \{q_a\}$ and $F_2 = \{q_b\}$:*



Büchi automata are often defined with only one set of accepting states. We call such automata *simple*.

Lemma 8. *For every Büchi-automaton $\mathcal{A}$, there is a simple Büchi-automaton $\mathcal{A}$, there is a simple Büchi-automaton $\mathcal{A}'$ such that $L(\mathcal{A}) = L(\mathcal{A}')$ and $|Q'| = |Q| \cdot k$.*

Proof. Let $\mathcal{A} = (Q, \Sigma, I, \Delta, F_1, \ldots, F_k)$. Set $\mathcal{A}' = (Q', \Sigma, I', \Delta', F')$ as follows:

- $Q' = Q \times \{0, \ldots, k\}$;
- $I' = I \times \{1\}$;
- $((q, i), a, (q', i')) \in \Delta'$ iff $(q, a, q') \in \Delta$ and one of the following holds:
  - $i > 0$, $q \notin F_i$, and $i' = i$;
  - $i > 0$, $q \in F_i$, and $i' = i + 1 \mod k + 1$;
  - $i = 0$ and $i' = 1$.
- $F' = Q \times \{0\}$.

The proof that $L(\mathcal{A}) = L(\mathcal{A}')$ is left as an exercise. Intuitively, when $\mathcal{A}'$ is in state (q, i) with $i > 0$, it is waiting for a accepting state from the set q_i. When it is in $(q, 0)$, it has seen a accepting state from each of $F_1, \ldots, F_k$, in this order, and starts again with F_1. This particular order can be stipulated since there have to be *infinitely* many states from each F_i. ❏

Büchi automata may be used for specifying temporal properties: when using $\Sigma = 2^{\mathsf{PL}}$, $L(\mathcal{A})$ is a set of LTL structures and thus a temporal property.

To decide satisfiability in LTL, we convert the input formula φ_0 into a Büchi automaton $\mathcal{A}_{\varphi_0}$ with alphabet $\Sigma = 2^{\mathsf{PL}}$ such that $\mathcal{A}_{\varphi_0}$ accepts an LTL structure M iff $M, 0 \models \varphi_0$. Thus, φ_0 is satisfiable iff $L(\mathcal{A}_{\varphi_0}) \neq \emptyset$.

The *emptyness problem* for Büchi automata asks, given an automaton $\mathcal{A}$, whether $L(\mathcal{A}) = \emptyset$. The following Lemma gives a hint on how to decide emptyness. Let $\mathcal{A} = (Q, \Sigma, I, \Delta, F_1, \ldots, F_k)$ be a Büchi automaton and $q, q' \in Q$. We say that q' is *reachable* from q if there are states $q_1, \ldots, q_k \in Q$ and $a_1, \ldots, a_{k-1} \in \Sigma$ such that $q_1 = q$, $q_k = q'$, and $(q_i, a_i, q_{i+1}) \in \Delta$ for $1 \leq i < k$.

Lemma 9. *Let $\mathcal{A} = (Q, \Sigma, I, \Delta, F_1, \ldots, F_k)$ be a Büchi automaton. Then $L(\mathcal{A}) \neq \emptyset$ iff there are $q_0, \ldots, q_k$ such that*

- *$q_0 \in I$;*
- *$q_i \in F_i$ for $1 \leq i \leq k$, and*
- *q_{i+1} is reachable from q_i for $i \leq k$, where $q_{k+1} := q_1$.*

Proof. "$\Rightarrow$". Then there is a word $w \in L(\mathcal{A})$ and an accepting run $r = q_0 q_1 \cdots \in Q^\omega$ of $\mathcal{A}$ on w. Since the F_i are finite, there is a $q_i \in F_i$ that occurs infinitely often in r, for all i with $1 \leq i \leq k$. Clearly, q_{i+1} is reachable from q_i for $i < k$.

"$\Leftarrow$". Let $q_0, \ldots, q_k \in Q$ such that the conditions in the lemma are satisfied and let $u_1, \ldots, u_k$ be words such that q_i reaches q_{i+1} with u_i for all $i \leq k$. Then $u_0(u_1 \cdots u_k)^\omega \in L(\mathcal{A}) \neq \emptyset$. ❑

Theorem 10. *The emptyness problem for Büchi automata $\mathcal{A} = (Q, \Sigma, I, \Delta, F_1, \ldots, F_k)$ can be decided in time $p(k \cdot |Q|)$, with $p()$ a polynomial.*

Proof. By Lemma 8, we can transform every automaton $\mathcal{A}$ into a simple one $\mathcal{A}'$ such that $L(\mathcal{A}) = L(\mathcal{A}')$ and $|Q'| \leq |Q| \cdot k$. It thus suffices to show that emptyness of simple automata can be decided in time $p(|Q|)$.

Let $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ be a simple automaton. By Lemma 9, $L(\mathcal{A}) \neq \emptyset$ iff there exist states $q \in I$ and $q' \in F$ such that q' is reachable in from q and q' is reachable from q'. Define a directed edge-labelled graph $G_\Delta = (V, E)$ by setting $V = Q$ and $E = \Delta$. Then reachability becomes graph reachability, which can be decided in time polynomial in $|V| = |Q|$. We require $2 \cdot |I| \cdot |F|$ reachability tests to check emptyness of $\mathcal{A}$. This gives the desired result. ❑

To construct the automaton $\mathcal{A}_{\varphi_0}$, we need some preliminaries. We first determine the set of formulas that are "relevant" for deciding satisfiability of φ_0.

Definition 11. The *closure* $\mathsf{cl}(\varphi_0)$ of φ_0 is the smallest set of formulas containing all subformulas of φ_0 and closed under single negations. Formally:

- $\mathsf{cl}(p) := \{p, \neg p\}$;
- $\mathsf{cl}(\neg\varphi) := \mathsf{cl}(\varphi)$;
- if $\vartheta \in \{\varphi \wedge \psi, \varphi \mathcal{U} \psi\}$, then $\mathsf{cl}(\vartheta) := \{\vartheta, \neg\vartheta\} \cup \mathsf{cl}(\varphi) \cup \mathsf{cl}(\psi)$;
- $\mathsf{cl}(\bigcirc\varphi) := \{\bigcirc\varphi, \neg\bigcirc\varphi\} \cup \mathsf{cl}(\varphi)$.

△

Each LTL structure $M = S_0 S_1 \cdots$ gives rise to a sequence of sets of formulas $T_0 T_1 \cdots$ by setting

$$T_i := \{\varphi \in \mathsf{cl}(\varphi_0) \mid M, i \models \varphi\}.$$

We say that M *realizes* the sequence $T_0 T_1 \cdots$. By the semantics, formula sets that occur in a sequence realized by an LTL structure satisfy certain properties. Formula sets satisfying these properties are called *types* and will be the states of $\mathcal{A}_{\varphi_0}$

Definition 12. A *type for* φ_0 is a subset $T \subseteq \mathsf{cl}(\varphi_0)$ such that:

1. $\psi \in T$ iff $\neg\psi \notin T$, for all $\neg\psi \in \mathsf{cl}(\varphi_0)$;
2. $\{\psi, \vartheta\} \subseteq T$ iff $\psi \wedge \vartheta \in T$, for all $\psi \wedge \vartheta \in \mathsf{cl}(\varphi_0)$;

We use $\mathsf{TP}(\varphi_0)$ to denote thet set of types for φ_0. For $T, T' \in \mathsf{TP}(\varphi_0)$, we write $T \to_{\bigcirc} T'$ if

- for all $\bigcirc\varphi \in \mathsf{cl}(\varphi_0)$, $\bigcirc\varphi \in T$ iff $\varphi \in T'$;
- for all $\varphi\,\mathcal{U}\,\psi \in \mathsf{cl}(\varphi_0)$, we have $\varphi\,\mathcal{U}\,\psi \in T$ iff
  - $\psi \in T$ or
  - $\varphi \in T$ and $\varphi\,\mathcal{U}\,\psi \in T'$.

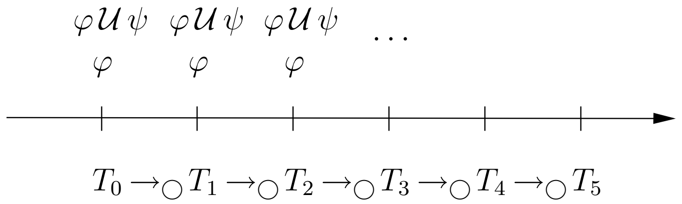
$\triangle$

The relation "$\to_{\bigcirc}$" will be used used to define the transition relation of $\mathcal{A}_{\varphi_0}$: if M is an LTL structure realizing $T_0, T_1, \ldots$, then for all $i \geq 0$, we have

(i) T_i is a type and

(ii) $T_i \to_{\bigcirc} T_{i+1}$

Still, not all sequences satisfying (i) and (ii) are induced by an LTL structure since formulas $\varphi\,\mathcal{U}\,\psi$, may be *delayed forever*.



In fact, a sequence $T_0, T_1, \ldots$ is realized by an LTL structure *if and only* if it satisfies (i) and (ii) and has no infinitely delayed until formulas. Using the set of accepting states, we can rule out such infinite delays.

Definition 13. Let there be k until formulas in $\mathsf{cl}(\varphi_0)$ and assume that these formulas are linearly ordered. The automaton $\mathcal{A}_{\varphi_0}$ is defined as $(Q, \Sigma, I, \Delta, F_1, \ldots, F_k)$:

- $Q = \mathsf{TP}(\varphi_0)$;
- $\Sigma = 2^{\mathsf{PL}}$;
- $I := \{T \in Q \mid \varphi_0 \in T\}$;
- $\Delta := \{(T, a, T') \mid a \cap \mathsf{cl}(\varphi_0) = T \cap \mathsf{PL}$ and $T \to_{\bigcirc} T'\}$;
- $F_i := \{T \in Q \mid \varphi\,\mathcal{U}\,\psi \notin T$ or $\psi \in T\}$ if the i-th until formula in $\mathsf{cl}(\varphi_0)$ is $\varphi\,\mathcal{U}\,\psi$.

$\triangle$

Lemma 14. *For all LTL structures $M = S_0 S_1 \cdots$, $M \in L(\mathcal{A}_{\varphi_0})$ iff $M, 0 \models \varphi_0$.*

Proof. "$\Leftarrow$". Suppose $M, 0 \models \varphi_0$. Define

$$T_i := \{\varphi \in \mathsf{cl}(\varphi_0) \mid M, i \models \varphi\}.$$

It is easily checked that $T_0 T_1 \cdots$ is a run of $\mathcal{A}_{\varphi_0}$ on M:

- $T_i \in \mathsf{TP}(\varphi_0)$ for all $i \geq 0$;
- $\varphi_0 \in T_0$ and thus $T_0 \in I$;
- $S_i \cap \mathsf{cl}(\varphi_0) = T_i \cap \mathsf{PL}$ and $T_i \rightarrow_{\bigcirc} T_{i+1}$, thus $(T_i, S_i, T_{i+1}) \in \Delta$ for all $i \geq 0$;

Moreover, $T_0 T_1 \cdots$ is accepting: suppose that, for some $j \in \{1, \ldots, k\}$, the set $\{i \in \mathbb{N} \mid T_i \in F_j\}$ is finite. Then there exists an $i_0 \in \mathbb{N}$ such that $T_i \notin F_j$ for all $i \geq i_0$. This means $\varphi \, \mathcal{U} \, \psi \in T_i$ and $\psi \notin T_i$, for all $i \geq i_0$, where $\varphi \, \mathcal{U} \, \psi$ is the j-th until formula. Hence, $M, i_0 \models \varphi \, \mathcal{U} \, \psi$ and $M, i \not\models \psi$ for all $i \geq i_0$, in contradiction to the semantics.

