Linear-time Temporal Logic

Pedro Cabalar

Department of Computer Science
University of Corunna, SPAIN
cabalar@udc.es

2013/2014
Propositional Linear-time Temporal Logic (LTL)

Syntax

- \( \Sigma = \) set of atoms or propositions. Example: \( \Sigma = \{ p, q, r \} \)
- usual propositional operators \( \bot, \top, \land, \lor, \neg, \rightarrow, \leftrightarrow \)
- plus modal operators to talk about (linear) time

Modal operators:

- unary operators:
  - \( \square = \) “forever”
  - \( \Diamond = \) “eventually”
  - \( \bigcirc = \) “next”

- binary operators:
  - \( \mathcal{U} = \) “until”
  - \( \mathcal{W} = \) “until” (weak version)
  - \( \mathcal{V} = \) “release” (dual of \( \mathcal{U} \)
Definition 1 (State)

Given a set of propositions $\Sigma$, a state $s$ is a truth valuation $s : \Sigma \rightarrow \{ \text{True}, \text{False} \}$.

It can be represented as the set of (true) atoms. Example: if $\Sigma = \{ p, q, r \}$ state $s = \{ p, r \}$ means $s(p) = \text{True}$, $s(q) = \text{False}$, $s(r) = \text{True}$.

Definition 2 (Interpretation)

An interpretation $M$ is an infinite sequence of states $s_0, s_1, s_2, \ldots$

Example:

$\{ p, q \} \quad \{ p, r \} \quad \{ q \} \quad \{ q, r \} \quad \emptyset$

\[ s_0 \quad \rightarrow \quad s_1 \quad \rightarrow \quad s_2 \quad \rightarrow \quad s_3 \quad \rightarrow \quad s_4 \quad \rightarrow \ldots \]
Definition 3 (Satisfaction)

Let $M = s_0, s_1, \ldots$ with $i \geq 0$. We say that $M, i \models \alpha$ when:

- $M, i \models p$ if $p \in s_i$ (for $p \in \Sigma$)
- $M, i \models \Box \alpha$ if $M, j \models \alpha$ for all $j \geq i$
- $M, i \models \Diamond \alpha$ if $M, j \models \alpha$ for some $j \geq i$
- $M, i \models \Diamond \alpha$ if $M, i + 1 \models \alpha$
- $M, i \models \alpha U \beta$ if there exists $n \geq i$, $M, n \models \beta$ and $M, j \models \alpha$ for all $i \leq j < n$.
- $M, i \models \alpha W \beta$ if $M, i \models \Box \alpha$ or $M, i \models \alpha U \beta$
Semantics

$\bigcirc \varphi$

$\varphi$

$\bullet \rightarrow \bullet \rightarrow \bullet \rightarrow \ldots \rightarrow \bullet \rightarrow \ldots$

$\Box \varphi$

$\varphi \quad \varphi \quad \varphi \quad \varphi$

$\bullet \rightarrow \bullet \rightarrow \bullet \rightarrow \ldots \rightarrow \bullet \rightarrow \ldots$

$\Diamond \varphi$

$\varphi$

$\bullet \rightarrow \bullet \rightarrow \bullet \rightarrow \ldots \rightarrow \bullet \rightarrow \ldots$
Semantics

- $\varphi U \psi$ = repeat $\varphi$ until (mandatorily) $\psi$

- $\varphi V \psi$ = there is a $\varphi$ before any state in which $\neg \psi$
Semantics

- $\mathbf{T} \mathbf{U} \psi = \text{repeat } \mathbf{T} \text{ until (mandatorily) } \psi$

  \[
  \begin{array}{cccccc}
  \mathbf{T} & \mathbf{T} & \mathbf{T} & \mathbf{T} & \mathbf{T} & \psi \\
  \bullet & \rightarrow & \bullet & \rightarrow & \bullet & \rightarrow & \bullet & \rightarrow & \bullet & \rightarrow & \bullet & \rightarrow & \bullet & \rightarrow & \bullet & \rightarrow & \bullet & \rightarrow & \bullet & \rightarrow & \bullet & \rightarrow & \bullet & \rightarrow & \bullet & \rightarrow \\
  \end{array}
  \]

  This is equivalent to $\Diamond \psi$.

