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

- 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:
  $R_n$: $n$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
- Needham-Schroeder, authentication protocol based on symmetric encryption. Published in 1978 by Needham and Schroeder.
  Flaw found by Lowe in 1995 (man in the middle).

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](image1.png)  ![E.A. Emerson](image2.png)  ![J. Sifakis](image3.png)

Prix Turing 2007.

References

<table>
<thead>
<tr>
<th>Bibliography</th>
</tr>
</thead>
</table>

Outline

<table>
<thead>
<tr>
<th>Introduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Models</td>
</tr>
<tr>
<td>- ... with variables</td>
</tr>
<tr>
<td>- Concurrent systems</td>
</tr>
<tr>
<td>- Synchronization and communication</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Time Specifications</td>
</tr>
</tbody>
</table>

| 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 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.

Example: Digicode ABA

Every discrete system may be described with a TS.

Transition system

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 $M = (S, \Sigma, V, (D_v)_{v \in V}, T, I, AP, \ell)$
- $V$: set of (typed) variables, e.g., boolean, $[0..4]$, ...
- Each variable $v \in V$ has a domain $D_v$ (finite or infinite)
- Guard or Condition: unary predicate over $D = \Pi_{v \in V} D_v$
  Symbolic descriptions: $x < 5$, $x + y = 10$, ...
- Instruction or Update: map $f: D \rightarrow D$
  Symbolic descriptions: $x := 0$, $x := (y + 1)^2$, ...
- Transitions: $T \subseteq S \times (2^D \times \Sigma \times D) \times S$
  Symbolic descriptions: $s \xrightarrow{g,a,f} s' \wedge \nu \models g$

Example: Vending machine
- coffee: 50 cents, orange juice: 1 euro, ...
- possible coins: 10, 20, 50 cents
- we may shuffle coin insertions and drink selection

Programs = Kripke structures with variables
- Program counter = states
- Instructions = transitions
- Variables = variables

Example: GCD

TS with variables ...

Example: Digicode

...and its semantics ($n = 2$)
Only variables

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

Definition: TSV 
\[ M = (V, (D_v)_{v \in V}, T, I, AP, \ell) \]

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

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

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

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

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 $S = \prod_i S_i$, $\Sigma = \prod_i (\Sigma_i \cup \{\epsilon\})$, and $I = \prod_i I_i$
- $T = \{(p_1, \ldots, p_n) \mapsto (q_1, \ldots, q_n) : \text{ for all } i, (p_i, a_i, q_i) \in T_i \text{ or } p_i = q_i \text{ and } a_i = \epsilon\}$
- $AP = \bigcup_i AP_i$ and $\ell(p_1, \ldots, p_n) = \bigcup_i \ell(p_i)$

Synchronized products: restrictions of the general product.

Parallel compositions
- Synchronous: $\Sigma_{\text{sync}} = \prod_i \Sigma_i$
- Asynchronous: $\Sigma_{\text{sync}} = \bigcup_i \Sigma_i'$ with $\Sigma_i' = \{\epsilon\}^{i-1} \times \Sigma_i \times \{\epsilon\}^{n-i}$

Synchronizations
- By states: $S_{\text{sync}} \subseteq S$
- By labels: $\Sigma_{\text{sync}} \subseteq \Sigma$
- By transitions: $T_{\text{sync}} \subseteq T$

**Example: Printer manager**

Example: Asynchronous product

Synchronization by states: $(P, P)$ is forbidden

Synchronization by Rendez-vous

Synchronization by transitions is universal but too low-level.

**Definition: Rendez-vous**

- $!m$ sending message $m$
- $?m$ receiving message $m$

**SOS: Structural Operational Semantics**

Local actions

\[
\begin{align*}
& s_1^1 \xrightarrow{a_1} s_1' \\
& (s_1, s_2) \xrightarrow{m} (s_1', s_2') \\
& s_2^a \xrightarrow{a_2} s_2' \\
& (s_1, s_2, s_3) \xrightarrow{m} (s_1', s_2', s_3')
\end{align*}
\]

Rendez-vous

\[
\begin{align*}
& s_1 \xrightarrow{!m} s_1^1 \land s_2 \xrightarrow{?m} s_2^a \\
& (s_1, s_2, s_3) \xrightarrow{m} (s_1', s_2', s_3')
\end{align*}
\]

- It is a kind of synchronization by actions.
- Essential feature of process algebra.

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

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

Example: Synchronization by Rendez-vous

Cabin:

```
<table>
<thead>
<tr>
<th>Cabin</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>
```

```
?down  !leave₀ !reach₁
|      |      |
|      |      |
?up    !leave₁ !reach₂
|      |      |
|      |      |
?up    !leave₁ !reach₂
|      |      |
|      |      |
```

Door for level $i$:

```
<table>
<thead>
<tr>
<th>Door</th>
</tr>
</thead>
<tbody>
<tr>
<td>Closed</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
```

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)

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

\[ (\bar{s}, \nu) \xrightarrow{a} (\bar{s}', f(\nu)) \]

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 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 swape the first two assignments?

