Initiation à la vérification
Basics of Verification


Paul Gastin

Paul.Gastin@lsv.ens-cachan.fr
http://www.lsv.ens-cachan.fr/~gastin/

MPRI – M1
2019 – 2020
Outline

1. Introduction

Models

Temporal Specifications

Satisfiability and Model Checking

More on Temporal Specifications
Need for formal verification methods

Critical systems

- Transport
- Energy
- Medicine
- Communication
- Finance
- Embedded systems
- ...

...
Disastrous software bugs


Mariner 1 probe, 1962

See http://en.wikipedia.org/wiki/Mariner_1

- Destroyed 293 seconds after launch
- Missing hyphen in the data or program? No!
- Overbar missing in the mathematical specification:
  \( \ddot{R}_n \): \( n \)th smoothed value of the time derivative of a radius.
Without the smoothing function indicated by the bar, the program treated normal minor variations of velocity as if they were serious, causing spurious corrections that sent the rocket off course.
Disastrous software bugs

Ariane 5 flight 501, 1996

See http://en.wikipedia.org/wiki/Ariane_5_Flight_501

- Destroyed 37 seconds after launch (cost: 370 millions dollars).
- Data conversion from a 64-bit floating point to 16-bit signed integer value caused a hardware exception (arithmetic overflow).
- Efficiency considerations had led to the disabling of the software handler (in Ada code) for this error trap.
- The fault occurred in the inertial reference system of Ariane 5. The software from Ariane 4 was re-used for Ariane 5 without re-testing.
- On the basis of those calculations the main computer commanded the booster nozzles, and somewhat later the main engine nozzle also, to make a large correction for an attitude deviation that had not occurred.
- The error occurred in a realignment function which was not useful for Ariane 5.
Disastrous software bugs

Spirit Rover (Mars Exploration), 2004


▶ Ceased communicating on January 21.
▶ Flash memory management anomaly: too many files on the file system
▶ Resumed to working condition on February 6.
Disastrous software bugs

Other well-known bugs

## Formal verifications methods

### Based on
- A formal model of the system
- A formal semantics of the modelling language
- A formal specification

### Complementary approaches
- Theorem prover
- Model checking
- Static analysis
- Test
Model Checking

- Purpose 1: automatically finding software or hardware bugs.
- Purpose 2: prove correctness of abstract models.
- Should be applied during design.
- Real systems can be analysed with abstractions.

E.M. Clarke  
E.A. Emerson  
J. Sifakis

Prix Turing 2007.
# Model Checking

## 3 steps

- Constructing the model $M$ (transition systems)
- Formalizing the specification $\varphi$ (temporal logics)
- Checking whether $M \models \varphi$ (algorithmics)

## Main difficulties

- Size of models (combinatorial explosion)
- Expressivity of models or logics
- Decidability and complexity of the model-checking problem
- Efficiency of tools

## Challenges

- Extend models and algorithms to cope with more systems.
  Infinite systems, parameterized systems, probabilistic systems, concurrent systems, timed systems, hybrid systems, . . . See Modules 2.8 & 2.9

- Scale current tools to cope with real-size systems.
  Needs for modularity, abstractions, symmetries, . . .
References

*Principles of Model Checking.*  

*Systems and Software Verification. Model-Checking Techniques and Tools.*  

*Model Checking.*  


*Temporal Verification of Reactive Systems: Safety.*  

*The Complexity of Temporal Logic Model Checking.*  
Outline

Introduction

2 Models

- Transition Systems
- ... with Variables
- Concurrent Systems
- Synchronization and Communication

Temporal Specifications

Satisfiability and Model Checking

More on Temporal Specifications
Model and abstractions

Example: Golden face

Each coin has a golden face and a silver face. At each step, we may flip simultaneously the 3 coins of a line, column or diagonal. Is it possible to have all coins showing its golden face? If yes, what is the smallest number of steps.

Model = Transition system

- States: configurations of the board $\begin{bmatrix} 2 & 9 \end{bmatrix} = 512$ states
- Transitions: flipping a line/column/diagonal
- Problem: reachability

Abstraction 1: number of golden faces in a configuration.
Abstraction 2: parity of the number of golden faces in the corners.
Model and Specification

Example: Men, Wolf, Goat, Cabbage

Model = Transition system
- State = who is on which side of the river
- Transition = crossing the river
- Specification
  - Safety: Never leave WG or GC alone
  - Liveness: Take everyone to the other side of the river.
Transition system or Kripke structure

Definition: TS

\[ M = (S, \Sigma, T, I, AP, \ell) \]

- \( S \): set of states (finite or infinite)
- \( \Sigma \): set of actions
- \( T \subseteq S \times \Sigma \times S \): set of transitions
- \( I \subseteq S \): set of initial states
- \( AP \): set of atomic propositions
- \( \ell : S \rightarrow 2^{AP} \): labelling function.

Every discrete system may be described with a TS.

Example: Digicode ABA
Description Languages

Pb: How can we easily describe big systems?

Description Languages (high level)

- Programming languages
- Boolean circuits
- Modular description, e.g., parallel compositions
  problems: concurrency, synchronization, communication, atomicity, fairness, ...
- Petri nets (intermediate level)
- Transition systems (intermediate level)
  with variables, stacks, channels, ...
  synchronized products
- Logical formulae (low level)

Operational semantics

High level descriptions are translated (compiled) to low level (infinite) TS.
# Transition systems with variables

**Definition: TSV**

\[ M = (S, \Sigma, \mathcal{V}, (D_v)_{v \in \mathcal{V}}, T, I, AP, \ell) \]

- Finite description with \( S, \Sigma, AP, \ell \) as before
- \( \mathcal{V} \): set of (typed) variables, e.g., boolean, \([0..4]\), \( \mathbb{N} \), ...
- Each variable \( v \in \mathcal{V} \) has a domain \( D_v \) (finite or infinite). Let \( D = \prod_{v \in \mathcal{V}} D_v \).
- Guard or Condition \( g \) with semantics \( \llbracket g \rrbracket \subseteq D \) (predicate)
  - Symbolic descriptions: \( x < 5, x + y = 10, \ldots \)
- Instruction or Update \( f \) with semantics \( \llbracket f \rrbracket : D \rightarrow D \) (or \( \llbracket f \rrbracket \subseteq D \times D \))
  - Symbolic descriptions: \( x := 0, x := (y + 1)^2, \ldots \)
- \( T \subseteq S \times (\text{Guard} \times \Sigma \times \text{Update}) \times S \)
  - Symbolic descriptions: \( s \xrightarrow{x < 50, ?\text{coin}, x := x + \text{coin}} s' \)
- \( I \subseteq S \times \text{Guard} \)
  - Symbolic descriptions: \( (s_0, x = 0) \)

**Example: Vending machine**

- coffee: 50 cents, orange juice: 1 euro, ...
- possible coins: 10, 20, 50 cents
- we may shuffle coin insertions and drink selection
Transition systems with variables

**Semantics: low level TS**

- $S' = S \times D$
- $I' = \{ (s, \nu) \mid \exists (s, g) \in I \text{ with } \nu \models g \}$
- Transitions: $T' \subseteq (S \times D) \times \Sigma \times (S \times D)$

\[
\begin{align*}
S & \xrightarrow{g, a, f} S' \land \nu \models g \\
(s, \nu) & \xrightarrow{a} (s', f(\nu))
\end{align*}
\]

**SOS: Structural Operational Semantics**

- $AP'$: we may use atomic propositions in $AP$ or guards such as $x > 0$.

**Programs = Kripke structures with variables**

- Program counter = states
- Instructions = transitions
- Variables = variables

**Example: GCD**
Example: Digicode

- $\text{cpt} < n$
- $B, C$
- $\text{cpt}++$

- $\text{cpt} = 0$
- $\text{OPEN}$
- $A$
- $B$
- $A$
- $B$
- $C$

- $\text{cpt} = n$
- $B, C$
- $\text{cpt}++$

- $\text{ERROR}$

- $\text{cpt} = n$
- $C$
- $\text{cpt}++$

- $\text{cpt} = n$
- $A, C$
- $\text{cpt}++$

- $\text{cpt} = n$
- $B, C$
- $\text{cpt}++$

- $\text{cpt} = n$
- $B, C$
- $\text{cpt}++$
Only variables

The state is nothing but a special variable: \( s \in \mathcal{V} \) with domain \( D_s = S \).

**Definition: TSV**

\[
M = (\mathcal{V}, (D_v)_{v \in \mathcal{V}}, T, I, \text{AP}, \ell)
\]

- \( D = \prod_{v \in \mathcal{V}} D_v \)
- \( I \subseteq D, T \subseteq D \times D \)

**Symbolic representations with logic formulae**

- \( I \) given by a formula \( \psi(\nu) \)
- \( T \) given by a formula \( \varphi(\nu, \nu') \)
  - \( \nu \): values before the transition
  - \( \nu' \): values after the transition
- Often we use boolean variables only: \( D_v = \{0, 1\} \)
- Concise descriptions of boolean formulae with Binary Decision Diagrams.

**Example: Boolean circuit: modulo 8 counter**

\[
\begin{align*}
    b'_0 &= \neg b_0 \\
    b'_1 &= b_0 \oplus b_1 \\
    b'_2 &= (b_0 \land b_1) \oplus b_2
\end{align*}
\]
Modular description of concurrent systems

\[ M = M_1 \parallel M_2 \parallel \cdots \parallel M_n \]

Semantics

- Various semantics for the parallel composition \( \parallel \)
- Various communication mechanisms between components:
  - Shared variables, FIFO channels, Rendez-vous, ...
- Various restrictions

Atomic propositions are inherited from the local systems.

Example: Elevator with 1 cabin, 3 doors, 3 calling devices

- Cabin:
- Door for level \( i \):
- Call for level \( i \):
Synchronized products (semantics)

Definition: General product

- Components: \( M_i = (S_i, \Sigma_i, T_i, I_i, AP_i, \ell_i) \)
- Product: \( M = (S, \Sigma, T, I, AP, \ell) \) with
  
  \[
  S = \prod_i S_i, \quad \Sigma = \prod_i (\Sigma_i \cup \{\epsilon\}), \quad \text{and} \quad I = \prod_i I_i
  \]

  \( T \) defined by
  
  \[
  \forall i, \quad (p_i \xrightarrow{a_i} q_i) \in T_i \lor (a_i = \epsilon \land p_i = q_i)
  \]

  \[
  (p_1, \ldots, p_n) \xrightarrow{(a_1, \ldots, a_n)} (q_1, \ldots, q_n)
  \]

  \( AP = \bigcup_i AP_i \) and \( \ell(p_1, \ldots, p_n) = \bigcup_i \ell(p_i) \)

Synchronized products: restrictions of the general product.

Parallel compositions: 2 special cases

- Synchronous: \( \Sigma_{\text{sync}} = \prod_i \Sigma_i \)
- Asynchronous: \( \Sigma_{\text{async}} = \bigcup_i \Sigma'_i \) with \( \Sigma'_i = \{\epsilon\}^{i-1} \times \Sigma_i \times \{\epsilon\}^{n-i} \)

Restrictions

- on states: \( S_{\text{restrict}} \subseteq S \)
- on labels: \( \Sigma_{\text{restrict}} \subseteq \Sigma \)
- on transitions: \( T_{\text{restrict}} \subseteq T \)
Shared variables

Define: Asynchronous product + shared variables
\bar{s} = (s_1, \ldots, s_n) denotes a tuple of states
\nu \in D = \prod_{v \in \mathcal{V}} D_v is a valuation of variables.

Semantics (SOS)
\nu \models g \land s_i \xrightarrow{g,a,f} s_i' \land s_j' = s_j \text{ for } j \neq i
\bar{s}, \nu \xrightarrow{a} (\bar{s}', f(\nu))

Example: Mutual exclusion for 2 processes satisfying

- **Safety**: never simultaneously in critical section (CS).
- **Liveness**: if a process wants to enter its CS, it eventually does.
- **Fairness**: if process 1 wants to enter its CS, then process 2 will enter its CS at most once before process 1 does.

using shared variables but without further restrictions: the atomicity is

- testing or reading or writing a single variable at a time
- no test-and-set: \{x = 0; x := 1\}
Peterson’s algorithm (1981)

Process \( i \): // \( i \) is not a variable

\[
\text{loop forever} \\
\text{req}[i] := \text{true}; \text{turn} := 1-i \\
\text{wait until (\text{turn} = i \text{ or req}[1-i] = \text{false})} \\
\text{Critical section} \\
\text{req}[i] := \text{false}
\]

Exercise:

- Draw the concrete TS assuming the first two assignments are atomic.
- Is the algorithm still correct if we swap the first two assignments?
Atomicity

Example:

Initially $x = 1 \land y = 2$

Program $P_1$: $x := x + y \parallel y := x + y$

Program $P_2$: \[
\begin{pmatrix}
\text{Load}R_1, x \\
\text{Add}R_1, y \\
\text{Store}R_1, x
\end{pmatrix}
\parallel
\begin{pmatrix}
\text{Load}R_2, x \\
\text{Add}R_2, y \\
\text{Store}R_2, y
\end{pmatrix}
\]

Assuming each instruction is atomic, what are the possible results of $P_1$ and $P_2$?
Atomicity

Definition: Atomic statements: \( \text{atomic}(ES) \)

Elementary statements (no loops, no communications, no synchronizations)

\[
ES ::= \text{skip} \mid \text{await } c \mid x := e \mid ES ; ES \mid ES \Box ES \mid \text{when } c \text{ do } ES \mid \text{if } c \text{ then } ES \text{ else } ES
\]

Atomic statements: if the ES can be fully executed then it is executed in one step.

\[
(\bar{s}, \nu) \xrightarrow{ES} (\bar{s}', \nu') \quad (\bar{s}, \nu) \xrightarrow{\text{atomic}(ES)} (\bar{s}', \nu')
\]

Example: Atomic statements

- \( \text{atomic}(x = 0; x := 1) \) (Test and set)
- \( \text{atomic}(y := y - 1; \text{await}(y = 0); y := 1) \) is equivalent to \( \text{await}(y = 1) \)
Communication by Rendez-vous

Restriction on transitions is universal but too low-level.