"$\Rightarrow$". Let $M = S_0 S_1 \cdots \in L(\mathcal{A}_{\varphi_0})$, and let $T_0 T_1 \cdots$ be an accepting run of $\mathcal{A}_{\varphi_0}$ on M. Then M is an LTL structure. Since we have $\varphi_0 \in T_0$ by def. of I, it suffices to show that, for all $i \geq 0$ and $\varphi \in \mathsf{cl}(\varphi_0)$, we have

$$\varphi \in T_i \text{ iff } M, i \models \varphi$$

The proof is by induction on the structure of φ:

- $\varphi = p \in \mathsf{PL}$. $p \in T_i$ iff $p \in T_i \cap \mathsf{PL}$ iff (by def. of Δ) $p \in S_i \cap \mathsf{cl}(\varphi_0)$.
- φ of the form $\neg\psi$ or $\psi \wedge \vartheta$. Easy using induction hypothesis (IH) and definition of types (Exercise).
- $\varphi = \bigcirc\psi$. $\bigcirc\psi \in T_i$ iff ($T_i \rightarrow_{\bigcirc} T_{i+1}$ by Δ) $\psi \in T_{i+1}$ iff (IH) $M, i+1 \models \psi$ iff (semantics) $M, i \models \bigcirc\psi$.
- $\varphi = \psi \, \mathcal{U} \, \vartheta$. "if". Let $M, i \models \varphi$. Then there is an $n \geq i$ such that $M, n \models \vartheta$ and $M, k \models \psi$ for $i \leq k < n$. We show by induction on j that $\psi \, \mathcal{U} \, \vartheta \in T_{n-j}$ for $j \leq n - i$. For the induction start, $M, n \models \vartheta$ and (outer) IH yield $\vartheta \in T_n$ and $T_n \rightarrow_{\bigcirc} T_{n+1}$ implies $\psi \, \mathcal{U} \, \vartheta \in T_n$. For the induction step, $M, n - j \models \psi$ and (outer) IH yield $\psi \in T_{n-j}$. Inner IH yields $\psi \, \mathcal{U} \, \vartheta \in T_{n-j+1}$, and thus $T_{n-j} \rightarrow_{\bigcirc} T_{n-j+1}$ implies $\psi \, \mathcal{U} \, \vartheta \in T_{n-j}$.

  "only if". Let $\varphi \in T_i$ and let φ be the j-th until formula. Since states from F_j appear infinitely often among $T_0 T_1 \cdots$, there is an $n \geq i$ such that $T_n \in F_j$. Let n be smallest with this property. Then, $\psi \, \mathcal{U} \, \vartheta \in T_k$ and $\vartheta \notin T_k$ for $i \leq k < n$. Using $\rightarrow_{\bigcirc}$, it is easy to show by induction on k that, for $k \in \{i, \ldots, n-1\}$:

  (i) $\psi \in T_k$;
  (ii) $\psi \, \mathcal{U} \, \vartheta \in T_{k+1}$.

  By IH, (i) yields $M, k \models \psi$ for $k \in \{i, \ldots, n-1\}$. Since $T_n \in F_j$, (ii) yields $\vartheta \in T_n$ and thus $M, n \models \vartheta$. The semantics yields $M, 0 \models \psi \, \mathcal{U} \, \vartheta$.

❑

Lemmas 5 and 14 imply that $L(\mathcal{A}_{\varphi_0}) \neq \emptyset$ iff φ_0 is satisfiable. Let $|\varphi_0|$ denote the *length* of φ_0, i.e., the number of symbols needed to write it. It is easily checked that $|\mathsf{cl}(\varphi_0)| \leq 2|\varphi_0|$, $|\mathsf{TP}(\varphi_0)| \leq 2^{2|\varphi_0|}$, and k (the number of until formulas) is bounded by $|\varphi_0|$. Thus, the automaton $\mathcal{A}_{\varphi_0}$ has at most $2^{2|\varphi_0|}$ states and $|\varphi_0|$ sets of accepting states. Together with Theorem 10, we thus get:

Theorem 15. *LTL satisfiability is decidable in* ExpTime.

This upper bound can be further improved to a PSpace one. Let us review the steps needed to decide satisfiability of an LTL formula φ_0:

1. construct automaton $\mathcal{A}_{\varphi_0}$ (exponentially large);

2. check emptyness of $\mathcal{A}_{\varphi_0}$.

Already Step 1 requires exponential space. To decide satisfiability in PSpace we give up the separation between the two steps above and construct required parts of the automaton $\mathcal{A}_\varphi$ "on the fly" without ever keeping the whole automaton in memory.

Theorem 16. *LTL satisfiability is in* PSpace.

Proof. Let φ_0 be an LTL formula with until formulas $\varphi_1 \mathcal{U} \psi_1, \ldots, \varphi_k \mathcal{U} \psi_k$ and $\mathcal{A}_{\varphi_0} = (Q, 2^{\mathsf{PL}}, I, \Delta, F_1, \ldots, F_k)$ the automaton from Definition 13. Note that we do no *construct* $\mathcal{A}_{\varphi_0}$ at this point.

By Lemma 9, $L(\mathcal{A}_\varphi) \neq \emptyset$ iff there are states $q_0, \ldots, q_k$ such that

- $q_0 \in I$

- $q_i \in F_i$ for $1 \leq i \leq k$, and

- q_{i+1} is reachable from q_i for $i \leq k$, where $q_{k+1} := q_1$.

For reachability, it obviously suffices to check paths of length at most $|Q|$. Based on this observation, non-emptyness of $\mathcal{A}_{\varphi_0}$ can be checked by the following non-deterministic algorithm:

procedure satisfiable?(φ_0)
  guess $q_0, \ldots, q_k \in Q$, i.e., subsets of $\mathsf{cl}(\varphi_0)$ satisfying the conditions of types
  if $q_0 \notin I$ (i.e. if $\varphi_0 \notin q_0$) **then**
    return "no"
  if $q_i \notin F_i$ for some i with $1 \leq i \leq k$ (i.e. if $\psi_i \notin q_i$ and $\varphi_i \,\mathcal{U}\, \psi_i \in q_i$) **then**
    return "no"
  if reachable?(q_i, q_{i+1})="no" for some i with $1 \leq i \leq k$, **then**
    return "no"
  return "yes"

procedure reachable?(q_B, q_E)
  guess number $n \leq |Q| = 2^{2|\varphi_0|}$
  $q := q_B$
  for $1 \leq i \leq n$ **do**
    guess a $q' \in Q'$ and an $a \in 2^{\mathsf{PL}}$ and check whether $(q, a, q') \in \Delta'$
      i.e. if $q = (T, i)$ and $q' = (T', i')$, then $a \cap \mathsf{cl}(\varphi_0) = T \cap \mathsf{PL}$,
        $T \to_{\bigcirc} T'$, and i' relates to i as in the proof of Lemma 8
    if the check fails **then**
      return "no"
    $q := q'$
  if $q \neq q_E$ **then**
    return "no"
  return "yes"

This algorithm needs only space polynomial in $|\varphi_0|$ since $|Q|$ is exponential in $|\varphi_0|$ and we store n, m in binary. It remains to refer to Savitch's theorem, which states that PSPACE = NPSPACE. ❏

This bound cannot be further improved. Let $LTL_{\bigcirc,\Diamond}$ be the variant of LTL that has only temporal operators $\bigcirc$ and $\Diamond$, but not $\mathcal{U}$. We will later show that $LTL_{\bigcirc,\Diamond}$ satisfiability is PSPACE-hard.

There is an interesting side-effect of the automaton $\mathcal{A}_\varphi$ and Lemma 9. We call a word $w \in \Sigma^\omega$ *ultimately periodic* if there are words $u, v \in \Sigma^*$ such that $w = uv^\omega$. An *ultimately peridoc model* of an LTL formula is an LTL structure that is an ultimately periodic word.

Theorem 17. *There is a polynomial p such that, if an LTL formula φ is satisfiable, then it has an ultimately periodic model $M = uv^\omega \in (2^{\mathsf{PL}})^\omega$ with $|u|, |v| \leq 2^{p(|\varphi|)}$.*

Proof. By Lemma 9, $L(\mathcal{A}_\varphi) \neq \emptyset$ iff there are $q_0, \ldots, q_k$ such that

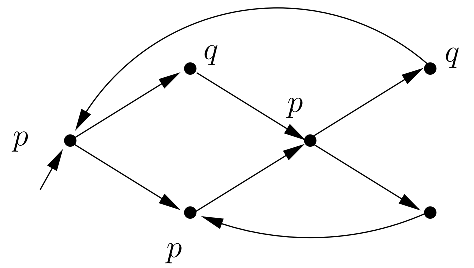
- $q_0 \in I$;
- $q_i \in F_i$ for $1 \leq i \leq k$, and
- q_{i+1} is reachable from q_i for $i \leq k$, where $q_{k+1} := q_1$.

Let $u_1, \ldots, u_k$ be the shortest words such that q_i reaches q_{i+1} with u_i for all $i \leq k$. Clearly, $|u_i| \leq |Q| = 2^{2|\varphi|}$. We have $u_0(u_1 \cdots u_k)^\omega \in L(\mathcal{A})$, $|u_0| \leq 2^{2|\varphi|}$, and $|u_1 \cdots u_k| \leq k \cdot 2^{2|\varphi|}$. ❑

2.4 Model Checking

The LTL model checking problem is as follows: given a Kripke structure $K = (S, S_I, R, V)$ and an LTL formula φ, check whether $K \models \varphi$. Recall: this is the case if all traces M of K satisfy $M, 0 \models \varphi$.

Example 18. *The following Kripke structure satisfies $\Box(q \to \bigcirc\bigcirc\bigcirc p)$. It does not satisfy $\Box(p \to p\,\mathcal{U}\,q)$.*



We can reuse the automata constructed for satisfiability to obtain an elegant approach to model checking.

Lemma 19. *If $K = (S, S_I, R, V)$ is a Kripke structure, then there exists an automaton $\mathcal{A}_K$ such that $L(\mathcal{A}_K)$ is the set of traces of K.*

Proof. Define $\mathcal{A}_K := (Q, \Sigma, I, \Delta)$ with:

- $Q = S$;
- $\Sigma = 2^{\mathsf{PL}}$;
- $I := S_I$;
- $\Delta := \{(s, S, s') \mid S = V(s) \text{ and } sRs'\}$;

❑

Note that the size of $\mathcal{A}_K$ is linear in the size of K. By Lemma 14, it is clear that $K \models \varphi$ iff

$$L(\mathcal{A}_K) \subseteq L(\mathcal{A}_\varphi). \tag{$*$}$$

As $L(\mathcal{A}_{\neg\varphi}) = (2^{\mathsf{PL}})^\omega \setminus L(\mathcal{A}_\varphi)$, $(*)$ can be rewritten as

$$L(\mathcal{A}_K) \cap L(\mathcal{A}_{\neg\varphi}) = \emptyset. \tag{$**$}$$

This suggests an algorithm for checking whether $K \models \varphi$: construct a Büchi automaton $\mathcal{A}$ such that $L(\mathcal{A}) = L(\mathcal{A}_K) \cap L(\mathcal{A}_{\neg\varphi})$ and perform an emptyness test.

