Initiation à la vérification
Basics of Verification

http://mpri.master.univ-paris7.fr/C-1-22.html

Paul Gastin

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

MPRI – M1
2010 – 2011
Outline

1 Introduction
   • Bibliography

Models

Specifications

Linear Time Specifications

Branching Time Specifications
Need for formal verifications 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:

\( \bar{R}_n \): \( n \)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

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

<table>
<thead>
<tr>
<th>Bibliography</th>
</tr>
</thead>
</table>
*Principles of Model Checking.*  
*Systems and Software Verification. Model-Checking Techniques and Tools.*  
*Model Checking.*  
*Temporal Verification of Reactive Systems: Safety.*  
Outline

Introduction

Models

- Transition systems
- ... with variables
- Concurrent systems
- Synchronization and communication

Specifications

Linear Time Specifications

Branching Time Specifications
Constructing the model

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
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.

Example: Digicode ABA

![Digicode ABA Diagram](image)

Every discrete system may be described with a TS.
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

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

- \( \mathcal{V} \): set of (typed) variables, e.g., boolean, [0..4], ...
- Each variable \( v \in \mathcal{V} \) has a domain \( D_v \) (finite or infinite)
- Guard or Condition: unary predicate over \( D = \prod_{v \in \mathcal{V}} D_v \)

Symbolic descriptions: \( x < 5 \), \( x + y = 10 \), ...

- Instruction or Update: map \( f : D \to D \)

Symbolic descriptions: \( x := 0 \), \( x := (y + 1)^2 \), ...

- \( T \subseteq S \times (2^D \times \Sigma \times D^D) \times S \)

Symbolic descriptions: \( s \xrightarrow{x < 50, ?\text{coin}, x := x + \text{coin}} s' \)

- \( I \subseteq S \times 2^D \)

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 in \( 2^D \) such as \( x > 0 \).

Programs = Kripke structures with variables

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

Example: GCD
Example: Digicode

cpt < n
B, C

cpt++

A

cpt++

B

cpt++

C

cpt++

A, C

cpt++

B

cpt++

C

cpt++

B

cpt++

C

cpt++

A

cpt++

OPEN

ERROR
...and its semantics \((n = 2)\)

Example: Digicode

\[
\begin{array}{ccc}
1,0 & \rightarrow & 2,0 \rightarrow 3,0 \rightarrow 4,0 \\
B,C & \rightarrow & C \rightarrow A \rightarrow B,C
\
1,1 & \rightarrow & 2,1 \rightarrow 3,1 \rightarrow 4,1 \\
B,C & \rightarrow & C \rightarrow A \rightarrow B,C
\
1,2 & \rightarrow & 2,2 \rightarrow 3,2 \rightarrow 4,2 \\
B,C & \rightarrow & A \rightarrow A,C \rightarrow B,C
\
5,3 & \rightarrow & \text{ERROR}
\end{array}
\]
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**

\[
\begin{align*}
    b'_0 &= \neg b_0 \\
    b'_1 &= b_0 \oplus b_1 \\
    b'_2 &= (b_0 \land b_1) \oplus b_2
\end{align*}
\]
Symbolic representation

Example: Logical representation

\[ \delta_B = \begin{align*}
& s = 1 \land \text{cpt} < n \land s' = 1 \land \text{cpt}' = \text{cpt} + 1 \\
& \lor s = 1 \land \text{cpt} = n \land s' = 5 \land \text{cpt}' = \text{cpt} + 1 \\
& \lor s = 2 \land s' = 3 \land \text{cpt}' = \text{cpt} \\
& \lor s = 3 \land \text{cpt} < n \land s' = 1 \land \text{cpt}' = \text{cpt} + 1 \\
& \lor s = 3 \land \text{cpt} = n \land s' = 5 \land \text{cpt}' = \text{cpt} + 1
\end{align*} \]
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 synchronization mechanisms

Example: Elevator with 1 cabin, 3 doors, 3 calling devices
Modular description of concurrent systems

Example: Elevator

- Cabin:
  - 0
  - 1
  - 2

- Door for level $i$:
  - Closed
  - Opened

- Call for level $i$:
  - False
  - True

The actual system is a synchronized product of all these automata. It consists of (at most) $3 \times 2^3 \times 2^3 = 192$ states.
Synchronized products

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
  \[
  S = \prod_i S_i, \quad \Sigma = \prod_i (\Sigma_i \cup \{\varepsilon\}), \quad \text{and} \quad I = \prod_i I_i
  \]
  \[
  T = \{(p_1, \ldots, p_n) \xrightarrow{(a_1, \ldots, a_n)} (q_1, \ldots, q_n) \mid \text{for all } i, (p_i, a_i, q_i) \in T_i \text{ or } p_i = q_i \text{ and } a_i = \varepsilon\}
  \]
  \[
  AP = \biguplus_i AP_i \quad \text{and} \quad \ell(p_1, \ldots, p_n) = \bigcup_i \ell(p_i)
  \]

Synchronized products: restrictions of the general product.

Parallel compositions

- **Synchronous:** \( \Sigma_{\text{sync}} = \prod_i \Sigma_i \)
- **Asynchronous:** \( \Sigma_{\text{sync}} = \bigoplus_i \Sigma_i' \) with \( \Sigma_i' = \{\varepsilon\}^{i-1} \times \Sigma_i \times \{\varepsilon\}^{n-i} \)

Synchronizations

- **By states:** \( S_{\text{sync}} \subseteq S \)
- **By labels:** \( \Sigma_{\text{sync}} \subseteq \Sigma \)
- **By transitions:** \( T_{\text{sync}} \subseteq T \)
Example: Asynchronous product
Synchronization by states: \((P, P)\) is forbidden
Example: digicode

Example: Synchronous product
Synchronization by transitions
Synchronization by Rendez-vous

Synchronization by transitions is universal but too low-level.

Definition: Rendez-vous

- \( \text{\texttt{!m}} \) sending message \( m \)
- \( \text{\texttt{?m}} \) receiving message \( m \)

SOS: Structural Operational Semantics

Local actions

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

Rendez-vous

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

It is a kind of synchronization by actions.

Essential feature of process algebra.

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

- \( \text{\texttt{?up}} \) is uncontrollable for the cabin
- \( \text{\texttt{?leave}_i} \) is uncontrollable for door \( i \)
- \( \text{\texttt{?call}_0} \) is uncontrollable for the system
Example: Elevator

Example: Synchronization by Rendez-vous

Cabin:

Door for level $i$:

We should design the controller
Shared variables

Definition: Asynchronous product + shared variables

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

Semantics (SOS)

\[ \nu \models g \land s_i \xrightarrow{g,a,f} s'_i \land s'_j = s_j \text{ for } j \neq i \]

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 no synchronization mechanisms: 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 $i$:
loop forever
  req[i] := true; turn := 1-i
  wait until (turn = i or req[1-i] = false)
Critical section
req[i] := 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)

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

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

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

**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 \(\text{await}(y = 1)\)
Example: Leader election

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

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

Definition: Channels

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

- 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

Send

\[ s_i \xrightarrow{c!e} s'_i \land \nu'(c) = \nu(e) \cdot \nu(c) \]

\[ (\bar{s}, \nu) \xrightarrow{c!e} (\bar{s}', \nu') \]

Receive

\[ s_i \xrightarrow{c?x} s'_i \land \nu(c) = \nu'(c) \cdot \nu'(x) \]

\[ (\bar{s}, \nu) \xrightarrow{c?e} (\bar{s}', \nu') \]

Lossy send

\[ s_i \xrightarrow{c!e} s'_i \]

\[ (\bar{s}, \nu) \xrightarrow{c!e} (\bar{s}', \nu) \]

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
- ...

Models: expressivity versus decidability

Definition: (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

Definition: 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
Specifications
Linear Time Specifications
Branching Time Specifications
**Static and dynamic properties**

**Definition: Static properties**

**Example:** Mutual exclusion

Safety properties are often static.

They can be reduced to reachability.

**Definition: Dynamic properties**

**Example:** Every request should be eventually granted.

\[
\bigwedge_i \forall t, (\text{Call}_i(t) \rightarrow \exists t' \geq t, (\text{atLevel}_i(t') \land \text{openDoor}_i(t')))
\]

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

\[
\bigwedge_i \forall t \forall t', (\text{Call}_i(t) \land t \leq t' \land \text{atLevel}_i(t')) \rightarrow \\
\exists t \leq t'' \leq t', (\text{atLevel}_i(t'') \land \text{openDoor}_i(t''))
\]
First Order specifications

First order logic

- These specifications can be written in FO(<).
- 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.

Definition: 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.
Linear versus Branching

Let \( M = (S, T, I, AP, \ell) \) be a Kripke structure.

**Definition: Linear specifications**

Example: The printer manager is **fair**. On each run, whenever some process requests the printer, it eventually gets it.