---

Atomicity

Example:

Initially $x = 1 \land y = 2$

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

Program $P_2$: \[ \left( \begin{array}{c}
LoadR_1, x \\
AddR_1, y \\
StoreR_1, x
\end{array} \right) \parallel \left( \begin{array}{c}
LoadR_2, x \\
AddR_2, y \\
StoreR_2, y
\end{array} \right) \]

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)

\[
ES ::= \text{skip} \mid \text{await } c \mid x := e \mid ES \mid ES \sqcap ES
\]

| when c do ES | if c then ES else ES |

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

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

**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 \( \text{await}(y = 1) \)

**Channels**

**Example:** Leader election

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

\[
\text{send}(id)
\]

loop forever

\[
\text{receive}(x)
\]

if \( (x = id) \) then STOP fi

if \( (x > id) \) then \( \text{send}(x) \)

**Definition:** Channels

- Declaration:
  - \( c \) : channel \([k]\) of bool size \( k \)
  - \( c \) : channel \([\infty]\) of int unbounded
  - \( c \) : channel \([0]\) of colors Rendez-vous

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

- Domain: Let \( D_m \) be the domain for a single message.
  - \( D_k = D^*_m \) \text{ size } k
  - \( D_\infty = D^*_m \) unbounded
  - \( D_e = \{e\} \) Rendez-vous

- Politics: FIFO, LIFO, BAG, . . .

**Semantics:** (lossy) FIFO

\[
\text{Send} \quad s_i \xrightarrow{c!e} s'_i \wedge \nu'(c) = \nu(c) \cdot \nu'(c)
\]

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

\[
\text{Receive} \quad s_i \xrightarrow{c?x} s'_i \wedge \nu'(c) = \nu'(c) \cdot \nu'(x)
\]

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

Lossy send

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

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.

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

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

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

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.

Linear versus Branching
Let \( M = (S, T, I, \text{AP}, \ell) \) be a Kripke structure.

References

Bibliography


A large list of references is given in this paper.

Bibliography


Some original References

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

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

Branching Time Specifications

Linear Temporal Logic (Pnueli 1977)

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

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

Definition: Semantics:

- $w, i \models p$ 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:

```
\begin{tikzpicture}[node distance=1cm]
  \node (p) {$p$};
  \node (q) [right of=p] {$q$};
  \node (r) [right of=q] {$r$};
  \node (s) [right of=r] {$s$};
  \node (t) [right of=s] {$t$};
  \node (u) [right of=t] {$u$};
  \node (v) [right of=u] {$v$};
  \node (w) [right of=v] {$w$};
  \node (x) [right of=w] {$x$};

  \draw (p) -- (q);
  \draw (q) -- (r);
  \draw (r) -- (s);
  \draw (s) -- (t);
  \draw (t) -- (u);
  \draw (u) -- (v);
  \draw (v) -- (w);
  \draw (w) -- (x);
\end{tikzpicture}
```

Linear Temporal Logic (Pnueli 1977)

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

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

Definition: Semantics:

- $w, i \models p$ 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:

```
\begin{tikzpicture}[node distance=1cm]
  \node (p) {$p$};
  \node (q) [right of=p] {$q$};
  \node (r) [right of=q] {$r$};
  \node (s) [right of=r] {$s$};
  \node (t) [right of=s] {$t$};
  \node (u) [right of=t] {$u$};
  \node (v) [right of=u] {$v$};
  \node (w) [right of=v] {$w$};
  \node (x) [right of=w] {$x$};

  \draw (p) -- (q);
  \draw (q) -- (r);
  \draw (r) -- (s);
  \draw (s) -- (t);
  \draw (t) -- (u);
  \draw (u) -- (v);
  \draw (v) -- (w);
  \draw (w) -- (x);
\end{tikzpicture}
```
Linear Temporal Logic (Pnueli 1977)

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

$$\varphi ::= \bot \mid p \mid \neg \varphi \mid \varphi \lor \varphi \mid \varphi \land \varphi \mid \varphi \lor \psi \mid \varphi \land \psi \mid \varphi \lor \varphi \mid \varphi \land \varphi$$

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

- $w, i \models p \iff p \in a_i$
- $w, i \models \neg \varphi \iff w, i \not\models \varphi$
- $w, i \models \varphi \lor \psi \iff w, i \models \varphi$ or $w, i \models \psi$
- $w, i \models \varphi \land \psi \iff \exists k, i \leq k \text{ and } w, k \models \varphi \text{ and } \forall j, (i \leq j < k) \to w, j \models \varphi$

**Example:**

$$\varphi \lor \psi \rightarrow \varphi \land \psi$$

---

**Linear Temporal Logic (Pnueli 1977)**

**Definition: Macros**

- **Eventually:** $\text{F} \varphi = \top \lor \varphi$
  \[\begin{array}{c}
  \hline
  \varphi \false \false \false \false
  \\
  \hline
  \\text{F} \varphi \false \false \false \false
  \\
  \hline
  \end{array}\]

- **Always:** $\text{G} \varphi = \neg \text{F} \neg \varphi$
  \[\begin{array}{c}
  \hline
  \varphi \false \false \false \false
  \\
  \hline
  \text{G} \varphi \false \false \false \false
  \\
  \hline
  \end{array}\]

- **Weak until:** $\varphi \text{W} \psi = \text{G} \varphi \lor \varphi \text{U} \psi$
  \[\begin{array}{c}
  \hline
  \varphi \false \false \false \false
  \\
  \hline
  \text{G} \varphi \false \false \false \false
  \\
  \hline
  \varphi \false \false \false \false
  \\
  \hline
  \end{array}\]

- **Release:** $\varphi \text{R} \psi = \psi \text{W} (\varphi \land \psi)$
  \[\begin{array}{c}
  \hline
  \varphi \false \false \false \false
  \\
  \hline
  \psi \false \false \false \false
  \\
  \hline
  \varphi \text{R} \psi \false \false \false \false
  \\
  \hline
  \end{array}\]