Theorem 20. *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be non-deterministic Büchi-automata with alphabet Σ, $|Q_i| = n_i$ and k_i sets of accepting states, for $i \in \{1, 2\}$. Then there exists a non-deterministic Büchi automaton $\mathcal{A}$ such that $L(\mathcal{A}) = L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$ and $\mathcal{A}$ has $n_1 \cdot n_2$ states and $k_1 + k_2$ sets of accepting states.*

Proof. Let $\mathcal{A}_i = (Q_i, \Sigma, I_i, \Delta_i, F_1^{(i)}, \dots, F_{f_i}^{(i)})$, for $i \in \{1, 2\}$. Set

$$\mathcal{A} := (Q_1 \times Q_2, \Sigma, I_1 \times I_2, \Delta, F_1, \dots, F_{f_1}, F'_1, \dots, F'_{f_2})$$

where

- $\Delta = \{((q_1, q_2), a, (q'_1, q'_2)) \mid (q_i, a, q'_i) \in \Delta_i, \text{ for } i \in \{1, 2\}\}$;
- $F_i := F_i^{(1)} \times Q_2$, for $1 \leq i \leq f_1$;
- $F'_i := Q_1 \times F_i^{(2)}$, for $1 \leq i \leq f_2$.

Exercise: prove that $\mathcal{A}$ is as required. ❑

By Theorems 10 and 20, $(**)$ is decidable. Thus, it is decidable whether a Kripke structure K satisfies a formula φ.

Theorem 21. *Let $K = (S, S_I, R, V)$ be a Kripke structure with $|S| = n$, and let φ be a formula of length m. Then it can be decided in deterministic time*

$$p(n) \cdot 2^{p(m)}$$

whether $K \models \varphi$, where p is a polynomial.

Proof. Easily derived from the following:

- $\mathcal{A}_K$ has n states and no set of accepting states;
- $\mathcal{A}_{\neg\varphi}$ has $2^{2(m+1)}$ states and at most $m + 1$ sets of accepting states;
- by Lemma 20, the intersection automaton for $\mathcal{A}_K$ and $\mathcal{A}_{\neg\varphi}$ has $n \cdot 2^{2(m+1)}$ states and at most $m + 1$ sets of accepting states.
- by Theorem 10, emptyness of the intersection automaton can thus be decided in time $p(n) \cdot 2^{p(m)}$, with p a polynomial.

❑

Thus, the algorithm is exponential in the size of the property φ, but *only polynomial* in the size of K. This is a very important observation since properties tend to be short while Kripke structures may be of excessive size: a program with only 100 Boolean variables has at least 2^{100} states. This is known as the *state explosion problem.*

As in the case of satisfiability, this upper bound is not optimal: we can prove a PSpace upper bound by reducing model checking to satisfiability:

Theorem 22. *LTL model checking is in* PSpace*, i.e., there is an algorithm verifying whether $K = (S, S_I, R, V)$ satisfies φ that needs space polynomial in $|S| + |\varphi|$.*

Proof. Let $K = (S, S_i, R, V)$ be a Kripke structure and φ an LTL formula using propositional letters from PL. Assume w.l.o.g. that S and PL are disjoint. We use the elements of S as additional propositional letters. Define a formula φ_K describing all traces of K as follows:

$$
\begin{aligned}
&\bigvee_{s\in S_I} s \wedge \\
&\Box\Big(\bigvee_{s\in S} s \wedge \bigwedge_{s,s'\in S, s\neq s'} \neg(s \wedge s') \wedge \\
&\quad \bigwedge_{s\in S} \Big(s \rightarrow \Big(\bigwedge_{p\in V(s)} p \wedge \bigwedge_{p\in \mathsf{PL}\setminus V(s)} \neg p \wedge \bigvee_{s'\in S \text{ with } sRs'} \bigcirc s' \Big) \Big) \Big)
\end{aligned}
$$

Then $K \models \varphi$ iff $\psi := \varphi_K \wedge \neg\varphi$ is unsatisfiable: define $K' := (S, S_I, R, V')$ with

$$V'(s) = V(s) \cup \{s\} \text{ for all } s \in S.$$

It is easy to see that $K \models \varphi$ iff $K' \models \varphi$. Additionally,

- every model of ψ is a trace of K' violating φ;
- every trace of K' violating φ is a model of ψ.

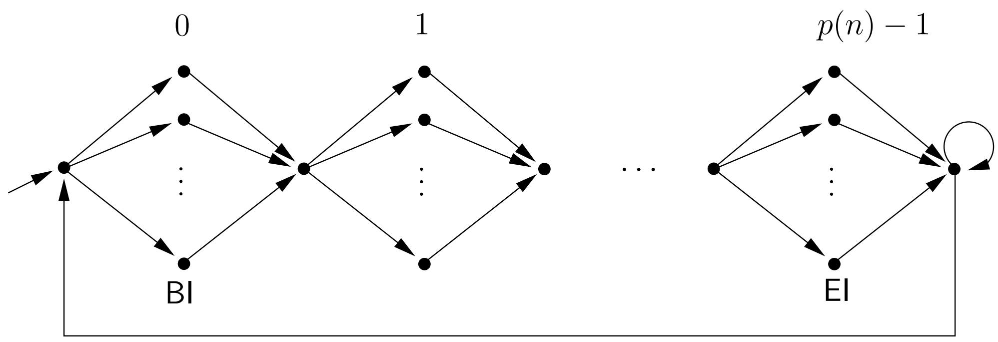
Since PSpace is closed under complement and by Theorem 16, LTL model checking is in PSpace. ❏

By the reduction, we get an algorithm whose space consumption is polynomial in the size of both the formula and the Kripke structure. In general, we cannot be better than PSpace:

Theorem 23. *$LTL_{\bigcirc,\Diamond}$ model checking is* PSpace*-hard.*

Proof. Let $M = (Q, \Sigma, \Delta, q_0, F)$ be a non-deterministic Turing machine with $\Delta \subseteq Q \times \Sigma \times Q \times \Sigma \times \{L, R\}$. Assume that M solves a PSpace-complete problem, and that there is a polynomial $p(\cdot)$ such that M consumes at most $p(n)$ tape cells if started on an input of length n.

Our goal is to construct, for each input w for M, a Kripke structure $K_{M,w}$ and a formula φ_w such that M accepts w iff $K_{M,w}$ does not satisfy $\neg\varphi_w$. Let $\mathsf{PL} := \Sigma \cup (\Sigma \times Q) \cup \{BI, EI\}$. We construct $K_{M,w} = (S, S_I, R, V)$ over PL as follows:



The vertical sequences are labelled via V with the elements of $\Sigma \cup (\Sigma \times Q)$, the intermediate points are labelled via V with the empty set, and S_I consists only of the left-most element.

Intuitively, traces $\emptyset S_0 \emptyset S_1 \emptyset S_2 \cdots$ of $K_{M,w}$ that satisfy φ_w will encode computations of M on w:

- each S_i describes one tape cell of a configuration;
- if $S_i = \{a\}$ with $a \in \Sigma$, then the tape cell is labelled a and the head is somewhere else in the current configuration;
- if $S_i = \{(a,q)\}$ with $a \in \Sigma$ and $q \in Q$, then the tape cell is labelled a, the current state is q, and the head is over this cell;
- a sequence $S_i, \ldots, S_{i+p(n)}$ with $\mathsf{BI} \in S_i$ and $\mathsf{EI} \in S_{i+p(n)}$ describes a configuration (ID) of M;
- the sequence $S_0 S_1 S_2 \cdots$ describes a computation of M.

The formula φ_w describes successful computations of M on w. It is the conjunction of the following:

- Input is $w = a_0, \ldots, a_{n-1}$, initial state is q_0, and after input there are only blanks $B \in \Sigma$:
$$\bigcirc\Big((a_0, q_0) \wedge \bigwedge_{1 \le i < n} \bigcirc^{2i} a_i \wedge \bigwedge_{n \le i < p(n)} \bigcirc^{2i} B\Big)$$
- Keeping contents of non-head tape cells: for all $a \in \Sigma$:
$$\Box\big(a \rightarrow \bigcirc^{2p(n)+1}\big(a \vee \bigvee_{q \in Q} (a,q)\big)\big)$$
- Only one head per configuration: for all i, j with $i < j < p(n)$:
$$\Box\big((BI \wedge \bigcirc^{2i+1} h) \rightarrow \bigcirc^{2j+1} \neg h\big)$$
where h abbreviates $\bigvee_{(a,q) \in \Sigma \times Q} (a,q)$.
- Making a transition. For all $a \in \Sigma$ and $q \in Q \setminus F$,
$$\Box\big((a,q) \rightarrow \bigvee_{(q,a,q',a',M) \in \Delta} \chi_{(q,a,q',a',M)}\big)$$
with
$$\begin{array}{lcl} \chi_{(q,a,q',a',R)} & := & \neg EI \wedge \bigcirc^{2p(n)+1}\big(a' \wedge \bigcirc^{2} \bigvee_{b \in \Sigma} (b, q')\ \big) \\ \chi_{(q,a,q',a',L)} & := & \neg BI \wedge \bigcirc^{2p(n)-1}\big(\ \bigvee_{b \in \Sigma} (b, q') \wedge \bigcirc^{2} a'\ \big) \end{array}$$
- The computation is accepting:
$$\Diamond \bigvee_{(a,q) \in \Sigma \times F} (a,q)$$

Now, M accepts w iff there exists a trace t of $K_{M,w}$ such that $t, 0 \models \varphi_w$ iff $K_{M,w}$ does not satisfy $\neg\varphi_w$. ❑

The same can be proved for $LTL_{\mathcal{U}}$, the variant of LTL that has $\mathcal{U}$ as the only temporal operator, but not $\bigcirc$.

Taking together Theorems 22 and 23, we obtain the following lower bound for satisfiability:

Corollary 24. *$LTL_{\bigcirc,\Diamond}$ satisfiability is* PSPACE*-hard, thus* PSPACE*-complete.*

Again, the same can be proved for $LTL_{\mathcal{U}}$.

2.5 Dropping Next and Until

For expressing some properties, the $\bigcirc$ and $\mathcal{U}$ operators are not needed. For example, none of our examples of fairness, safety, and liveness conditions uses them. Let $\mathrm{LTL}_\Diamond$ be the fragment of LTL that has the only temporal operator $\Diamond$, but neither $\bigcirc$ nor $\mathcal{U}$. Reasoning in $\mathrm{LTL}_\Diamond$ turns out to be simpler than reasoning in LTL: satisfiability in $\mathrm{LTL}_\Diamond$ is NP-complete, and model checking is co-NP-complete. Thus, reasoning in $\mathrm{LTL}_\Diamond$ is not harder than in propositional logic.

To prove that satisfiability in $\mathrm{LTL}_\Diamond$ is in NP, we first show the following.

Lemma 25. *Let φ be a satisfiable LTL formula. Then there is an LTL structure $M = uv^\omega$, $u, v \in (2^{\mathsf{PL}})^*$, such that $M, 0 \models \varphi$ and $|u| + |v| \leq |\varphi| + 1$.*

The proof of this lemma is based on the following observation. Let φ be an LTL formula, $M = S_0 S_1 \cdots$ an LTL structure such that $M, 0 \models \varphi$, and $i \in \mathbb{N}$ such that the following holds: for all $\Diamond\psi \in \mathsf{cl}(\varphi)$, $M, i \models \psi$ implies that there is a $j \geq i$ with $M, j \models \psi$. Then we can "drop" i without affecting the truth of any formulas from $\mathsf{cl}(\varphi)$ at any time point. In particular, the resulting structure $M' = S_0 \cdots S_{i-1} S_{i+1} S_{i+2} \cdots$ also satisfies $M', 0 \models \varphi$.