**Execution sequences (runs):** \( \sigma = s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \cdots \) with \( s_i \rightarrow s_{i+1} \in T \)

Two Kripke structures having the same execution sequences satisfy the same linear specifications.

Actually, linear specifications only depend on the **label** of the execution sequence

\[
\ell(\sigma) = \ell(s_0) \rightarrow \ell(s_1) \rightarrow \ell(s_2) \rightarrow \cdots
\]

Models are words in \( \Sigma^\omega \) with \( \Sigma = 2^{AP} \).

**Definition: Branching specifications**

Example: Each process has the **possibility** to print first.

Such properties depend on the execution tree.

**Execution tree = unfolding of the transition system**
References

Bibliography

*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

A large list of references is given in this paper.

Bibliography

First-order definable languages.
http://www.lsv.ens-cachan.fr/~gastin/mes-publis.php

A large overview of formalisms expressively equivalent to First-Order.
Some original References

_Tense Logic and the Theory of Linear Order._

_Fast LTL to Büchi automata translation._
In _CAV’01_, vol. 2102, _Lecture Notes in Computer Science_, pp. 53–65.
http://www.lsv.ens-cachan.fr/~gastin/mes-publis.php

The tableau method for temporal logic: An overview,

The complexity of propositional linear temporal logic.
Checking that finite state concurrent programs satisfy their linear specification. 

On the temporal analysis of fairness. 

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

Introduction

Models

Specifications

4 Linear Time Specifications
- Definitions
- Main results
- Büchi automata
- From LTL to BA
- Hardness results

Branching Time Specifications
Linear Temporal Logic (Pnueli 1977)

Definition: Syntax: \( \text{LTL}(\text{AP}, \text{X}, \text{U}) \)
\[
\varphi ::= \bot \mid p \ (p \in \text{AP}) \mid \neg \varphi \mid \varphi \lor \varphi \mid \text{X} \varphi \mid \varphi \lor \varphi
\]

Definition: Semantics: \( w = a_0 a_1 a_2 \cdots \in \Sigma^\omega \) with \( \Sigma = 2^{\text{AP}} \) and \( i \in \mathbb{N} \)
\[
w, i \models p \quad \text{if} \quad p \in a_i
\]
\[
w, i \models \neg \varphi \quad \text{if} \quad w, i \not\models \varphi
\]
\[
w, i \models \varphi \lor \psi \quad \text{if} \quad w, i \models \varphi \text{ or } w, i \models \psi
\]
\[
w, i \models \text{X} \varphi \quad \text{if} \quad w, i + 1 \models \varphi
\]
\[
w, i \models \varphi \lor \psi \quad \text{if} \quad \exists k. \ i \leq k \text{ and } w, k \models \psi \text{ and } \forall j. \ (i \leq j < k) \rightarrow w, j \models \varphi
\]

Example:

\[\bullet \rightarrow \bullet \rightarrow \bullet \rightarrow \bullet \rightarrow \bullet \rightarrow \bullet \rightarrow \bullet \rightarrow \bullet \rightarrow \cdots\]

\[p \quad \emptyset \quad p,q \quad p \quad q \quad \emptyset \quad p,r \quad q,r \quad q \quad \cdots\]
Linear Temporal Logic (Pnueli 1977)

Definition: Syntax: \( LTL(\text{AP}, X, U) \)

\[ \varphi ::= \bot \mid p \; (p \in \text{AP}) \mid \neg \varphi \mid \varphi \lor \varphi \mid X \varphi \mid \varphi U \varphi \]

Definition: Semantics: \( w = a_0a_1a_2 \cdots \in \Sigma^\omega \) with \( \Sigma = 2^{\text{AP}} \) and \( i \in \mathbb{N} \)

\[
\begin{align*}
    w, i \models p & \quad \text{if} \quad p \in a_i \\
    w, i \models \neg \varphi & \quad \text{if} \quad w, i \not\models \varphi \\
    w, i \models \varphi \lor \psi & \quad \text{if} \quad w, i \models \varphi \text{ or } w, i \models \psi \\
    w, i \models X \varphi & \quad \text{if} \quad w, i + 1 \models \varphi \\
    w, i \models \varphi U \psi & \quad \text{if} \quad \exists k. \; i \leq k \text{ and } w, k \models \psi \text{ and } \forall j. \; (i \leq j < k) \rightarrow w, j \models \varphi
\end{align*}
\]

Example:
Linear Temporal Logic (Pnueli 1977)

**Definition: Syntax:** \( \text{LTL}(\text{AP}, X, U) \)

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

**Definition: Semantics:** \( w = a_0a_1a_2 \cdots \in \Sigma^\omega \) with \( \Sigma = 2^{\text{AP}} \) and \( i \in \mathbb{N} \)

- \( w, i \models p \) if \( p \in a_i \)
- \( w, i \models \neg \varphi \) if \( w, i \not\models \varphi \)
- \( w, i \models \varphi \lor \psi \) if \( w, i \models \varphi \) or \( w, i \models \psi \)
- \( w, i \models X \varphi \) if \( w, i + 1 \models \varphi \)
- \( w, i \models \varphi U \psi \) if \( \exists k. \ i \leq k \) and \( w, k \models \psi \) and \( \forall j. \ (i \leq j < k) \rightarrow w, j \models \varphi \)

**Example:**

\[
\varphi U \psi
\]

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

\[
\varphi \quad \varphi \quad \psi
\]
Linear Temporal Logic (Pnueli 1977)

**Definition: Macros**

- **Eventually:** \( F \varphi = T \cup \varphi \)

- **Always:** \( G \varphi = \neg F \neg \varphi \)

- **Weak until:** \( \varphi W \psi = G \varphi \lor \varphi U \psi \)

- **Release:** \( \varphi R \psi = \psi W (\varphi \land \psi) = \neg (\neg \varphi U \neg \psi) \)

- **Next until:** \( \varphi XU \psi = X(\varphi U \psi) \)

- \( X \psi = \bot XU \psi \) and \( \varphi U \psi = \psi \lor (\varphi \land \varphi XU \psi) \).
## Linear Temporal Logic (Pnueli 1977)

### Definition: Specifications:

<table>
<thead>
<tr>
<th>Specification</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Safety</td>
<td>$G \text{good}$</td>
</tr>
<tr>
<td>MutEx</td>
<td>$\neg F(\text{crit}_1 \land \text{crit}_2)$</td>
</tr>
<tr>
<td>Liveness</td>
<td>$GF \text{active}$</td>
</tr>
<tr>
<td>Response</td>
<td>$G(\text{request} \rightarrow F\text{grant})$</td>
</tr>
<tr>
<td>Response'</td>
<td>$G(\text{request} \rightarrow X(\neg \text{request} U \text{grant}))$</td>
</tr>
<tr>
<td>Release</td>
<td>reset $R$ alarm</td>
</tr>
<tr>
<td>Strong fairness</td>
<td>$GF \text{request} \rightarrow GF \text{grant}$</td>
</tr>
<tr>
<td>Weak fairness</td>
<td>$FG \text{request} \rightarrow GF \text{grant}$</td>
</tr>
</tbody>
</table>
Linear Temporal Logic (Pnueli 1977)

Examples:

Every elevator request should be eventually satisfied.

$$\bigwedge_i G(\text{Call}_i \rightarrow F(\text{atLevel}_i \land \text{openDoor}_i))$$

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

$$\bigwedge_i G(\text{Call}_i \rightarrow \neg\text{atLevel}_i W (\text{atLevel}_i \land \text{openDoor}_i))$$
Past LTL

Definition: Semantics: \( w = a_0a_1a_2 \cdots \in \Sigma^\omega \) with \( \Sigma = 2^{AP} \) and \( i \in \mathbb{N} \)

\[
\begin{align*}
w, i \models Y \varphi & \quad \text{if} \quad i > 0 \text{ and } w, i - 1 \models \varphi \\
w, i \models \varphi S \psi & \quad \text{if} \quad \exists k. \ k \leq i \text{ and } w, k \models \psi \text{ and } \forall j. \ (k < j \leq i) \rightarrow w, y \models \varphi
\end{align*}
\]

Example:

\[
\begin{array}{cccccccc}
& & & & & & & \varphi S \psi \\
& & & \psi & & & \varphi & \\
& & & & & & & \cdots
\end{array}
\]

Example: LTL versus PLTL

\[ G(\text{grant } \rightarrow Y(\neg \text{grant } S \text{ request})) \]

Theorem (Laroussinie & Markey & Schnoebelen 2002)

PLTL may be exponentially more succinct than LTL.
### Past LTL

**Definition: Semantics:** 

\[ w = a_0a_1a_2 \cdots \in \Sigma^\omega \text{ with } \Sigma = 2^{2^{AP}} \text{ and } i \in \mathbb{N} \]

\[ w, i \models Y \varphi \quad \text{if} \quad i > 0 \text{ and } w, i - 1 \models \varphi \]

\[ w, i \models \varphi S \psi \quad \text{if} \quad \exists k. \; k \leq i \text{ and } w, k \models \psi \text{ and } \forall j. \; (k < j \leq i) \rightarrow w, y \models \varphi \]