- **Next until:** $\varphi \text{XU} \psi = \text{X} (\varphi \text{U} \psi)$
  \[\begin{array}{c}
  \hline
  \varphi \false \false \false \false
  \\
  \hline
  \psi \false \false \false \false
  \\
  \hline
  \varphi \text{XU} \psi \false \false \false \false
  \\
  \hline
  \end{array}\]

- $\text{X} \psi = \bot$ \text{XU} \psi \land \varphi \text{U} \psi = \psi \lor (\varphi \land \varphi \text{XU} \psi)$

---

**Linear Temporal Logic (Pnueli 1977)**

**Examples:**

- Every elevator request should be eventually satisfied.
  $$\bigwedge \text{G}(\text{Call}_i \rightarrow \text{F}(\text{atLevel}_i \land \text{openDoor}_i))$$

- The elevator should not cross a level for which a call is pending without stopping.
  $$\bigwedge \text{G}(\text{Call}_i \rightarrow \neg \text{atLevel}_i \text{W} (\text{atLevel}_i \land \text{openDoor}_i))$$
Elegant algebraic proof of

Corollary:

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

Theorem [8, Kamp 68]

Example: LTL versus PLTL

Example: LTL versus PLTL

Example: LTL versus PLTL

Example: LTL versus PLTL

Theorem (Laroussinie & Markey & Schnoebelen 2002)

Past LTL

Past LTL

PLTL may be exponentially more succinct than LTL.

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

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

Example:

Example: LTL versus PLTL

\( G(\text{grant} \rightarrow Y(\neg \text{grant} \, \text{request})) \)

= \( (\text{request} \, R \neg \text{grant}) \land G(\text{grant} \rightarrow (\text{request} \lor X(\text{request} \, R \neg \text{grant}))) \)

Theorem (Laroussinie & Markey & Schnoebelen 2002)

PLTL may be exponentially more succinct than LTL.

Expressivity

Theorem [8, Kamp 68]

\( \text{LTL}(Y, S, X, U) = \text{FO}_2(\leq) \)

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

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

\[
w, k \models \varphi \iff w, k \models \bigvee\limits_i \vec{\varphi}_i \land \vec{\varphi}_i'
\]

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

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

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

Elegant algebraic proof of \( \text{LTL}(X, U) = \text{FO}_2(\leq) \) due to Wilke 98.

Model checking for LTL

Definition: Model checking problem

Input: A Kripke structure \( M = (S, T, I, \ell, \alpha) \)
A formula \( \varphi \in \text{LTL}(\text{AP}, Y, S, X, U) \)

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

- Universal MC: \( M \models \varphi \) 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 \( M \).

\[
M \models \varphi \iff M \models \exists \neg \neg \varphi
\]

Theorem [11, Sistla, Clarke 85], [12, Lichtenstein & Pnueli 85]

The Model checking problem for LTL is PSPACE-complete.
Satisfiability for LTL
Let $AP$ be the set of atomic propositions and $\Sigma = 2^{AP}$.

Definition: Satisfiability problem
Input: A formula $\varphi \in \text{LTL}(AP, Y, S, X, U)$
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 $\varphi \in \text{LTL}(AP, Y, S, X, U)$
Question: Existence of $w \in \Sigma^\omega$ such that $w, 0 \models \varphi$.
Remark: $\varphi$ is satisfiable iff $\text{F} \varphi$ 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 $\varphi \in \text{LTL}(AP, \ldots)$, construct a Büchi automaton $A_\varphi$ such that $\mathcal{L}(A) = \mathcal{L}(\varphi) = \{w \in \Sigma^\omega \mid w, 0 \models \varphi\}$.

Satisfiability (initial)
Check the Büchi automaton $A_\varphi$ for emptiness.

Model checking
Construct a synchronized product $B = M \otimes A_\neg \varphi$ so that the successful runs of $B$ correspond to the initial runs of $M$ satisfying $\neg \varphi$.
Then, check $B$ for emptiness.

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

Büchi automata
Definition:
$A = (Q, \Sigma, I, T, F)$ where
- $Q$: finite set of states
- $\Sigma$: finite set of labels
- $I \subseteq Q$: set of initial states
- $T \subseteq Q \times \Sigma \times Q$: transitions
- $F \subseteq Q$: set of accepting states (repeated, final)

Example:
$\varphi$: $1 \xrightarrow{a} 2 \xrightarrow{b} 1$
$\mathcal{L}(A) = \{w \in \{a, b\}^\omega \mid |w|_a = \omega\}$

Büchi automata for some LTL formulae
Definition:
Recall that $\Sigma = 2^{AP}$. For $\psi \in B(AP)$ we let $\Sigma_\psi = \{a \in \Sigma \mid a \models \psi\}$.
For instance, for $p, q \in AP$,
- $\Sigma_p = \{a \in \Sigma \mid p \in a\}$ and $\Sigma_{\neg p} = \Sigma \setminus \Sigma_p$
- $\Sigma_{p \lor q} = \Sigma_p \cup \Sigma_q$ and $\Sigma_{p \land q} = \Sigma_p \cap \Sigma_q$
- $\Sigma_{p \land \neg q} = \Sigma_p \setminus \Sigma_q$

Examples:
$\text{F} p$: $1 \xrightarrow{\Sigma} 2 \xrightarrow{\Sigma_p} 1$
$\text{XX} p$: $1 \xrightarrow{\Sigma} 2 \xrightarrow{\Sigma} 3 \xrightarrow{\Sigma_p} 4 \xrightarrow{\Sigma}$
$\text{G} p$: $1 \xrightarrow{\Sigma_p}$
Büchi automata for some LTL formulae

Examples:

FGp:
\[ \Sigma_1 \rightarrow \Sigma_p \rightarrow \Sigma_p \rightarrow \Sigma_2 \]
no deterministic Büchi automaton.

