Linear-time Temporal Logic

Pedro Cabalar

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

2015/2016
Propositional Linear-time Temporal Logic (LTL)

Syntax

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

Modal operators:

- unary operators:
  - $\square = \text{“forever”}$
  - $\Diamond = \text{“eventually”}$
  - $\circ = \text{“next”}$

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

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

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

Definition 2 (Interpretation)

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

Example:

\[
\begin{align*}
\{p, q\} & \rightarrow \{p, r\} & \rightarrow \{q\} & \rightarrow \{q, r\} & \rightarrow \emptyset \\
\bullet & \rightarrow \bullet & \rightarrow \bullet & \rightarrow \bullet & \rightarrow \bullet & \rightarrow \ldots \\
S_0 & \rightarrow S_1 & \rightarrow S_2 & \rightarrow S_3 & \rightarrow S_4
\end{align*}
\]
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

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

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

- \( T \cup \psi = \text{repeat } T \text{ until (mandatorily) } \psi \)

  \[ T \quad T \quad T \quad T \quad T \quad \psi \]

  
  
  \[ \bullet \rightarrow \bullet \rightarrow \bullet \rightarrow \ldots \rightarrow \bullet \rightarrow \bullet \rightarrow \ldots \]

  This is equivalent to \( \Diamond \psi \).

- \( \bot \lor \psi \) = there is a \( \bot \) before any state with \( \neg \psi \).

  That is, we cannot have \( \neg \phi \), i.e., \( \psi \) must hold forever \( \square \psi \)

  \[ \psi \quad \psi \quad \psi \quad \psi \quad \psi \quad \psi \]

  
  
  \[ \bullet \rightarrow \bullet \rightarrow \bullet \rightarrow \ldots \rightarrow \bullet \rightarrow \bullet \rightarrow \ldots \]
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 $\models \alpha \leftrightarrow \beta$. 
Some interesting equivalences

\[\begin{align*}
\Diamond \alpha & \iff \top \mathcal{U} \alpha \quad (1) \\
\Box \alpha & \iff \bot \mathcal{V} \alpha \quad (2) \\
\Box \alpha & \iff \neg \Diamond \neg \alpha \quad (3) \\
\Diamond \alpha & \iff \neg \Box \neg \alpha \quad (4) \\
\Box \alpha & \iff \alpha \land \Box \Box \alpha \quad (5) \\
\Diamond \alpha & \iff \alpha \lor \Box \Diamond \alpha \quad (6) \\
\alpha \mathcal{U} \beta & \iff (\alpha \mathcal{W} \beta) \land \Diamond \beta \quad (7) \\
\alpha \mathcal{W} \beta & \iff (\alpha \mathcal{U} \beta) \lor \Box \alpha \quad (8) \\
\alpha \mathcal{U} \beta & \iff \beta \lor \alpha \land \Box (\alpha \mathcal{U} \beta) \quad (9) \\
\alpha \mathcal{V} \beta & \iff \neg (\neg \alpha \mathcal{U} \neg \beta) \quad (10) \\
\alpha \mathcal{V} \beta & \iff \beta \mathcal{W} (\beta \land \alpha) \quad (11)
\end{align*}\]
LTL can be seen as a fragment of Predicate Calculus.

\[ \text{MFO}(<) = \text{Monadic First Order Logic with } < \text{ relation} \]

- All predicates are monadic (1 argument) \( p(x), q(y), \ldots \)
- excepting a linear order predicate \( x \leq y \), binary and infix
- arguments \( x, y, \ldots \) represent time points
- constant 0 represents the initial time point

Example \( \square p \) can be translated as \( \forall x (x \geq 0 \rightarrow p(x)) \)
We adopt some abbreviations

\[ x = y \overset{\text{def}}{=} x \leq y \land y \leq x \]

\[ x < y \overset{\text{def}}{=} x \leq y \land \neg(y \leq x) \]

\[ x \leq y \leq z \overset{\text{def}}{=} x \leq y \land y \leq z \]

\[ \forall x \geq i : \alpha(x) \overset{\text{def}}{=} \forall (i \leq x \rightarrow \alpha(x)) \]

\[ \exists x \geq i : \alpha(x) \overset{\text{def}}{=} \exists (i \leq x \land \alpha(x)) \]

\[ \forall x \in i..j : \alpha(x) \overset{\text{def}}{=} \forall (i \leq x \leq j \rightarrow \alpha(x)) \]

\[ \exists x \in i..j : \alpha(x) \overset{\text{def}}{=} \exists (i \leq x \leq j \land \alpha(x)) \]