Proof. By Theorem 17, satisfiability of φ implies that there is an LTL structure $M = uv^\omega$, $u, v \in (2^{\mathsf{PL}})^*$, such that $M, 0 \models \varphi$. Let $|u| = m$, $|v| = n$, and $uv = S_0 \cdots S_{m+n-1}$. We select elements from this sequence to obtain new words u', v' that are of appropriate length: let $R \subseteq \{0, \ldots, n\}$ be minimal such that for all $i < m + n$,

- $0 \in R$;
- if $\Diamond\psi \in \mathsf{cl}(\varphi)$, $M, i \models \psi$, and $M, j \not\models \psi$ for all $j \in \{i+1, \ldots, m+n\}$, then $i \in R$.

Thus, R contains all the *last occurrences* of formulas that appear inside a diamond formula.

Let $i_0, \ldots, i_{m'+n'-1}$ be the elements of R in ascending order. Obviously, $m' + n' \leq |\varphi| + 1$. Now let m' be such that $i_j < m$ for all $j < m'$ and $i_j \geq m$ for all $j \geq m'$. Set $u' := S_{i_0} \cdots S_{i_{m'-1}}$, $v' := S_{i_{m'}} \cdots S_{j_{m'+n'-1}}$, and $M' := u' \cdot v'^\omega$. We can show the following by induction on the structure of ψ:

Claim. For all $\psi \in \mathsf{cl}(\varphi)$ and all $j < m' + n'$, $M', j \models \varphi$ iff $M, i_j \models \varphi$.

The only interesting case is $\psi = \Diamond\chi$. Here we exploit the regular structure of M and M' and the fact that the last occurrence of every formula inside a diamond formula is kept. Since $0 \in R$, we have $i_0 = 0$ and the claim implies $M', 0 \models \varphi$ as required. ❏

We can now prove:

Theorem 26. *Satisfiability in $LTL_\Diamond$ is in* NP.

Proof. Since propositional logic is a fragment of $\text{LTL}_\Diamond$, NP-hardness of satisfiability is immediate. For the upper bound, we use the following algorithm. By Lemmas 25 and 14, we can decide satisfiability of an LTL formula φ by checking whether $L(\mathcal{A}_\varphi)$ accepts a word $u \cdot v^\omega$ with $|u| + |v| \leq |\varphi| + 1$. To do this, guess numbers $m, n \in \mathbb{N}$ such that $m + n \leq |\varphi| + 1$, a word $w = a_1 \cdots a_{m+n}$, and a sequence of states $q_0, \ldots, q_{m+n}$. Then check whether the following holds, accept if this is the case and reject otherwise:

1. $q_0 \in I$;
2. $(q_i, a_{i+1}, q_{i+1}) \in \Delta$ for all $i \leq m + n$, where $a_{n+m+1} := a_m$ and $q_{n+m+1} := q_m$.
3. there is a $j \in \{m, \ldots, n\}$ such that $q_j \in F_i$, for all $i \in \{1, \ldots, k\}$;

It is easy to see that Properties 1 to 3 can be checked in polynomial time without computing the automaton $\mathcal{A}_\varphi$ explicitly. ❏

Containment in co-NP of model checking can be proved by showing that the complement of this problem is in NP. This, in turn, can be done by adapting the proofs of Lemma 25 and Theorem 26 to the intersection automaton of $\mathcal{A}_K$ and $\mathcal{A}_\varphi$. Co-NP-hardness can be shown by a reduction from 3SAT which is similar to the proof of Theorem 23.

2.6 LTL and ω-regular Languages

We have seen that many relevant properties can be formalized in LTL. Are there properties that LTL cannot capture? And if so, how can we characterize the properties that *can* be described?

Let $M(\varphi)$ denote the set of LTL structures M such that $M, 0 \models \varphi$.

Theorem 27. *There is no LTL-formula such that*

$$M(\varphi) = EVEN(p) := \{S_0 S_1 \cdots \mid p \in S_i \textit{ for all even } i \in \mathbb{N}\}.$$

Proof. For $i \geq 0$ and φ an LTL-formula, let $M_i := \{p\}^i \emptyset \{p\}^\omega$ be the LTL structure that makes p true everywhere except at position i, let $V_i(\varphi)$ denote the truth value of φ at Point 0 of M_i, and let n_φ be the number of $\bigcirc$-operators in φ.

Claim: For all LTL formulas φ and all $i, j \in \mathbb{N}$ with $n_\varphi < i \leq j$, $V_i(\varphi) = V_j(\varphi)$.

Intuitively: if the $\neg p$ position is further away than there are $\bigcirc$-operators in φ, then φ does not know how far exactly it is away.

Assume the claim has been proved. Assume to the contrary of what is to be shown that there is an LTL formula φ with $M(\varphi) = EVEN(p)$. By the claim, we have $V_{2n_\varphi}(\varphi) = V_{2n_\varphi+1}(\varphi)$: contradiction to $M(\varphi) = EVEN(p)$ since $M_{2n_\varphi} \notin EVEN(p)$ and $M_{2n_\varphi+1} \in EVEN(p)$.

The proof of the claim is by induction on the structure of φ:

- φ is a propositional letter: immediate;
- φ is $\psi \wedge \chi$ or $\neg\psi$: easy using IH;
- φ is $\bigcirc\psi$. We have $V_i(\varphi) = V_{i-1}(\psi)$ and $V_j(\varphi) = V_{j-1}(\psi)$ since M_{i-1} is the substructure of M_i starting at Point 1, and likewise for M_{j-1}. Since $n_\psi = n_\varphi - 1$, IH yields $V_{i-1}(\psi) = V_{j-1}(\psi)$, and thus $V_i(\varphi) = V_j(\varphi)$.
- φ is $\psi\,\mathcal{U}\,\chi$. By applying the equivalence

  $$\varphi \equiv \chi \vee (\psi \wedge \bigcirc\psi\,\mathcal{U}\,\chi),$$

  $i - n_\varphi$ times, we get

  $$\begin{aligned} \varphi \;=\; & \chi \vee \\ & (\psi \wedge (\bigcirc\chi \vee \\ & \quad (\bigcirc\psi \wedge (\bigcirc^2\chi \vee \\ & \qquad (\bigcirc^2\psi \wedge \cdots \wedge (\bigcirc^{(i-n_\varphi)-1}\chi \vee \\ & \qquad\quad (\bigcirc^{(i-n_\varphi)-1}\psi \wedge \bigcirc^{(i-n_\varphi)}(\psi\,\mathcal{U}\,\chi)) \cdots))))) \end{aligned}$$

  We can thus describe $V_i(\varphi)$ as follows:

  $$\begin{aligned} V_i(\varphi) \;=\; & \underline{V_i(\chi)} \vee \\ & (\underline{V_i(\psi)} \wedge (\underline{V_{i-1}(\chi)} \vee \\ & \quad (\underline{V_{i-1}(\psi)} \wedge \cdots \wedge (\underline{V_{n_\varphi+1}(\chi)} \vee \\ & \qquad (\underline{V_{n_\varphi+1}(\psi)} \wedge V_{n_\varphi}(\psi\,\mathcal{U}\,\chi)) \cdots)))) \end{aligned}$$

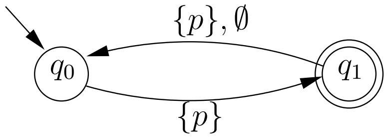
  By IH, all red (upper underlined) and blue (lower underlined) truth values are identical. Thus, the above expression simplifies to

  $$V_i(\varphi) = V_{n_\varphi+1}(\chi) \vee (V_{n_\varphi+1}(\psi) \wedge V_{n_\varphi}(\psi\,\mathcal{U}\,\chi)) \qquad (*)$$

  Exactly the same argument can be used that $V_j(\varphi)$ is equal to the right-hand side of $(*)$. Thus $V_i(\varphi) = V_j(\varphi)$.

❏

On the other hand, there is a simple Büchi-automaton expressing $EVEN(p)$: let $\mathsf{PL} = \{p\}$, $\Sigma = 2^{\mathsf{PL}}$, and $F_1 = \{q_1\}$:



The languages recognizable by a Büchi-automaton are called *ω-regular*. Together with Lemma 14, we get:

Theorem 28. *Büchi automata are strictly more expressive than LTL:*

1. *For every LTL formula φ, $M(\varphi)$ is ω-regular;*
2. *There are ω-regular languages $L \subseteq (2^{\mathsf{PL}})^\omega$ such that $L \neq M(\varphi)$ for all LTL formulas φ.*

2.7 Connection to First-Order Logic

LTL to First-Order

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.

We may interprete 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 letters.

We write $\varphi(x_1, \ldots, x_n)$ to indicate that the variables in the $FO_<$ formula φ are $x_1, \ldots, x_n$. For an $\mathrm{FO}_<$ formula $\varphi(x_1, \ldots, x_n)$, an LTL structure M, and $n_1, \ldots, n_k \in \mathbb{N}$, we write $M \models \varphi[n_1, \ldots, n_k]$ if φ is true in M with variable x_i bound to value n_i, for $1 \leq i \leq k$.

For example, consider the following formulas:

- For
$$\varphi(x_1, x_2) = \neg p(x_1) \wedge p(x_2) \wedge \forall x_3.(x_1 < x_3 \to \neg p(x_3),$$
we have $\emptyset\{p\}^\omega \models \varphi[0, 1]$.
- The following formula $\psi(x)$ expresses that there exists a future point that agrees with the current point (identified by the free variable) on the unary predicates $p_1, \ldots, p_n$:
$$\psi(x) = \exists y.\Big(x < y \wedge \bigwedge_{1 \leq i \leq n} (p_i(x) \leftrightarrow p_i(y))\Big)$$

We say that an $FO_<$ formula $\varphi(x)$ with exactly one free variable is *equivalent* to an LTL formula ψ if for all LTL models M and $n \in \mathbb{N}$, we have $M, n \models \psi$ iff $M \models \varphi[n]$.

Theorem 29. *For every LTL formula φ, there exists an equivalent $FO_<$ formula.*

Proof. The following translation takes LTL formulas φ to equivalent $FO_<$ formulas $\varphi^*(x)$:

$$\begin{array}{rcl}
p^x &=& p(x) \\
(\neg\varphi)^x &=& \neg\varphi^x \\
(\varphi \wedge \psi)^x &=& \varphi^x \wedge \psi^x \\
(\bigcirc\varphi)^x &=& \exists y.(\mathsf{suc}(x, y) \wedge \varphi^y) \\
(\varphi\,\mathcal{U}\,\psi)^x &=& \exists y.(x < y \wedge \psi^y \wedge \forall z.(x \leq z < y \to \chi^z))
\end{array}$$

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

So we cannot say more in LTL than in first-order logic. What about the converse? Is there an LTL formula that is equivalent to the $\mathrm{FO}_<$ formula $\psi(x)$ above? (yes, there is. Exercise: find it). In general, are there $FO_<$ formulas $\varphi(x)$ 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. Proof: by Lemma 5, every satisfiable LTL formula is satisfiable in some LTL structure M at time point 0. The formula above is satisfiable, but not at time point 0.