Definition: Rendez-vous

- !m sending message m
- ?m receiving message m
- SOS: Structural Operational Semantics

Local actions:

\[
\begin{align*}
\frac{s_1 \xrightarrow{a_1} s'_1}{(s_1, s_2) \xrightarrow{a_1} (s'_1, s_2)} \\
\end{align*}
\]

Rendez-vous:

\[
\begin{align*}
\frac{\text{!}m}{s_1 \xrightarrow{1} s'_1 \land s_2 \xrightarrow{2} s'_2} \\
\frac{?m}{m \xrightarrow{} (s'_1, s'_2)} \\
\frac{\text{!}m}{s_1 \xrightarrow{1} s'_1 \land s_2 \xrightarrow{2} s'_2} \\
\frac{?m}{m \xrightarrow{} (s'_1, s'_2)}
\end{align*}
\]

- It is a restriction on actions.
- Essential feature of process algebra.

Example: Elevator with 1 cabin, 3 doors, 3 calling devices

- ?up is uncontrollable for the cabin
- ?leave_i is uncontrollable for door i
- ?call_0 is uncontrollable for the system
Example: Leader election

We have $n$ processes on a directed ring, each having a unique $id \in \{1, \ldots, n\}$.

```plaintext
send(id)
loop forever
  receive(x)
  if (x = id) then STOP fi
  if (x > id) then send(x)
```

Channels
Definition: Channels

- **Declaration:**
  - $c : \text{channel } [k] \text{ of bool} \quad \text{size } k$
  - $c : \text{channel } [\infty] \text{ of int} \quad \text{unbounded}$
  - $c : \text{channel } [0] \text{ of colors} \quad \text{Rendez-vous}$

- **Primitives:**
  - $\text{empty}(c)$
  - $c!e \quad \text{add the value of expression } e \text{ to channel } c$
  - $c?x \quad \text{read a value from } c \text{ and assign it to variable } x$

- **Domain:** Let $D_m$ be the domain for a single message.
  - $D_c = D_m^k \quad \text{size } k$
  - $D_c = D_m^* \quad \text{unbounded}$
  - $D_c = \{\varepsilon\} \quad \text{Rendez-vous}$

- **Politics:** FIFO, LIFO, BAG, …
Channels

Semantics: (lossy) FIFO

<table>
<thead>
<tr>
<th>Send</th>
<th>$s_i \xrightarrow{c!e} s_i' \land \nu'(c) = \nu(e) \cdot \nu(c)$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$(\bar{s}, \nu) \xrightarrow{c!e} (\bar{s}', \nu')$</td>
</tr>
<tr>
<td>Receive</td>
<td>$s_i \xrightarrow{c?x} s_i' \land \nu'(c) = \nu'(c) \cdot \nu'(x)$</td>
</tr>
<tr>
<td></td>
<td>$(\bar{s}, \nu) \xrightarrow{c?e} (\bar{s}', \nu')$</td>
</tr>
<tr>
<td>Lossy send</td>
<td>$s_i \xrightarrow{c!e} s_i'$</td>
</tr>
<tr>
<td></td>
<td>$(\bar{s}, \nu) \xrightarrow{c!e} (\bar{s}', \nu')$</td>
</tr>
</tbody>
</table>

Implicit assumption: all variables that do not occur in the premise are not modified.

Exercises:

1. Implement a FIFO channel using rendez-vous with an intermediary process.
2. Give the semantics of a LIFO channel.
3. Model the alternating bit protocol (ABP) using a lossy FIFO channel.

Fairness assumption: For each channel, if infinitely many messages are sent, then infinitely many messages are delivered.
High-level descriptions

Summary

- Sequential program = transition system with variables
- Concurrent program with shared variables
- Concurrent program with Rendez-vous
- Concurrent program with FIFO communication
- Petri net
- ...
Remark: (Un)decidability

- Automata with 2 integer variables = Turing powerful
  Restriction to variables taking values in finite sets
- Asynchronous communication: unbounded fifo channels = Turing powerful
  Restriction to bounded channels or lossy channels

Remark: Some infinite state models are decidable

- Petri nets. Several unbounded integer variables but no zero-test.
- Pushdown automata. Model for recursive procedure calls.
- Timed automata.
- ...
Outline

Introduction

Models

Temporal Specifications
- General Definitions
- (Linear) Temporal Specifications
- Branching Temporal Specifications
- CTL*
- CTL

Satisfiability and Model Checking

More on Temporal Specifications
**Static and dynamic properties**

**Example: Static properties**

- **Mutual exclusion**
- Safety properties are often static.
- They can be reduced to reachability.

**Example: Dynamic properties**

- Every elevator request should be eventually granted.

- The elevator should not cross a level for which a call is pending without stopping.
Temporal Structures

Definition: Flows of time

A flow of time is a strict order $(\mathbb{T}, <)$ where $\mathbb{T}$ is the nonempty set of time points and $<$ is an irreflexive transitive relation on $\mathbb{T}$.

Example: Flows of time

- $(\{0, \ldots, n\}, <)$: Finite runs of sequential systems.
- $(\mathbb{N}, <)$: Infinite runs of sequential systems.
- $(\mathbb{R}, <)$: Runs of real-time sequential systems.
- Trees: Finite or infinite run-trees of sequential systems.
- Mazurkiewicz traces: Runs of distributed systems (partial orders).
- and also $(\mathbb{Z}, <)$ or $(\mathbb{Q}, <)$ or $(\omega^2, <)$, 

Definition: Temporal Structures

Let $\text{AP}$ be a set of atoms (atomic propositions) and let $\mathcal{C}$ be a class of time flows. A temporal structure over $(\mathcal{C}, \text{AP})$ is a triple $(\mathbb{T}, <, \lambda)$ where $(\mathbb{T}, <)$ is a time flow in $\mathcal{C}$ and $\lambda: \mathbb{T} \rightarrow 2^{\text{AP}}$ labels time points with atomic propositions.

The temporal structure $(\mathbb{T}, <, \lambda)$ is also denoted $(\mathbb{T}, <, h)$ where $h: \text{AP} \rightarrow 2^{\mathbb{T}}$ assigns time points to atomic propositions: $h(p) = \{t \in \mathbb{T} \mid p \in \lambda(t)\}$ for $p \in \text{AP}$. 
Let $M = (S, T, I, AP, \ell)$ be a Kripke structure.

**Definition: Runs as temporal structures**

An infinite run $\sigma = s_0s_1s_2 \cdots$ of $M$ with $(s_i, s_{i+1}) \in T$ for all $i \geq 0$ defines a linear temporal structure $\ell(\sigma) = (\mathbb{N}, <, \lambda)$ where $\lambda(i) = \ell(s_i)$ for $i \in \mathbb{N}$.

Such a temporal structure can be seen as an infinite word over $\Sigma = 2^{AP}$: 

$$\ell(\sigma) = \ell(s_0)\ell(s_1)\ell(s_2) \cdots \in \Sigma^\omega$$

**Linear specifications** only depend on runs.

Example: The printer manager is starvation free.

On each run, whenever some process requests the printer, it eventually gets it.

**Remark:**

Two Kripke structures having the same linear temporal structures satisfy the same linear specifications.
Branching behaviors and specifications

The system has an infinite active run, along which it may always reach an inactive state.

Definition: Computation-tree or run-tree: unfolding of the TS

Let $M = (S, T, I, AP, \ell)$ be a Kripke structure. Wlog. $I = \{s_0\}$ is a singleton. Let $D$ be a finite set with $|D|$ the outdegree of the transition relation $T$.

The computation-tree of $M$ is an unordered tree $t : D^* \to S$ (partial map) s.t.

- $t(\varepsilon) = s_0$,
- For every node $u \in \text{dom}(t)$ labelled $s = t(u)$, if $T(s) = \{s_1, \ldots, s_k\}$ then $u$ has exactly $k$ children which are labelled $s_1, \ldots, s_k$

Associated temporal structure $\ell(t) = (\text{dom}(t), <, \lambda)$ where

- $<$ is the strict prefix relation over $D^*$,
- and $\lambda(u) = \ell(t(u))$ for $u \in \text{dom}(t)$.

(Linear) runs of $M$ are branches of the computation-tree $t$. 
First-order Specifications

Definition: Syntax of FO(AP, <)

Let \( \text{Var} = \{x, y, \ldots\} \) be first-order variables.

\[
\varphi ::= \bot \mid p(x) \mid x = y \mid x < y \mid \neg \varphi \mid \varphi \lor \varphi \mid \exists x \varphi
\]

where \( p \in \text{AP} \).

Definition: Semantics of FO(AP, <)

Let \( w = (\mathbb{T}, <, \lambda) \) be a temporal structure over AP.
Let \( \nu : \text{Var} \to \mathbb{T} \) be an assignment of first-order variables to time points.

\[
\begin{align*}
  w, \nu \models p(x) & \quad \text{if} \quad p \in \lambda(\nu(x)) \\
  w, \nu \models x = y & \quad \text{if} \quad \nu(x) = \nu(y) \\
  w, \nu \models x < y & \quad \text{if} \quad \nu(x) < \nu(y) \\
  w, \nu \models \exists x \varphi & \quad \text{if} \quad w, \nu[x \mapsto t] \models \varphi \quad \text{for some} \ t \in \mathbb{T}
\end{align*}
\]

where \( \nu[x \mapsto t] \) maps \( x \) to \( t \) and \( y \neq x \) to \( \nu(y) \).

Previous specifications can be written in FO(<) (except the branching one).
First-order vs Temporal

First-order logic
- FO(⟨) has a good expressive power
  ... but FO(⟨)-formulae are not easy to write and to understand.
- FO(⟨) is decidable
  ... but satisfiability and model checking are non elementary.

Temporal logics
- no variables: time is implicit.
- quantifications and variables are replaced by modalities.
- Usual specifications are easy to write and read.
- Good complexity for satisfiability and model checking problems.
- Good expressive power.

Linear Temporal Logic (LTL) over (ℕ, <) introduced by Pnueli (1977) as a convenient specification language for verification of systems.
Temporal Specifications

Definition: Syntax of \( \text{TL}(\text{AP}, \text{SU}, \text{SS}) \)

\[
\varphi ::= \bot \mid p \ (p \in \text{AP}) \mid \neg \varphi \mid \varphi \lor \varphi \mid \varphi \text{ SU } \varphi \mid \varphi \text{ SS } \varphi
\]

Definition: Semantics: \( w = (T, <, \lambda) \) temporal structure and \( i \in T \)

\[
\begin{align*}
w, i \models p & \quad \text{if} \quad p \in \lambda(i) \\
w, i \models \neg \varphi & \quad \text{if} \quad w, i \not\models \varphi \\
w, i \models \varphi \lor \psi & \quad \text{if} \quad w, i \models \varphi \text{ or } w, i \models \psi \\
w, i \models \varphi \text{ SU } \psi & \quad \text{if} \quad \exists k \ i < k \text{ and } w, k \models \psi \text{ and } \forall j \ (i < j < k \rightarrow w, j \models \varphi) \\
w, i \models \varphi \text{ SS } \psi & \quad \text{if} \quad \exists k \ i > k \text{ and } w, k \models \psi \text{ and } \forall j \ (i > j > k \rightarrow w, j \models \varphi)
\end{align*}
\]

Previous specifications can be written in \( \text{TL}(\text{AP}, \text{SU}, \text{SS}) \) (except the branching one).