- $\mathbf{\bot} \mathbf{V} \psi = \text{there is a } \bot \text{ before any state with } \neg \psi$

  That is, we cannot have $\neg \phi$, i.e., $\psi$ must hold forever $\Box \psi$

  \[
  \begin{array}{ccccccc}
  \psi & \psi & \psi & \psi & \psi & \psi & \psi \\
  \bullet & \rightarrow & \bullet & \rightarrow & \bullet & \rightarrow & \bullet & \rightarrow & \bullet & \rightarrow & \bullet & \rightarrow & \bullet & \rightarrow & \bullet & \rightarrow & \bullet & \rightarrow & \bullet & \rightarrow & \bullet & \rightarrow & \bullet & \rightarrow \\
  \end{array}
  \]
Some standard logical terminology

- Interpretation $M$ is a model of theory $\Gamma$, written $M \models \Gamma$, iff $M, 0 \models \alpha$ for each formula $\alpha \in \Gamma$.

- Formula $\alpha$ is inconsistent or unsatisfiable iff it has no models. $\alpha$ is a tautology or is valid iff any interpretation is a model of $\alpha$.

- $\alpha$ is a “logical consequence of” or “is entailed by” $\Gamma$, written $\Gamma \models \alpha$, iff any model of $\Gamma$ satisfies $\alpha$. Therefore, when $\Gamma = \emptyset$, what does $\models \alpha$ mean?

- Two formulas are equivalent iff they have the same models.

- LTL satisfies $\{\alpha\} \models \beta$ iff $\models \alpha \rightarrow \beta$

  In particular, $\alpha$ and $\beta$ are equivalent iff $\models \alpha \leftrightarrow \beta$. 

P. Cabalar (Department of Computer Science University of Corunna, SPAIN cabalar@udc.es)

Linear Temporal Logic 2013/2014 9 / 64
Some interesting equivalences

\[ \diamond \alpha \iff \top \cup \alpha \]  
\[ \square \alpha \iff \bot \vee \alpha \]  
\[ \square \alpha \iff \neg \diamond \neg \alpha \]  
\[ \diamond \alpha \iff \neg \square \neg \alpha \]  
\[ \square \alpha \iff \alpha \land \square \square \alpha \]  
\[ \diamond \alpha \iff \alpha \lor \square \diamond \alpha \]  
\[ \alpha \cup \beta \iff (\alpha \uhr \beta) \land \diamond \beta \]  
\[ \alpha \uhr \beta \iff (\alpha \cup \beta) \lor \square \alpha \]  
\[ \alpha \cup \beta \iff \beta \lor \alpha \land \square (\alpha \cup \beta) \]  
\[ \alpha \lor \beta \iff \neg (\neg \alpha \cup \neg \beta) \]
Some interesting equivalences

Exercise 1
Prove validity of (6) and (9).

Exercise 2
Which are the models of $\bot \cup p$? Which are the models of $(\circ p) \cup \neg p$?

Exercise 3
Define an operator $\mathcal{B}$ (“before”) so that $\alpha \mathcal{B} \beta$ means for any state in which $\beta$ will occur, then some $\alpha$ will occur before.