Past Operators

When we want to compare $FO_<$ with LTL, we should extend the latter with *past-time temporal operators* $\bigcirc^- \varphi$ and $\varphi \mathcal{S} \psi$, whose semantics is:

$$\begin{array}{lll} M, n \models \bigcirc^- \varphi & \text{iff} & n > 0 \text{ and } M, n-1 \models \varphi \\ M, n \models \varphi \mathcal{S} \psi & \text{iff} & \exists m \leq n : M, m \models \psi \wedge M, k \models \varphi \text{ for all } k \in \{m+1, \ldots, n\} \end{array}$$

The abbreviations $\Diamond^- \varphi$, $\Box^- \varphi$, etc. are defined as in the future-time case. This variant of LTL is called *LTL with past operators (LTLP)*. As an example, consider the following property:

$$\varphi := \Box(\Diamond^- \mathsf{executed}_i \rightarrow \Diamond \mathsf{executed}_i)$$

It states that $\mathsf{executed}_i$ is false everywhere or true infinitely often.

First-order to LTLP

In LTLP, we find a formula that is equivalent to $\exists y.y < x$, namely $\bigcirc^- \top$. Indeed, LTLP has the same expressive power of $\mathrm{FO}_<$. The following is one of the most classical results in temporal logic.

Theorem 30 (Kamp). *For every $FO_<$ formula with one free variable, there exists an equivalent LTLP formula.*

A proof of this result is out of scope. Instead, we consider the fragment $LTLP_{\bigcirc,\Diamond}$ of $LTLP$ that has $\bigcirc\varphi$, $\bigcirc^-\varphi$, $\Diamond\varphi$, and $\Diamond^-\varphi$ as the only temporal operators. Formulas of $LTLP_{\bigcirc,\Diamond}$ can be translated into $FO^2_<$ formulas, i.e., $FO_<$ formulas that use only two variables x and y: we proceed as in the proof of Theorem 31, translating the temporal operators as follows:

$$\begin{array}{rcl} (\bigcirc\varphi)^x & = & \exists y.(\mathsf{suc}(x,y) \wedge \varphi^y) \\ (\bigcirc^-\varphi)^x & = & \exists y.(\mathsf{suc}(y,x) \wedge \varphi^y) \\ (\Diamond\varphi)^x & = & \exists y.(x < y \wedge \varphi^y) \\ (\Diamond^-\varphi)^x & = & \exists y.(y < x \wedge \varphi^y) \end{array}$$

Instead of introducing new variables for every temporal operator, we simply alternate between the two available variables x and y. For example,

$$\Diamond(p \wedge \Diamond^-(q \wedge \neg\bigcirc p))$$

is translated to

$$\exists y.(x < y \wedge p(y) \wedge \exists x.(x < y \wedge q(x) \wedge \neg(\exists y.(\mathsf{suc}(x,y) \wedge p(y))))).$$

It can be shown that the until operator cannot be translated to $FO^2_<$.

We show the following, restricted version of Kamp's theorem:

Theorem 31. *For every $FO^2_<$ formula with one free variable, there exists an equivalent $LTLP_{\bigcirc,\diamond}$-formula.*

The proof of this result is much easier than that of Theorem 32. We first introduce the following:

- An $FO^2_<$-formula $\rho(x,y)$ is called a *binary atom* if it is an atom with both x and y free variables, i.e. $x < y$, $x = y$, $\mathsf{suc}(x,y)$, and the same with variables swapped;
- A formula γ is called a *unary atom* if it is an atom with one free variable, i.e., $x = x$, $x < x$, $\mathsf{suc}(x,x)$, $p(x)$, and the same with x replaced by y.

Proof. Let $\varphi(x) \in FO^2_<$. We assume $\varphi(x)$ is built using $\exists$, $\wedge$, and $\neg$ only. We inductively define two mappings $\cdot^x$ and $\cdot^y$ where $\cdot^x$ takes each $FO^2_<$-formula $\varphi(x)$ to the corresponding $\text{LTLP}_{\bigcirc,\diamond}$-formula φ^x and $\cdot^y$ does the same for $FO^2_<$-formulas $\varphi(y)$. We only give the details of $\cdot^x$ since $\cdot^y$ is defined symmterically by switching the roles of x and y.

Case 1. If $\varphi(x) = p(x)$, then $\varphi^x = p$.

Case 2. If $\varphi(x) = x < x$, then $\varphi^x = \bot$.

Case 3. If $\varphi(x) = \mathsf{suc}(x,x)$, then $\varphi^x = \bot$.

Case 4. If $\varphi(x) = x = x$, then $\varphi^x = \top$.

Case 5. If $\varphi(x) = \neg\chi$, then $\varphi^x = \neg\chi^x$.

Case 6. If $\varphi(x) = \chi_1 \wedge \chi_2$, then we can assume w.l.o.g. that x occurs free in both χ_1 and χ_2. Put $\varphi^x = \chi_1^x \wedge \chi_2^x$.

Case 7. If $\varphi(x) = \exists y.\chi(x,y)$, then $\chi(x,y)$ can be written as a Boolean combination

$$\chi(x,y) = \gamma[\rho_1,\ldots,\rho_r,\gamma_1(x),\ldots,\gamma_l(x),\beta_1(y),\ldots,\beta_s(y)],$$

of formulas ρ_i, $\gamma_i(x)$, and $\beta_i(y)$, where

- the ρ_i are binary atoms;
- the $\gamma_i(x)$ are unary atoms or of the form $\exists y.\gamma_i'$;
- and the $\beta_i(y)$ are unary atoms or of the form $\exists x.\beta_i'$.

We may assume w.l.o.g. that x occurs free in all γ_i and y occurs free in all β_i. Observe that no other free variable can occur in these formulas.

Our first step is to move the formulas $\gamma_1, \ldots, \gamma_\ell$ (which do not have y as a free variable) out of the scope of $\exists$: $\varphi(x)$ is equivalent to

$$\bigvee_{\langle t_1,\ldots,t_\ell\rangle \in \{\top,\bot\}^\ell} \Big(\bigwedge_{1 \le i \le \ell} (\gamma_i \leftrightarrow t_i) \wedge \exists y.\gamma(\rho_1, \ldots, \rho_r, t_1, \ldots, t_l, \beta_1, \ldots, \beta_s)\Big). \tag{1}$$

Now, let Γ be the set containing the following formulas:

1. $x < y \wedge \neg\mathsf{suc}(x, y)$;
2. $\mathsf{suc}(x, y)$;
3. $x = y$;
4. $\mathsf{suc}(y, x)$;
5. $y < x \wedge \neg\mathsf{suc}(y, x)$.

These formulas give rise to five cases for the relation between x and y in χ. These cases are exhaustive and mutually exclusive. Each case uniquely determines the truth of each binary atom ρ_i. For example, Case 1 makes $x < y$ true and $x = y$ false and Case 4 makes $y < x$ true and $\mathsf{suc}(x, y)$ false. Given $\sigma \in \Gamma$ and a binary atom ρ_i, let ρ_i^σ be the truth value of ρ_i in case σ, i.e., either $\top$ or $\bot$. Now we can move the binary atoms inside χ so that they appear directly after the quantifier: $\varphi(x)$ is equivalent to

$$\begin{aligned}\bigvee_{\langle t_1,\ldots,t_\ell\rangle \in \{\top,\bot\}^\ell} \Big(&\bigwedge_{1 \le i \le \ell} (\gamma_i \leftrightarrow t_i) \wedge \\ &\bigvee_{\sigma \in \Gamma} \exists y(\sigma \wedge \gamma(\rho_1^\sigma, \ldots, \rho_r^\sigma, t_1, \ldots, t_l, \beta_1, \ldots, \beta_s))\Big).\end{aligned} \tag{2}$$

The quantifier followed by a binary atom is equivalent to an LTL operator: for $\sigma \in \Gamma$, set

$$\mathsf{op}_\sigma \varphi := \begin{cases} \bigcirc^- \bigcirc^- \Diamond^- \varphi & \text{if } \sigma = x < y \wedge \neg\mathsf{suc}(x, y) \\ \bigcirc^- \varphi & \text{if } \sigma = \mathsf{suc}(x, y) \\ \varphi & \text{if } \sigma \text{ is } x = y \\ \bigcirc \varphi & \text{if } \sigma = \mathsf{suc}(y, x) \\ \bigcirc\bigcirc\Diamond\varphi & \text{if } \sigma = y < x \wedge \neg\mathsf{suc}(y, x) \end{cases}$$

The remaining formulas $\beta_1, \ldots, \beta_s$ in the scope of $\exists$ have y as only free variable. The formulas $\gamma_1, \ldots, \gamma_\ell$ have x as only free variable. Hence we can recursively compute γ_i^x and β_i^y. Now we define $\varphi(x)^\sigma$ as

$$\bigvee_{\langle t_1,\ldots,t_\ell\rangle \in \{\top,\bot\}^\ell} \Big(\bigwedge_{1 \le i \le \ell} (\gamma_i^x \leftrightarrow t_i) \wedge \bigvee_{\sigma \in \Gamma} \mathsf{op}_\sigma \gamma(\rho_1^\sigma, \ldots, \rho_r^\sigma, t_1, \ldots, t_l, \beta_1^y, \ldots, \beta_s^y)\Big).$$

❑

Some further remarks on LTLP and Theorem 32:

- It follows from Theorems 32 and 27 that, just like LTL, $FO_<$ cannot express $EVEN(p)$. Logics that are able to define all ω-regular properties include the temporal μ-calculus and the monadic *second*-order theory of $(\mathbb{N}, <)$.

- Although LTLP and $FO_<$ have the same expressive power, they differ in computational complexity and succinctness:

  1. by extending our techniques for LTL, one can show that satisfiability in LTLP is PSpace-complete. Thus, introducing past operators comes at no additional costs.
  2. there exists no bound $k \in \mathbb{N}$ such that satisfiability of $FO_<$ formulas can be decided in time

     $$2^{2^{\cdot^{\cdot^{\cdot^{2^n}}}}} \quad \text{(with } k \text{ height of the tower).}$$

     In other words, the complexity is *non-elementary.*

  There is also a non-elementary gap in formula length between LTLP and $FO_<$.

- Let two LTLP formulas φ and ψ be *initially equivalent* iff for all LTL structures M, we have $M, 0 \models \varphi$ iff $M, 0 \models \psi$. Altough introducing past operators trivially increases the expressive power regarding standard equivalence, this is not the case for initial equivalence: it can be shown that for every LTLP formula there is an LTL formula that is initially equivalent. Observe that initial equivalence is relevant for model checking.

3 Branching Time Logic: CTL

When doing model checking, we effectively use LTL in a branching time environment: every state in a Kripke structure 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 has more than a single computation. Branching time logics allow us to explicitly talk about such branches in time.

3.1 CTL

Definition 32. The class of *computational tree logic (CTL) formulas* is the smallest set such that

- each propositional letter $p \in \mathsf{PL}$ is a formula;

- if φ and ψ are formulas, then so are $\neg\varphi$ and $\varphi \wedge \psi$;

- if φ and ψ are formulas, then so are $\mathsf{A}\bigcirc\varphi$, $\mathsf{E}\bigcirc\varphi$, $\mathsf{A}(\varphi\,\mathcal{U}\,\psi)$, and $\mathsf{E}(\varphi\,\mathcal{U}\,\psi)$.