#### Example:

\[ \text{Example: LTL versus PLTL} \]

\[ G(\text{grant} \rightarrow Y(\neg \text{grant} S \text{ request})) = (\text{request} R \neg \text{grant}) \land G(\text{grant} \rightarrow (\text{request} \lor X(\text{request} R \neg \text{grant}))) \]

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

PLTL may be exponentially more succinct than LTL.
Expressivity

Theorem [8, Kamp 68]

\[ \text{LTL}(Y, S, X, U) = \text{FO}_\Sigma(\leq) \]

Separation Theorem [13, Gabbay, Pnueli, Shelah & Stavi 80]

For all \( \varphi \in \text{LTL}(Y, S, X, U) \) there exist \( \overleftarrow{\varphi}_i \in \text{LTL}(Y, S) \) and \( \overrightarrow{\varphi}_i \in \text{LTL}(X, U) \) such that for all \( w \in \Sigma^\omega \) and \( k \geq 0 \),

\[ w, k \models \varphi \iff w, k \models \bigvee_i \overleftarrow{\varphi}_i \land \overrightarrow{\varphi}_i \]

Corollary: \( \text{LTL}(Y, S, X, U) = \text{LTL}(X, U) \)

For all \( \varphi \in \text{LTL}(Y, S, X, U) \) there exist \( \overrightarrow{\varphi} \in \text{LTL}(X, U) \) such that for all \( w \in \Sigma^\omega \),

\[ w, 0 \models \varphi \iff w, 0 \models \overrightarrow{\varphi} \]

Elegant algebraic proof of \( \text{LTL}(X, U) = \text{FO}_\Sigma(\leq) \) due to Wilke 98.
## Model checking for LTL

### Definition: Model checking problem

**Input:** A Kripke structure \( M = (S, T, I, AP, \ell) \)

A formula \( \varphi \in \text{LTL}(AP, Y, S, X, U) \)

**Question:** Does \( M \models \varphi \) ?

- **Universal MC:** \( M \models \forall \varphi \) if \( \ell(\sigma), 0 \models \varphi \) for all initial infinite run 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], [12, Lichtenstein & Pnueli 85]

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

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

**Definition:** Satisfiability problem

Input: A formula $\varphi \in \text{LTL}(\text{AP}, Y, S, X, U)$

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 $\varphi \in \text{LTL}(\text{AP}, Y, S, X, U)$

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

Remark: $\varphi$ is satisfiable iff $F \varphi$ is *initially* satisfiable.

**Theorem (Sistla, Clarke 85, Lichtenstein et. al 85)**

The satisfiability problem for LTL is PSPACE-complete.

**Definition:** (Initial) validity

$\varphi$ is valid iff $\neg \varphi$ is not satisfiable.
Decision procedure for LTL

**Definition: The core**

From a formula $\varphi \in \text{LTL}(\text{AP}, \ldots)$, construct a Büchi automaton $A_\varphi$ such that

$$L(A) = L(\varphi) = \{ w \in \Sigma^\omega \mid w, 0 \models \varphi \}.$$  

**Satisfiability (initial)**

Check the Büchi automaton $A_\varphi$ for emptiness.

**Model checking**

Construct a synchronized product $B = M \otimes A_{\lnot \varphi}$ so that the successful runs of $B$ correspond to the initial runs of $M$ satisfying $\lnot \varphi$.

Then, check $B$ for emptiness.

**Theorem:**

Checking Büchi automata for emptiness is NLOGSPACE-complete.
Büchi automata

Definition:

\[ A = (Q, \Sigma, I, T, F) \]

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

Example:

\[ \mathcal{L}(A) = \{w \in \{a, b\}^\omega \mid |w|_a = \omega\} \]
Büchi automata for some LTL formulae

**Definition:**

Recall that $\Sigma = 2^{\text{AP}}$. For $\psi \in \mathbb{B}(\text{AP})$ we let $\Sigma_\psi = \{ a \in \Sigma \mid a \models \psi \}$.

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 $\Sigma_{p \lor q} = \Sigma_p \cup \Sigma_q$
- $\Sigma_{p \land \neg q} = \Sigma_p \setminus \Sigma_q$ ...

**Examples:**

\[
\begin{align*}
\text{F } p: & \quad \begin{array}{c}
\Sigma \xrightarrow{} 1 \xrightarrow{\Sigma_p} 2 \xrightarrow{\Sigma} \Sigma \\
\text{or} & \\
\Sigma_{-p} \xrightarrow{} 1 \xrightarrow{\Sigma_p} 2 \xrightarrow{\Sigma} \Sigma
\end{array} \\
\text{X } X \text{ p: } & \quad \begin{array}{c}
\Sigma \xrightarrow{} 1 \xrightarrow{\Sigma} 2 \xrightarrow{\Sigma} 3 \xrightarrow{\Sigma_p} 4 \xrightarrow{\Sigma} \Sigma \\
\Sigma
\end{array} \\
\text{G } p: & \quad \begin{array}{c}
\Sigma \xrightarrow{} 1 \xrightarrow{\Sigma_p} \Sigma
\end{array}
\end{align*}
\]
Büchi automata for some LTL formulae

Examples:

F G p:

![Diagram](image1.png)

no deterministic Büchi automaton.

G F p:

![Diagram](image2.png)

deterministic Büchi automata are not closed under complement.

G(p → F q):

![Diagram](image3.png)
Büchi automata for some LTL formulae

Examples:

\( p \mathcal{U} q: \)

\[
\begin{align*}
\Sigma_p & \xrightarrow{} 1 \\
\Sigma_q & \xrightarrow{} 2
\end{align*}
\]

or

\[
\begin{align*}
\Sigma_p & \xrightarrow{} 1 \\
\Sigma_{p \land \neg q} & \xrightarrow{} 2
\end{align*}
\]

\( p \mathcal{W} q: \)

\[
\begin{align*}
\Sigma_p & \xrightarrow{} 1 \\
\Sigma_q & \xrightarrow{} 2
\end{align*}
\]

or

\[
\begin{align*}
\Sigma_p & \xrightarrow{} 1 \\
\Sigma_{p \land \neg q} & \xrightarrow{} 2
\end{align*}
\]

\( p \mathcal{R} q: \)

\[
\begin{align*}
\Sigma_q & \xrightarrow{} 1 \\
\Sigma_{p \land q} & \xrightarrow{} 2
\end{align*}
\]

or

\[
\begin{align*}
\Sigma_q & \xrightarrow{} 1 \\
\Sigma_{q \land \neg p} & \xrightarrow{} 2
\end{align*}
\]
Büchi automata

Properties

Büchi automata are closed under union, intersection, complement.

- Union: trivial
- Intersection: easy (exercice)
- Complement: hard

Let $\varphi = F((p \land X^n \neg p) \lor (\neg p \land X^n p))$

Any non deterministic Büchi automaton for $\neg \varphi$ has at least $2^n$ states.
Büchi automata

Exercise:

Given Büchi automata for \( \varphi \) and \( \psi \),

- Construct a Büchi automaton for \( X \varphi \) (trivial)
- Construct a Büchi automaton for \( \varphi U \psi \)

This gives an inductive construction of \( A_\varphi \) from \( \varphi \in \text{LTL}(\text{AP}, X, U) \) . . .

. . . but the size of \( A_\varphi \) might be non-elementary in the size of \( \varphi \).
**Definition: acceptance on states**

\[ A = (Q, \Sigma, I, T, F_1, \ldots, F_n) \text{ with } F_i \subseteq Q. \]

An infinite run \( \sigma \) is successful if it visits infinitely often each \( F_i \).

**Definition: acceptance on transitions**

\[ A = (Q, \Sigma, I, T, T_1, \ldots, T_n) \text{ with } T_i \subseteq T. \]

An infinite run \( \sigma \) is successful if it uses infinitely many transitions from each \( T_i \).
Proof: Synchronized product with $\mathcal{B}$

Transitions:
$$t = s_1 \xrightarrow{a} s_1' \in A \land s_2 \xrightarrow{t} s_2' \in B$$

$$\langle s_1, s_2 \rangle \xrightarrow{a} \langle s_1', s_2' \rangle$$

Accepting states: $Q \times \{n\}$
Negative normal form

Definition: Syntax \((p \in AP)\)

\[
\varphi ::= \top | \bot | p | \neg p | \varphi \lor \varphi | \varphi \land \varphi | X\varphi | \varphi U \varphi | \varphi R \varphi
\]

Proposition: Any formula can be transformed in NNF

\[
\neg (\varphi \lor \psi) \equiv (\neg \varphi) \land (\neg \psi) \quad \neg (\varphi \land \psi) \equiv (\neg \varphi) \lor (\neg \psi)
\]

\[
\neg (\varphi \mathrel U \psi) \equiv (\neg \varphi) \mathrel R (\neg \psi) \quad \neg (\varphi \mathrel R \psi) \equiv (\neg \varphi) \mathrel U (\neg \psi)
\]

\[
\neg X \varphi \equiv X \neg \varphi \quad \neg \neg \varphi \equiv \varphi
\]

This does not increase the number of Temporal subformulae.
Definition: Temporal formulae

- literals
- formulae with outermost connective $X$, $U$ or $R$.