The meaning of function ‘+1’ can be defined with the axiom

\[ x + 1 = y \leftrightarrow x < y \land \neg\exists z(x < z \land z < y) \]
Kamp’s translation

Temporal formula $\alpha$ at state $i$ becomes MFO($<$) formula $\alpha(i)$.

$$(p)(i) \overset{\text{def}}{=} p(i)$$

$$(\neg \alpha)(i) \overset{\text{def}}{=} \neg \alpha(i)$$

$$(\alpha \land \beta)(i) \overset{\text{def}}{=} \alpha(i) \land \beta(i)$$

$$(\alpha \lor \beta)(i) \overset{\text{def}}{=} \alpha(i) \lor \beta(i)$$

$$(\Box \alpha)(i) \overset{\text{def}}{=} \alpha(i + 1)$$

$$(\Diamond \alpha)(i) \overset{\text{def}}{=} \forall j \geq i : \alpha(j)$$

$$(\alpha \mathcal{U} \beta)(i) \overset{\text{def}}{=} \exists j \geq i : (\beta(j) \land \forall k \in i..j - 1 : \alpha(k))$$

$$(\alpha \mathcal{V} \beta)(i) \overset{\text{def}}{=} \forall j \geq i : (\beta(j) \lor \exists k \in i..j - 1 : \alpha(k))$$

Theorem: $M, i \models \alpha$ in LTL iff $M \models \alpha(i)$ in First-Order Logic.
Kamp’s translation

Example of use: prove the tautology \( \neg(\alpha \cup \beta) \iff \neg \alpha \lor \neg \beta \)

\[
(\neg(\alpha \cup \beta))(i) \iff \neg \exists j \geq i: (\beta(j) \land \forall k \in i..j - 1: \alpha(k)) \\
\iff \forall j \geq i: \neg(\beta(j) \land \forall k \in i..j - 1: \alpha(k)) \\
\iff \forall j \geq i: (\neg \beta(j) \lor \forall k \in i..j - 1: \alpha(k)) \\
\iff \forall j \geq i: (\neg \beta(j) \lor \exists k \in i..j - 1: \neg \alpha(k)) \\
\iff \forall j \geq i: (((\neg \beta)(j) \lor \exists k \in i..j - 1: (\neg \alpha)(k)) \\
\iff (\neg \alpha \lor \neg \beta)(i)
\]
Another example: prove the tautology $\square \alpha \leftrightarrow \alpha \land \lozenge \square \alpha$

\[
(\square \alpha)(i) = \forall j \geq i : \alpha(j)
\]

\[
\leftrightarrow \alpha(i) \land \forall j \geq i + 1 : \alpha(j)
\]

\[
\leftrightarrow \alpha(i) \land (\square \alpha)(i + 1)
\]

\[
\leftrightarrow \alpha(i) \land (\lozenge \square \alpha)(i)
\]

\[
\leftrightarrow (\alpha \land \lozenge \square \alpha)(i)
\]
Exercises

Exercise 1

Prove validity of (6) and (9).

Exercise 2

Prove the validity of the following formulas:

\[ \beta \rightarrow \Diamond \beta \]

\[ \beta \rightarrow \alpha U \beta \]

\[ \alpha U \beta \rightarrow \Diamond \beta \]
Theorem 4 (Kamp 1968)

LTL is exactly as expressive as MFO(\(<\)).

- As we saw, any LTL formula can be interpreted in MFO(\(<\)) in a natural way.
- The real interest of Kamp’s proof is the other direction: any MFO(\(<\)) formula can be “rearranged” as an equivalent modal LTL formula.
Exercise 3

Which are the models of $\perp \cup p$? Which are the models of $(\bigcirc p) \cup \neg p$?

Exercise 4

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

Exercise 5

Try to express the formula whose models satisfy: $p$ is true in all even states $0, 2, 4, \ldots$ leaving all the rest free.
1. Syntax and semantics
2. Specification with LTL
3. Model checking algorithms
4. Complexity and expressiveness
5. Deductive system
6. Semantic tableaux
Examples of properties specification

Figure out the meaning of these example formulas:

- □((¬passport ∨ ¬ticket) → ◇¬board))
- □(requested → ♦received)
- □(received → ○processed)
- □(processed → ◇□done)
- “It can’t be that we continually resend a request that is never done.” The statement: □requested ∧ □¬done should be inconsistent.
  That is, we should be able to derive □requested → ◇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 \textit{bad} never happens = \Box \neg \textit{bad}.