$\triangle$

The symbols A and E are called *path quantifiers*. Apart from the Boolean abbreviations, we use:

$$\begin{array}{rcl} \mathsf{A}\Diamond\varphi & \text{for} & \mathsf{A}(\top\,\mathcal{U}\,\varphi) \\ \mathsf{E}\Diamond\varphi & \text{for} & \mathsf{E}(\top\,\mathcal{U}\,\varphi) \\ \mathsf{A}\Box\varphi & \text{for} & \neg\mathsf{E}\Diamond\neg\varphi \\ \mathsf{E}\Box\varphi & \text{for} & \neg\mathsf{A}\Diamond\neg\varphi \end{array}$$

Note that formulas such as $\mathsf{E}\neg\Diamond\neg p$ and $\mathsf{E}(\Box q \wedge \Diamond p)$ are not CTL formulas.

The semantics of CTL is based on Kripke structures.

Definition 33. Let $K = (S, S_I, R, V)$ be a Kripke structure. We define satisfaction of CTL formulas in K at states $s \in S$ as follows:

$$\begin{array}{rcl} K, s \models p & \text{iff} & p \in V(s),\ \text{for all } p \in \mathsf{PL} \\ K, s \models \neg\varphi & \text{iff} & K, s \models \varphi \text{ is } \textit{not} \text{ the case} \\ K, s \models \varphi \wedge \psi & \text{iff} & K, s \models \varphi \text{ and } K, s \models \psi \\ K, s \models \mathsf{E}\bigcirc\varphi & \text{iff} & K, t \models \varphi \text{ for some } t \in S \text{ with } sRt \\ K, s \models \mathsf{A}\bigcirc\varphi & \text{iff} & K, t \models \varphi \text{ for all } t \in S \text{ with } sRt \\ K, s \models \mathsf{A}(\varphi\,\mathcal{U}\,\psi) & \text{iff} & \text{for all computations } s_0 s_1 \cdots \text{ of } K \text{ with } s_0 = s, \text{ there is} \\ & & \text{an } m \geq 0 \text{ such that } K, s_m \models \psi \text{ and } K, s_k \models \varphi \text{ for all } k < m \\ K, s \models \mathsf{E}(\varphi\,\mathcal{U}\,\psi) & \text{iff} & \text{there exists a computation } s_0 s_1 \cdots \text{ of } K \text{ with } s_0 = s \\ & & \text{such that there is an } m > 0 \text{ with } K, s_m \models \psi \text{ and } K, s_k \models \varphi \text{ for all } k < m \end{array}$$

$\triangle$

Some example formulas in CTL:

- $\mathsf{E}\Diamond(A = 2 \wedge B = 2)$. It is possible to reach a state where both processes are in the critical section.
- $\mathsf{A}\Box(\mathsf{enabled}_1 \vee \cdots \vee \mathsf{enabled}_k)$: freedom from deadlocks (a safety property);
- $\mathsf{A}\Box(\mathsf{Req} \rightarrow \mathsf{A}\Diamond\mathsf{Ack})$: every request will eventually be acknowledged (a liveness property);
- $\mathsf{A}\Box(\mathsf{A}\Diamond\mathsf{enabled}_i)$: process i is enabled infinitely often on every computation path (unconditional fairness)
- $\mathsf{A}\Box(\mathsf{E}\Diamond\mathsf{Restart})$: from every state it is possible to get to a restart state.

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). More formally, let $K = (S, S_I, R, V)$ be a Kripke structure with $S_I = \{\hat{s}\}$. We define a tree-shaped Kripke structure $T(K) = (S', S_I, R', V')$ as follows:

- S' is the set of all finite computations of K, i.e.,

$$S' = \{s_0 \cdots s_k \mid s_0 = \hat{s} \text{ and } s_i R s_{i+1} \text{ for all } i < k\};$$

- $R' = \{(p, p') \in S' \times S' \mid p = qs,\ p' = ps'$ for some $s, s' \in S$ with $sRs'\}$;
- $V'(p) = V(s)$ if $p = p's$ for some $p' \in \{\varepsilon\} \cup S'$ and $s \in S$.

$T(K)$ is called the *unravelling* of K. Observe that $T(K)$ has no leaves because K is total.

Intuitively, the next lemma says that CTL formulas cannot distinguish between a state in a Kripke structure and the corresponding states in the tree-shaped unravelling.

Lemma 34. *Let K be a Kripke structure, s a state of K, $p = s_0 \cdots s_k$ a state of $T(K)$ such that $s_k = s$, and φ a CTL formula. Then $K, s \models \varphi$ iff $T(K), p \models \varphi$.*

The proof is by induction on the structure of φ.

The lemma implies that if a formula is CTL satisfiable in a Kripke structure, then it is satisfiable in an (infinite) tree-shaped Kripke structure.

CTL vs LTL

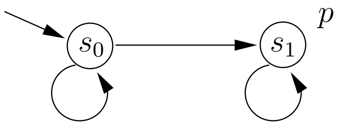
We want to compare the expressive power of LTL and CTL. To do this, we give a branching time reading to LTL formulas that is inspired by our interpretation of LTL formulas in model checking: for $K = (S, S_I, R, V)$ a Kripke structure with $s \in S$ and φ an LTL formula, we write $K, s \models \varphi$ if *all* traces M of $K' = (S, \{s\}, R, V)$ satisfy $M, 0 \models \varphi$. Thus, we view LTL formulas as implicitly universally quantified.

Now we call two LTL or CTL formulas φ and ψ *equivalent* if, for all Kripke structures K and states s of K, we have $K, s \models \varphi$ iff $K, s \models \psi$.

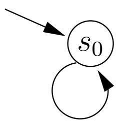
Theorem 35.

1. *For the following CTL formula, there is no equivalent LTL formula:* $\mathsf{E}\Diamond p$
2. *For the following LTL formula, there is no equivalent CTL formula:* $\Diamond\Box p$

Proof. We only prove (1). Assume that there is an LTL formula φ that is equivalent to $\mathsf{E}\Diamond p$. Consider the following Kripke structure K:



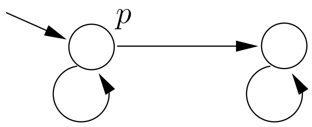
Then we have $K, s_0 \models \varphi$. Thus, every trace of K is a model of φ. Now consider K' defined as follows:



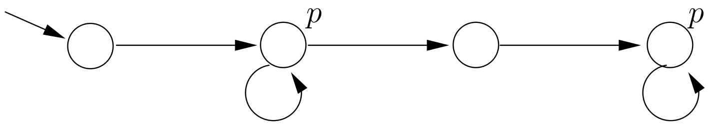
Then every trace of K' is also a trace of K. It follows that $K', s_0 \models \varphi$, in contradiction to $K', s_0 \not\models \mathsf{E}\Diamond p$. ❏

Regarding Point 2 of Lemma 37, let us consider two candidate CTL formulas for equivalence with $\Diamond\Box p$

- $\mathsf{A}\Diamond\mathsf{E}\Box p$. This formula is too weak: it is satisfied in the following structure, which does not satisfy $\Diamond\Box p$:



- $\mathsf{A}\Diamond\mathsf{A}\Box p$. This formula is too strong: it is not satisfied in the following structure, which satisfies $\Diamond\Box p$:



The formal proof of Point 2 of Lemma 37 uses structures of this shape.

We even have the following (without proof):

Theorem 36. *Let φ be a CTL formula and ψ the LTL formula obtained from φ by eliminating all path quantifiers. Then one of the following holds:*

- *φ and ψ are equivalent;*
- *there is no LTL formula that is equivalent to φ.*

To find a nice characterization of those LTL formulas that can be translated into CTL formulas is an open research problem.

3.2 Model Checking CTL

The CTL model checking problem is as follows: given a Kripke structure $K = (S, S_I, R, V)$ and a CTL formula φ, check whether K *satifies* φ, i.e., whether $K, s \models \varphi$ for all $s \in S_I$.

Let $K = (S, S_I, R, V)$ and φ_0 a CTL formula. The general idea to check whether K satisfies φ_0 is as follows:

1. arrange all subformulas ψ of φ_0 in a sequence $\psi_0, \ldots, \psi_k$ in ascending order w.r.t. formula length: for $1 \leq i < j \leq k$, ψ_i is not longer than ψ_j;
2. for all subformulas ψ of φ_0, compute the set $\mathsf{sat}_K(\psi) := \{s \in S \mid K, s \models \psi\}$ in this ordering;
3. check whether $S_I \subseteq \mathsf{sat}(\varphi_0)$.

The computation of $\mathsf{sat}(\psi_i)$ depends on the topmost operator in ψ_i.

- if ψ_i is a propositional letter, then $\mathsf{sat}(\psi_i) := \{s \in S \mid p \in V(s)\}$;
- if $\psi_i = \varphi \wedge \psi$, then $\mathsf{sat}(\psi_i) := \mathsf{sat}(\varphi) \cap \mathsf{sat}(\psi)$;
- if $\psi_i = \neg\varphi$, then $\mathsf{sat}(\psi_i) := S \setminus \mathsf{sat}(\varphi)$;
- if $\psi_i = \mathsf{E}\bigcirc\varphi$, then $\mathsf{sat}(\psi_i) := \{s \in S \mid \exists t \in S : sRt \wedge t \in \mathsf{sat}(\varphi)\}$;
- if $\psi_i = \mathsf{A}\bigcirc\varphi$, then $\mathsf{sat}(\psi_i) := \{s \in S \mid \forall t \in S : sRt \wedge t \in \mathsf{sat}(\varphi)\}$;
- if $\psi_i = \mathsf{E}(\varphi \mathcal{U} \psi)$, we use the following procedure:

```
sat(ψ_i) := T := sat(ψ)
while T ≠ ∅ do
    choose s ∈ T
    T := T \ {s}
    for all t ∈ S with tRs do
        if t ∈ sat(φ) and t ∉ sat(ψ_i) then
            sat(ψ_i) := sat(ψ_i) ∪ {t}
            T := T ∪ {t}
```

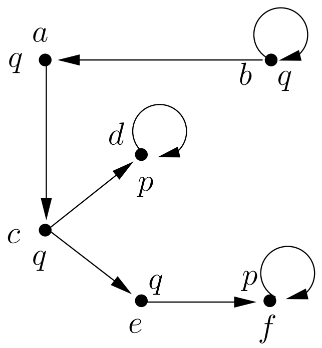
- if $\psi_i = \mathsf{A}(\varphi \mathcal{U} \psi)$, we use the following procedure:

```
sat(ψ_i) := T := sat(ψ)
while T ≠ ∅ do
    choose s ∈ T
    T := T \ {s}
    for all t ∈ S with tRs do
        flag = 1
        for all t' ∈ S with tRt' do
            if t' ∉ sat(ψ_i) then
                flag = 0
        if t ∈ sat(φ), t ∉ sat(ψ_i), and flag = 1, then
            sat(ψ_i) := sat(ψ_i) ∪ {t}
            T := T ∪ {t}
```

The following can be proved by induction on the length of the formula ψ.

Lemma 37. *For all $s \in S$ and subformulas ψ of φ_0, we have $K, s \models \psi$ iff $s \in \mathsf{sat}(\psi)$.*

Example. Consider the following Kripke structure K:



Compute the sets $\mathsf{sat}(\varphi)$ for the following φ:

- $\mathsf{E}\bigcirc p$;
- $\mathsf{E} q \,\mathcal{U}\, (\mathsf{E}\bigcirc p)$;
- $\mathsf{A} q \,\mathcal{U}\, p$.