Theorem: \( \text{TL} \subseteq \text{FO}^3 \)

For each \( \varphi \in \text{TL}(\text{AP}, \text{SU}, \text{SS}) \) we can construct an equivalent formula with one free variable \( \tilde{\varphi}(x) \in \text{FO}^3(\text{AP}, <) \).
Temporal Specifications

Definition: non-strict versions of until and since

\[ \varphi U \psi \stackrel{\text{def}}{=} \psi \lor (\varphi \land \varphi \text{SU } \psi) \quad \varphi S \psi \stackrel{\text{def}}{=} \psi \lor (\varphi \land \varphi \text{SS } \psi) \]

\( w, i \models \varphi U \psi \) if \( \exists k \; i \leq k \) and \( w, k \models \psi \) and \( \forall j \; (i \leq j < k \rightarrow w, j \models \varphi) \)

\( w, i \models \varphi S \psi \) if \( \exists k \; i \geq k \) and \( w, k \models \psi \) and \( \forall j \; (i \geq j > k \rightarrow w, j \models \varphi) \)

Definition: Derived modalities

\[ X \varphi \stackrel{\text{def}}{=} \bot \text{ SU } \varphi \quad \text{Next} \quad Y \varphi \stackrel{\text{def}}{=} \bot \text{ SS } \varphi \quad \text{Yesterday} \]

\( w, i \models X \varphi \) if \( \exists k \; i < k \) and \( w, k \models \varphi \) and \( \neg \exists j \; (i < j < k) \)

\( w, i \models Y \varphi \) if \( \exists k \; i > k \) and \( w, k \models \varphi \) and \( \neg \exists j \; (i > j > k) \)

\[ SF \varphi \stackrel{\text{def}}{=} \top \text{ SU } \varphi \quad SP \varphi \stackrel{\text{def}}{=} \top \text{ SS } \varphi \]

\[ F \varphi \stackrel{\text{def}}{=} \top \text{ U } \varphi \quad P \varphi \stackrel{\text{def}}{=} \top \text{ S } \varphi \]

\[ G \varphi \stackrel{\text{def}}{=} \neg F \neg \varphi \quad H \varphi \stackrel{\text{def}}{=} \neg P \neg \varphi \]

\[ \varphi W \psi \stackrel{\text{def}}{=} (G \varphi) \lor (\varphi \text{U } \psi) \quad \text{Weak Until} \]

\[ \varphi R \psi \stackrel{\text{def}}{=} (G \psi) \lor (\psi \text{U } (\varphi \land \psi)) \quad \text{Release} \]
Temporal Specifications

Example: Specifications on the time flow \((\mathbb{N}, <)\)

- **Safety:** \(G\) good
- **MutEx:** \(\neg F(\text{crit}_1 \land \text{crit}_2)\)
- **Liveness:** \(G F \text{ active}\)
- **Response:** \(G(request \rightarrow F \text{ grant})\)
- **Response’:** \(G(request \rightarrow (\neg request \text{ SU grant}))\)
- **Release:** reset \(R\) alarm
- **Strong fairness:** \((G F \text{ request}) \rightarrow (G F \text{ grant})\)
- **Weak fairness:** \((F G \text{ request}) \rightarrow (G F \text{ grant})\)
- **Stability:** \(G \neg p \lor (\neg p \cup G p)\)
Discrete linear time flows

Definition: discrete linear time flows \((\mathbb{T}, <)\)

A linear time flow is discrete if \(SF \top \rightarrow X\top\) and \(SP \top \rightarrow Y\top\) are valid formulae.

\((\mathbb{N}, <)\) and \((\mathbb{Z}, <)\) are discrete.

\((\mathbb{Q}, <)\) and \((\mathbb{R}, <)\) are not discrete.

Exercise: For discrete linear time flows \((\mathbb{T}, <)\)

\[ \varphi SU \psi \equiv X(\varphi U \psi) \]
\[ \varphi SS \psi \equiv Y(\varphi S \psi) \]
\[ \neg X \varphi \equiv \neg X \top \lor X \neg \varphi \]
\[ \neg Y \varphi \equiv \neg Y \top \lor Y \neg \varphi \]
\[ \neg (\varphi U \psi) \equiv (G \neg \psi) \lor (\neg \psi U (\neg \varphi \land \neg \psi)) \]
\[ \equiv \neg \psi W (\neg \varphi \land \neg \psi) \]
\[ \equiv \neg \varphi R \neg \psi \]

Remark: Dense time flow \(\mathbb{R} = \mathbb{Q}\) or \(\mathbb{R} = \mathbb{R}\)

\[ \neg (\varphi U \psi) \] does not imply \(\neg \varphi R \neg \psi\).
Model checking for linear behaviors

Definition: Model checking problem

Input: A Kripke structure $M = (S, T, I, AP, \ell)$
A formula $\varphi \in LTL(AP, SU, SS)$

Question: Does $M \models \varphi$?

- Universal MC: $M \models \forall \varphi$ if $\ell(\sigma), 0 \models \varphi$ for all initial infinite runs $\sigma$ of $M$.
- Existential MC: $M \models \exists \varphi$ if $\ell(\sigma), 0 \models \varphi$ for some initial infinite run $\sigma$ of $M$.

$M \models \forall \varphi$ iff $M \not\models \exists \neg \varphi$

Theorem [11, Sistla, Clarke 85], [10, Lichtenstein & Pnueli 85]
The Model checking problem for LTL is PSPACE-complete. Proof later
Weaknesses of linear behaviors

Example:

$\varphi$: Whenever $p$ holds, it is possible to reach a state where $q$ holds.

$\varphi$ cannot be checked on linear behaviors.

We need to consider the computation-trees.
Weaknesses of FO specifications

Example:

\( \psi \): The system has an infinite active run, along which it may always reach an inactive state.

\( \psi \) cannot be expressed in FO.

We need quantifications on runs:

\[ \psi = EG(\text{Active} \land EF \neg \text{Active}) \]

- E: for some infinite run
- A: for all infinite runs
MSO Specifications

Definition: Syntax of MSO(AP, <)

\[ \varphi ::= \bot \mid p(x) \mid x = y \mid x < y \mid x \in X \mid \neg \varphi \mid \varphi \lor \varphi \mid \exists x \varphi \mid \exists X \varphi \]

where \( p \in \text{AP} \), \( x, y \) are first-order variables and \( X \) is a second-order variable.

Definition: Semantics of MSO(AP, <)

Let \( w = (T, <, \lambda) \) be a temporal structure over \( \text{AP} \).

An assignment \( \nu \) maps first-order variables to time points in \( T \) and second-order variables to sets of time points.

The semantics of first-order constructs is unchanged.

\[ w, \nu \models x \in X \quad \text{if} \quad \nu(x) \in \nu(X) \]
\[ w, \nu \models \exists X \varphi \quad \text{if} \quad w, \nu[X \mapsto T] \models \varphi \text{ for some } T \subseteq T \]

where \( \nu[X \mapsto T] \) maps \( X \) to \( T \) and keeps unchanged the other assignments.
MSO vs Temporal

MSO logic

- MSO(\(<\)) has a good expressive power
  ... but MSO(\(<\))-formulae are not easy to write and to understand.
- MSO(\(<\)) is decidable on computation trees
  ... but satisfiability and model checking are non elementary.

We need a temporal logic

- with no explicit variables,
- allowing quantifications over runs,
- usual specifications should be easy to write and read,
- with good complexity for satisfiability and model checking problems,
- with good expressive power.

Computation Tree Logic CTL* introduced by Emerson & Halpern (1986).
**Definition: Syntax of the Computation Tree Logic $\text{CTL}^\ast(\text{AP}, \text{SU})$**

\[ \varphi ::= \bot \mid p \quad (p \in \text{AP}) \mid \neg \varphi \mid \varphi \lor \varphi \mid \varphi \text{ SU } \varphi \mid \text{E } \varphi \mid \text{A } \varphi \]

We may also add the past modality $\text{SS}$.

**Definition: Semantics of $\text{CTL}^\ast(\text{AP}, \text{SU})$**

Let $M = (S, T, I, \text{AP}, \ell)$ be a Kripke structure.
Let $\sigma = s_0s_1s_2 \cdots$ be an infinite run of $M$.

- $M, \sigma, i \models p$ if $p \in \ell(s_i)$
- $M, \sigma, i \models \varphi \text{ SU } \psi$ if $\exists k > i, M, \sigma, k \models \psi$ and $\forall i < j < k, M, \sigma, j \models \varphi$
- $M, \sigma, i \models \text{E } \varphi$ if $M, \sigma', i \models \varphi$ for some infinite run $\sigma'$ such that $\sigma'[i] = \sigma[i]$
- $M, \sigma, i \models \text{A } \varphi$ if $M, \sigma', i \models \varphi$ for all infinite runs $\sigma'$ such that $\sigma'[i] = \sigma[i]$

where $\sigma[i] = s_0 \cdots s_i$.

**Remark:**

- $\sigma'[i] = \sigma[i]$ means that future is branching but past is not.
Example: Some specifications

- **EF \( \varphi \):** \( \varphi \) is possible
- **AG \( \varphi \):** \( \varphi \) is an invariant
- **AF \( \varphi \):** \( \varphi \) is unavoidable
- **EG \( \varphi \):** \( \varphi \) holds globally along some path

Remark: Some equivalences

- **A \( \varphi \) \equiv \neg E \neg \varphi**
- **E(\( \varphi \lor \psi \)) \equiv E \varphi \lor E \psi**
- **A(\( \varphi \land \psi \)) \equiv A \varphi \land A \psi**

Theorem: **CTL* \( \subseteq \) MSO**

For each \( \varphi \in \text{CTL}^*(\text{AP, SU}) \) we can construct an equivalent formula with two free variables \( \tilde{\varphi}(X, x) \in \text{MSO}(\text{AP, } \prec) \).
Model checking of $\text{CTL}^*$

Definition: Existential and universal model checking

Let $M = (S, T, I, \text{AP}, \ell)$ be a Kripke structure and $\varphi \in \text{CTL}^*$ a formula.

$M \models \exists \varphi$ if $M, \sigma, 0 \models \varphi$ for some initial infinite run $\sigma$ of $M$.

$M \models \forall \varphi$ if $M, \sigma, 0 \models \varphi$ for all initial infinite runs $\sigma$ of $M$.

Remark: $M \models \forall \varphi$ iff $M \not\models \exists \neg \varphi$

Definition: Model checking problems $\text{MC}_{\text{CTL}^*}^\forall$ and $\text{MC}_{\text{CTL}^*}^\exists$

Input: A Kripke structure $M = (S, T, I, \text{AP}, \ell)$ and a formula $\varphi \in \text{CTL}^*$

Question: Does $M \models \forall \varphi$? or Does $M \models \exists \varphi$?

Theorem:

The model checking problem for $\text{CTL}^*$ is PSPACE-complete. \hspace{1cm} \text{Proof later}
**State formulae and path formulae**

**Definition: State formulae**

\[ \varphi \in \text{CTL}^* \text{ is a state formula if } \forall M, \sigma, \sigma', i, j \text{ such that } \sigma(i) = \sigma'(j) \text{ we have} \]

\[ M, \sigma, i \models \varphi \iff M, \sigma', j \models \varphi \]

If \( \varphi \) is a state formula and \( M = (S, T, I, AP, \ell) \), define

\[ M, s \models \varphi \text{ if } M, \sigma, 0 \models \varphi \text{ for some infinite run } \sigma \text{ of } M \text{ with } \sigma(0) = s \]

and

\[ \llbracket \varphi \rrbracket^M = \{ s \in S \mid M, s \models \varphi \} \]

**Example: State formulae**

Atomic propositions are state formulae:

\[ \llbracket p \rrbracket = \{ s \in S \mid p \in \ell(s) \} \]

State formulae are closed under boolean connectives.

\[ \llbracket \neg \varphi \rrbracket = S \setminus \llbracket \varphi \rrbracket \quad \llbracket \varphi_1 \lor \varphi_2 \rrbracket = \llbracket \varphi_1 \rrbracket \cup \llbracket \varphi_2 \rrbracket \]

Formulae of the form \( E \varphi \) or \( A \varphi \) are state formulae, provided \( \varphi \) is future.

Remark:

\[ M \models \exists \varphi \text{ iff } I \cap \llbracket E \varphi \rrbracket \neq \emptyset \quad M \models \forall \varphi \text{ iff } I \subseteq \llbracket A \varphi \rrbracket \]

**Definition: Alternative syntax**

State formulae

\[ \varphi ::= \bot \mid p \ (p \in AP) \mid \neg \varphi \mid \varphi \lor \varphi \mid E \psi \mid A \psi \]

Path formulae

\[ \psi ::= \varphi \mid \neg \psi \mid \psi \lor \psi \mid \psi \mathbf{SU} \psi \]
**Definition: Computation Tree Logic CTL**

**CTL(AP, X, U)**

**Syntax:**

\[ \varphi ::= \bot \mid p \ (p \in AP) \mid \neg \varphi \mid \varphi \lor \varphi \mid \text{EX} \varphi \mid \text{AX} \varphi \mid E \varphi \mathbin{U} \varphi \mid A \varphi \mathbin{U} \varphi \]

The semantics is inherited from **CTL\(^*\).**

**Remark: All CTL formulae are state formulae**

\[ [\varphi]^M = \{ s \in S \mid M, s \models \varphi \} \]

**Examples: Macros**

- **EF** \(\varphi = \text{E} \top \mathbin{U} \varphi\) and **AG** \(\varphi = \neg \text{EF} \neg \varphi\)
- **AF** \(\varphi = \text{A} \top \mathbin{U} \varphi\) and **EG** \(\varphi = \neg \text{AF} \neg \varphi\)
- **AG**(req \(\rightarrow\) **EF** grant)
- **AG**(req \(\rightarrow\) **AF** grant)
**Definition: Semantics**

All CTL-formulae are state formulae. Hence, we have a simpler semantics.
Let $M = (S, T, I, AP, \ell)$ be a Kripke structure without deadlocks and let $s \in S$.