Exercise 4
Try to express the formula whose models satisfy: $p$ is true in all even states $0, 2, 4, \ldots$ leaving all the rest free.
Outline

1. Syntax and semantics
2. Specification with LTL
3. Model checking algorithms
4. Deductive system
5. Complexity and expressiveness
6. Semantic tableaux

P. Cabalar (Department of Computer Science University of Corunna, SPAIN cabalar@udc.es)
Linear Temporal Logic 2013/2014 13 / 64
Examples of properties specification

Figure out the meaning of these example formulas:

- $\Box((\neg \text{passport} \lor \neg \text{ticket}) \rightarrow \Box \neg \text{board})$

- $\Box(\text{requested} \rightarrow \Diamond \text{received})$

- $\Box(\text{received} \rightarrow \Box \text{processed})$

- $\Box(\text{processed} \rightarrow \Diamond \Box \text{done})$

- “It can’t be that we continually resend a request that is never done.” The statement: $\Box \text{requested} \land \Box \neg \text{done}$ should be inconsistent. That is, we should be able to derive $\Box \text{requested} \rightarrow \Diamond \text{done}$. 
An example: trains crossing

- Railroad, single rail and a road level-crossing.
- Goal: **specifying properties** to be satisfied.
- Propositions representing events
  - $a = \text{“A train is approaching”}$
  - $c = \text{“A train is crossing”}$
  - $l = \text{“The light is blinking”}$
  - $b = \text{“The barrier is down”}$
Safety properties

Safety property = something *bad* never happens = \( \square \neg \text{bad} \).

- When a train is crossing, the barrier must be down
  Solution: \( \square (c \rightarrow b) \equiv \square \neg (c \land \neg b) \)

- If a train is approaching or crossing, the light must be blinking
  Solution: \( \square (a \lor c \rightarrow l) \equiv \square \neg ((a \lor c) \land \neg l) \)

- If the barrier is up and the light is off, then no train is coming or crossing.
  Solution: \( \square (\neg b \land \neg l \rightarrow \neg a \land \neg c) \equiv \square (\neg b \land \neg l \land (a \lor c)) \)


Liveness properties

Liveness property = something \textit{initiated} eventually \textit{terminates} = 
\[ \Box(initiated \rightarrow \Diamond \text{terminates}) \]

- When a train is approaching, a train will eventually cross
  Solution: \[ \Box(a \rightarrow \Diamond c) \]

- Sometimes we can use \(\mathcal{U}\), \(\mathcal{W}\) or \(\mathcal{V}\) to propagate a condition until termination.

- When a train is approaching (and nobody is crossing), the barrier will be eventually down before it crosses (if it does so)
  Solution: \[ \Box(a \land \neg c \rightarrow \neg c \mathcal{W} b) \]

- If a train finishes crossing, the barrier will be eventually risen
  Solution: \[ \Box\neg(c \land c \mathcal{U}(\neg c \land \Box b)) \]
  \[ \equiv \Box(c \rightarrow \neg c \mathcal{V} \neg(c \rightarrow \Diamond \neg b)) \]
Infinitely often vs latching condition

- Something happens infinitely often = $\square \Diamond \text{something}$.  
  Example: The barrier is risen infinitely often = $\square \Diamond \neg b$

- The dual is a latching condition = $\Diamond \square \alpha$.  
  Example: at a given point, no more trains are approaching = $\Diamond \square \neg a$
Fairness means that if a choice holds sufficiently often, then it is taken sufficiently often. Some examples:

- **Unconditional or absolute fairness** (a.k.a. impartiality)
  every process should be executed infinitely often $\square \Diamond executed_i$

- **Strong fairness** every process enabled infinitely often should be executed infinitely often $\square \Diamond enabled_i \rightarrow \square \Diamond executed_i$

- **Weak fairness** every process permanently enabled after some point should be executed infinitely often $\Diamond \square enabled_i \rightarrow \square \Diamond executed_i$
An example (from “A primer on Model Checking”, M. Ben-Ari 2010).

Two processes P and Q may increment the value of a memory cell n using a register regP, regQ respectively.

They run concurrently. An interruption may arbitrarily stop the execution, store the register values, and then restore them on return.

Process P

```plaintext
integer regP=0;
p1: load n into regP
p2: increment regP
p3: store regP into n
p4: end
```

Process Q

```plaintext
integer regQ=0;
q1: load n into regQ
q2: increment regQ
q3: store regQ into n
q4: end
```
A simple example of concurrent program

- A **state** has \((\text{IPP}, \text{IPQ}, \text{regP}, \text{regQ}, n)\) where \(\text{IPP}\) and \(\text{IPQ}\) are the respective **instruction pointers**.