Reducing the number of temporal subformulae

<table>
<thead>
<tr>
<th>Expression</th>
<th>Equivalent Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(X \varphi) \land (X \psi)$</td>
<td>$X(\varphi \land \psi)$</td>
</tr>
<tr>
<td>$(\varphi \mathcal{R} \psi_1) \land (\varphi \mathcal{R} \psi_2)$</td>
<td>$\varphi \mathcal{R} (\psi_1 \land \psi_2)$</td>
</tr>
<tr>
<td>$(G \varphi) \land (G \psi)$</td>
<td>$G(\varphi \land \psi)$</td>
</tr>
<tr>
<td>$(X \varphi) \cup (X \psi)$</td>
<td>$X(\varphi \cup \psi)$</td>
</tr>
<tr>
<td>$(\varphi_1 \mathcal{R} \psi) \lor (\varphi_2 \mathcal{R} \psi)$</td>
<td>$(\varphi_1 \lor \varphi_2) \mathcal{R} \psi$</td>
</tr>
<tr>
<td>$G F \varphi \lor G F \psi$</td>
<td>$G F (\varphi \lor \psi)$</td>
</tr>
</tbody>
</table>
From LTL to BA [6, Demri & Gastin 10]

Definition:

- $Z \subseteq \text{NNF}$ is **consistent** if $\bot \notin Z$ and $\{p, \neg p\} \not\subseteq Z$ for all $p \in \text{AP}$.
- For $Z \subseteq \text{NNF}$, we define $\bigwedge Z = \bigwedge_{\psi \in Z} \psi$.
  Note that $\bigwedge \emptyset = \top$ and if $Z$ is inconsistent then $\bigwedge Z \equiv \bot$.

Intuition for the BA $A_{\varphi} = (Q, \Sigma, I, T, (T_\alpha)_{\alpha \in U(\varphi)})$

Let $\varphi \in \text{NNF}$ be a formula.

- $\text{sub}(\varphi)$ is the set of sub-formulae of $\varphi$.
- $U(\varphi)$ the set of until sub-formulae of $\varphi$.
- We construct a BA $A_{\varphi}$ with $Q = 2^{\text{sub}(\varphi)}$ and $I = \{\varphi\}$.
- A state $Z \subseteq \text{sub}(\varphi)$ is a set of obligations.
- If $Z \subseteq \text{sub}(\varphi)$, we want $\mathcal{L}(A^Z_{\varphi}) = \{u \in \Sigma^\omega \mid u, 0 \models \bigwedge Z\}$
  where $A^Z_{\varphi}$ is $A_{\varphi}$ using $Z$ as unique initial state.
Reduced formulae

Definition: Reduced formulae

- A formula is **reduced** if it is a literal ($p$ or $\neg p$) or a next-formula ($X \beta$).
- $Z \subseteq \text{NNF}$ is **reduced** if all formulae in $Z$ are reduced.

For $Z \subseteq \text{NNF}$ consistent and reduced, we define

- $\text{next}(Z) = \{ \alpha \mid X \alpha \in Z \}$
- $\Sigma_Z = \bigcap_{p \in Z} \Sigma_p \cap \bigcap_{\neg p \in Z} \Sigma_{\neg p}$

Lemma: Next step

Let $Z \subseteq \text{NNF}$ be consistent and reduced.

Let $u = a_0a_1a_2 \cdots \in \Sigma^\omega$ and $n \geq 0$. Then

$$u, n \models \bigwedge Z \iff u, n + 1 \models \bigwedge \text{next}(Z) \text{ and } a_n \in \Sigma_Z$$

- $A_\varphi$ will have transitions $Z \xrightarrow{\Sigma_Z} \text{next}(Z)$.
  
  Note that $\emptyset \xrightarrow{\Sigma} \emptyset$.

- **Problem**: $\text{next}(Z)$ is not reduced in general (it may even be inconsistent).
Definition: Reduction of obligations to literals and next-formulae

Let $Y \subseteq \text{NNF}$ and let $\psi \in Y$ maximal not reduced.

If $\psi = \psi_1 \land \psi_2$:

$Y \xrightarrow{\varepsilon} (Y \setminus \{\psi\}) \cup \{\psi_1, \psi_2\}$

If $\psi = \psi_1 \lor \psi_2$:

$Y \xrightarrow{\varepsilon} (Y \setminus \{\psi\}) \cup \{\psi_1\}$

$Y \xrightarrow{\varepsilon} (Y \setminus \{\psi\}) \cup \{\psi_2\}$

If $\psi = \psi_1 \mathcal{R} \psi_2$:

$Y \xrightarrow{\varepsilon} (Y \setminus \{\psi\}) \cup \{\psi_1, \psi_2\}$

$Y \xrightarrow{\varepsilon} (Y \setminus \{\psi\}) \cup \{\psi_2, X \psi\}$

If $\psi = \text{G} \psi_2$:

$Y \xrightarrow{\varepsilon} (Y \setminus \{\psi\}) \cup \{\psi_2, X \psi\}$

If $\psi = \psi_1 \mathcal{U} \psi_2$:

$Y \xrightarrow{\varepsilon} (Y \setminus \{\psi\}) \cup \{\psi_2\}$

$Y \xrightarrow{\varepsilon \psi} (Y \setminus \{\psi\}) \cup \{\psi_1, X \psi\}$

If $\psi = \text{F} \psi_2$:

$Y \xrightarrow{\varepsilon} (Y \setminus \{\psi\}) \cup \{\psi_2\}$

$Y \xrightarrow{\varepsilon \psi} (Y \setminus \{\psi\}) \cup \{X \psi\}$

Note the mark $!\psi$ on the second transitions for $\mathcal{U}$ and $\text{F}$.
Reduction rules

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

State = set of obligations.
Reduce obligations to literals and next-formulae.
Note again the mark $!F q$ on the last edge
Reduction

Lemma:
- if there is only one rule $Y \xrightarrow{\varepsilon} Y_1$ then $\bigwedge Y \equiv \bigwedge Y_1$
- if there are two rules $Y \xrightarrow{\varepsilon} Y_1$ and $Y \xrightarrow{\varepsilon} Y_2$ then $\bigwedge Y \equiv \bigwedge Y_1 \lor \bigwedge Y_2$

Definition:
For $Y \subseteq \text{NNF}$ and $\alpha \in U(\varphi)$, let

$$\text{Red}(Y) = \{ Z \text{ consistent and reduced} \mid \text{there is a path } Y \xrightarrow{\ast} Z \}$$

$$\text{Red}_\alpha(Y) = \{ Z \text{ consistent and reduced} \mid \text{there is a path } Y \xrightarrow{\varepsilon} Z$$

without using an edge marked with $!\alpha\}$$

Lemma: Soundness
- Let $Y \subseteq \text{NNF}$, then $\bigwedge Y \equiv \bigvee Z \in \text{Red}(Y) \bigwedge Z$
- Let $u = a_0a_1a_2 \cdots \in \Sigma^\omega$ and $n \geq 0$ with $u, n \models \bigwedge Y$. Then, $\exists Z \in \text{Red}(Y)$ such that $u, n \models \bigwedge Z$

and $Z \in \text{Red}_\alpha(Y)$ for all $\alpha = \alpha_1 \cup \alpha_2 \in U(\varphi)$ such that $u, n \models \alpha_2$. 
**Definition: Automaton $A_\varphi$**

- **States:** $Q = 2^{\text{sub}(\varphi)}$, $I = \{\varphi\}$
- **Transitions:** $T = \{Y \xrightarrow{a} \text{next}(Z) \mid Y \in Q, a \in \Sigma_Z \text{ and } Z \in \text{Red}(Y)\}$
- **Acceptance:** $T_\alpha = \{Y \xrightarrow{a} \text{next}(Z) \mid Y \in Q, a \in \Sigma_Z \text{ and } Z \in \text{Red}_\alpha(Y)\}$ for each $\alpha \in U(\varphi)$. 
Automaton $A_\varphi$

**Example:** $\varphi = G(p \rightarrow F q)$

Transition = check literals and move forward.

Simplification
Correctness of $A_\varphi$

**Proposition:** $L(\varphi) \subseteq L(A_\varphi)$

**Lemma:**
Let $\rho = Y_0 \xrightarrow{a_0} Y_1 \xrightarrow{a_1} Y_2 \cdots$ be an accepting run of $A_\varphi$ on $u = a_0a_1a_2 \cdots \in \Sigma^\omega$.