- $M, s \models p$ if $p \in \ell(s)$
- $M, s \models \text{EX } \varphi$ if $\exists s \rightarrow s'$ with $M, s' \models \varphi$
- $M, s \models \text{AX } \varphi$ if $\forall s \rightarrow s'$ we have $M, s' \models \varphi$
- $M, s \models \text{E } \varphi \text{ U } \psi$ if $\exists s = s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \cdots s_k$ finite path, with $M, s_k \models \psi$ and $M, s_j \models \varphi$ for all $0 \leq j < k$
- $M, s \models \text{A } \varphi \text{ U } \psi$ if $\forall s = s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \cdots$ infinite paths, $\exists k \geq 0$ with $M, s_k \models \psi$ and $M, s_j \models \varphi$ for all $0 \leq j < k$

**Theorem:** $\text{CTL} \subseteq \text{MSO}$

For each $\varphi \in \text{CTL}(AP, X, U)$ we can construct an equivalent formula with one free variable $\tilde{\varphi}(x) \in \text{MSO}(AP, <)$.

NB. Here models are computation trees.
Example:

\[ \begin{align*}
[\text{EX } p] &= \{1, 2, 3, 5, 6\} \\
[\text{AX } p] &= \{3, 6\} \\
[\text{EF } p] &= \{1, 2, 3, 4, 5, 6, 7, 8\} \\
[\text{AF } p] &= \{2, 3, 5, 6, 7\} \\
[\text{E } q \cup r] &= \{1, 2, 3, 4, 5, 6\} \\
[\text{A } q \cup r] &= \{2, 3, 4\}
\end{align*} \]
Remark: Equivalent formulae

- $AX \varphi \equiv \neg EX \neg \varphi,$
- $\neg (\varphi U \psi) \equiv G \neg \psi \lor (\neg \psi U (\neg \varphi \land \neg \psi))$
- $A \varphi U \psi \equiv \neg EG \neg \psi \land \neg E(\neg \psi U (\neg \varphi \land \neg \psi))$
- $AG(req \to F\,grant) \equiv AG(req \to AF\,grant)$
- $AGF \varphi \equiv AGAF \varphi$
- $EFG \varphi \equiv EFEG \varphi$
- $EGEF \varphi \neq EGF \varphi \neq EGAF \varphi$
- $AFAG \varphi \neq AFAG \varphi \neq AFEG \varphi$
- $EGEX \varphi \neq EGX \varphi \neq EGAX \varphi$
**Model checking of CTL**

**Definition: Existential and universal model checking**

Let $M = (S, T, I, AP, \ell)$ be a Kripke structure and $\varphi \in \text{CTL}$ a formula.

- $M \models_\exists \varphi$ if $M, s \models \varphi$ for some $s \in I$.
- $M \models_\forall \varphi$ if $M, s \models \varphi$ for all $s \in I$.

**Remark:**

- $M \models_\exists \varphi$ iff $I \cap \llbracket \varphi \rrbracket \neq \emptyset$
- $M \models_\forall \varphi$ iff $I \subseteq \llbracket \varphi \rrbracket$
- $M \models_\forall \varphi$ iff $M \not\models_\exists \neg \varphi$

**Definition: Model checking problems $\text{MC}_{\text{CTL}}^\forall$ and $\text{MC}_{\text{CTL}}^\exists$**

**Input:** A Kripke structure $M = (S, T, I, AP, \ell)$ and a formula $\varphi \in \text{CTL}$

**Question:** Does $M \models_\exists \varphi$? or Does $M \models_\forall \varphi$?

**Theorem:**

Let $M = (S, T, I, AP, \ell)$ be a Kripke structure and $\varphi \in \text{CTL}$ a formula. The model checking problem $M \models_\exists \varphi$ is decidable in time $O(|M| \cdot |\varphi|)$.
References

*Principles of Model Checking.*  

*Systems and Software Verification. Model-Checking Techniques and Tools.*  

*Model Checking.*  


*Temporal Verification of Reactive Systems: Safety.*  

*The Complexity of Temporal Logic Model Checking.*  
References


Outline

Introduction

Models

Temporal Specifications

4 Satisfiability and Model Checking

- CTL
- Fair CTL
- Büchi automata
- From LTL to BA
- LTL
- CTL*

More on Temporal Specifications
Model checking of CTL

Theorem

Let $M = (S, T, I, AP, \ell)$ be a Kripke structure and $\varphi \in CTL$ a formula. The set $[\varphi] = \{s \in S \mid M, s \models \varphi\}$ can be computed in time $O(|M| \cdot |\varphi|)$. Hence, the model checking problem $M \models \exists \varphi$ is decidable in time $O(|M| \cdot |\varphi|)$.

Proof:

Compute $[\varphi]$ by induction on the formula.

The set $[\varphi]$ is represented by a boolean array: $L[s] = \top$ if $s \in [\varphi]$.

For each $t \in S$, the set $T^{-1}(t)$ is represented as a list.

$T^{-1}$ is an array of lists, its size is $|S| + |T|$.

for all $t \in S$ do for all $s \in T^{-1}(t)$ do ... od takes time $O(|S| + |T|)$.
Model checking of CTL

Definition: function semantics(\(\varphi\)) returns boolean array \(L\)

\[
\begin{align*}
\text{case } \varphi &= p \in AP \\
& \quad \text{for all } s \in S \text{ do } L[s] := (p \in \ell(s)) \od \quad O(|S|) \\
\text{case } \varphi &= \neg \varphi_1 \\
& \quad L_1 := \text{semantics}(\varphi_1) \\
& \quad \text{for all } s \in S \text{ do } L[s] := \neg L_1[s] \od \quad O(|S|) \\
\text{case } \varphi &= \varphi_1 \lor \varphi_2 \\
& \quad L_1 := \text{semantics}(\varphi_1); L_2 := \text{semantics}(\varphi_2) \\
& \quad \text{for all } s \in S \text{ do } L[s] := L_1[s] \lor L_2[s] \od \quad O(|S|) \\
\text{case } \varphi &= EX \varphi_1 \\
& \quad L_1 := \text{semantics}(\varphi_1) \\
& \quad \text{for all } s \in S \text{ do } L[s] := \bot \od \quad O(|S|) \\
& \quad \text{for all } t \in S \text{ do if } L_1[t] \text{ then } \text{for all } s \in T^{-1}(t) \text{ do } L[s] := \top \quad O(|S| + |T|) \\
\text{case } \varphi &= AX \varphi_1 \\
& \quad L_1 := \text{semantics}(\varphi_1) \\
& \quad \text{for all } s \in S \text{ do } L[s] := \top \od \quad O(|S|) \\
& \quad \text{for all } t \in S \text{ do if } \neg L_1[t] \text{ then } \text{for all } s \in T^{-1}(t) \text{ do } L[s] := \bot \quad O(|S| + |T|)
\end{align*}
\]
Model checking of CTL

Definition: function semantics(\(\phi\)) returns boolean array \(L\)

case \(\phi = E \varphi_1 \cup \varphi_2\)  \(O(|S| + |T|)\)

\[
\begin{align*}
L_1 & := \text{semantics}(\varphi_1); \quad L_2 := \text{semantics}(\varphi_2) \\
\text{for all } s \in S \text{ do} \\
L[s] & := L_2[s] \\
\text{if } L_2[s] \text{ then } \text{Todo.push}(s) \quad \text{// Todo is implemented with a stack} \\
\text{while } \text{Todo} \neq \emptyset \text{ do} \\
\text{Invariant 1: } \quad [\varphi_2] \cup \text{Todo} \subseteq L \subseteq [E \varphi_1 \cup \varphi_2] \\
t & := \text{Todo.pop()} \quad O(1) \\
\text{for all } s \in T^{-1}(t) \text{ do} \\
\text{if } L_1[s] \land \neg L[s] \text{ then } \text{Todo.push}(s); \quad L[s] := \top \quad O(1) \\
\text{od}
\end{align*}
\]
Model checking of CTL

Definition: function semantics(\(\varphi\)) returns boolean array \(L\)

\[
\text{case } \varphi = \mathsf{A} \varphi_1 \mathsf{U} \varphi_2 \\
L_1 := \text{semantics}(\varphi_1); \quad L_2 := \text{semantics}(\varphi_2) \\
\text{for all } s \in S \text{ do} \\
L[s] := L_2[s] \\
\text{if } L_2[s] \text{ then } \text{Todo.push}(s) /\!\!/ \text{ Todo is implemented with a stack} \\
\text{for all } s \in S \text{ do } d[s] := 0; \quad c[s] := 0 \\
\text{while } \text{Todo} \neq \emptyset \text{ do} \\
\text{Invariant 1: } \forall s \in S, \; d[s] = |T(s)| \text{ and } c[s] = |T(s) \cap (L \setminus \text{Todo})| \\
\text{Invariant 2: } \llbracket \varphi_2 \rrbracket \cup \text{Todo} \subseteq L \subseteq \llbracket \mathsf{A} \varphi_1 \mathsf{U} \varphi_2 \rrbracket \\
t := \text{Todo.pop()} \\
\text{for all } s \in T^{-1}(t) \text{ do} \\
c[s] := c[s] + 1 \\
\text{if } c[s] = d[s] \land L_1[s] \land \neg L[s] \text{ then } \text{Todo.push}(s); \quad L[s] := \top \\
\text{od} \\
\mathcal{O}(|S| + |T|) \\
\mathcal{O}(|S|) \\
\mathcal{O}(|S|) \\
\mathcal{O}(|S| + |T|) \\
|S| \times \\
\mathcal{O}(1) \\
|T| \times \\
\mathcal{O}(1) \\
\mathcal{O}(1) \\
\mathcal{O}(1)
Definition: SAT(CTL)

Input: A formula $\varphi \in \text{CTL}$

Question: Existence of a model $M$ and a state $s$ such that $M, s \models \varphi$?

Theorem: Complexity

- The model checking problem for CTL is PTIME-complete.
- The satisfiability problem for CTL is EXPTIME-complete.
Example: Fairness

Only fair runs are of interest

- Each process is enabled infinitely often: \( \bigwedge_i G F \text{run}_i \)
- No process stays ultimately in the critical section: \( \bigwedge_i \neg F_G \text{cs}_i = \bigwedge_i G F \neg \text{cs}_i \)

Definition: Fair Kripke structure

\[
M = (S, T, I, AP, \ell, F_1, \ldots, F_n) \text{ with } F_i \subseteq S.
\]

An infinite run \( \sigma \) is **fair** if it visits infinitely often each \( F_i \).
Definition: Syntax of fair-CTL

\[ \varphi ::= \bot \mid p \ (p \in \text{AP}) \mid \neg \varphi \mid \varphi \land \varphi \mid E_f X \varphi \mid A_f X \varphi \mid E_f \varphi U \varphi \mid A_f \varphi U \varphi \]

Definition: Semantics as a fragment of CTL*

Let \( M = (S, T, I, \text{AP}, \ell, F_1, \ldots, F_n) \) be a fair Kripke structure.