GFGp:
\[ \Sigma_1 \rightarrow \Sigma_p \rightarrow \Sigma_p \rightarrow \Sigma_2 \]
deterministic Büchi automata are not closed under complement.

G(p \rightarrow Fq):
\[ \Sigma_1 \rightarrow \Sigma_{p \land \neg q} \rightarrow \Sigma_{\neg q} \rightarrow \Sigma_2 \]

Properties

Büchi automata are closed under union, intersection, complement.

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

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

Exercise:
Given Büchi automata for \( \varphi \) and \( \psi \),
- Construct a Büchi automaton for \( X \varphi \) (trivial)
- Construct a Büchi automaton for \( \varphi \lor \psi \)

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 \( \varphi \).
**Generalized Büchi automata**

**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 \( F_i \).

![Generalized Büchi automata diagram](image)

**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 \( T_i \).

![Generalized Büchi automata diagram](image)

**GBA to BA**

**Proof:** Synchronized product with \( B \)

![GBA to BA diagram](image)

**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
\[
\begin{align*}
\neg(\varphi \lor \psi) & \equiv \neg \varphi \land \neg \psi \\
\neg(\varphi \land \psi) & \equiv \neg \varphi \lor \neg \psi \\
\neg(\varphi U \psi) & \equiv \neg \varphi R \neg \psi \\
\neg(\varphi R \psi) & \equiv \neg \varphi U \neg \psi \\
\neg X\varphi & \equiv X\neg \varphi \\
\neg \neg \varphi & \equiv \varphi 
\end{align*}
\]

This does not increase the number of Temporal subformulae.

**Temporal formulae**

**Definition:** Temporal formulae
- literals
- formulae with outermost connective X, U or R.

**Reducing the number of temporal subformulae**
\[
\begin{align*}
(X\varphi) \land (X\psi) & \equiv X(\varphi \land \psi) \\
(\varphi R \psi_1) \land (\varphi R \psi_2) & \equiv \varphi R (\psi_1 \land \psi_2) \\
(G\varphi) \land (G\psi) & \equiv G(\varphi \land \psi) \\
GF \varphi \lor GF \psi & \equiv GF(\varphi \lor \psi) \\
(\varphi R \psi_1) \lor (\varphi R \psi_2) & \equiv (\varphi R \psi_1) \lor (\varphi R \psi_2) \\
(\varphi_1 \lor \varphi_2) R \psi & \equiv (\varphi_1 \lor \varphi_2) R \psi \\
\end{align*}
\]
From LTL to BA [6, Demri & Gastin 10]

**Definition:**
- $Z \subseteq \text{NNF}$ is **consistent** if $\bot \not\in Z$ 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 $\bigwedge \emptyset = T$ 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 $\varphi$.
- $\text{U}(\varphi)$ the set of until-subformulæ 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 $\mathcal{L}(A_\varphi^Z) = \{u \in \Sigma^+ | u, 0 \models \bigwedge Z\}$
  where $A_\varphi^Z$ is $A_\varphi$ using $Z$ as unique initial state.

**Reduction rules**

**Definition:** Reduction of obligations to literals and next-formulæ

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 $\psi = \psi_1 \lor \psi_2$: $Y \xrightarrow{\varepsilon} (Y \setminus \{\psi\}) \cup \{\psi_1\}$
- If $\psi = \neg \psi_2$: $Y \xrightarrow{\varepsilon} (Y \setminus \{\psi\}) \cup \{\psi_2\}$
- If $\psi = \psi_1 \rightarrow \psi_2$: $Y \xrightarrow{\varepsilon} (Y \setminus \{\psi\}) \cup \{\psi_2\}$
- If $\psi = \bigwedge \psi_2$: $Y \xrightarrow{\varepsilon} (Y \setminus \{\psi\}) \cup \{\psi_2\}$
- If $\psi = \bigvee \psi_2$: $Y \xrightarrow{\varepsilon} (Y \setminus \{\psi\}) \cup \{\psi_2\}$
- If $\psi = \neg \bigvee \psi_2$: $Y \xrightarrow{\varepsilon} (Y \setminus \{\psi\}) \cup \{\psi_2\}$
- If $\psi = \neg \bigwedge \psi_2$: $Y \xrightarrow{\varepsilon} (Y \setminus \{\psi\}) \cup \{\psi_2\}$

Note the mark $!\psi$ on the second transitions for $U$ and $F$.
Reduction

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

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

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

$\text{Red}_\alpha(Y) = \{ Z \text{ consistent and reduced} \mid \text{there is a path } Y \xrightarrow{*} Z \text{ without using an edge marked with } \lambda\alpha \}$

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

Automaton $A_\varphi$

Definition: Automaton $A_\varphi$
- States: $Q = \{ Y \in \text{sub}(\varphi) \}$, $I = \{ \varphi \}$
- 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 $\alpha \in U(\varphi)$.