Then, for all $\psi \in \text{sub}(\varphi)$ and $n \geq 0$,
for all reduction path $Y_n \xrightarrow{\varepsilon} Y \xrightarrow{\varepsilon} Z$ with $a_n \in \Sigma_Z$ and $Y_{n+1} = \text{next}(Z)$,

$$\psi \in Y \implies u, n \models \psi$$

**Corollary:** $L(A_\varphi) \subseteq L(\varphi)$
\( L(\varphi) \subseteq L(A_\varphi) \)

**Proof:**

Let \( u = a_0a_1a_2 \cdots \in \Sigma^\omega \) be such that \( u, 0 \models \varphi \). By induction, we build a run

\[
\rho = Y_0 \xrightarrow{a_0} Y_1 \xrightarrow{a_1} Y_2 \cdots
\]

We start with \( Y_0 = \{\varphi\} \). Assume that \( u, n \models \bigwedge Y_n \) for some \( n \geq 0 \). By Lemma [Soundness], there is \( Z_n \in \text{Red}(Y_n) \) such that \( u, n \models \bigwedge Z_n \) and for all until subformulae \( \alpha = \alpha_1 U \alpha_2 \in U(\varphi) \), if \( u, n \models \alpha_2 \) then \( Z_n \in \text{Red}_\alpha(Y_n) \). Then we define \( Y_{n+1} = \text{next}(Z_n) \). Since \( u, n \models \bigwedge Z_n \), Lemma [Next Step] implies \( a_n \in \Sigma_{Z_n} \) and \( u, n + 1 \models \bigwedge Y_{n+1} \). Therefore, \( \rho \) is a run for \( u \) in \( A_\varphi \).

It remains to show that \( \rho \) is successful. By definition, it starts from the initial state \( \{\varphi\} \). Now let \( \alpha = \alpha_1 U \alpha_2 \in U(\varphi) \). Assume there exists \( N \geq 0 \) such that \( Y_n \xrightarrow{a_n} Y_{n+1} \notin T_\alpha \) for all \( n \geq N \). Then \( Z_n \notin \text{Red}_\alpha(Y_n) \) for all \( n \geq N \) and we deduce that \( u, n \not\models \alpha_2 \) for all \( n \geq N \). But, since \( Z_N \notin \text{Red}_\alpha(Y_N) \), the formula \( \alpha \) has been reduced using an \( \varepsilon \)-transition marked \( !\alpha \) along the path from \( Y_N \) to \( Z_N \). Therefore, \( X\alpha \in Z_N \) and \( \alpha \in Y_{N+1} \). By construction of the run we have \( u, N + 1 \models \bigwedge Y_{N+1} \). Hence, \( u, N + 1 \models \alpha \), a contradiction with \( u, n \not\models \alpha_2 \) for all \( n \geq N \). Consequently, the run \( \rho \) is successful and \( u \) is accepted by \( A_\varphi \).
\[ \mathcal{L}(A_\varphi) \subseteq \mathcal{L}(\varphi) \]

**Lemma:**

Let \( \rho = Y_0 \xrightarrow{a_0} Y_1 \xrightarrow{a_1} Y_2 \cdots \) be an accepting run of \( A_\varphi \) on \( u = a_0a_1a_2 \cdots \in \Sigma^\omega \).

Then, for all \( \psi \in \text{sub}(\varphi) \) and \( n \geq 0 \),

for all reduction path \( Y_n \xrightarrow{\varepsilon} Y \xrightarrow{\varepsilon} Z \) with \( a_n \in \Sigma_Z \) and \( Y_{n+1} = \text{next}(Z) \),

\[ \psi \in Y \implies u, n \models \psi \]

**Proof: by induction on \( \psi \)**

- \( \psi = \top \). The result is trivial.
- \( \psi = p \in \text{AP}(\varphi) \). Since \( p \) is reduced, we have \( p \in Z \) and it follows \( \Sigma_Z \subseteq \Sigma_p \). Therefore, \( p \in a_n \) and \( u, n \models p \). The proof is similar if \( \psi = \neg p \) for some \( p \in \text{AP}(\varphi) \).
- \( \psi = X \psi_1 \). Then \( \psi \in Z \) and \( \psi_1 \in Y_{n+1} \). By induction we obtain \( u, n + 1 \models \psi_1 \) and we deduce \( u, n \models X \psi_1 = \psi \).
- \( \psi = \psi_1 \land \psi_2 \). Along the path \( Y \xrightarrow{\varepsilon} Y' \xrightarrow{\varepsilon} Z \) the formula \( \psi \) must be reduced so \( Y \xrightarrow{\varepsilon} Y' \xrightarrow{\varepsilon} Z \) with \( \psi_1, \psi_2 \in Y' \). By induction, we obtain \( u, n \models \psi_1 \) and \( u, n \models \psi_2 \). Hence, \( u, n \models \psi \). The proof is similar for \( \psi = \psi_1 \lor \psi_2 \).
\[ \mathcal{L}(A_\varphi) \subseteq \mathcal{L}(\varphi) \]

Proof:

- \( \varphi = \psi_1 \cup \psi_2 \). Along the path \( Y \xrightarrow{\varepsilon} Z \) the formula \( \varphi \) must be reduced so \( Y \xrightarrow{\varepsilon} Y' \xrightarrow{\varepsilon} Z \) with either \( Y'' = Y' \setminus \{\psi\} \cup \{\psi_2\} \) or \( Y'' = Y' \setminus \{\psi\} \cup \{\psi_1, X \psi\} \).

In the first case, we obtain by induction \( u, n \models \psi_2 \) and therefore \( u, n \models \varphi \). In the second case, we obtain by induction \( u, n \models \psi_1 \). Since \( X \varphi \) is reduced we get \( X \varphi \in Z \) and \( \varphi \in \text{next}(Z) = Y_{n+1} \).

Let \( k > n \) be minimal such that \( Y_k \xrightarrow{a_k} Y_{k+1} \in T_\varphi \) (such a value \( k \) exists since \( \rho \) is accepting). We first show by induction that \( u, i \models \psi_1 \) and \( \psi \in Y_{i+1} \) for all \( n \leq i < k \). Recall that \( u, n \models \psi_1 \) and \( \psi \in Y_{n+1} \). So let \( n < i < k \) be such that \( \psi \in Y_i \). Let \( Z' \in \text{Red}(Y_i) \) be such that \( a_i \in \Sigma_{Z'} \) and \( Y_{i+1} = \text{next}(Z') \). Since \( k \) is minimal we know that \( Z' \notin \text{Red}_\psi(Y_i) \). Hence, along any reduction path from \( Y_i \) to \( Z' \) we must use a step \( Y' \xrightarrow{\varepsilon} Y' \setminus \{\psi\} \cup \{\psi_1, X \psi\} \). By induction on the formula we obtain \( u, i \models \psi_1 \). Also, since \( X \varphi \) is reduced, we have \( X \varphi \in Z' \) and \( \varphi \in \text{next}(Z') = Y_{i+1} \).

Second, we show that \( u, k \models \psi_2 \). Since \( Y_k \xrightarrow{a_k} Y_{k+1} \in T_\varphi \), we find some \( Z' \in \text{Red}_\psi(Y_k) \) such that \( a_k \in \Sigma_{Z'} \) and \( Y_{k+1} = \text{next}(Z') \). Since \( \psi \in Y_k \), along some reduction path from \( Y_k \) to \( Z' \) we use a step \( Y' \xrightarrow{\varepsilon} Y' \setminus \{\psi\} \cup \{\psi_2\} \). By induction we obtain \( u, k \models \psi_2 \). Finally, we have shown \( u, n \models \psi_1 \cup \psi_2 = \psi \).
\[ \mathcal{L}(A_\varphi) \subseteq \mathcal{L}(\varphi) \]

**Proof:**

\( \psi = \psi_1 \mathsf{R} \psi_2 \). Along the path \( Y \xrightarrow{\varepsilon} Z \) the formula \( \psi \) must be reduced so \( Y \xrightarrow{\varepsilon} Y' \xrightarrow{\varepsilon} Y'' \xrightarrow{\ast} Z \) with either \( Y'' = Y' \setminus \{ \psi \} \cup \{ \psi_1, \psi_2 \} \) or \( Y'' = Y' \setminus \{ \psi \} \cup \{ \psi_2, X \psi \} \).