Let,

\[ E_f \varphi = E(\text{FairRun} \land \overline{\varphi}) \quad \text{and} \quad A_f \varphi = A(\text{FairRun} \rightarrow \overline{\varphi}) \]

where

\[ \text{FairRun} = \bigwedge_i \text{GF} F_i \]

Then,

\[ [\varphi]_f = [\overline{\varphi}] \]

Lemma: CTL\(_f\) cannot be expressed in CTL
Proof: $CTL_f$ cannot be expressed in CTL

Consider the Kripke structure $M_k$ defined by:

\[
\begin{array}{cccccc}
2k & \rightarrow & 2k - 1 & \rightarrow & 2k - 2 & \rightarrow & 2k - 3 & \rightarrow & \cdots & 4 & \rightarrow & 3 & \rightarrow & 2 & \rightarrow & 1 \\
p & \rightarrow & \neg p & \rightarrow & p & \rightarrow & \neg p & \rightarrow & \neg p & \rightarrow & p & \rightarrow & \neg p & \rightarrow & p & \rightarrow & \neg p \\
\end{array}
\]

- $M_k, 2k \models E\ G\ F\ p$ but $M_k, 2k - 2 \nvDash E\ G\ F\ p$
- If $\varphi \in CTL$ and $|\varphi| \leq m \leq k$ then
  
  $M_k, 2k \models \varphi$ iff $M_k, 2m \models \varphi$
  
  $M_k, 2k - 1 \models \varphi$ iff $M_k, 2m - 1 \models \varphi$

If the fairness condition is $\ell^{-1}(p)$ then $E_f \top$ cannot be expressed in CTL.
Model checking of $\text{CTL}_f$

Theorem

The model checking problem for $\text{CTL}_f$ is decidable in time $O(|M| \cdot |\varphi|)$

Proof: Computation of FairStates $= \{ s \in S \mid M, s \models E_f \top \}$

Compute the SCC of $M$ in time $O(|M|)$, e.g., with Tarjan’s algorithm.
Let $S'$ be the union of the (non trivial) SCCs which intersect each $F_i$.
Then, FairStates is the set of states that can reach $S'$: $\text{FairStates} = [E \top U S']$.
Note that reachability can be computed in linear time.

Proof: Reductions

$E_f X \varphi = E X(\text{FairStates} \land \varphi)$ \hspace{1cm} and \hspace{1cm} $E_f \varphi U \psi = E \varphi U (\text{FairStates} \land \psi)$

It remains to deal with $A_f \varphi U \psi$. We have

$A_f \varphi U \psi = \neg E_f G \neg \psi \land \neg E_f (\neg \psi U (\neg \varphi \land \neg \psi))$

Hence, we only need to compute the semantics of $E_f G \varphi$.

Let $M_\varphi$ be the restriction of $M$ to $[\varphi]_f$. Then,

$M, s \models E_f G \varphi$ iff $M_\varphi, s \models E_f \top$.

We apply the above algorithm for $E_f \top$ to $M_\varphi$. 
Büchi automata

Definition:

A Büchi automaton (BA) is a tuple $A = (Q, \Sigma, I, T, F)$ where

- $Q$: finite set of states
- $\Sigma$: finite set of labels
- $I \subseteq Q$: set of initial states
- $T \subseteq Q \times \Sigma \times Q$: set of transitions (non-deterministic)
- $F \subseteq Q$: set of final (repeated) states

Run: $\rho = q_0, a_0, q_1, a_1, q_2, a_2, q_3, \ldots$ with $(q_i, a_i, q_{i+1}) \in T$ for all $i \geq 0$.

$\rho$ is initial if $q_0 \in I$.

$\rho$ is final (successful) if $q_i \in F$ for infinitely many $i$’s.

$\rho$ is accepting if it is both initial and final.

$L(A) = \{ a_0 a_1 a_2 \cdots \in \Sigma^\omega \mid \exists \rho = q_0, a_0, q_1, a_1, q_2, a_2, q_3, \ldots \text{ accepting run} \}$

A language $L \subseteq \Sigma^\omega$ is $\omega$-regular if it can be accepted by some Büchi automaton.
Büchi automata

Examples:

Infinitely many $a$’s:

No deterministic Büchi automaton for this language.

Finitely many $a$’s:

Whenever $a$ then later $b$:
Büchi automata

Properties

Büchi automata are closed under union, intersection, complement.

- Union: trivial
- Intersection: easy (exercise)
- Complement: difficult

Let $L = \Sigma^* \left( a\Sigma^{n-1}b \cup b\Sigma^{n-1}a \right) \Sigma^\omega$

Any non deterministic Büchi automaton for $\Sigma^\omega \setminus L$ has at least $2^n$ states.
Theorem: Büchi

Let \( L \subseteq \Sigma^\omega \) be a language. The following are equivalent:

- \( L \) is \( \omega \)-regular
- \( L \) is \( \omega \)-rational, i.e., \( L \) is a finite union of languages of the form \( L_1 \cdot L_2^\omega \) where \( L_1, L_2 \subseteq \Sigma^+ \) are rational.
- \( L \) is MSO-definable, i.e., there is a sentence \( \varphi \in \text{MSO}_\Sigma(\prec) \) such that 
  \[ L = L(\varphi) = \{ w \in \Sigma^\omega \mid w \models \varphi \} \].

Exercises:

1. Construct a BA for \( L(\varphi) \) where \( \varphi \) is the FO\(\Sigma(\prec) \) sentence
   \[
   (\forall x, (P_a(x) \rightarrow \exists y > x, P_a(y))) \rightarrow (\forall x, (P_b(x) \rightarrow \exists y > x, P_c(y)))
   \]

2. Given BA for \( L_1 \subseteq \Sigma^\omega \) and \( L_2 \subseteq \Sigma^\omega \), construct BA for
   
   next\( (L_1) = \Sigma \cdot L_1 \)

   strict – until\( (L_1, L_2) = \{ uv \in \Sigma^\omega \mid u \in \Sigma^+ \land v \in L_2 \land 
   u''v \in L_1 \text{ for all } u', u'' \in \Sigma^+ \text{ with } u = u'u'' \} \)
Definition: final condition on states or on transitions

$A = (Q, \Sigma, I, T, F_1, \ldots, F_n)$ with $F_i \subseteq Q$.
An infinite run $\sigma$ is final (successful) if it visits infinitely often each $F_i$.

$A = (Q, \Sigma, I, T, T_1, \ldots, T_n)$ with $T_i \subseteq T$.
An infinite run $\sigma$ is final if it uses infinitely many transitions from each $T_i$.

Example: Infinitely many $a$’s and infinitely many $b$’s

Theorem:

1. GBA and BA have the same expressive power.
2. Checking whether a BA or GBA has an accepting run is NLOGSPACE-complete.
Unambiguous, Complete, Prophetic (G)BA

Definition: Unambiguous, Complete, Prophetic Büchi automata

A BA or GBA $\mathcal{A}$ is unambiguous if every word has at most one accepting run in $\mathcal{A}$. A BA or GBA $\mathcal{A}$ is complete if every word has at least one accepting run in $\mathcal{A}$. A BA or GBA $\mathcal{A}$ is prophetic if every word has exactly one final run in $\mathcal{A}$.

Rem: when $I = Q$ then accepting = final. Hence, when $I = Q$ then prophetic = unambiguous and complete.

Examples: Unambiguous, Complete, Prophetic

- Finitely many $a$'s.
- $G(a \rightarrow F b)$ with $\Sigma = \{a, b, c\}$.

Proposition: Closure properties

Prophetic BA or GBA are closed under boolean operations (union, intersection, complement).

Theorem: Prophetic Büchi automata (Carton-Michel 2003)

Every $\omega$-regular language can be accepted by a prophetic BA.
**Definition: SBT: Synchronous (letter to letter) Büchi transducer**

Let $A$ and $B$ be two alphabets.

A synchronous Büchi transducer from $A$ to $B$ is a tuple $\mathcal{A} = (Q, A, I, T, F, \mu)$ where $(Q, A, I, T, F)$ is a Büchi automaton (input) and $\mu : T \rightarrow B$ is the output function.

It computes the relation

$$ [\mathcal{A}] = \{ (u, v) \in A^\omega \times B^\omega \mid \exists \rho = q_0, a_0, q_1, a_1, q_2, a_2, q_3, \ldots \text{ accepting run} $$

with $u = a_0 a_1 a_2 \cdots$ and $v = \mu(\rho)$,

i.e., $v = b_0 b_1 b_2 \cdots$ with $b_i = \mu(q_i, a_i, q_{i+1})$ for $i \geq 0$}

If $(Q, A, I, T, F)$ is unambiguous then $[\mathcal{A}] : A^\omega \rightarrow B^\omega$ is a (partial) function, in which case we also write $[\mathcal{A}](u) = v$ for $(u, v) \in [\mathcal{A}]$.

We will also use SGBT: synchronous transducers with generalized Büchi acceptance.

**Example: Left shift with $A = B = \{a, b\}$**

![Diagram of a left shift automaton with alphabets $a$ and $b$.]
Composition of Büchi transducers

Definition: Composition

Let $A$, $B$, $C$ be alphabets.
Let $A = (Q, A, I, T, (F_i)_i, \mu)$ be an SGBT from $A$ to $B$.
Let $A' = (Q', B, I', T', (F'_j)_j, \mu')$ be an SGBT from $B$ to $C$.
Then $A \cdot A' = (Q \times Q', A, I \times I', T'', (F_i \times Q')_i, (Q \times F'_j)_j, \mu'')$ defined by:

$$\tau'' = (p, p') \xrightarrow{a} (q, q') \in T'' \text{ and } \mu''(\tau'') = c$$

iff

$$\tau = p \xrightarrow{a} q \in T \text{ and } \tau' = p' \xrightarrow{\mu(\tau)} q' \in T' \text{ and } c = \mu'(\tau')$$

is an SGBT from $A$ to $C$.
When the transducers define functions, we also denote the composition by $A' \circ A$.

Proposition: Composition

1. We have $[A \cdot A'] = [A] \cdot [A']$.
2. If $(Q, A, I, T, (F_i)_i)$ and $(Q', B, I', T', (F'_j)_j)$ are unambiguous (resp. complete, prophetic) then $(Q \times Q', A, I \times I', T'', (F_i \times Q')_i, (Q \times F'_j)_j)$ is also unambiguous (resp. complete, prophetic), and
   $\forall u \in A^\omega$ we have $[A' \circ A](u) = [A']( [A](u))$. 
**Product of Büchi transducers**

**Definition: Product**

Let $A$, $B$, $C$ be alphabets.

Let $A = (Q, A, I, T, (F_i)_i, \mu)$ be an SGBT from $A$ to $B$.

Let $A' = (Q', A, I', T', (F'_j)_j, \mu')$ be an SGBT from $A$ to $C$.

Then $A \times A' = (Q \times Q', A, I \times I', T'', (F_i \times Q')_i, (Q \times F'_j)_j, \mu'')$ defined by:

$$
\tau'' = (p, p') \xrightarrow{a} (q, q') \in T'' \text{ and } \mu''(\tau'') = (b, c)
$$

iff

$$
\tau = p \xrightarrow{a} q \in T \text{ and } b = \mu(\tau) \text{ and } \tau' = p' \xrightarrow{a} q' \in T' \text{ and } c = \mu'(\tau')
$$

is an SGBT from $A$ to $B \times C$.

**Proposition: Product**

We identify $(B \times C)^\omega$ with $B^\omega \times C^\omega$.

1. We have $[A \times A'] = \{(u, v, v') | (u, v) \in [A] \text{ and } (u, v') \in [A']\}$.

2. If $(Q, A, I, T, (F_i)_i)$ and $(Q', A, I', T', (F'_j)_j)$ are unambiguous (resp. complete, prophetic) then $(Q \times Q', A, I \times I', T'', (F_i \times Q')_i, (Q \times F'_j)_j)$ is also unambiguous (resp. complete, prophetic), and

   $\forall u \in A^\omega$ we have $[A \times A'](u) = ([A](u), [A'](u))$. 
Definition:

For a propositional formula $\xi$ over $\text{AP}$, we let $\Sigma_\xi = \{ a \in \Sigma \mid a \models \xi \}$. For instance, for $p, q \in \text{AP}$,

- $\Sigma_p = \{ a \in \Sigma \mid p \in a \}$ and $\Sigma_{\neg p} = \Sigma \setminus \Sigma_p$
- $\Sigma_{p \land q} = \Sigma_p \cap \Sigma_q$ and $\Sigma_{p \lor q} = \Sigma_p \cup \Sigma_q$
- $\Sigma_{p \land \neg q} = \Sigma_p \setminus \Sigma_q$ ...

Notation:

In automata, $s \xrightarrow{\Sigma_\xi} s'$ stands for the set of transitions $\{s\} \times \Sigma_\xi \times \{s'\}$.

To simplify the pictures, we use $s \xrightarrow{\xi} s'$ instead of $s \xrightarrow{\Sigma_\xi} s'$.

Example: $G(p \rightarrow F q)$

\[
\begin{align*}
\neg p \lor q & \quad p \land \neg q & \quad \neg q \\
\text{1} & \quad \text{1} & \quad \text{2}
\end{align*}
\]
Semantics of \textit{LTL} with sequential functions

**Definition:** Semantics of \( \varphi \in \text{LTL}(\text{AP}, \text{SU}, \text{SS}) \)

Let \( \Sigma = 2^{\text{AP}} \) and \( B = \{0, 1\} \).