- Initial state is always \((p_1, q_1, 0, 0, 0)\). Thus, each variable can take values \(0, 1, 2\) (at most, two increments are made).

- There exist \(4 \times 4 \times 3 \times 3 \times 3 = 432\) states.

- We may build a non-deterministic finite automaton (NFDA) with all the transitions.
A simple example of concurrent program

Example of computation:

\[(p_1, q_1, 0, 0, 0) \rightarrow (p_2, q_1, 0, 0, 0)\]
\[(p_3, q_1, 1, 0, 0) \rightarrow (p_4, q_1, 1, 0, 1)\]
\[(p_4, q_2, 1, 1, 1) \rightarrow (p_4, q_3, 1, 2, 1)\]
\[(p_4, q_2, 1, 2, 2)\]

\[p_1: \text{load } n \text{ into regP} \quad q_1: \text{load } n \text{ into regQ}\]
\[p_2: \text{increment } \text{regP} \quad q_2: \text{increment } \text{regQ}\]
\[p_3: \text{store } \text{regP} \text{ into } n \quad q_3: \text{store } \text{regQ} \text{ into } n\]
\[p_4: \text{end} \quad q_4: \text{end}\]
A simple example of concurrent program

- We want to check the assertion: after termination, \( n=2 \). That is, \( p4 \land q4 \Rightarrow n = 2 \).
- We check whether there exists an execution satisfying the negation: \( p4 \land q4 \land n \neq 2 \).
- After building the automaton, we obtain a counterexample path:

\[
(p1, q1, 0, 0, 0) \rightarrow (p2, q1, 0, 0, 0) \\
(p2, q2, 0, 0, 0) \rightarrow (p3, q2, 1, 0, 0) \\
(p3, q3, 1, 1, 0) \rightarrow (p4, q3, 1, 1, 1) \\
(p4, q4, 1, 1, 1)
\]

- \( p1 \): load \( n \) into regP
- \( p2 \): increment regP
- \( p3 \): store regP into \( n \)
- \( p4 \): end

- \( q1 \): load \( n \) into regQ
- \( q2 \): increment regQ
- \( q3 \): store regQ into \( n \)
- \( q4 \): end
Real problems have finite number of states (computers deal with a finite number of bits).

But still, we deal with an unfeasible, astronomical number of cases: possible values in the memory \( \times \) possible transitions in a path the NFDA \( \times \) number of possible paths in the NFDA.

Keypoint: not all the states are reachable. In our example, from 432, fixing initial state \((p1, q1, 0, 0, 0)\) only 22 are reachable.

Model checking algorithms generate reachable states on-the-fly from initial state and property to check.

To check whether a generated state was obtained before, a hash table storing the states is used.
It uses a programming language **Promela** (PROcess MEta LAnguage) derived from Dijkstra’s Guarded Command Language-

Properties are specified either using local `assert` statements or globally using linear temporal logic (LTL).

No pointers, functions, parameters, classes, etc. Focused on concurrency and verification.
Assume we execute now our two processes in a loop of 10 times. The code using SPIN programming language, Promela, would be:

```plaintext
byte n=0, finish=0;

active [2] proctype P() {
   byte register, counter=0;
do :: counter = 10 -> break
   :: else ->
      register=n;
      register++;
      n=register;
      counter++
   od;
   finish++
}
```
(continued)

```c
active proctype WaitForFinish() {
    finish==2;
    printf("n= %d \n", n)
}
```
Model checking algorithms try to build an automaton $A_P$ that captures the program behaviour.

Given the property $\alpha$ to check, we generate a second automaton $A_{\neg \alpha}$ for its negation.

We take the intersection automaton $A_P \cap A_{\neg \alpha}$.