In the first case, we obtain by induction \( u, n \models \psi_1 \) and \( u, n \models \psi_2 \). Hence, \( u, n \models \psi \) and we are done. In the second case, we obtain by induction \( u, n \models \psi_2 \) and we get also \( \psi \in Y_{n+1} \). Continuing with the same reasoning, we deduce easily that either \( u, n \models G \psi_2 \) or \( u, n \models \psi_2 \mathsf{U} (\psi_1 \land \psi_2) \).
Example with two until sub-formulae

Example: Nested until: \( \varphi = p \cup \psi \) with \( \psi = q \cup r \)

\[
\begin{align*}
\text{Red}(\{\varphi\}) &= \{\{p, X \varphi\}, \{q, X \psi\}, \{r\}\} \\
\text{Red}_\varphi(\{\varphi\}) &= \{\{q, X \psi\}, \{r\}\} \\
\text{Red}_\psi(\{\varphi\}) &= \{\{p, X \varphi\}, \{r\}\} \\
\text{Red}(\{\psi\}) &= \{\{q, X \psi\}, \{r\}\} \\
\text{Red}_\varphi(\{\psi\}) &= \{\{q, X \psi\}, \{r\}\} \\
\text{Red}_\psi(\{\psi\}) &= \{\{r\}\}
\end{align*}
\]

\[
\begin{tikzpicture}
  \node (phi) at (0,0) {$\varphi = p \cup (q \cup r)$};
  \node (psi) at (2,2) {$\psi = q \cup r$};
  \node (empty) at (1,-2) {$\emptyset$};

  \draw[->] (phi) -- node[above] {$\Sigma_q$} (psi);
  \draw[->] (phi) -- node[below] {$\Sigma_p$} (empty);
  \draw[->] (psi) -- node[above] {$\Sigma_q$} (psi);
  \draw[->] (psi) -- node[below] {$\Sigma_r$} (empty);
  \draw[->] (empty) -- node[above] {$\Sigma_r$} (phi);
  \draw[->] (empty) -- node[below] {$\Sigma_r$} (psi);

  \draw[->] (phi) -- node[above] {$\varphi, \psi$} (empty);
  \draw[->] (psi) -- node[below] {$\varphi, \psi$} (empty);
\end{tikzpicture}
\]
Corollary: PSPACE upper bound for satisfiability and model checking

- Let $\varphi \in \text{LTL}$, we can check whether $\varphi$ is satisfiable (or valid) in space polynomial in $|\varphi|$.

- Let $\varphi \in \text{LTL}$ and $M = (S, T, I, \text{AP}, \ell)$ be a Kripke structure. We can check whether $M \models \forall \varphi$ (or $M \models \exists \varphi$) in space polynomial in $|\varphi| + \log |M|$.

Proof:

For $M \models \forall \varphi$ we construct a synchronized product $M \otimes A_{\neg \varphi}$:

Transitions:

$$\frac{s \rightarrow s' \in M \land Y \xrightarrow{\ell(s)} Y' \in A_{\neg \varphi}}{(s, Y) \xrightarrow{\ell(s)} (s', Y')}$$

Initial states: $I \times \{\neg \varphi\}$.

Acceptance conditions: inherited from $A_{\neg \varphi}$.

Check $M \otimes A_{\neg \varphi}$ for emptiness.
On the fly simplifications $\mathcal{A}_\varphi$

Built-in: reduction of a maximal formula.

**Definition: Additional reduction rules**

If $\bigwedge Y \equiv \bigwedge Y'$ then we may use $Y \xrightarrow{\varepsilon} Y'$.

Remark: checking equivalence is as hard as building the automaton. Hence we only use syntactic equivalences.

- If $\psi = \psi_1 \lor \psi_2$ and $\psi_1 \in Y$ or $\psi_2 \in Y$: $Y \xrightarrow{\varepsilon} Y \setminus \{\psi\}$
- If $\psi = \psi_1 \lor \psi_2$ and $\psi_2 \in Y$: $Y \xrightarrow{\varepsilon} Y \setminus \{\psi\}$
- If $\psi = \psi_1 \land \psi_2$ and $\psi_1 \in Y$: $Y \xrightarrow{\varepsilon} Y \setminus \{\psi\} \cup \{\psi_2\}$
On the fly simplifications $\mathcal{A}_\varphi$

**Definition: Merging equivalent states**

Let $A = (Q, \Sigma, I, T, T_1, \ldots, T_n)$ and $s_1, s_2 \in Q$. We can merge $s_1$ and $s_2$ if they have the same outgoing transitions:

\[ \forall a \in \Sigma, \forall s \in Q, \quad (s_1, a, s) \in T \iff (s_2, a, s) \in T \]

and

\[ (s_1, a, s) \in T_i \iff (s_2, a, s) \in T_i \quad \text{for all } 1 \leq i \leq n. \]

**Remark: Sufficient condition**

Two states $Y, Y'$ of $\mathcal{A}_\varphi$ have the same outgoing transition if

\[ \text{Red}(Y) = \text{Red}(Y') \]

and

\[ \text{Red}_\alpha(Y) = \text{Red}_\alpha(Y') \quad \text{for all } \alpha \in \mathcal{U}(\varphi). \]

**Example: Let $\varphi = GFp \land GFq$.**

Without merging states $\mathcal{A}_\varphi$ has 4 states. These 4 states have the same outgoing transitions. The simplified automaton has only one state.
Other constructions

- Tableau construction. See for instance [9, Wolper 85]
  + Easy definition, easy proof of correctness
  + Works both for future and past modalities
  - Inefficient without optimizations

- Using Very Weak Alternating Automata [10, Gastin & Oddoux 01].
  + Very efficient
  - Only for future modalities
  Online tool: http://www.lsv.ens-cachan.fr/~gastin/ltl2ba/

- The domain is still very active.

- See other references in [6, Demri & Gastin 10].
\[
\text{MC}^3(X, U) \leq_P \text{SAT}(X, U)
\]
[11, Sistla & Clarke 85]

Let \( M = (S, T, I, AP, \ell) \) be a Kripke structure and \( \varphi \in \text{LTL}(AP, X, U) \)

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

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

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

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

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

Let \( w = a_0 a_1 a_2 \cdots \in \Sigma'\omega \). Then, \( w \models \psi_M \) iff there exists an initial infinite run \( \sigma \) of \( M \) such that \( \pi(w) = \ell(\sigma) \) and \( a_i \cap AP_S = \{ \text{at}_{s_i} \} \) for all \( i \geq 0 \).

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

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

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

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

\( Q_i \in \{\forall, \exists\} \) 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 \( \{v_i, v'_i\} = \{0, 1\} \).

---

**Proposition:**

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

Consider the KS $M$: 

$$
\begin{array}{c}
\xymatrix{
\cdot & e_0 \ar[rd] & \cdot \\
\cdot & s_1 & e_1 \ar[ld] & \cdot \\
\cdot & x_1^t & s_2 & x_2^t \ar[ld] & e_2 & \cdots \\
\cdot & x_1^f & e_1 & s_2 & x_2^f & \cdots \\
\cdot & f_0 & a_{11} & f_1 & a_{21} & f_2 & \cdots \\
\cdot & \vdots & a_{12} & \vdots & a_{22} & \vdots & \cdots \\
\cdot & a_{1k_1} & f_0 & a_{2k_2} & f_1 & a_{2k_2} & f_2 & \cdots \\
\cdot & \vdots & a_{1j_1} & \vdots & a_{2j_2} & \vdots & \cdots \\
\cdot & a_{mk_m} & f_{m-1} & a_{mk_m} & f_m & a_{mk_m} & \cdots \\
\cdot & e_m \\
\end{array}
$$

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

Let $\varphi_j = G(e_{j-1} \to (\neg s_{j-1} U x_j^t) \wedge (\neg s_{j-1} U x_j^f))$ and $\varphi = \bigwedge_{j \mid Q_j = \forall} \varphi_j$.

Then, $\gamma$ is valid iff $M \models \exists \psi \wedge \varphi$. 

\[ \mathsf{QBF} \leq_P \mathsf{MC}^\exists(U) \quad [11, \text{Sistla & Clarke 85}] \]
QBF $\leq_P$ MC$^3$(U) [11, Sistla & Clarke 85]

Proof: If $M \models_{\exists} \psi \land \varphi$ then $\gamma$ is valid

Each finite path $\tau = e_0 \rightarrow f_m$ in $M$ defines a valuation $v^\tau$ by:

$$
v^\tau_k = \begin{cases} 
1 & \text{if } \tau, |\tau| \models \neg s_k S x^t_k \\
0 & \text{if } \tau, |\tau| \models \neg s_k S x^f_k
\end{cases}
$$

Let $\sigma$ be an initial infinite path of $M$ s.t. $\sigma, 0 \models \psi \land \varphi$.

Let $V = \{v^\tau \mid \tau = e_0 \rightarrow f_m \text{ is a prefix of } \sigma\}$.

Claim: $V$ is nonempty, valid and closed.
Proof: If $\gamma$ is valid then $M \models \exists \psi \land \varphi$

Let $V \subseteq \{0, 1\}^n$ be nonempty, valid and closed.