Define \( [\varphi] : \Sigma^\omega \to B^\omega \) by \( [\varphi](u) = b_0 b_1 b_2 \cdots \) with \( b_i = \begin{cases} 1 & \text{if } u, i \models \varphi \\ 0 & \text{otherwise.} \end{cases} \)

**Example:**

\[
[p \text{ SU } q](\emptyset\{q\}\{p\}\emptyset\{p\}\{p\}\{q\}\emptyset\{p\}\{p, q\}\emptyset^\omega) = 1001110110^\omega \\
[\text{X } p](\emptyset\{q\}\{p\}\emptyset\{p\}\{p\}\{q\}\emptyset\{p\}\{p, q\}\emptyset^\omega) = 0101100110^\omega \\
[F \text{ p}](\emptyset\{q\}\{p\}\emptyset\{p\}\{p\}\{q\}\emptyset\{p\}\{p, q\}\emptyset^\omega) = 1111111110^\omega
\]

The aim is to compute \([\varphi]\) with synchronous Büchi transducers (actually, SGBT).

For past formulas, we use deterministic and complete GBA.

For future formulas, we use prophetic GBA.
Synchronous Büchi transducer for $p \mathbf{SU} q$

Example: An SBT for $[p \mathbf{SU} q]$

Lemma: The input BA is prophetic

For all $u = a_0a_1a_2\cdots \in \Sigma^\omega$, there is a unique final run $\rho = s_0, a_0, s_1, a_1, s_2, a_2, s_3, \ldots$ of $A$ on $u$.

The run $\rho$ satisfies for all $i \geq 0$, $s_i = \begin{cases} 1 & \text{if } u, i \models q \\ 2 & \text{if } u, i \models \neg q \land (p \mathbf{U} q) \\ 3 & \text{if } u, i \models \neg (p \mathbf{U} q) \end{cases}$

Hence, the SBT computes $[p \mathbf{SU} q]$. 
Example: An SBT for $[p \mathcal{U} q]$

The automaton is prophetic (same input BA as for $p \mathcal{U} q$).
This SBT computes $[p \mathcal{U} q]$. 
Special cases of Until: Future and Next

Example: $F q = \top U q$ and $X q = \bot SU q$

Exercise: Give SBT’s for the following formulae:

$SF q$, $SG q$, $p SR q$, $p SS q$, $Y q$, $G q$, $p R q$, $p S q$, $G(p \rightarrow F q)$. 
From LTL to Büchi automata

Definition: SBT for LTL modalities

- $A_T$ from $\Sigma$ to $\mathbb{B} = \{0, 1\}$: 
  
  $\begin{array}{c}
  \circ \\
  \rightarrow \\
  \rightarrow \\
  \circ \\
  \Sigma/1
  \end{array}$

- $A_p$ from $\Sigma$ to $\mathbb{B} = \{0, 1\}$: 
  
  $\begin{array}{c}
  \circ \\
  \rightarrow \\
  \rightarrow \\
  \circ \\
  p/1
  \end{array}$

- $A_{\neg}$ from $\mathbb{B}$ to $\mathbb{B}$: 
  
  $\begin{array}{c}
  \circ \\
  \rightarrow \\
  \rightarrow \\
  \circ \\
  0/1
  \end{array}$

- $A_\lor$ from $\mathbb{B}^2$ to $\mathbb{B}$: 
  
  $\begin{array}{c}
  \circ \\
  \rightarrow \\
  \rightarrow \\
  \circ \\
  0,0/0
  \end{array}$

- $A_\land$ from $\mathbb{B}^2$ to $\mathbb{B}$: 
  
  $\begin{array}{c}
  \circ \\
  \rightarrow \\
  \rightarrow \\
  \circ \\
  0,1/0
  \end{array}$
Definition: SBT for LTL modalities (cont.)

- $A_{SU}$ from $\mathbb{B}^2$ to $\mathbb{B}$:
  Prophetic

- $A_{SS}$ from $\mathbb{B}^2$ to $\mathbb{B}$:
  Deterministic & Complete
  Not prophetic
From LTL to Büchi automata

Definition: Translation from LTL to SGBT

For each $\xi \in \text{LTL}(\text{AP}, \text{SU}, \text{SS})$ we define inductively an SGBT $A_\xi$ as follows:

- $A_T$ and $A_p$ for $p \in \text{AP}$ are already defined
- $A_{\neg \varphi} = A_{\neg} \circ A_{\varphi}$
- $A_{\varphi \lor \psi} = A_{\lor} \circ (A_{\varphi} \times A_{\psi})$
- $A_{\varphi \text{SS} \psi} = A_{\text{SS}} \circ (A_{\varphi} \times A_{\psi})$
- $A_{\varphi \text{SU} \psi} = A_{\text{SU}} \circ (A_{\varphi} \times A_{\psi})$

Theorem: Correctness of the translation

For each $\xi \in \text{LTL}(\text{AP}, \text{SU}, \text{SS})$, we have $[A_\xi] = [\xi]$ and $A_\xi$ is unambiguous.
Moreover, the number of states of $A_\xi$ is at most $2|\xi_{\text{SS}} \cdot 3|\xi_{\text{SU}}$
the number of acceptance conditions is $|\xi_{\text{SU}}$
where $|\xi_{\text{SS}}$ (resp. $|\xi_{\text{SU}}$) is the number of SS (resp. SU) occurring in $\xi$.

Remark:

- If a subformula $\varphi$ occurs serveral time in $\xi$, we only need one copy of $A_{\varphi}$.
- We may also use automata for other modalities: $A_X$ (2 states), $A_U$, …
Useful simplifications

Reducing the number of temporal subformulae

\[(X \varphi) \land (X \psi) \equiv X(\varphi \land \psi)\]
\[(X \varphi) \SU (X \psi) \equiv X(\varphi \SU \psi)\]
\[(G \varphi) \land (G \psi) \equiv G(\varphi \land \psi)\]
\[GF \varphi \lor GF \psi \equiv GF(\varphi \lor \psi)\]
\[(\varphi_1 \SU \psi) \land (\varphi_2 \SU \psi) \equiv (\varphi_1 \land \varphi_2) \SU \psi\]
\[(\varphi \SU \psi_1) \lor (\varphi \SU \psi_2) \equiv \varphi \SU (\psi_1 \lor \psi_2)\]

Merging equivalent states

Let \(\mathcal{A} = (Q, \Sigma, I, T, (F_i)_i, \mu)\) be an SGBT and \(s_1, s_2 \in Q\). We can merge \(s_1\) and \(s_2\) if they satisfy the same final conditions:

\[s_1 \in F_i \iff s_2 \in F_i\quad \text{for all } i\]

and they have the same outgoing transitions: \(\forall a \in \Sigma, \forall s \in Q\),

\[\tau_1 = (s_1, a, s) \in T \iff \tau_2 = (s_2, a, s) \in T\]

and \(\mu(\tau_1) = \mu(\tau_2)\).
Other constructions

▶ Tableau construction. See for instance [16, Wolper 85]
  + : Easy definition, easy proof of correctness
  + : Works both for future and past modalities
  − : Inefficient without strong optimizations

▶ Using Very Weak Alternating Automata [17, Gastin & Oddoux 01].
  + : Very efficient
  − : Only for future modalities
  Online tool: http://www.lsv.ens-cachan.fr/~gastin/ltl2ba/

▶ Using reduction rules [7, Demri & Gastin 10].
  + : Efficient and produces small automata
  + : Can be used by hand on real examples
  − : Only for future modalities

▶ The domain is still very active.
Some References

Checking that finite state concurrent programs satisfy their linear specification.

The tableau method for temporal logic: An overview,

The complexity of propositional linear temporal logic.

*Fast LTL to Büchi automata translation*.
http://www.lsv.ens-cachan.fr/~gastin/mes-publis.php

*Specification and Verification using Temporal Logics*.
In Modern applications of automata theory, IISc Research Monographs 2.
http://www.lsv.ens-cachan.fr/~gastin/mes-publis.php
Satisfiability for LTL over \((\mathbb{N}, <)\)

Let \(\text{AP}\) be the set of atomic propositions and \(\Sigma = 2^{\text{AP}}\).

**Definition: Satisfiability problem**

**Input:** A formula \(\varphi \in \text{LTL}(\text{AP}, \text{SU}, \text{SS})\)

**Question:** Existence of \(w \in \Sigma^\omega\) and \(i \in \mathbb{N}\) such that \(w, i \models \varphi\).

**Definition: Initial Satisfiability problem**

**Input:** A formula \(\varphi \in \text{LTL}(\text{AP}, \text{SU}, \text{SS})\)

**Question:** Existence of \(w \in \Sigma^\omega\) such that \(w, 0 \models \varphi\).

Remark: \(\varphi\) is satisfiable iff \(F \varphi\) is *initially* satisfiable.

**Definition: (Initial) validity**

\(\varphi\) is valid iff \(\neg \varphi\) is not satisfiable.

**Theorem [11, Sistla, Clarke 85], [10, Lichtenstein & Pnueli 85]**

The satisfiability problem for LTL is PSPACE-complete.
Model checking for LTL

Definition: Model checking problem

Input: A Kripke structure $M = (S, T, I, AP, \ell)$
A formula $\varphi \in \text{LTL}(AP, SU, SS)$

Question: Does $M \models \varphi$?

- Universal MC: $M \models \forall \varphi$ if $\ell(\sigma), 0 \models \varphi$ for all initial infinite runs of $M$.
- Existential MC: $M \models \exists \varphi$ if $\ell(\sigma), 0 \models \varphi$ for some initial infinite run of $M$.

$$M \models \forall \varphi \iff M \not\models \exists \neg \varphi$$

Theorem [11, Sistla, Clarke 85], [10, Lichtenstein & Pnueli 85]

The Model checking problem for LTL is PSPACE-complete
\[MC^3(SU) \leq_P SAT(SU)\]

[11, Sistla & Clarke 85]

Let \( M = (S, T, I, AP, \ell) \) be a Kripke structure and \( \varphi \in LTL(AP, SU) \)

Introduce new atomic propositions: \( AP_S = \{ at_s \mid s \in S \} \)

Define \( AP' = AP \sqcup AP_S \quad \Sigma' = 2^{AP'} \quad \pi : \Sigma^\omega \rightarrow \Sigma^\omega \) by \( \pi(a) = a \cap AP \).

Let \( w \in \Sigma^\omega \). We have \( w \models \varphi \) iff \( \pi(w) \models \varphi \)

Define \( \psi_M \in LTL(AP', X, F) \) of size \( O(|M|^2) \) by

\[
\psi_M = \left( \bigvee_{s \in I} at_s \right) \land G \left( \bigvee_{s \in S} \left( at_s \land \bigwedge_{t \neq s} \neg at_t \land \bigwedge_{p \in \ell(s)} p \land \bigwedge_{p \notin \ell(s)} \neg p \land \bigvee_{t \in T(s)} X at_t \right) \right)
\]

Let \( w = a_0a_1a_2 \cdots \in \Sigma^\omega \). Then, \( w \models \psi_M \) iff there exists an initial infinite run \( \sigma = s_0, s_1, s_2, \ldots \) of \( M \) such that \( \pi(w) = \ell(\sigma) \) and \( a_i \cap AP_S = \{ at_{s_i} \} \) for all \( i \geq 0 \).

Therefore, \( M \models \exists \varphi \) iff \( \psi_M \land \varphi \) is initially satisfiable

\( M \models \forall \varphi \) iff \( \psi_M \land \neg \varphi \) is not initially satisfiable

Remark: we also have \( MC^3(X, F) \leq_P SAT(X, F) \).
Definition: QBF

Input: A formula \( \gamma = Q_1 x_1 \cdots Q_n x_n \gamma' \) with \( \gamma' = \bigwedge_{1 \leq i \leq m} \bigvee_{1 \leq j \leq k_i} a_{ij} \) (CNF)

\( Q_i \in \{\forall, \exists\} \) and \( a_{ij} \in \{x_1, \neg x_1, \ldots, x_n, \neg x_n\} \).

Question: Is \( \gamma \) valid?

Definition:

An assignment of the variables \( \{x_1, \ldots, x_n\} \) is a word \( v = v_1 \cdots v_n \in \{0, 1\}^n \).
We write \( v[i] \) for the prefix of length \( i \).
Let \( V \subseteq \{0, 1\}^n \) be a set of assignments.

- \( V \) is valid (for \( \gamma' \)) if \( v \models \gamma' \) for all \( v \in V \),
- \( V \) is closed (for \( \gamma \)) if \( \forall v \in V, \forall 1 \leq i \leq n \) s.t. \( Q_i = \forall \),
  \( \exists v' \in V \) s.t. \( v[i - 1] = v'[i - 1] \) and \( v'_i = 1 - v_i \).

Proposition:

\( \gamma \) is valid \iff \( \exists V \subseteq \{0, 1\}^n \) s.t. \( V \) is nonempty valid and closed
Let $\gamma = Q_1 x_1 \cdots Q_n x_n \bigwedge \bigvee_{1 \leq i \leq m} a_{ij}$ with $Q_i \in \{\forall, \exists\}$ and $a_{ij}$ literals.