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

- If a train is approaching or crossing, the light must be blinking
  Solution: \Box (a \lor c \rightarrow l) \equiv \Box \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: \Box (\neg b \land \neg l \rightarrow \neg a \land \neg c) \equiv \Box \neg (\neg b \land \neg l \land (a \lor c))
Liveness property = something \textit{initiated} eventually \textit{terminates} = 
\(\Box (\text{initiated} \rightarrow \Diamond \text{terminates})\)

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

- Sometimes we can use \(U\), \(W\) or \(V\) 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: \(\Box (a \land \neg c \rightarrow \neg c \ W b)\)

- If a train finishes crossing, the barrier will be eventually risen
  Solution: \(\Box (c \land \bigcirc \neg c \rightarrow \bigcirc \Diamond \neg b)\)
Something happens infinitely often = $\Box \Diamond \text{something}$.  
Example: The barrier is risen infinitely often = $\Box \Diamond \neg b$

The dual is a latching condition = $\Diamond \Box \alpha$.  
Example: at a given point, no more trains are approaching = $\Diamond \Box \neg a$
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 $\square \Diamond \text{executed}_i$

- **Strong fairness** every process enabled infinitely often should be executed infinitely often $\square \Diamond \text{enabled}_i \rightarrow \square \Diamond \text{executed}_i$

- **Weak fairness** every process permanently enabled after some point should be executed infinitely often $\Diamond \square \text{enabled}_i \rightarrow \square \Diamond \text{executed}_i$
Outline

1. Syntax and semantics
2. Specification with LTL
3. Model checking algorithms
4. Complexity and expressiveness
5. Deductive system
6. Semantic tableaux
A simple example of concurrent program

- 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 $\text{regP}$, $\text{regQ}$ respectively.
- They run concurrently. An interruption may arbitrarily stop the execution, store the register values, and then restore them on return.

```c
byte n=0;
active proctype P() {
    byte regP=0;
    p1: regP=n;
    p2: regP++;
    p3: n=regP;
    p4: skip
}

active proctype Q() {
    byte regQ=0;
    q1: regQ=n;
    q2: regQ++;
    q3: n=regQ;
    q4: skip
}
```

P. Cabalar (Department of Computer Science University of Corunna, SPAIN
cabalar@udc.es)
A state has \((IPP, IPQ, \text{regP}, \text{regQ}, n)\) where \(IPP\) and \(IPQ\) 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:

\[(p_1,q_1,0,0,0) \rightarrow (p_2,q_1,0,0,0)\]
\[(p_3,q_1,1,0,0) \rightarrow (p_4,q_1,1,0,1)\]
\[(p_4,q_2,1,1,1) \rightarrow (p_4,q_3,1,2,1)\]
\[(p_4,q_4,1,2,2)\]

\[\begin{align*}
p_1: & \text{ regP=n;} & q_1: & \text{ regQ=n;} \\
p_2: & \text{ regP++;} & q_2: & \text{ regQ++;} \\
p_3: & \text{ n=regP;} & q_3: & \text{ n=regQ;} \\
p_4: & \text{ skip} & q_4: & \text{ skip}
\end{align*}\]
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:

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

\[
\begin{align*}
p1 &: \text{regP=n}; & q1 &: \text{regQ=n}; \\
p2 &: \text{regP++}; & q2 &: \text{regQ++}; \\
p3 &: \text{n=regP}; & q3 &: \text{n=regQ}; \\
p4 &: \text{skip} & q4 &: \text{skip}
\end{align*}
\]
Model checking algorithms

- 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 $(p_1, q_1, 0, 0, 0)$ 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.
Automata construction

- 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:

- **Implicit 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.
Model checking techniques

- **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 Complexity and expressiveness
5 Deductive system
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, \textit{SAT} (propositional satisfiability, i.e., “does a formula \(\alpha\) have any model?”) is a decision problem, and its complexity class is \textbf{NP}-complete.

Other examples of \textbf{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 ’Halts’), 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.
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

  \[ P \neq NP \]

- 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/

  The Clay Mathematics Institute designated $1 million prize for its solution!
Meaning of NP-completeness

- A problem $X$ is $\textbf{C}$-complete, for some complexity class $\textbf{C}$, iff any problem $Y$ in $\textbf{C}$ is reducible to $X$ in polynomial-time.

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

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

- $\textit{SAT}$ is commonly used nowadays for showing that a problem $X$ is at least as complex as $\textbf{NP}$. To this aim, just encode $\textit{SAT}$ into $X$. 
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, \( NP \subseteq PSPACE \). Again, unsolved question
  \( NP \overset{?}{=} PSPACE \) but strongly suspected to be \( \neq \).
- 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.
Outline

1. Syntax and semantics
2. Specification with LTL
3. Model checking algorithms
4. Complexity and expressiveness
5. Deductive system
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.
**Inference methods**

- **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 $\mathcal{L}$).