First ingredient: extension of a run.
Assume $\tau = e_0 \rightarrow f_m$ satisfies $v^\tau \in V$ and $\tau, 0 \models \psi$.
Let $1 \leq i \leq n$ with $Q_i = \forall$.
Let $v' \in V$ s.t. $v'[i - 1] = v[i - 1]$ and $\{v_i, v'_i\} = \{0, 1\}$.
We can extend $\tau$ in $\tau' = \tau \rightarrow s_i \rightarrow e_n \rightarrow f_0 \rightarrow f_m$ with $v'^\tau = v'$ and $\tau', 0 \models \psi$.
We say that $\tau'$ is an extension of $\tau$ wrt. $i$

Second step: the sequence of indices for the extensions.
Let $1 \leq i_\ell < \cdots < i_1 \leq n$ be the indices of universal quantifications ($Q_{i_j} = \forall$).
Define by induction $w_1 = i_1$ and if $k < \ell$, $w_{k+1} = w_k i_{k+1} w_k$. Let $w = (w_\ell 1)^\omega$.

Final step: the infinite run.
Let $v \in V \neq \emptyset$ and let $\tau = e_0 \rightarrow f_m$ with $v^\tau \in V$ and $\tau, 0 \models \psi$.
We build an infinite run $\sigma$ by extending $\tau$ inductively wrt. the sequence of indices defined by $w$.

Claim: $\sigma, 0 \models \psi \land \varphi$. 
Theorem: Complexity of LTL

The following problems are PSPACE-complete:

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

The following problems are NP-complete:

- \( \text{SAT}(\text{LTL}(F)), \text{MC}^\exists(\text{LTL}(F)) \)
Possibility is not expressible in LTL

Example:

\( \varphi \): Whenever \( p \) holds, it is possible to reach a state where \( q \) holds.
\( \varphi \) cannot be expressed in LTL.

Consider the two models:

\[ M_1: \]

1 \( \rightarrow \) 2 \( \rightarrow \) 3 \( \rightarrow \) 4

\[ M_2: \]

1 \( \rightarrow \) 2' \( \rightarrow \) 4

\( M_1 \models \varphi \) but \( M_2 \not\models \varphi \)

\( M_1 \) and \( M_2 \) satisfy the same LTL formulae.

We need quantifications on runs: \( \varphi = AG(p \rightarrow EFq) \)

- E: for some infinite run
- A: for all infinite runs
**Definition: Syntax of the Computation Tree Logic CTL***

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

**Definition: Semantics:**

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

- \( M, \sigma, i \models E \varphi \) if \( M, \sigma', 0 \models \varphi \) for some infinite run \( \sigma' \) such that \( \sigma'(0) = \sigma(i) \)
- \( M, \sigma, i \models A \varphi \) if \( M, \sigma', 0 \models \varphi \) for all infinite runs \( \sigma' \) such that \( \sigma'(0) = \sigma(i) \)

**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:**

\[ A \varphi \equiv \neg E \neg \varphi \]
Definition: State formulae

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

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

Example: State formulae

Formulae of the form \( p \) or \( E \varphi \) or \( A \varphi \) are state formulae.
State formulae are closed under boolean connectives.

\[
[p] = \{ s \in S \mid p \in \ell(s) \} \quad [\neg \varphi] = S \setminus [\varphi] \quad [\varphi_1 \lor \varphi_2] = [\varphi_1] \cup [\varphi_2]
\]

Definition: Alternative syntax

State formulae \( \varphi ::= \perp \mid p \ (p \in AP) \mid \neg \varphi \mid \varphi \lor \varphi \mid E \psi \mid A \psi \)

Path formulae \( \psi ::= \varphi \mid \neg \psi \mid \psi \lor \psi \mid X \psi \mid \psi U \psi \)
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 $M, \sigma, 0 \models \varphi$ for some initial infinite run $\sigma$ of $M$.

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

Remark:

$M \models \exists \varphi \iff I \cap [E \varphi] \neq \emptyset$

$M \models \forall \varphi \iff I \subseteq [A \varphi]$

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

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 $M \models \forall \varphi$ ? or Does $M \models \exists \varphi$ ?
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.
\( MC^{\forall}_{\text{CTL}^*} \) in PSPACE

**Proof:**

For \( Q \in \{ \exists, \forall \} \) and \( \psi \in \text{LTL} \), let \( MC^{Q}_{\text{LTL}}(M, t, \psi) \) be the function which computes in polynomial space whether \( M, t \models Q \psi \), i.e., if \( M, t \models Q\psi \).

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

\[ MC^{\forall}_{\text{CTL}^*}(M, s, \varphi) \]

If \( E, A \) do not occur in \( \varphi \) then return \( MC^{\forall}_{\text{LTL}}(M, s, \varphi) \) fi

Let \( Q\psi \) be a subformula of \( \varphi \) with \( \psi \in \text{LTL} \) and \( Q \in \{ E, A \} \)

Let \( p_{Q\psi} \) be a new propositional variable

Define \( \ell' : S \to 2^\text{AP'} \) with \( \text{AP}' = \text{AP} \cup \{ p_{Q\psi} \} \) by

\[ \ell'(t) \cap \text{AP} = \ell(t) \text{ and } p_{Q\psi} \in \ell'(t) \text{ iff } MC^{Q}_{\text{LTL}}(M, t, \psi) \]

Let \( M' = (S, T, I, \text{AP}', \ell') \)

Let \( \varphi' = \varphi[p_{Q\psi}/Q\psi] \) be obtained from \( \varphi \) by replacing each \( Q\psi \) by \( p_{Q\psi} \)

Return \( MC^{\forall}_{\text{CTL}^*}(M', s, \varphi') \)
### Definition: SAT($\text{CTL}^*$)

<table>
<thead>
<tr>
<th>Input:</th>
<th>A formula $\varphi \in \text{CTL}^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question:</td>
<td>Existence of a model $M$ and a run $\sigma$ such that $M, \sigma, 0 \models \varphi$?</td>
</tr>
</tbody>
</table>

### Theorem

The satisfiability problem for $\text{CTL}^*$ is $2$-EXPTIME-complete.
Definition: Computation Tree Logic (CTL)

Syntax:

$$\varphi ::= \bot \mid p \ (p \in AP) \mid \neg \varphi \mid \varphi \lor \varphi \mid \text{EX} \varphi \mid \text{AX} \varphi \mid \text{E} \varphi \cup \varphi \mid \text{A} \varphi \cup \varphi$$

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

- $$\text{EF} \varphi = \text{E} \top \cup \varphi$$ and $$\text{AF} \varphi = \text{A} \top \cup \varphi$$
- $$\text{EG} \varphi = \neg \text{AF} \neg \varphi$$ and $$\text{AG} \varphi = \neg \text{EF} \neg \varphi$$
- $$\text{AG}(\text{req} \rightarrow \text{EF grant})$$
- $$\text{AG}(\text{req} \rightarrow \text{AF grant})$$
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 $s \in S$.

\[
\begin{align*}
  s \models p & \quad \text{if} \quad p \in \ell(s) \\
  s \models \text{EX} \varphi & \quad \text{if} \quad \exists s \to s' \text{ with } s' \models \varphi \\
  s \models \text{AX} \varphi & \quad \text{if} \quad \forall s \to s' \text{ we have } s' \models \varphi \\
  s \models \text{E} \varphi \text{ U } \psi & \quad \text{if} \quad \exists s = s_0 \to s_1 \to s_2 \to \cdots s_j \text{ finite path, with } \\
  & \quad \text{ } s_j \models \psi \text{ and } s_k \models \varphi \text{ for all } 0 \leq k < j \\
  s \models \text{A} \varphi \text{ U } \psi & \quad \text{if} \quad \forall s = s_0 \to s_1 \to s_2 \to \cdots \text{ infinite path, } \exists j \geq 0 \text{ with } \\
  & \quad \text{ } s_j \models \psi \text{ and } s_k \models \varphi \text{ for all } 0 \leq k < j
\end{align*}
\]
Example:

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

▷ $AX \varphi = \neg EX \neg \varphi,$

▷ $\neg (\varphi U \psi) = G \neg \psi \lor (\neg \psi U (\neg \varphi \land \neg \psi))$

▷ $A \varphi U \psi = \neg EG \neg \psi \land \neg E \neg \psi U (\neg \varphi \land \neg \psi)$

▷ $AG(req \rightarrow F grant) = AG(req \rightarrow AF grant)$

▷ $AGF \varphi = AGAF \varphi$

▷ $EFG \varphi = EFEG \varphi$

▷ $EGEF \varphi \neq EGF \varphi$

▷ $AFAG \varphi \neq AFG \varphi$

▷ $EGEX \varphi \neq EGX \varphi$
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 $M \not\models_\exists \lnot \varphi$

Definition: Model checking problems $MC^\forall_{\text{CTL}}$ and $MC^\exists_{\text{CTL}}$

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

Question: Does $M \models_\forall \varphi$? or Does $M \models_\exists \varphi$?
Model checking of $\mathsf{CTL}$

Theorem

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

Proof:

Compute $\llbracket \varphi \rrbracket = \{ s \in S \mid M, s \models \varphi \}$ by induction on the formula.