Correctness of $A_\varphi$

Proposition: $\mathcal{L}(\varphi) \subseteq \mathcal{L}(A_\varphi)$

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 $Y_n \xrightarrow{a_n} Y_1 \xrightarrow{a_1} Z$ with $a_n \in \Sigma_Z$ and $Y_{n+1} = \text{next}(Z)$,

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

Corollary: $\mathcal{L}(A_\varphi) \subseteq \mathcal{L}(\varphi)$
Let \( \rho \) be the second case, we obtain by induction that \( \psi \) is minimal. We know that \( \psi \) is accepting. We first show by induction that \( \rho \) is a run for \( \psi \) in \( A_\varphi \). It remains to show that \( \psi \) is successful. We start with \( \varphi = \psi \). Now let \( \alpha = \alpha_1 \cup \alpha_2 \in U(\varphi) \). Assume there exists \( N \geq 0 \) such that \( Y_n \not\in \text{Red}(Y_n) \) for all \( n \geq N \). Then \( Z_n \not\in \text{Red}_n(Y_n) \) for all \( n \geq N \) and we deduce that \( u, n \not\in \alpha_2 \) for all \( n \geq N \). But, since \( Z_n \not\in \text{Red}_n(Y_n) \), the formula \( \alpha \) has been reduced using an \( \epsilon \)-transition marked \( \alpha_2 \) along the path from \( Y_n \) to \( Z_n \). Therefore, \( X \alpha \in Z_n \) and \( \alpha \in Y_n \). By construction of the run we have \( u, n+1 \in Y_n \). Hence, \( u, n+1 \not\in \alpha \), a contradiction with \( u, n \not\in \alpha_2 \) for all \( n \geq N \). Consequently, the run \( \rho \) is successful and \( u \) is accepted by \( A_\varphi \).

**Proof:**

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

In the first case, we obtain by induction \( u, n \not\equiv \psi_2 \) and therefore \( u, n \not\equiv \psi \). In the second case, we obtain by induction \( u, n \not\equiv \psi_1 \). Since \( X \psi_1 \) is reduced we get \( X \psi \) in \( Y_n \).

Let \( Y_k \xrightarrow{\alpha} Y_{k+1} \in T_\varphi \) (such a value \( k \) exists since \( \rho \) is a run). We verify by induction that \( u, i \not\equiv \psi \) and \( \psi \in Y_{k+1} \) for all \( n \leq i < k \). Recall that \( u, n \not\equiv \psi \) and \( \psi \in Y_{k+1} \). 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 \( Y_{k+1} \in \text{next}(Z') \). Since \( k \) is minimal we know that \( Z' \not\in \text{Red}_n(Y_i) \). Hence, any reduction path from \( Y_i \) to \( Z' \) must use a step \( Y_i \xrightarrow{\alpha} Y' \setminus \{ \psi \} \cup \{ \psi_1, \psi \} \). By induction on the formula we obtain \( u, i \not\equiv \psi \). Also, since \( X \psi \) is reduced, we have \( X \psi \in Z' \) and \( \psi \in \text{next}(Z') \).

Second, we show that \( u, k \not\equiv \psi_2 \). Since \( Y_k \xrightarrow{\alpha} Y_{k+1} \in T_\varphi \), we find some \( Z' \in \text{Red}(Y_k) \) such that \( a_k \in \Sigma_{Z'} \) and \( Y_{k+1} \in \text{next}(Z') \). Since \( \psi \in Y_k \), along some reduction path from \( Y_k \) to \( Z' \) we use a step \( Y' \xrightarrow{\alpha} Y' \setminus \{ \psi \} \cup \{ \psi_2 \} \). By induction we obtain \( u, k \not\equiv \psi_2 \). Finally, we have shown \( u, n \not\equiv \psi_1 \cup \psi_2 \).
Example with two until sub-formulae

Example: Nested until: \( \varphi = p \mathsf{U} \psi \) with \( \psi = q \mathsf{U} r \)

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

\[
\begin{array}{c}
\varphi \mathrel{\mathsf{U}} \psi \\
\Sigma_r \mathrel{\mathsf{U}} \psi \\
\Sigma_q \mathrel{\mathsf{U}} \psi \\
\Sigma_y \\
\end{array}
\]

On the fly simplifications \( \mathcal{A}_\varphi \)

Built-in: reduction of a maximal formula.

Definition: Additional reduction rules

If \( \varphi \equiv \wedge Y' \) then we may use \( Y \overset{\varphi}{\rightarrow} Y' \).

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 \( \psi_2 \in Y \):
  \( Y \overset{\psi}{\rightarrow} Y \setminus \{\psi\} \)
- If \( \psi = \psi_1 \mathsf{U} \psi_2 \) and \( \psi_2 \in Y \):
  \( Y \overset{\psi}{\rightarrow} Y \setminus \{\psi\} \)
- If \( \psi = \psi_1 \mathsf{R} \psi_2 \) and \( \psi_1 \in Y \):
  \( Y \overset{\psi}{\rightarrow} Y \setminus \{\psi\} \cup \{\psi_2\} \)

Satisfiability and Model Checking

Corollary: PSPACE upper bound for satisfiability and model checking

- Let \( \varphi \in \mathsf{LTL} \), we can check whether \( \varphi \) is satisfiable (or valid) in space polynomial in \(|\varphi|\).
- Let \( \varphi \in \mathsf{LTL} \) and \( M = (S, T, I, \mathsf{AP}) \) be a Kripke structure. We can check whether \( M \models \varphi \) (or \( M \models \exists \varphi \)) in space polynomial in \(|\varphi| + \log |M|\).