1. If no path, the property is satisfied
2. If we find a path, it is a counterexample

“On-the-fly” techniques [Gerth, Peled, Vardi & Wolper 95] allow detecting that a property does not hold before completely constructing the full automaton.
Explicit vs Symbolic

Two possibilities:

- **Explicit model checking**: each automaton node is an individual state. A hash table indexes all the expanded states. SPIN uses this method.

- **Symbolic model checking**: each node actually represents a set of states. Typically, each set of states is represented with a Binary Decision Diagram (BDD). SMV uses this method.
Partial order reduction: the keypoint is detecting when the ordering of interleavings is irrelevant.

Example: \( n \) processes can execute instructions \( I_1, I_2, \ldots, I_n \) in any ordering. We have \( n! \) combinations, but we can fix an arbitrary one when ordering is irrelevant for the property to check.

Bounded model checking: when we want to check if property \( \alpha \) is violated in \( k \) or fewer steps (\( k \geq 0 \) finite).

Fixing the path length \( i \leq k \) we can translate the problem to SAT (propositional satisfiability). Iterative deepening goes increasing \( i = 1, 2, \ldots, k \) until a counterexample is found or \( k \) reached.
Outline

1 Syntax and semantics
2 Specification with LTL
3 Model checking algorithms
4 Deductive system
5 Complexity and expressiveness
6 Semantic tableaux
Inference or formal proof: we make syntactic manipulation of formulae. To do so, we use:

- An initial set of formulae: axioms.
- Syntactic manipulation rules: inference rules.
- As a result of applying these rules, we go obtaining new formulae: theorems.
Notation: $\Gamma \vdash \alpha$ means that formula $\alpha$ can be derived or inferred from theory $\Gamma$.

Usually, axioms are not represented inside $\Gamma$. Thus, $\vdash \alpha$ means that $\alpha$ is a theorem (from logic $L$).

Given a language $\mathcal{L}$, a logic $L$ is a subset of $\mathcal{L}$. It can be defined:

- Semantically: $L = \{ \alpha \in \mathcal{L} | \models \alpha \}$.
- Syntactically: $L = \{ \alpha \in \mathcal{L} | \vdash \alpha \}$.

What should we expect from an inference method?

- Soundness (or correctness): if $\vdash \alpha$ then $\models \alpha$
- Completeness: if $\models \alpha$ then $\vdash \alpha$
A deductive system

We define the *LTL* deductive system as follows.

**Axioms:**

<table>
<thead>
<tr>
<th>Axiom</th>
<th>Formula</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ax0</td>
<td>$PC$</td>
<td>Any substitution instance of any Propositional Calculus tautology</td>
</tr>
<tr>
<td>Ax1</td>
<td>$\vdash \Box (\alpha \rightarrow \beta) \rightarrow (\Box \alpha \rightarrow \Box \beta)$</td>
<td>Distribution of $\Box$ over $\rightarrow$</td>
</tr>
<tr>
<td>Ax2</td>
<td>$\vdash \Diamond (\alpha \rightarrow \beta) \rightarrow (\Diamond \alpha \rightarrow \Diamond \beta)$</td>
<td>Distribution of $\Diamond$ over $\rightarrow$</td>
</tr>
<tr>
<td>Ax3</td>
<td>$\vdash \Box \alpha \rightarrow (\alpha \land \Diamond \alpha \land \Diamond \Box \alpha)$</td>
<td>Expansion of $\Box$</td>
</tr>
<tr>
<td>Ax4</td>
<td>$\vdash \Box (\alpha \rightarrow \Diamond \alpha) \rightarrow (\alpha \rightarrow \Box \alpha)$</td>
<td>Induction</td>
</tr>
<tr>
<td>Ax5</td>
<td>$\vdash \Diamond \alpha \leftrightarrow \neg \Diamond \neg \alpha$</td>
<td>Linearity</td>
</tr>
</tbody>
</table>

**Inference rules:**

<table>
<thead>
<tr>
<th>Rule</th>
<th>Premise</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>MP</td>
<td>$\vdash \alpha, \vdash \alpha \rightarrow \beta$</td>
<td>$\vdash \beta$</td>
</tr>
<tr>
<td>N</td>
<td>$\vdash \alpha$</td>
<td>$\vdash \Box \alpha$</td>
</tr>
</tbody>
</table>
A deductive system