Lemma 39 yields that CTL model checking is decidable. Concerning the complexity, we observe the following: if φ_0 is of length n, then at most n sets $\mathsf{sat}(\psi)$ need to be computed. How complex is it to compute each such set?

- ψ is a propositional letter or of the form $\psi_1 \wedge \psi_2$ or $\neg\psi'$: $\mathcal{O}(|S|)$ steps needed;
- ψ is of the form $\mathsf{E}\bigcirc\varphi$ or $\mathsf{E}(\psi \,\mathcal{U}\, \psi')$: $\mathcal{O}(|S| + |R|)$ steps needed

  the maximum cardinality of the initial set $\mathsf{sat}(\psi_i)$ is $|S|$, and, in the forall loop, each edge from R is "touched" at most once (in *all* iterations of the while);

- ψ is of the form $\mathsf{A}(\psi \,\mathcal{U}\, \psi')$: $\mathcal{O}(|S| + |R|^2)$ steps needed

  the maximum cardinality of the initial set $\mathsf{sat}(\psi_i)$ 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.

We thus obtain an upper bound of at most $n \cdot \mathcal{O}(|S| + |R|^2)$ steps. This bound can be improved to an $n \cdot \mathcal{O}(|S| + |R|)$ one by refining the procedure for computing the sets $\mathsf{sat}(\mathsf{A}(\psi \,\mathcal{U}\, \psi'))$. The new procedure uses a counter $c(s)$ for every state $s \in S$:

$\mathsf{sat}(\psi_i) := T := \mathsf{sat}(\psi)$
for all $s \in S$ do
    $c(s) = 0$
for all $(s, t) \in R$ do
    $c(s){+}{+}$
while $T \neq \emptyset$ do
    choose $s \in T$

$T := T \setminus \{s\}$
for all $t \in S$ with tRs do
  $c(t){-}{-}$
  if $t \in \mathsf{sat}(\varphi)$, $t \notin \mathsf{sat}(\psi_i)$, and $c(t) = 0$, then
    $\mathsf{sat}(\psi_i) := \mathsf{sat}(\psi_i) \cup \{t\}$
    $T := T \cup \{t\}$

Theorem 38. *There is a CTL model checking algorithm with time complexity* $|\varphi| \cdot \mathcal{O}(|S| + |R|)$.

Thus, CTL model checking is simpler than model checking LTL and $\mathrm{LTL}_{\Diamond}$! For this reason, CTL is the predominant logic implemented in industrial model checkers.

Why is CTL model checking easy? Because in some sense, the expressive power of CTL is weak. For example, it cannot express strong fairness, i.e., the LTL formula

$$\Diamond^{\infty}\mathsf{enabled} \rightarrow \Diamond^{\infty}\mathsf{executed}.$$

For suppose that there exists a CTL formula χ equivalent to the LTL formula

$$\begin{aligned} & \Diamond^{\infty} p \rightarrow \Diamond^{\infty} q \\ = \; & \Box\Diamond p \rightarrow \Box\Diamond q \\ = \; & \neg(\Box\Diamond p) \vee \Box\Diamond q \\ \equiv \; & \Diamond\Box\neg p \vee \Box\Diamond q \end{aligned}$$

Then the CTL formula $\chi[\neg\varphi/p, \bot/q]$ is equivalent to the LTL formula $\Diamond\Box\varphi$: contradiction to Lemma 37, which states that $\Diamond\Box p$ is not CTL definable.

3.3 Satisfiability in CTL

A CTL formula φ is *satisfiable* if there is a Kripke structure K and a state s of K such that $K, s \models \varphi$. Such a structure is called a *model* of φ. By Lemma 36, to check satisfiability of a formula φ it suffices to consider Kripke structures that have the form of an infinite tree.

Thus, we can use *automata on infinite trees* to decide satisfiability of CTL formulas. This can be done similarly to the LTL case, using non-deterministic Büchi tree automata. We use a different approach based on *alternating automata*. Before we can introduce these automata, we require some definitions:

- A *tree* is a non-empty prefix-closed set of words over the set of symbols $\mathbb{N}^+$ (i.e., the positive integers).
- Let Σ be an alphabet. A Σ-*labelled tree* is a pair (T, ℓ) with T a tree and $\ell : T \rightarrow \Sigma$ a labelling function. (T, ℓ) is called a Σ-*labelled* m-*tree* if T is the set of all words over $\{1, \ldots, m\}$.

- A *positive Boolean formula (PBF)* is a propositional formula built only from $\wedge$, $\vee$, true, and false.
- For a set X, we use $\mathcal{B}^+(X)$ to denote the set of all PBFs with the elements of X used as variables.
- For $n \in \mathbb{N}$, we use $[n]$ to denote the set $\{0, \dots, k\}$.

Let Σ be a finite alphabet and $m > 0$. An *alternating Büchi (tree) automaton* $\mathcal{A}$ on Σ-labelled m-trees is a tuple $\mathcal{A} = (Q, \Sigma, I, \delta, F_1, \dots, F_k)$, where all components except δ are as for non-deterministic Büchi automata on ω-words and

$$\delta : Q \times \Sigma \to \mathcal{B}^+([m] \times Q).$$

Thus, δ maps every state and symbol to a positive Boolean formula over propositional variables of the form (i, q), where $i \in \{0, \dots, m\}$ and $q \in Q$.

Suppose that the automaton reads a Σ-labelled tree (T, ℓ) and currently processes node $x \in T$ with $\ell(x) = a$. Here are some examples of what may happen:

- $\delta(q, a) = (3, q')$. The automaton proceeds to node $x \cdot 3$ and switches to state q'.
- $\delta(q, a) = (3, q') \vee (4, q'')$. This implements non-determinism: the automaton can either proceed to node $x \cdot 3$ and switch to state 3, or proceed to state $x \cdot 4$ and switch to state q''.
- $\delta(q, a) = (0, q')$. The automaton stays at node x, switching to state q'.
- $\delta(q, a) = (1, q') \wedge (4, q'')$. The automaton makes a copy of itself so that there are now two incarnations. It sends one copy to $x \cdot 1$ in state q' and the other to $x \cdot 4$ in state q''.
- $\delta(q, a) = (0, q') \vee ((1, q) \wedge (1, q''))$ shows e.g. that the automaton can send different copies of itself to the same node.
- $\delta(q, a) = \mathsf{true}$ means that this copy of the automaton terminates successfully and $\delta(q, a) = \mathsf{false}$ means that this copy of the automaton is stuck (and fails).

To formally define the behaviour of alternating automata, we need to introduce runs. We first need two additional definitions:

- Let $\varphi \in \mathcal{B}^+([m] \times Q)$ and $V \subseteq [m] \times Q$. We can view V as a valuation for the set of variables $[m] \times Q$: all variables in V are true and all others are false. We say that V *satisfies* φ if φ evaluates to true under the valuation V.
- Let $x \in \{1, 2, \dots\}^*$ and $c \in \mathbb{N}$. Then $x \cdot c$ denotes concatenation as usual with the exception that $x \cdot c = x\ x$ if $c = 0$.

Definition 39. A *run* of an alternating automaton $\mathcal{A} = (Q, \Sigma, I, \delta, F_1, \dots, F_k)$ on a Σ-labelled m-tree (T, ℓ) is a $T \times Q$-labelled tree (T_r, r). Intuitively, a node in T_r labelled (x, q) describes a copy of $\mathcal{A}$ that processes x and is in state q. (T_r, r) has to satisfy the following conditions:

- $r(\varepsilon) = (\varepsilon, q_0)$ for some $q_0 \in I$;
- Let $y \in T_r$ with $r(y) = (x, q)$ and $\delta(q, \ell(x)) = \theta$. Then there is a subset $S = \{(c_1, q_1), \ldots, (c_n, q_n)\} \subseteq [m] \times Q$ such that
  - S satisfies θ;
  - for $1 \leq i \leq n$, $y \cdot i \in T_r$ and $r(y \cdot i) = (x \cdot c_i, q_i)$.

The run (T_r, r) is *accepting* if every infinite branch of T_r contains infinitely many nodes labelled with an element of F_i for all $i \in \{1, \ldots, k\}$. The language $L(\mathcal{A})$ accepted by $\mathcal{A}$ is defined as the set of all those Σ-labelled m-trees on which there is an accepting run of $\mathcal{A}$. $\triangle$

We state the following result without proof:

Theorem 40. *The emptyness an of alternating Büchi automata is* ExpTime*-complete (in the number of states).*

We now show how to translate a CTL formula φ into an alternating automaton $\mathcal{A}_\varphi$ on 2^{PL}-labelled trees such that φ is satisfiable iff $L(\mathcal{A}_\varphi) \neq \emptyset$.

Intuitively, the automaton $\mathcal{A}_\varphi$ accepts tree-shaped Kripke. structures. Note that a tree-shaped Kripke structure $K = (S, S_I, R, V)$ can be viewed as a 2^{PL}-labelled tree in a natural way: fix an injection $f : S \to \{1, 2, \ldots\}^*$ such that

- the root of K is mapped onto ε;
- if sRs', then $f(s') = f(s) \cdot i$ for some $i > 0$.

Now, define the labelled tree corresponding to K as (T, ℓ), where $T = \mathsf{ran}(f)$ and $\ell(x) = V(f^{-1}(x))$ for all $x \in T$. In the following, we will not distinguish between K and the corresponding tree.

A *generating formula* has the form $\mathsf{E}\bigcirc\varphi$, $\mathsf{E}\varphi\,\mathcal{U}\,\psi$, $\neg\mathsf{A}\bigcirc\varphi$, or $\neg\mathsf{A}\varphi\,\mathcal{U}\,\psi$. An *until formula* has the form $\mathsf{E}\varphi\,\mathcal{U}\,\psi$ or $\mathsf{A}\varphi\,\mathcal{U}\,\psi$. Let

- φ_0 be a CTL formula,
- $\mathsf{cl}(\varphi_0)$ the set of all subformulas of φ_0 and their negations,
- m the number of generating formulas in $\mathsf{cl}(\varphi_0)$, and
- k the number of until formulas in $\mathsf{cl}(\varphi_0)$ (which we assume to be linearly ordered as usual).

The alternating automaton $\mathcal{A}_{\varphi_0} = (Q, 2^{\mathsf{PL}}, I, \delta, F_1, \ldots, F_k)$ runs on 2^{PL}-labelled m-trees since, in every state, we need at most as many successors as there are generating formulas. It is defined as

- $Q = \mathsf{cl}(\varphi_0)$;
- $I = \{\varphi_0\}$;

- The transition relation δ is defined as follows:
  - $\delta(p, a) = \mathsf{true}$ if $p \in a$
  - $\delta(p, a) = \mathsf{false}$ if $p \notin a$
  - $\delta(\varphi \wedge \psi, a) = (0, \varphi) \wedge (0, \psi)$
  - $\delta(\mathsf{E}\bigcirc\varphi, a) = \bigvee_{1 \le i \le m}(i, \varphi)$
  - $\delta(\mathsf{A}\bigcirc\varphi, a) = \bigwedge_{1 \le i \le m}(i, \varphi)$
  - $\delta(\mathsf{E}\varphi\,\mathcal{U}\,\psi, a) = (0, \psi) \vee ((0, \varphi) \wedge \bigvee_{1 \le i \le m}(i, \mathsf{E}\varphi\,\mathcal{U}\,\psi))$
  - $\delta(\mathsf{A}\varphi\,\mathcal{U}\,\psi, a) = (0, \psi) \vee ((0, \varphi) \wedge \bigwedge_{1 \le i \le m}(i, \mathsf{A}\varphi\,\mathcal{U}\,\psi))$
  - $\delta(\neg\varphi, a)$ is defined as the dual of $\delta(\varphi, a)$, i.e., by taking $\delta(\varphi, a)$ and then swapping conjunction with disjunction, swapping true with false, and negating all occurring CTL formulas.
- The set F_i, $1 \le i \le k$, is simply $\mathsf{cl}(\varphi_0)$ without the i-th until formula.