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

The labelling $\ell$ is encoded in $L$: for $p \in \text{AP}$ we have $L[s][p] = \top$ if $p \in \ell(s)$. 
Model checking of CTL

Definition: procedure semantics(\(\varphi\))

\[
\begin{align*}
\text{case } \varphi &= \neg \varphi_1 \\
\text{semantics}(\varphi_1) \\
[\varphi] &= S \setminus [\varphi_1] \\
& \quad \mathcal{O}(|S|) \\
\text{case } \varphi &= \varphi_1 \lor \varphi_2 \\
\text{semantics}(\varphi_1); \text{semantics}(\varphi_2) \\
[\varphi] &= [\varphi_1] \cup [\varphi_2] \\
& \quad \mathcal{O}(|S|) \\
\text{case } \varphi &= \text{EX} \varphi_1 \\
\text{semantics}(\varphi_1) \\
[\varphi] &= \emptyset \\
\text{for all } (s,t) \in T \text{ do if } t \in [\varphi_1] \text{ then } [\varphi] &= [\varphi] \cup \{s\} \\
& \quad \mathcal{O}(|S|) \quad \mathcal{O}(|T|) \\
\text{case } \varphi &= \text{AX} \varphi_1 \\
\text{semantics}(\varphi_1) \\
[\varphi] &= S \\
\text{for all } (s,t) \in T \text{ do if } t \notin [\varphi_1] \text{ then } [\varphi] &= [\varphi] \setminus \{s\} \\
& \quad \mathcal{O}(|S|) \quad \mathcal{O}(|T|)
\end{align*}
\]
Model checking of CTL

Definition: procedure \text{semantics}(\varphi)

case \varphi = E\varphi_1 U \varphi_2

\text{semantics}(\varphi_1); \text{semantics}(\varphi_2)

L := \llbracket \varphi_2 \rrbracket \quad \text{// the set } L \text{ is the “todo” list}

Z := \emptyset \quad \text{// the set } Z \text{ is the “done” list}

\text{while } L \neq \emptyset \text{ do}

\text{Invariant: } \llbracket \varphi_2 \rrbracket \cup (\llbracket \varphi_1 \rrbracket \cap T^{-1}(Z)) \subseteq Z \cup L \subseteq \llbracket E \varphi_1 U \varphi_2 \rrbracket

\text{take } t \in L; \quad L := L \setminus \{t\}; \quad Z := Z \cup \{t\}

\text{for all } s \in T^{-1}(t) \text{ do}

\quad \text{if } s \in \llbracket \varphi_1 \rrbracket \setminus (Z \cup L) \text{ then } L := L \cup \{s\}

\llbracket \varphi \rrbracket := Z

\text{\(O(|S| + |T|)\)}

\text{\(O(|S|)\)}

\text{\(O(|S|)\)}

\text{\(|S| \text{ times}\)}

\text{\(|T| \text{ times}\)}

Z is only used to make the invariant clear.
Z \cup L \text{ can be replaced by } \llbracket \varphi \rrbracket.
Model checking of CTL

Definition: procedure semantics(ϕ)

Replacing \( Z \cup L \) by \([\varphi]\)

case \( \varphi = E\varphi_1 \cup \varphi_2 \)

\[ \text{semantics}(\varphi_1); \text{semantics}(\varphi_2) \]
\[ L := [\varphi_2] \quad // \text{the set } L \text{ is implemented with a list} \]
\[ [\varphi] := [\varphi_2] \]

while \( L \neq \emptyset \) do

\[ \text{take } t \in L; \quad L := L \setminus \{t\} \]

for all \( s \in T^{-1}(t) \) do

\[ \text{if } s \in [\varphi_1] \setminus [\varphi] \quad \text{then } L := L \cup \{s\}; \quad [\varphi] := [\varphi] \cup \{s\} \]

\( O(|S| + |T|) \)
\( O(|S|) \)
\( O(|S|) \)
\( |S| \text{ times} \)
\( O(1) \)
\( |T| \text{ times} \)
\( O(1) \)
Model checking of CTL

Definition: procedure semantics(ϕ)

case ϕ = Aφ₁ U φ₂

semantics(φ₁); semantics(φ₂)

L := [φ₂] // the set L is the “todo” list
Z := ∅ // the set Z is the “done” list

for all s ∈ S do c[s] := |T(s)|

while L ≠ ∅ do

Invariant: ∀s ∈ S, c[s] = |T(s) \ Z| and

[φ₂] ∪ ([φ₁] ∩ {s ∈ S | T(s) ⊆ Z}) ⊆ Z ∪ L ⊆ [Aφ₁ U φ₂]

take t ∈ L; L := L \ {t}; Z := Z ∪ {t}

for all s ∈ T⁻¹(t) do

c[s] := c[s] - 1

if c[s] = 0 ∧ s ∈ [φ₁] \ (Z ∪ L) then L := L ∪ {s}

[ϕ] := Z

Z is only used to make the invariant clear.
Z ∪ L can be replaced by [ϕ].

O(|S| + |T|)
O(|S|)
O(|S|)
O(|S|)
O(1)
O(|S|)
O(1)
O(|S|)
O(|S|)
O(|S|)
O(|T|) times
O(1)
## Definition: procedure semantics($\varphi$)

Replacing $Z \cup L$ by $[\varphi]$

### case $\varphi = A\varphi_1 \cup \varphi_2$

- semantics($\varphi_1$); semantics($\varphi_2$)
- $L := [\varphi_2]$ // the set $L$ is implemented with a list
- $[\varphi] := [\varphi_2]$
- for all $s \in S$ do $c[s] := |T(s)|$
- while $L \neq \emptyset$ do
  - take $t \in L$; $L := L \setminus \{t\}$
  - for all $s \in T^{-1}(t)$ do
    - $c[s] := c[s] - 1$
    - if $c[s] = 0 \land s \in [\varphi_1] \setminus [\varphi]$ then
      - $L := L \cup \{s\}$; $[\varphi] := [\varphi] \cup \{s\}$
Complexity of $\text{CTL}$

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

**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 $\text{CTL}$ is PTIME-complete.
- The satisfiability problem for $\text{CTL}$ 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: $\bigwedge_i \neg F G CS_i = \bigwedge_i G F \neg CS_i$

Definition: Fair Kripke structure

$M = (S, T, I, AP, \ell, F_1, \ldots, F_n)$ with $F_i \subseteq S$.

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

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

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.

Then, \( E_f \varphi = E(\text{fair} \land \varphi) \) and \( A_f \varphi = A(\text{fair} \rightarrow \varphi) \)

where \( \text{fair} = \bigwedge_i \text{GF} F_i \)

Lemma: \( \text{CTL}_f \) cannot be expressed in CTL
Proof: CTL\(_f\) cannot be expressed in CTL

Consider the Kripke structure \(M_k\) defined by:

\[
\begin{align*}
2k &\quad 2k - 1 &\quad 2k - 2 &\quad 2k - 3 &\cdots \\
p &\quad \neg p &\quad p &\quad \neg p &\quad \cdots \\
\end{align*}
\]

- \(M_k, 2k \models EGFp\) but \(M_k, 2k - 2 \not\models EGFp\)
- If \(\varphi \in \text{CTL}\) 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 $\text{Fair} = \{ s \in S \mid M, s \models E_f \top \}$

Compute the SCC of $M$ with Tarjan's algorithm (in time $O(|M|)$).

Let $S'$ be the union of the (non trivial) SCCs which intersect each $F_i$.

Then, $\text{Fair}$ is the set of states that can reach $S'$.

Note that reachability can be computed in linear time.
Model checking of $\text{CTL}_f$

Proof: Reductions

$E_f X \varphi = EX(\text{Fair} \land \varphi)$ and $E_f \varphi \mathbin{U} \psi = E \varphi \mathbin{U} (\text{Fair} \land \psi)$

It remains to deal with $A_f \varphi \mathbin{U} \psi$.

Recall that $A \varphi \mathbin{U} \psi = \neg EG \neg \psi \land \neg E \neg \psi \mathbin{U} (\neg \varphi \land \neg \psi)$

This formula also holds for fair quantifications $A_f$ and $E_f$.

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

Proof: Computation of $E_f G \varphi$

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

Compute the SCC of $M_\varphi$ with Tarjan’s algorithm (in linear time).

Let $S'$ be the union of the (non trivial) SCCs of $M_\varphi$ which intersect each $F_i$.

Then, $M, s \models E_f G \varphi$ iff $M, s \models E \varphi \mathbin{U} S'$ iff $M_\varphi, s \models EF S'$.

This is again a reachability problem which can be solved in linear time.