Consider the KS $M$:

Let $\psi_{ij} = \begin{cases} G(x_k^f \rightarrow s_k R \neg a_{ij}) & \text{if } a_{ij} = x_k \\ G(x_k^t \rightarrow s_k R \neg a_{ij}) & \text{if } a_{ij} = \neg x_k \end{cases}$ and $\psi = \bigwedge_{i,j} \psi_{ij}$.

Let $\varphi_i = G(e_{i-1} \rightarrow (\neg s_{i-1} U x_i^t) \land (\neg s_{i-1} U x_i^f))$ and $\varphi = \bigwedge_{i | Q_i = \forall} \varphi_i$.

Then, $\gamma$ is valid iff $M \models_\exists \psi \land \varphi$. 
Theorem: Complexity of LTL

The following problems are PSPACE-complete:

- \( \text{SAT}(\text{LTL}(\text{SU}, \text{SS})) \), \( \text{MC}^\forall(\text{LTL}(\text{SU}, \text{SS})) \), \( \text{MC}^\exists(\text{LTL}(\text{SU}, \text{SS})) \)
- \( \text{SAT}(\text{LTL}(X, F)) \), \( \text{MC}^\forall(\text{LTL}(X, F)) \), \( \text{MC}^\exists(\text{LTL}(X, F)) \)
- \( \text{SAT}(\text{LTL}(U)) \), \( \text{MC}^\forall(\text{LTL}(U)) \), \( \text{MC}^\exists(\text{LTL}(U)) \)
- The restriction of the above problems to a unique propositional variable

The following problems are NP-complete:

- \( \text{SAT}(\text{LTL}(F)) \), \( \text{MC}^\exists(\text{LTL}(F)) \)
Definition: Syntax of the Computation Tree Logic $\text{CTL}^*$

$$\varphi ::= \bot \mid p \ (p \in \text{AP}) \mid \neg \varphi \mid \varphi \lor \varphi \mid X \varphi \mid \varphi U \varphi \mid E \varphi \mid A \varphi$$

Theorem

The model checking problem for $\text{CTL}^*$ is PSPACE-complete.

Proof:

PSPACE-hardness: follows from $\text{LTL} \subseteq \text{CTL}^*$.

PSPACE-easiness: reduction to LTL-model checking by inductive eliminations of path quantifications.
## Satisfiability for $\text{CTL}^*$

### Definition: $\text{SAT}(\text{CTL}^*)$

**Input:** A formula $\varphi \in \text{CTL}^*$

**Question:** Existence of a model $M$ and a run $\sigma$ such that $M, \sigma, 0 \models \varphi$?

### Theorem

The satisfiability problem for $\text{CTL}^*$ is 2-EXPTIME-complete.
Outline

Introduction

Models

Temporal Specifications

Satisfiability and Model Checking

More on Temporal Specifications
  - Expressivity
  - Ehrenfeucht-Fraïssé games
  - Separation
Expressivity

Definition: Equivalence

Let $C$ be a class of time flows.

Two formulae $\varphi, \psi \in \text{TL}(\text{AP}, \text{SU}, \text{SS})$ are equivalent over $C$ if for all temporal structures $w = (T, <, \lambda)$ over $C$ and all time points $t \in T$ we have

$$w, t \models \varphi \text{ iff } w, t \models \psi$$

Two formulae $\varphi \in \text{TL}(\text{AP}, \text{SU}, \text{SS})$ and $\psi(x) \in \text{FO}_{\text{AP}}(<)$ are equivalent over $C$ if for all temporal structures $w = (T, <, \lambda)$ over $C$ and all time points $t \in T$ we have

$$w, t \models \varphi \text{ iff } w, x \mapsto t \models \psi$$

We also write $w \models \psi(t)$.

Remark: $\text{TL}(\text{AP}, \text{SU}, \text{SS}) \subseteq \text{FO}^3_{\text{AP}}(<) \subseteq \text{FO}_{\text{AP}}(<)$

$\forall \varphi \in \text{TL}(\text{AP}, \text{SU}, \text{SS}), \exists \psi(x) \in \text{FO}^3_{\text{AP}}(<)$ such that $\varphi$ and $\psi(x)$ are equivalent.

Expressivity problem: $\text{LTL} = \text{FO}$?
**Expressivity**

**Definition: complete linear time flows**

A time flow \((\mathbb{T}, <)\) is **linear** if \(<\) is a **total** strict order.

A linear time flow \((\mathbb{T}, <)\) is **complete** if every nonempty and bounded subset of \(\mathbb{T}\) has a **least upper bound** and a **greatest lower bound**.

\((\mathbb{N}, <)\), \((\mathbb{Z}, <)\) and \((\mathbb{R}, <)\) are complete.

\((\mathbb{Q}, <)\) and \((\mathbb{R} \setminus \{0\}, <)\) are **not** complete.

**Theorem: Expressive completeness [12, Kamp 68]**

For **complete** linear time flows, \(\text{TL}(\text{AP}, \text{SU}, \text{SS}) = \text{FO}_{\text{AP}}(<)\)

**Example: Translate in \(\text{TL}(\text{AP}, \text{SU}, \text{SS})\)**

\[
\psi(x) = \neg P_a(x) \land \neg P_b(x) \land \forall y \forall z \ (P_a(y) \land P_b(z) \land y < z) \rightarrow \\
\exists v \ y < v < z \land \left( \begin{array}{c}
P_c(v) \land x < y \\
\lor \ P_d(v) \land z < x \\
\lor \ P_e(v) \land y < x < z \\
\end{array} \right)
\]

(1)
Initial equivalence

Note that $\text{FO}_{\text{AP}}(<)$ is strictly more expressive than $\text{TL}(\text{AP, SU})$ or $\text{TL}(\text{AP, SS})$.

**Definition: Initial Equivalence**

Let $\mathcal{C}$ be a class of time flows having a least element (denoted $0$). Two formulae $\varphi, \psi \in \text{TL}(\text{AP, SU, SS})$ are initially equivalent over $\mathcal{C}$ if for all temporal structures $w = (T, <, \lambda)$ over $\mathcal{C}$ we have

$$w, 0 \models \varphi \iff w, 0 \models \psi$$

Two formulae $\varphi \in \text{TL}(\text{AP, SU, SS})$ and $\psi(x) \in \text{FO}_{\text{AP}}(<)$ are initially equivalent over $\mathcal{C}$ if for all temporal structures $w = (T, <, \lambda)$ over $\mathcal{C}$ we have

$$w, 0 \models \varphi \iff w \models \psi(0)$$

Elegant algebraic proof of $\text{TL}(\text{AP, SU})$ initially equivalent to $\text{FO}_{\text{AP}}(<)$ over $(\mathbb{N}, <)$ due to Wilke 98.

See also Diekert-Gastin [18]: $\text{TL} = \text{FO} = \text{SF} = \text{AP} = \text{CFBA} = \text{VWAA}$. 
**Stavi connectives: Time flows with gaps**

### Definition: Stavi Until: \( \overline{U} \)

Let \( w = (\mathbb{T}, <, \lambda) \) be a temporal structure and \( i \in \mathbb{T} \). Then, \( w, i \models \varphi \overline{U} \psi \) if

\[
\exists k \; i < k \\
\wedge \exists j \; (i < j < k \wedge w, j \models \neg \varphi) \\
\wedge \exists j \; (i < j < k \wedge \forall \ell \; (i < \ell < j \rightarrow w, \ell \models \varphi)) \\
\wedge \forall j \; \left[ i < j < k \rightarrow \left[ \exists k' \; [j < k' \wedge \forall j' \; (i < j' < k' \rightarrow w, j' \models \varphi)] \\
\text{or} \; \left[ \forall \ell \; (j < \ell < k \rightarrow w, \ell \models \psi) \wedge \exists \ell \; (i < \ell < j \wedge w, \ell \models \neg \varphi) \right] \right] \right]
\]

Similar definition for the Stavi Since \( \overline{S} \).

### Example:

(2)

- Let \( w = (\mathbb{R} \setminus \{0\}, <, h) \) with \( h(p) = \mathbb{R}_- \) and \( h(q) = \mathbb{R}_+ \).
  Then, \( w, -1 \not\models p \overline{U} q \) but \( w, -1 \models p \overline{U} q \).

- Let \( w' = (\mathbb{R} \setminus \{0\}, <, h') \) with \( h'(p) = \mathbb{R} \setminus \{1, \frac{1}{2}, \frac{1}{4}, \ldots, 0\} \) and \( h'(q) = \mathbb{R}_+ \).
  Then, \( w', -1 \models p \overline{U} q \).
Theorem: [14, Gabbay, Hodkinson, Reynolds]

$TL(\text{AP}, \text{SU}, \text{SS}, \overline{S}, \overline{U})$ is expressively complete for $\text{FO}_{\text{AP}}(\prec)$ over the class of all linear time flows.

Exercise: Isolated gaps

Let $\varphi_p = p \text{ SU } p \land \text{SF } \neg p \land \neg (p \text{ SU } \neg p) \land \neg (p \text{ SU } \neg (p \text{ SU } \top))$.

Let $w = (T, <, \lambda)$ with $T \subseteq \mathbb{R}$ and $t \in T$.

Show that if $w, t \models \varphi_p$ then $T$ has a gap.

Let $\psi_{p,q} = \varphi_p \land (q \lor \varphi_p) \text{ SU } (q \land \neg p)$.

Show that $\psi_{p,q}$ is equivalent to $p \text{ SU } q$ over the time flow $(\mathbb{R} \setminus \{0\}, \prec)$.

Show that $TL(\text{AP}, \text{SU}, \text{SS})$ is $\text{FO}_{\text{AP}}(\prec)$-complete over the time flow $(\mathbb{R} \setminus \mathbb{Z}, \prec)$.
Temporal depth

Definition: Temporal depth of $\varphi \in TL(\mathbb{AP}, SU, SS)$

$$
\begin{align*}
\text{td}(p) &= 0 & \text{if } p \in \mathbb{AP} \\
\text{td}(\lnot \varphi) &= \text{td}(\varphi) \\
\text{td}(\varphi \lor \psi) &= \max(\text{td}(\varphi), \text{td}(\psi)) \\
\text{td}(\varphi SS \psi) &= \max(\text{td}(\varphi), \text{td}(\psi)) + 1 \\
\text{td}(\varphi SU \psi) &= \max(\text{td}(\varphi), \text{td}(\psi)) + 1
\end{align*}
$$

Lemma:
Let $B \subseteq \mathbb{AP}$ be finite and $k \in \mathbb{N}$.
There are (up to equivalence) finitely many formulae in $TL(B, SU, SS)$ of temporal depth at most $k$. 
**$k$-equivalence**

**Definition:**
Let $w_0 = (T_0, <, h_0)$ and $w_1 = (T_1, <, h_1)$ be two temporal structures. Let $i_0 \in T_0$ and $i_1 \in T_1$. Let $k \in \mathbb{N}$.

We say that $(w_0, i_0)$ and $(w_1, i_1)$ are $k$-equivalent, denoted $(w_0, i_0) \equiv_k (w_1, i_1)$, if they satisfy the same formulae in $TL(AP, SU, SS)$ of temporal depth at most $k$.

**Lemma:** $\equiv_k$ is an equivalence relation of finite index.

**Example:**
Let $a = \{p\}$ and $b = \{q\}$. Let $w_0 = babaababaa$ and $w_1 = baababaaba$.

$(w_0, 3) \equiv_0 (w_1, 4)$  
$(w_0, 3) \equiv_1 (w_1, 4)$?  
$(w_0, 3) \equiv_1 (w_1, 6)$?

Here, $T_0 = T_1 = \{0, 1, 2, \ldots, 9\}$. 
EF-games for $\text{TL}(\text{AP, SU, SS})$

The EF-game has two players: Spoiler (Player I) and Duplicator (Player II).

The game board consists of 2 temporal structures:

$w_0 = (\mathbb{T}_0, <, h_0)$ and $w_1 = (\mathbb{T}_1, <, h_1)$.

There are two tokens, one on each structure: $i_0 \in \mathbb{T}_0$ and $i_1 \in \mathbb{T}_1$.

A configuration is a tuple $(w_0, i_0, w_1, i_1)$

or simply $(i_0, i_1)$ if the game board is understood.

Let $k \in \mathbb{N}$.

The $k$-round EF-game from a configuration proceeds with (at most) $k$ moves.

There are 2 available moves for $\text{TL}(\text{AP, SU, SS})$: SU-move or SS-move (see below).

Spoiler chooses which move is played in each round.

Spoiler wins if

- Either duplicator cannot answer during a move (see below).
- Or a configuration such that $(w_0, i_0) \not\equiv_0 (w_1, i_1)$ is reached.

Otherwise, duplicator wins.
Strict Until and Since moves

Definition: SU-move

- Spoiler chooses $\varepsilon \in \{0, 1\}$ and $k_\varepsilon \in \mathbb{T}_\varepsilon$ such that $i_\varepsilon < k_\varepsilon$.
- Duplicator chooses $k_{1-\varepsilon} \in \mathbb{T}_{1-\varepsilon}$ such that $i_{1-\varepsilon} < k_{1-\varepsilon}$.
  