Proof:

For \( M \models \varphi \) we construct a synchronized product \( M \otimes \mathcal{A}_\varphi \):

\[
\begin{align*}
\text{Transitions:} & \quad s \overset{a, s}{\rightarrow} \ \wedge \ Y = \ell(a) \rightarrow Y' \in \mathcal{A}_\varphi \\
\text{Initial states:} & \quad I \times \{\{\neg \varphi\}\} \\
\text{Acceptance conditions:} & \quad \text{inferred from} \ \mathcal{A}_\varphi. \\
\text{Check} & \quad M \otimes \mathcal{A}_\varphi \text{ for emptiness.}
\end{align*}
\]

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:

\[
\forall a \in \Sigma, \forall s \in Q, \quad \left( (s_1, a, s) \in T \iff (s_2, a, s) \in T \right) \quad \text{and} \quad \left( (s_1, a, s) \in T_i \iff (s_2, a, s) \in T_i \right) \quad \text{for all} \quad 1 \leq i \leq n.
\]

Remark: Sufficient condition

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

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

Example: Let \( \varphi = \mathsf{GF} p \land \mathsf{GF} q \).

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].

QBF Quantified Boolean Formulae

Definition: QBF
Input: A formula \( \gamma = Q_1 x_1 \cdots Q_n x_n \gamma' \) with \( \gamma' = \bigwedge_{1 \leq i \leq m, 1 \leq j \leq k_i} a_{ij} \)
\( Q_i \in \{ \forall, \exists \} \) and \( a_{ij} \in \{ x_i, \neg x_i, x_j, \neg x_j \} \).

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. \( Q_i = \forall \),
  \( \exists v' \in V. v[i-1] = v'[i-1] \) and \( \{ v, v' \} = \{ 0, 1 \} \).

Proposition:
\( \gamma \) is valid \iff \( \exists V \subseteq \{ 0, 1 \}^n \) s.t. \( V \) is nonempty, valid and closed.

MC^3(X, U) \leq_P SAT(X, U)
[11, Sistla & Clarke 85]
Let \( M = (S, T, I, AP, \ell) \) be a Kripke structure and \( \varphi \in LTL(AP, X, U) \)
Introduce new atomic propositions: \( AP_S = \{ a_s \mid s \in S \} \)
Define \( AP' = AP \uplus AP_S \)
\( \Sigma' = 2AP' \)
\( \pi : \Sigma' \rightarrow \Sigma \) by \( \pi(a) = a \cap AP \).
Let \( w \in \Sigma' \). We have \( w \models \pi(w) \iff \varphi \models \varphi \).
Define \( \psi_M \in LTL(AP', X, F) \) of size \( O(|M|^2) \) by
\[
\psi_M = \left( \bigvee_{s \in S} at_s \right) \land \left( \bigvee_{s \in S} \left( at_s \land \bigwedge_{i \neq s} \neg at_i \land \bigwedge_{p \in \ell(s)} p \land \bigwedge_{p \in \ell(s)} \neg p \land \bigvee_{t \in \ell(s)} X at_t \right) \right)
\]
Let \( w = a_{01}a_{23} \cdots \in \Sigma'. \) Then, \( w \models \psi_M \) iff there exists an initial infinite run \( \sigma \)
of \( M \) such that \( \pi(w) = \ell(\sigma) \) and \( a_i \cap AP_S = \{ at_s \} \) for all \( i \geq 0 \).
Therefore, \( M \models \exists \varphi \) iff \( \psi_M \land \varphi \) is satisfiable
\( M \models \varphi \) iff \( \psi_M \land \neg \varphi \) is not satisfiable

Remark: we also have \( MC^3(X, F) \leq_P SAT(X, F) \).
The following problems are NP-complete:

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

Proof: If \( M \models \psi \land \varphi \) then \( \gamma \) is valid.

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

\[ v_k^\tau = \begin{cases} 1 & \text{if } \tau, |\tau| \models \neg s_k x_k^i \\ 0 & \text{if } \tau, |\tau| \models s_k x_k^i \end{cases} \]

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

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

Claim: \( V \) is nonempty, valid and closed.

Theorem: Complexity of LTL

The following problems are PSPACE-complete:

- \( \text{SAT}(\text{LTL}(X, U, Y, S)) \), \( \text{MC}^3(\text{LTL}(X, U, Y, S)) \)
- \( \text{SAT}(\text{LTL}(X, F)) \), \( \text{MC}^3(\text{LTL}(X, F)) \), \( \text{MC}^2(\text{LTL}(X, F)) \)
- \( \text{SAT}(\text{LTL}(U)) \), \( \text{MC}^3(\text{LTL}(U)) \), \( \text{MC}^2(\text{LTL}(U)) \)
- The restriction of the above problems to a unique propositional variable

The following problems are NP-complete:

- \( \text{SAT}(\text{LTL}(F)) \), \( \text{MC}^2(\text{LTL}(F)) \)

Outline

Introduction
Models
Specifications
Linear Time Specifications
- Branching Time Specifications
  - CTL^*
  - CTL
  - Fair CTL
**Possibility is not expressible in LTL**

Example:
\[ \varphi: \text{Whenever } p \text{ holds, it is possible to reach a state where } q \text{ holds.} \]
\[ \varphi \text{ cannot be expressed in LTL.} \]

Consider the two models:

- **Model 1 (M1):**
  - Initial state: 1
  - Transitions:
    - From 1 to 2
    - From 1 to 3

- **Model 2 (M2):**
  - Initial state: 1
  - Transitions:
    - From 1 to 2
    - From 1 to 3

We need quantifications on runs:
\[ M_1 \models \varphi \quad \text{but} \quad M_2 \not\models \varphi \]

\[ M_1 \text{ and } M_2 \text{ satisfy the same LTL formulae.} \]