An example of a proof

**Theorem 4 (transitivity)**

\[ \vdash \square\square p \leftrightarrow \square p \]

**Proof:**

1. \[ \vdash \square\square p \rightarrow \square p \] \hspace{1cm} Expansion
2. \[ \vdash \square p \rightarrow \square\square p \] \hspace{1cm} Expansion
3. \[ \vdash \square(\square p \rightarrow \square\square p) \] \hspace{1cm} Necessitation on 2
4. \[ \vdash \square(\square p \rightarrow \square\square p) \rightarrow (\square p \rightarrow \square\square p) \] \hspace{1cm} Induction
5. \[ \vdash \square p \rightarrow \square\square p \] \hspace{1cm} Modus Ponens on 3, 4
6. \[ \vdash \square\square p \leftrightarrow \square p \] \hspace{1cm} P.C. 1, 5

Q.E.D.
A deductive system

Derived inference rules:

- **G □**
  \[
  \frac{\Gamma \vdash \alpha \rightarrow \beta}{\Gamma \vdash \Box \alpha \rightarrow \Box \beta}
  \]
  □ – Generalization

- **G ○**
  \[
  \frac{\Gamma \vdash \alpha \rightarrow \beta}{\Gamma \vdash \Diamond \alpha \rightarrow \Diamond \beta}
  \]
  ○ – Generalization

- **Ind**
  \[
  \frac{\Gamma \vdash \alpha \rightarrow \Diamond \alpha}{\Gamma \vdash \alpha \rightarrow \Box \alpha}
  \]
  Induction

These rules can be derived from previous axioms and rules.
A deductive system

Exercises

Exercise 5

Prove the following theorems:

\[ \vdash \Box (p \land q) \leftrightarrow \Box p \land \Box q \]
\[ \vdash \Diamond (p \lor q) \leftrightarrow \Diamond p \lor \Diamond q \]

Exercise 6

Prove the theorem

\[ \vdash \Box p \lor \Box q \rightarrow \Box (p \lor q) \]

and find a counterexample for:

\[ \Box (p \lor q) \rightarrow \Box p \lor \Box q \]
Outline

1. Syntax and semantics
2. Specification with LTL
3. Model checking algorithms
4. Deductive system
5. Complexity and expressiveness
6. Semantic tableaux
In complexity theory, solving a decision problem means building an algorithm that, in a finite number of steps, answers yes or no to a given input query.

For instance, SAT (propositional satisfiability, i.e., “does a formula $\alpha$ have any model?”) is a decision problem, and its complexity class is $\text{NP}$-complete.

Other examples of $\text{NP}$-complete problems are: the Travelling Salesman problem, the Graph Coloring problem, Subset Sum problem (find non-empty subset of integers that sum 0).
Meaning of NP-completeness

A Turing Machine (TM) is a theoretical device that operates on an infinite tape with cells containing symbols in a finite alphabet (including the blank or ’0’).

The TM has a current state $S_i$ among a finite set of states (including ’Halt’), and a head pointing to the “current” cell in the tape.

It has an associated transition function that describes the next step.
Example: with scanned symbol 0 and state $q_4$, write 1, move $Left$ and go to state $q_2$. That is:

$$t(0, q_4) = (1, Left, q_2)$$
A decision problem consists in providing a given tape input and asking the Turing Machine for a final output symbol answering Yes or No.

Example: SAT = given (an encoding of) a propositional formula, does it have at least one model?

A decision problem is in complexity class \( P \) iff the number of steps carried out by the TM is polynomial on the size \( n \) of the input.
Meaning of NP-completeness

- Now, a non-deterministic Turing Machine (NDTM) is such that the transition function is replaced by a transition relation.
- We may have different possibilities for the next step.
- Example: \( t(0, q_4, 1, \text{Left}, q_2), t(0, q_4, 0, \text{Right}, q_3) \)
Meaning of NP-completeness

- **Keypoint:** An NDTM provides an affirmative answer to a decision problem when at least one of the executions for the same input answers *Yes*.