  Spoiler wins if there is no such $k_{1-\varepsilon}$.
  
  Either spoiler chooses $(k_0, k_1)$ as next configuration of the EF-game, or the move continues as follows.

- Spoiler chooses $j_{1-\varepsilon} \in \mathbb{T}_{1-\varepsilon}$ with $i_{1-\varepsilon} < j_{1-\varepsilon} < k_{1-\varepsilon}$.
- Duplicator chooses $j_\varepsilon \in \mathbb{T}_\varepsilon$ with $i_\varepsilon < j_\varepsilon < k_\varepsilon$.
  
  Spoiler wins if there is no such $j_\varepsilon$.
  
  The next configuration is $(j_0, j_1)$.

Similar definition for the SS-move.
Definition: Winning strategy

Duplicator has a winning strategy in the $k$-round EF-game starting from $(w_0, i_0, w_1, i_1)$ if he can win all plays starting from this configuration. This is denoted by $(w_0, i_0) \sim_k (w_1, i_1)$.

Spoiler has a winning strategy in the $k$-round EF-game starting from $(w_0, i_0, w_1, i_1)$ if she can win all plays starting from this configuration.

Example:

Let $a = \{p\}$, $b = \{q\}$, $c = \{r\}$. Let $w_0 = aaabbc$ and $w_1 = aababc$.

$(w_0, 0) \sim_1 (w_1, 0)$

$(w_0, 0) \not\sim_2 (w_1, 0)$

Here, $T_0 = T_1 = \{0, 1, 2, \ldots, 5\}$. 
**EF-games for TL(AP, SU, SS)**

**Lemma: Determinacy**

The $k$-round EF-game for TL(AP, SU, SS) is determined:
For each initial configuration, either spoiler or duplicator has a winning strategy.

**Theorem: Soundness and completeness of EF-games**

For all $k \in \mathbb{N}$ and all configurations $(w_0, i_0, w_1, i_1)$, we have

$$(w_0, i_0) \sim_k (w_1, i_1) \text{ iff } (w_0, i_0) \equiv_k (w_1, i_1)$$

**Example:**

Let $a = \{p\}$, $b = \{q\}$, $c = \{r\}$.

Then, $aaaabbc, 0 \models p \text{ SU } (q \text{ SU } r)$ but $aaababc, 0 \not\models p \text{ SU } (q \text{ SU } r)$.

$p \text{ SU } (q \text{ SU } r)$ cannot be expressed with a formula of temporal depth at most 1.

$p \text{ SU } (q \land X q)$ cannot be expressed with a formula of temporal depth at most 1.

**Exercise:**

On finite linear time flows, “even length” cannot be expressed in TL(AP, SU, SS).
Moves for Strict Future and Past modalities

Definition: SF-move

- Spoiler chooses $\varepsilon \in \{0, 1\}$ and $j_\varepsilon \in T_\varepsilon$ such that $i_\varepsilon < j_\varepsilon$.
- Duplicator chooses $j_{1-\varepsilon} \in T_{1-\varepsilon}$ such that $i_{1-\varepsilon} < j_{1-\varepsilon}$.

Spoiler wins if there is no such $j_{1-\varepsilon}$.
The new configuration is $(j_0, j_1)$.

Similar definition for the SP-move.

Example:

$p \mathsf{SU} q$ is not expressible in $\mathsf{TL}(\mathsf{AP}, \mathsf{SP}, \mathsf{SF})$ over linear flows of time.

Let $a = \emptyset$, $b = \{p\}$ and $c = \{q\}$.

Let $w_0 = (abc)^na(abc)^n$ and $w_1 = (abc)^n(abc)^n$.

Note that $w_0, 3n \models (\neg p \land \neg q) \land X(\neg p \land \neg q)$ and $w_1, 3n \not\models (\neg p \land \neg q) \land X(\neg p \land \neg q)$.

If $n > k$ then, starting from $(w_0, 3n, w_1, 3n)$, duplicator has a winning strategy in the $k$-round EF-game using SF-moves and SP-moves.
Moves for Next and Yesterday modalities

Notation: $i \triangleleft j \overset{\text{def}}{=} i < j \land \neg \exists k (i < k < j)$.

**Definition: X-move**

- Spoiler chooses $\epsilon \in \{0, 1\}$ and $j_\epsilon \in \mathbb{T}_\epsilon$ such that $i_\epsilon < j_\epsilon$.
- Duplicator chooses $j_{1-\epsilon} \in \mathbb{T}_{1-\epsilon}$ such that $i_{1-\epsilon} < j_{1-\epsilon}$.

Spoiler wins if there is no such $j_{1-\epsilon}$.

The new configuration is $(j_0, j_1)$.

Similar definition for the Y-move.

**Exercise:**

Show that $p \mathbf{SU} q$ is not expressible in $\mathbf{TL}(\mathbf{AP}, \mathbf{Y}, \mathbf{SP}, \mathbf{X}, \mathbf{SF})$ over linear time flows.
Non-strict Until and Since moves

Definition: U-move

- Spoiler chooses $\varepsilon \in \{0, 1\}$ and $k_\varepsilon \in \mathbb{T}_\varepsilon$ such that $i_\varepsilon \leq k_\varepsilon$.
- Duplicator chooses $k_{1-\varepsilon} \in \mathbb{T}_{1-\varepsilon}$ such that $i_{1-\varepsilon} \leq k_{1-\varepsilon}$.

Either spoiler chooses $(k_0, k_1)$ as new configuration of the EF-game, or the move continues as follows:

- Spoiler chooses $j_{1-\varepsilon} \in \mathbb{T}_{1-\varepsilon}$ with $i_{1-\varepsilon} \leq j_{1-\varepsilon} < k_{1-\varepsilon}$.
- Duplicator chooses $j_\varepsilon \in \mathbb{T}_\varepsilon$ with $i_\varepsilon \leq j_\varepsilon < k_\varepsilon$.

Spoiler wins if there is no such $j_\varepsilon$. The new configuration is $(j_0, j_1)$.

- If duplicator chooses $k_{1-\varepsilon} = i_{1-\varepsilon}$ then the new configuration must be $(k_0, k_1)$.
- If spoiler chooses $k_\varepsilon = i_\varepsilon$ then duplicator must choose $k_{1-\varepsilon} = i_{1-\varepsilon}$, otherwise he loses.

Similar definition for the S-move.

Exercise:

1. Show that SU is not expressible in $\text{TL}(\text{AP}, S, U)$ over $(\mathbb{R}, <)$.
2. Show that SU is not expressible in $\text{TL}(\text{AP}, S, U)$ over $(\mathbb{N}, <)$.
Definition: Syntactically pure formulae and separation

A formula $\varphi \in TL(AP, SU, SS)$ is

- syntactically pure present if it is a boolean combination of formulae in $AP$,
- syntactically pure future if it is a boolean combination of formulae of the form $\alpha SU \beta$ where $\alpha, \beta \in TL(AP, SU)$,
- syntactically pure past if it is a boolean combination of formulae of the form $\alpha SS \beta$ where $\alpha, \beta \in TL(AP, SS)$,
- syntactically separated if it is a boolean combination of syntactically pure formulae.

A logic $\mathcal{L}$ is syntactically separable over a class $\mathcal{C}$ of time flows if each formula $\varphi \in \mathcal{L}$ is equivalent to some (finite) boolean combination of syntactically pure formulae.

Example:

The formulae $\varphi_1 = SF(q \land SP p)$ and $\varphi_2 = SF(q \land \neg SP \neg p)$ are not separated but we can find equivalent syntactically separated formulae.
Separation

Theorem: [9, Gabbay, Pnueli, Shelah & Stavi 80]

TL(AP, SU, SS) is syntactically separable over discrete and complete linear orders.

Definition: Discrete linear order

A linear time flow \((\mathbb{T}, <)\) is discrete if every non-maximal element has an immediate successor and every non-minimal element has an immediate predecessor.

\>
\( (\mathbb{N}, <) \) is the unique (up to isomorphism) discrete and complete linear order with a first point and no last point.

\>
\( (\mathbb{Z}, <) \) is the unique (up to isomorphism) discrete and complete linear order with no first point and no last point.

\>
Any discrete and complete linear order is isomorphic to a sub-flow of \((\mathbb{Z}, <)\).

Theorem: Gabbay, Reynolds, see [8]

TL(AP, SU, SS) is syntactically separable over \((\mathbb{R}, <)\).
Initial equivalence

**Corollary:** of the separation theorem

For each $\varphi \in \text{TL}(\text{AP}, \text{SU}, \text{SS})$ there exists $\psi \in \text{TL}(\text{AP}, \text{SU})$ such that $\varphi$ and $\psi$ are initially equivalent over $(\mathbb{N}, <)$.

**Example:** $\text{TL}(\text{AP}, \text{SU}, \text{SS})$ versus $\text{TL}(\text{AP}, \text{SU})$

$$G(\text{grant} \rightarrow (\neg \text{grant SS request}))$$

is initially equivalent to

$$(\text{request R } \neg \text{grant}) \land G(\text{grant} \rightarrow (\text{request } \lor (\text{request SR } \neg \text{grant})))$$

**Theorem:** (Laroussinie & Markey & Schnoebelen 2002)

$\text{TL}(\text{AP}, \text{SU}, \text{SS})$ may be exponentially more succinct than $\text{TL}(\text{AP}, \text{SU})$ over $(\mathbb{N}, <)$. 
Semantic Separation

Definition:
Let \( w = (\mathbb{T}, <, \lambda) \) and \( w' = (\mathbb{T}, <, \lambda') \) be temporal structures over the same time flow, and let \( t, t' \in \mathbb{T} \) be time points.

- \( w, w' \) agree on \( t, t' \) if \( \lambda(t) = \lambda'(t') \)
- \( w, w' \) agree on the past of \( t, t' \) if \( (\{ s \in \mathbb{T} \mid s < t \}, <, \lambda) \) and \( (\{ s \in \mathbb{T} \mid s < t' \}, <, \lambda') \) are isomorphic.
- \( w, w' \) agree on the future of \( t \) if \( (\{ s \in \mathbb{T} \mid s > t \}, <, \lambda) \) and \( (\{ s \in \mathbb{T} \mid s > t' \}, <, \lambda') \) are isomorphic.

Definition: Pure formulae
Let \( C \) be a class of time flows. A formula \( \varphi \) over some logic \( \mathcal{L} \) is pure past (resp. pure present, pure future) over \( C \) if

\[
w, t \vDash \varphi \quad \text{iff} \quad w', t' \vDash \varphi
\]

for all temporal structures \( w = (\mathbb{T}, <, \lambda) \) and \( w' = (\mathbb{T}, <, \lambda') \) over \( C \) and all time points \( t, t' \in \mathbb{T} \) such that

\( w, w' \) agree on the past of \( t, t' \) (resp. on \( t, t' \), on the future of \( t, t' \)).
Separation

Remark: Syntax versus semantic

Every formula $\varphi \in \text{TL}(\text{AP}, \text{SU}, \text{SS})$ which is syntactically pure present (resp. future, past) is also semantically pure present (resp. future, past).

Definition: Separation

A logic $\mathcal{L}$ is separable over a class $\mathcal{C}$ of time flows if each formula $\varphi \in \mathcal{L}$ is equivalent to some (finite) boolean combination of pure formulae.

Theorem: [13, Gabbay 89] (already stated by Gabbay in 81)

Let $\mathcal{C}$ be a class of linear time flows. Let $\mathcal{L}$ be a temporal logic able to express SF and SP. Then, $\mathcal{L}$ is separable over $\mathcal{C}$ iff it is expressively complete for $\text{FO}_{\text{AP}}(<)$ over $\mathcal{C}$.

Exercise: Checking semantically pure

Is the following problem decidable? If yes, what is his complexity?

Input: A formula $\varphi \in \text{TL}(\text{AP}, \text{SU}, \text{SS})$

Question: Is the formula $\varphi$ semantically pure future?
Some References

Tense Logic and the Theory of Linear Order.

On the temporal analysis of fairness.
In 7th Annual ACM Symposium PoPL’80, 163–173. ACM Press.

The declarative past and imperative future: Executable temporal logics for interactive systems.

Temporal expressive completeness in the presence of gaps.
In Logic Colloquium ’90, Springer Lecture Notes in Logic 2, pp. 89-121, 1993.

Separation — Past, Present and Future.
In “We Will Show Them: Essays in Honour of Dov Gabbay”.
Great survey on separation properties.
Some References

*Temporal logic: mathematical foundations and computational aspects.*

First-order definable languages.
Overview of formalisms expressively equivalent to First-Order for words.
http://www.lsv.ens-cachan.fr/~gastin/mes-publis.php

*Finite automata, formal logic, and circuit complexity.*

An until hierarchy and other applications of an Ehrenfeucht-Fraïssé game for temporal logic.