- **Given a language $\mathcal{L}$, a logic $\mathcal{L}$ is a subset of $\mathcal{L}$. It can be defined:**
  - Semantically: $\mathcal{L} = \{ \alpha \in \mathcal{L} \mid \models \alpha \}$.  
  - Syntactically: $\mathcal{L} = \{ \alpha \in \mathcal{L} \mid \vdash \alpha \}$.

- **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 \textbf{LTL} deductive system as follows.

Axioms:

\textbf{Ax0} \quad PC \quad \text{Any substitution instance of any Propositional Calculus tautology}

\textbf{Ax1} \quad \vdash \Box(\alpha \rightarrow \beta) \rightarrow (\Box \alpha \rightarrow \Box \beta) \quad \text{Distribution of } \Box \text{ over } \rightarrow

\textbf{Ax2} \quad \vdash \Diamond(\alpha \rightarrow \beta) \rightarrow (\Diamond \alpha \rightarrow \Diamond \beta) \quad \text{Distribution of } \Diamond \text{ over } \rightarrow

\textbf{Ax3} \quad \vdash \Box \alpha \rightarrow (\alpha \land \Diamond \alpha \land \Diamond \Box \alpha) \quad \text{Expansion of } \Box

\textbf{Ax4} \quad \vdash \Box(\alpha \rightarrow \Diamond \alpha) \rightarrow (\alpha \rightarrow \Box \alpha) \quad \text{Induction}

\textbf{Ax5} \quad \vdash \Diamond \alpha \leftrightarrow \neg \Diamond \neg \alpha \quad \text{Linearity}

Inference rules:

\quad \begin{array}{c}
\textbf{MP} \quad \vdash \alpha, \quad \vdash \alpha \rightarrow \beta \\
\hline
\vdash \beta
\end{array} \quad \text{Modus Ponens}

\quad \begin{array}{c}
\textbf{N} \quad \vdash \alpha \\
\hline
\vdash \Box \alpha
\end{array} \quad \text{Necessitation}
A deductive system

An example of a proof

Theorem 6 (transitivity)

\[ \vdash \Box\Box p \leftrightarrow \Box p \]

Proof:

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

Q.E.D.
A deductive system

Derived inference rules:

\[ G \square \quad \frac{\vdash \alpha \rightarrow \beta}{\vdash \square \alpha \rightarrow \square \beta} \quad \square \rightarrow \text{Generalization} \]

\[ G \circ \quad \frac{\vdash \alpha \rightarrow \beta}{\vdash \circ \alpha \rightarrow \circ \beta} \quad \circ \rightarrow \text{Generalization} \]

\[ \text{Ind} \quad \frac{\vdash \alpha \rightarrow \circ \alpha}{\vdash \alpha \rightarrow \square \alpha} \quad \text{Induction} \]

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

Exercises

Exercise 6

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 7

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. Complexity and expressiveness
5. Deductive system
6. Semantic tableaux
For simplicity, we assume $\alpha \rightarrow \beta \overset{\text{def}}{=} \neg \alpha \vee \beta$ 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>Formula</td>
<td>Branch 1</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 $\Diamond$-formulas like $\Diamond \alpha$ or $\neg \Diamond \alpha$. 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>$\Diamond \alpha$</td>
<td>$\alpha$</td>
</tr>
<tr>
<td>$\neg \Diamond \alpha$</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 \((p \lor q) \land \Box (\neg p \land \neg q)\)

\[
\begin{align*}
\neg p \land \neg q
\end{align*}
\]

- Both open branches yield to a transition to a new state where:
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 Hintikka structures. They can be expanded to interpretations (arbitrarily completing the truth of the rest of atoms)
Example 2: is $\square(p \land q) \rightarrow \square p$ valid?

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

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

$l_0 : \square(p \land q), \lozenge \neg p$

$p \land q, \lozenge \square(p \land q), \lozenge \neg p$

$p, q, \lozenge \square(p \land q), \lozenge \neg p$

$p, q, \lozenge \square(p \land q), \lozenge \neg p$

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

\[
\begin{align*}
  &s_0 &s_1 &s_2 \\
  &p, q &p, q &p, q \\
  \end{align*}
\]

or simply

\[
\begin{align*}
  &s_0 \\
  &p, q \\
\end{align*}
\]

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 $\square \Diamond p$

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