We need quantifications on runs:
\[ \varphi = \text{AG}(p \rightarrow \text{EF } q) \]
- \( \text{E: for some infinite run} \)
- \( \text{A: for all infinite runs} \)

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

**Example:** State formulae
Formulae of the form \( p \) or \( \text{E } \varphi \) or \( \text{A } \varphi \) are state formulae.
State formulae are closed under boolean connectives.
\[ [p] = \{ s \in S \mid \ell(s) \} \]
\[ [\neg \varphi] = S \setminus [\varphi] \]
\[ [\varphi_1 \lor \varphi_2] = [\varphi_1] \cup [\varphi_2] \]

**Definition:** Alternative syntax
- State formulae: \[ \varphi ::= \bot \mid p \mid \neg \varphi \mid \varphi \lor \varphi \mid \text{EF } \varphi \mid \text{AG } \varphi \]
- Path formulae: \[ \psi ::= \varphi \mid \neg \psi \mid \psi \lor \psi \mid X \psi \mid \psi U \psi \]

**CTL* (Emerson & Halpern 86)**

**Definition:** Syntax of the Computation Tree Logic \( \text{CTL}^* \)
\[ \varphi ::= \bot \mid p \mid \neg \varphi \mid \varphi \lor \varphi \mid X \varphi \mid \varphi U \varphi \mid \text{E } \varphi \mid \text{A } \varphi \]

**Definition:** Semantics:
Let \( M = (S, T, I, AP, \ell) \) be a Kripke structure and \( \sigma \) an infinite run of \( M \).
\[ M, \sigma, i \models \text{E } \varphi \quad \text{if} \quad M, \sigma', 0 \models \varphi \text{ for some infinite run } \sigma' \text{ such that } \sigma'(0) = \sigma(i) \]
\[ M, \sigma, i \models \text{A } \varphi \quad \text{if} \quad M, \sigma', 0 \models \varphi \text{ for all infinite runs } \sigma' \text{ such that } \sigma'(0) = \sigma(i) \]

**Example:** Some specifications
- \( \text{EF } \varphi: \varphi \text{ is possible} \)
- \( \text{AG } \varphi: \varphi \text{ is an invariant} \)
- \( \text{AF } \varphi: \varphi \text{ is unavoidable} \)
- \( \text{EG } \varphi: \varphi \text{ holds globally along some path} \)

**Remark:**
\[ A \varphi \equiv \neg \text{E } \neg \varphi \]

**Model checking of \( \text{CTL}^* \)**

**Definition:** Existential and universal model checking
Let \( M = (S, T, I, AP, \ell) \) be a Kripke structure and \( \varphi \in \text{CTL}^* \) a formula.
\[ M \models \exists \varphi \quad \text{if} \quad M, \sigma, 0 \models \varphi \text{ for some initial infinite run } \sigma \text{ of } M. \]
\[ M \models \forall \varphi \quad \text{if} \quad M, \sigma, 0 \models \varphi \text{ for all initial infinite runs } \sigma \text{ of } M. \]

**Remark:**
\[ M \models \exists \varphi \quad \text{iff} \quad I \cap [\text{E } \varphi] \neq \emptyset \]
\[ M \models \forall \varphi \quad \text{iff} \quad I \subseteq [\text{A } \varphi] \]
\[ M \models \forall \varphi \quad \text{iff} \quad M \not\models \exists \neg \varphi \]

**Definition:** Model checking problems \( \text{MC}_{\text{CTL}^*}^{\exists} \) and \( \text{MC}_{\text{CTL}^*}^{\forall} \)
**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 \( M \models \forall \varphi \) ?
Complexity of CTL* 

**Definition: Syntax of the Computation Tree Logic CTL***

\[
\phi ::= \bot \mid p \ (p \in AP) \mid \neg \phi \mid \phi \lor \phi \mid X \phi \mid \phi U \phi \mid E \phi \mid A \phi
\]

**Theorem**

The model checking problem for CTL* is PSPACE-complete

**Proof:**

- **PSPACE-hardness:** follows from LTL ⊆ CTL*.
- **PSPACE-easiness:** reduction to LTL-model checking by inductive eliminations of path quantifications.

Satisfiability for CTL*

**Definition: SAT(CTL*)**

**Input:** A formula \( \phi \in CTL^* \)

**Question:** Existence of a model \( M \) and a run \( \sigma \) such that \( M, \sigma, 0 \models \phi \)?

**Theorem**

The satisfiability problem for CTL* is 2-EXPTIME-complete

CTL (Clarke & Emerson 81)

**Definition: Computation Tree Logic (CTL)**

\[
\phi ::= \bot \mid p \ (p \in AP) \mid \neg \phi \mid \phi \lor \phi \mid EX \phi \mid AX \phi \mid \phi U \phi \mid A \phi \]

The semantics is inherited from CTL*.