Let us illustrate duals using the formula $\neg\mathsf{E}\varphi\,\mathcal{U}\,\psi$. It is easily seen that

$$\neg\mathsf{E}\varphi\,\mathcal{U}\,\psi \equiv \neg(\psi \vee (\varphi \wedge \mathsf{E}\bigcirc\mathsf{E}\varphi\,\mathcal{U}\,\psi)) \equiv \neg\psi \wedge (\neg\varphi \vee \mathsf{A}\bigcirc\neg\mathsf{E}\varphi\,\mathcal{U}\,\psi)$$

and from the definition of duals we get

$$\delta(\neg\mathsf{E}\varphi\,\mathcal{U}\,\psi, a) = (0, \neg\psi) \wedge ((0, \neg\varphi) \vee \bigwedge_{1 \le i \le m}(i, \neg\mathsf{E}\varphi\,\mathcal{U}\,\psi)).$$

We show that the automaton $\mathcal{A}_{\varphi_0}$ is as required.

Lemma 41.

1. *If $K \in L(\mathcal{A}_{\varphi_0})$ and K has root s, then $K, s \models \varphi_0$.*
2. *If φ_0 is satisfiable, then $L(\mathcal{A}_{\varphi_0}) \neq \emptyset$.*

Proof. (sketch). For Point 1, let (T_r, r) be an accepting run of $\mathcal{A}_{\varphi_0}$ on K . We can show by induction on the structure of φ that

for all formulas φ and $y \in T_r$, if $r(y) = (x, \varphi)$ then $K, x \models \varphi$.

Since $r(\varepsilon) = (\varepsilon, \varphi_0)$, it follows that φ_0 is satisfied at the root of K. In the inductive proof, the sets F_i are used to ensure that there are no infinite delays of until formulas: such an infinite delay corresponds to a path in T_r on which from some node on, every node is labelled with the until formula.

For Point 2, let φ_0 be satisfiable. By Lemma 36, φ_0 is satisfiable in the root of a tree-shaped Kripke structure K. We can remove and duplicate states from K to get a new Kripke structure K' that has out-degree exactly m at every node and still satisfies φ_0 at the root. It is now easy to define an accepting run (T_r, r) of $\mathcal{A}_{\varphi_0}$ on K' by induction on the depth of nodes in T_r. ❏

Together with the fact that $|Q| \leq 2|\varphi_0|$ and Theorem 42, we get the following.

Theorem 42. *Satisfiability in CTL is in* ExpTime.

In contrast to LTL, this bound cannot be further improved. Using a reduction of polynomially space-bounded alternating Turing machines (see Lecture "Logic & Complexity"), we can show:

Theorem 43. *Satisfiability in CTL is* ExpTime*-hard.*

Thus, when moving from LTL to CTL, model checking got (a lot!) simpler, but satisfiability got harder.

4 Branching Time Logic: CTL*

We now define a natural branching time logic containing both CTL and LTL as fragments.

4.1 CTL*

Definition 44. The class of *computational tree logic (CTL*) state formulas* and *path formulas* are the smallest sets such that

(s1) each propositional letter $p \in \mathsf{PL}$ is a state formula;

(s2) if φ and ψ are state formulas, then so are $\varphi \wedge \psi$ and $\neg\varphi$;

(s3) if ϑ is a path formula, then $\mathsf{E}\vartheta$ is a state formula;

(p1) any state formula is a path formula;

(p2) if ϑ and χ are path formulas, then so are $\vartheta \wedge \chi$, $\neg\vartheta$, $\bigcirc\vartheta$, $\vartheta\,\mathcal{U}\,\chi$.

$\triangle$

We use the usual Boolean abbreviations, the abbreviations known from LTL (for path formulas), and write $\mathsf{A}\vartheta$ for $\neg\mathsf{E}\neg\vartheta$.

The difference between CTL and CTL* is that the latter allows more complex formulas inside path quantifiers such as Boolean combinations and nestings of temporal operators. For example, the following are CTL* formulas, but not CTL formulas:

$$\mathsf{E}(\Diamond p \wedge \Box q) \quad \mathsf{A}(\Diamond\Box p) \quad \mathsf{EA}(\Box q).$$

Definition 45. Let $K = (S, S_I, R, V)$ be a Kripke structure. For π a computation in K and $k \in \mathbb{N}$, we use π^k to denote the result of dropping the first k symbols of π.We define satisfaction of CTL* state formulas in K at states $s \in S$ and of CTL^* path formulas at computations π of K as follows:

$$
\begin{array}{rll}
K, s \models p & \text{iff} & p \in V(s), \text{ for all } p \in \mathsf{PL} \\
K, s \models \neg\varphi & \text{iff} & K, s \not\models \varphi \\
K, s \models \varphi \wedge \psi & \text{iff} & K, s \models \varphi \text{ and } K, s \models \psi \\
K, s \models \mathsf{E}\vartheta & \text{iff} & \text{there is a computation } \pi \text{ starting at } s \text{ with } K, \pi \models \vartheta \\
K, \pi \models \varphi & \text{iff} & \text{the first state } s \text{ of } \pi \text{ satisfies } K, s \models \varphi, \text{ for all state formulas } \varphi \\
K, \pi \models \neg\vartheta & \text{iff} & K, \pi \not\models \vartheta \\
K, \pi \models \vartheta \wedge \chi & \text{iff} & K, \pi \models \vartheta \text{ and } K, \pi \models \chi \\
K, \pi \models \bigcirc\vartheta & \text{iff} & K, \pi^1 \models \vartheta \\
K, \pi \models \vartheta \,\mathcal{U}\, \chi & \text{iff} & \text{there is a } k \geq 0 \text{ with } K, \pi^k \models \chi \text{ and, for all } j < k,\ M, \pi^j \models \vartheta
\end{array}
$$

△

Clearly, every CTL is a state formula of CTL*, and thus CTL is a fragment of CTL*. To compare CTL* to LTL on Kripke structures, recall that for an LTL formula φ, we have $K, s \models \varphi$ iff all traces of $K' = (S, \{s\}, R, V)$ satisfy φ. Clearly, every LTL formula φ is a CTL* path formula. It is not difficult to see that $K, s \models \varphi$ (in LTL) iff $K, s \models \mathsf{A}\varphi$ (in CTL*).

Not surprisingly, CTL* is more than just the union of LTL and CTL:

Theorem 46. *There is no LTL formula and no CTL formula equivalent to* $(\mathsf{A}\Diamond\Box p) \vee \mathsf{E}\Diamond q$.

Proof. Suppose there is a CTL formula φ equivalent to $(\mathsf{A}\Diamond\Box p) \vee E\Diamond q$. Then the formula $\varphi[\bot/q]$ is equivalent to $\mathsf{A}\Diamond\Box p$, which is not expressible in CTL by Lemma 37. For LTL, we can argue analogously. ❑

We will not address satisfiability of CTL* formulas in detail and only note that

Theorem 47. *Satisfiability in CTL* is 2-*EXPTIME *complete.*

4.2 Model Checking

The CTL* model checking problem is defined as follows: given a Kripke structure $K = (S, S_I, R, V)$ and a CTL* *state* formula φ, check whether $K, s \models \varphi_0$ for all $s \in S_I$.

We could also define model checking for path formulas ϑ. Then, we would have to say on *which* computations. The only reasonable option seems to be as in LTL model checking: we consider *all* computations starting at an initial state. But then, model checking ϑ is equivalent to model checking the state formula $\mathsf{A}\vartheta$. In the following, we only consider model checking state formulas.

The idea for checking whether K satisfies a state formula φ_0 is as follows:

- compute sets $\mathsf{sat}_K(\psi)$ for all state subformulas ψ of φ_0 as for CTL, and finally check whether $S_I \subseteq \mathsf{sat}(\varphi_0)$;
- the sets $\mathsf{sat}_K(\mathsf{E}\vartheta)$ are computed as follows:
  - we convert ϑ into an LTL formula ϑ' by replacing all outermost state subformulas $\mathsf{E}\chi$ with a fresh propositional letter $p_{\mathsf{E}\chi}$;
  - if $p_{\mathsf{E}\chi}$ occurs in ϑ', then $\mathsf{sat}_K(\mathsf{E}\chi)$ has already been computed;
  - we can use LTL model checking to compute the states that satisfy the formula $\neg\vartheta'$ using $\mathsf{sat}_K(\mathsf{E}\chi)$ as the set of states where $p_{\mathsf{E}\chi}$ is true;
  - LTL formulas are implicitly universally quantified. Thus, $\mathsf{sat}_K(\mathsf{E}\vartheta)$ is the complement of the computed set of states (because $\mathsf{E}\vartheta \equiv \neg\mathsf{A}\neg\vartheta$).

Formally, arange the state subformulas of φ_0 in a sequence $\psi_0, \ldots, \psi_k$ in ascending order w.r.t. formula length and compute the sets $\mathsf{sat}_K(\psi_i)$ in this order. The cases where ψ_i is a propositional letter or of the form $\neg\varphi$ or $\varphi \wedge \psi$ are as in CTL. For the case $\psi_i = \mathsf{E}\vartheta$, use the following procedure:

if ψ_i is a CTL formula then
    apply CTL model checking to ψ_i and K to compute $\mathsf{sat}(\psi_i)$
else
    let $\mathsf{E}\chi_1, \ldots, \mathsf{E}\chi_k$ be all outermost state subformulas of ϑ of this form
    $\vartheta' := \vartheta[p_{\mathsf{E}\chi_1}/\mathsf{E}\chi_1, \ldots, p_{\mathsf{E}\chi_k}/\mathsf{E}\chi_k]$
    $V'(s) := V(s) \cup \{p_{\mathsf{E}\chi_i} \mid 1 \leq i \leq k \text{ and } s \in \mathsf{sat}_K(\mathsf{E}\chi_i)\}$
    $\mathsf{sat}(\psi_i) := \{s \in S \mid (S, \{s\}, R, V') \not\models_{\text{LTL}} \vartheta'\}$

Correctness (in the sense of Lemma 39) is easily proved. To analyze the runtime requirements of this algorithm, let $|S| = n$ and $|\varphi_0| = m$. Then

- there are at most m subformulas;
- every sat set $\mathsf{sat}(\varphi)$ with φ a propositional letter or of the form $\neg\psi$ or $\psi \wedge \psi'$ can be computed in time $\mathcal{O}(n)$;
- LTL model checking requires time $p(n) \cdot 2^{p(m)}$, where p is a polynomial (Theorem 21).

We thus get the following.

Theorem 48. *There is a CTL* model checking algorithm with time complexity* $p(|S|) \cdot 2^{p(|\varphi|)}$*, with* p *a polynomial.*

Thus, CTL* model checking is not more difficult as LTL model checking: polynomial in the size of K and exponential in the length of φ. As in the case of LTL, this result can be improved to a PSpace one.