- A decision problem is in class **NP** iff the number of steps carried out by the NDTM is polynomial on the size $n$ of the input.

- For **SAT**, we can build an NDTM that performs two steps:
  1. For each atom, generate 1 or 0 nondeterministically. This provides an arbitrary interpretation in linear time.
  2. Test whether the current interpretation is a model or not.

  The sequence of these two steps takes polynomial time.
Meaning of NP-completeness

- Unsolved problem

\[ P \supseteq NP \]

- The most accepted conjecture is that \( P \subset NP \). But remains unproved.

- It is one of the 7 Millenium Prize Problems
  [http://www.claymath.org/millennium/P_vs_NP/](http://www.claymath.org/millennium/P_vs_NP/)
  The Clay Mathematics Institute designated $1 million prize for its solution!
A problem $X$ is $C$-complete, for some complexity class $C$, iff any problem $Y$ in $C$ is reducible to $X$ in polynomial-time.

A complete problem is a representative of the class. Example: if an $NP$-complete problem happened to be in $P$ then $P = NP$.

$SAT$ was the first problem to be identified as $NP$-complete (Cook’s theorem, 1971).

$SAT$ is commonly used nowadays for showing that a problem $X$ is at least as complex as $NP$. To this aim, just encode $SAT$ into $X$. 

P. Cabalar (Department of Computer Science University of Corunna, SPAIN cabalar@udc.es)
LTL-satisfiability is PSPACE-complete

Theorem 5

[Halpern & Reif 1981], [Sistla & Clarke, 1982]
LTL-satisfiability is **PSPACE-complete**.

- **PSPACE** is the set of decision problems that can be solved by a Turing Machine using a polynomial amount of space (for a finite, unlimited time).

- There is no difference when the machine is non-deterministic: **NPSPACE = PSPACE** [Savitch 1970].

- On the other hand, **NP ⊆ PSPACE**. Again, unsolved question **NP ⊈ PSPACE** but strongly suspected to be ≠.

- Other **PSPACE**-complete problems are: Quantified Boolean Formula satisfiability, AI-Planning (STRIPS) existence of plan.
An finite state machine or finite automaton is a tuple \((Q, A, \delta, q_0, F)\) where

- \(Q\) is a finite set of states
- \(A\) is a finite set called the alphabet
- \(\delta : Q \times A \rightarrow Q\) is the transition function
- \(q_0\) is the initial state
- \(F\) is the set of accepting or final states

Example: this automaton recognizes words containing an even number of 0’s
ω-automata are a variation where the accepted language consists of words of infinite length. They define different acceptance conditions (when we consider a word to be “accepted”)

A Büchi automaton (BA) is an ω-automaton with the acceptance condition:

There is some run that visits (at least) one of the states in $F$ infinitely often

Example: this automaton recognizes the language $(0 + 1)^*0^\omega$
During model checking, LTL properties are translated into “equivalent” BA’s.

By equivalent we mean they recognize the same language. The BA alphabet $A$ corresponds to the set of possible LTL states.

Example: if the formula uses atoms $\Sigma = \{p, q\}$ then $A = 2^\Sigma = \{\emptyset, \{p\}, \{q\}, \{p, q\}\}$

Usually, each BA arc is labelled with a set of states that yield the same transition. This set of states is actually represented as an LTL formula.
A language accepted by a non-deterministic BA is called regular \( \omega \)-language.

An important restriction: LTL is less expressive than Büchi automata.

For instance, Exercise 4 (make \( p \) true in even states and free in all the rest) cannot be represented in LTL whereas it is accepted by the Büchi automaton:

Other temporal logics do cover regular \( \omega \)-languages.
Semantic tableaux

- For simplicity, we assume $\alpha \rightarrow \beta \overset{\text{def}}{=} \neg \alpha \lor \beta$ and $\alpha \leftrightarrow \beta \overset{\text{def}}{=} (\alpha \land \beta) \lor (\neg \alpha \land \neg \beta)$

- With respect to Propositional Calculus tableaux, we add unfolding rules for modal operators as follows:

<table>
<thead>
<tr>
<th>Propositional Calculus rules</th>
<th>Modal rules</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Formula</strong></td>
<td><strong>Branch 1</strong></td>
</tr>
<tr>
<td>$\alpha \lor \beta$</td>
<td>$\alpha$</td>
</tr>
<tr>
<td>$\alpha \land \beta$</td>
<td>$\alpha, \beta$</td>
</tr>
<tr>
<td>$\neg (\alpha \lor \beta)$</td>
<td>$\neg \alpha, \neg \beta$</td>
</tr>
<tr>
<td>$\neg (\alpha \land \beta)$</td>
<td>$\neg \alpha$</td>
</tr>
</tbody>
</table>
When these rules are **exhausted**, each tableau leaf is boxed and (partially) represents a **state**.

The state usually contains $\bigcirc$-formulas like $\bigcirc \alpha$ or $\neg \bigcirc \alpha$. In such a case, we **generate a transition** to a next state whose content is fixed with the new rules:

<table>
<thead>
<tr>
<th>Formula</th>
<th>Next state</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\bigcirc \alpha$</td>
<td>$\alpha$</td>
</tr>
<tr>
<td>$\neg \bigcirc \alpha$</td>
<td>$\neg \alpha$</td>
</tr>
</tbody>
</table>

We can reach a state **repeated** in previous tableau node. If so, we just label the previous node and **reuse** it.
Semantic tableaux

- Example: take \((p \lor q) \land \bigcirc(\neg p \land \neg q)\)

\[
(p \lor q) \land \bigcirc(\neg p \land \neg q) \\
(p \lor q), \bigcirc(\neg p \land \neg q) \\
p, \bigcirc(\neg p \land \neg q) \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad q, \bigcirc(\neg p \land \neg q)
\]

- Both open branches yield to a transition to a new state where:

\[
\neg p \land \neg q \\
\neg p, \neg q
\]
That is, any model of \((p \lor q) \land \Box(\neg p \land \neg q)\) must contain one of the following structures:

These are called \textit{Hintikka structures}. They can be expanded to interpretations (arbitrarily completing the truth of the rest of atoms)
Example 2: is $\Box(p \land q) \rightarrow \Box p$ valid?

We negate the formula and check if we obtain a closed tableau

$$\neg(\Box(p \land q) \rightarrow \Box p)$$

$l_0 : \Box(p \land q), \Diamond \neg p$

$p \land q, \bigcirc \Box(p \land q), \Diamond \neg p$

$p, q, \bigcirc \Box(p \land q), \Diamond \neg p$

$p, q,$

$\bigcirc \Box(p \land q), \neg p$

$\times$

We would create a new state with $\Box(p \land q), \Diamond \neg p = l_0$
The tableau is open but generates the following Hintikka structure:

\[ s_0 \rightarrow p, q \rightarrow s_1 \rightarrow p, q \rightarrow s_2 \rightarrow \ldots \]

or simply

\[ s_0 \rightarrow p, q \rightarrow \]

which is never a model because \( \diamond \neg p \) is never fulfilled.

For open tableaux, we will have to check fulfillment of \( \diamond \alpha \) formulas.
Example $\Box \Diamond p$

To $l_1$:

$l_1 : \Box \Diamond p$

To $l_2$:

$l_2 : \Diamond p, \Box \Box \Diamond p$

$l_3 : p, \Box \Box \Diamond p$

To $l_1$:

$l_4 : \Box \Diamond p, \Box \Box \Diamond p$

$l_5 : \Diamond p, \Box \Diamond p$

To $l_2$:

$\Diamond \alpha$ formulas are fulfilled, so the Hintikka structure represents possible models:

$s_0$ $s_1$