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

\[
\mathcal{M} = \{ s \in S \mid M, s \models \phi \}
\]

**Examples: Macros**

- \( EF \phi = E \top U \phi \) and \( AF \phi = A \top U \phi \)
- \( EG \phi = \neg AF \neg \phi \) and \( AG \phi = \neg EF \neg \phi \)
- \( AG(\text{req} \rightarrow \text{EF grant}) \)
- \( AG(\text{req} \rightarrow \text{AF grant}) \)
**Remark:** Equivalent formulae

- AXφ = ¬EX¬φ,
- (φ U ψ) = G¬ψ V (¬ψ U (¬φ ∧ ¬ψ))
- Aφ U ψ = ¬EG¬ψ ∧ ¬E¬ψ U (¬φ ∧ ¬ψ)
- AG(req → F grant) = AG(req → AF grant)
- AGFφ = AG AFφ
- EFGφ = EFGφ
- EGEFφ ≠ E G Fφ
- AF AGφ ≠ AF Gφ
- EG EXφ ≠ E GXφ

---

**Definition: Model checking problems MC^∀_{CTL} and MC^∃_{CTL}**

**Input:** A Kripke structure M = (S, T, I, AP, ³) and a formula φ ∈ CTL

**Question:** Does M |=_∀φ?  or  Does M |=_∃φ?
Model Checking of CTL

Definition: procedure semantics(ϕ)

```plaintext
\text{case } ϕ = \neg ϕ_1 \\
\text{semanatics}(ϕ_1) \\
[ϕ] := S \setminus [ϕ_1] \\
O(|S|) \\
\text{case } ϕ = ϕ_1 \lor ϕ_2 \\
\text{semanatics}(ϕ_1); \text{semanatics}(ϕ_2) \\
[ϕ] := [ϕ_1] \cup [ϕ_2] \\
O(|S|) \\
\text{case } ϕ = EX ϕ_1 \\
\text{semanatics}(ϕ_1) \\
[ϕ] := \emptyset \\
\text{for all } (s, t) \in T \text{ do if } t \in [ϕ_1] \text{ then } [ϕ] := [ϕ] \cup \{t\} \\
O(|T|) \\
\text{case } ϕ = AX ϕ_1 \\
\text{semanatics}(ϕ_1) \\
[ϕ] := S \\
\text{for all } (s, t) \in T \text{ do if } t \not\in [ϕ_1] \text{ then } [ϕ] := [ϕ] \setminus \{s\} \\
O(|T|)
```

Replacing Z ∪ L by [ϕ]

```plaintext
\text{case } ϕ = Eϕ_1 \lor ϕ_2 \\
\text{semanatics}(ϕ_1); \text{semanatics}(ϕ_2) \\
L := [ϕ_2] \quad // \text{the set } L \text{ is implemented with a list} \\
[ϕ] := [ϕ_2] \\
O(|S|) \\
\text{while } L \neq \emptyset \text{ do} \\
\text{take } t \in L; L := L \setminus \{t\}; \quad Z := Z \cup \{t\} \\
O(1) \\
\text{for all } s \in T^{-1}(t) \text{ do} \\
\text{if } s \in [ϕ_1] \setminus (Z \cup L) \text{ then } L := L \cup \{s\} \\
O(|T|) \\
[ϕ] := Z \\
```

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

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

- case \(\varphi = A \varphi_1 \cup \varphi_2\) \(\text{O}(|S| + |T|)\)
- semantics(\(\varphi_1\)); semantics(\(\varphi_2\))
- \(L := [\varphi_2]\) // the set \(L\) is the "todo" list \(\text{O}(|S|)\)
- \(Z := \emptyset\) // the set \(Z\) is the "done" list \(\text{O}(|S|)\)
- for all \(s \in S\) do \(c[s] := [T(s)]\) \(\text{O}(|S|)\)
- while \(L \neq \emptyset\) do \(|S|\) times
- Invariant: \(\forall s \in S, c[s] = |T(s) \setminus Z|\) and
- \([\varphi_2] \cup ([\varphi_1] \cap \{s \in S | T(s) \subseteq Z\}) \subseteq Z \cup L \subseteq [A \varphi_1 \cup \varphi_2]\)
- take \(t \in L, L := L \setminus \{t\}; Z := Z \cup \{t\}\) \(\text{O}(1)\)
- for all \(s \in T^{-1}(t)\) do \(|T|\) times
- \(c[s] := c[s] - 1\) \(\text{O}(1)\)
- if \(c[s] = 0 \land s \in [\varphi_1] \setminus (Z \cup L)\) then \(L := L \cup \{s\}\)

\([\varphi] := Z\)

\(Z\) is only used to make the invariant clear.
\(Z \cup L\) can be replaced by \([\varphi]\).

Complexity of CTL

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.

fairness

Example: Fairness

Only fair runs are of interest

- Each process is enabled infinitely often: \(\bigwedge_i GF \text{run}_i\)
- No process stays ultimately in the critical section: \(\bigwedge_i \neg GCS_i = \bigwedge_i GF \neg CS_i\)

Definition: Fair Kripke structure

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

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

Definition: Syntax of fair-CTL

\[ \varphi ::= \bot \mid p \mid \neg \varphi \mid \varphi \lor \varphi \mid E_f X \varphi \mid A_f X \varphi \mid E_f \varphi U \varphi \mid A_f \varphi U \varphi \]

Definition: Semantics as a fragment of CTL*

Let \( M = (S, T, I, F_1, \ldots, F_n) \) be a fair Kripke structure.

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

where \( fair = \bigwedge_i GF F_i \)

Lemma: CTL\(_f\) cannot be expressed in CTL

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

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

\[ p \quad \neg p \quad \neg p \quad \neg p \quad \neg p \quad p \]

\[ M_k, 2k \models EG F \]

But \( M_k, 2k - 2 \npre \not\models EG F \)

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 CTL\(_f\)**

Theorem

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

Proof: Computation of 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, Fair is the set of states that can reach \( S' \).

Note that reachability can be computed in linear time.

---

Proof: Computation of \( E_f G \varphi \)

Let \( M_\varphi \) be the restriction of \( M \) to \( [\varphi] \).

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, \( M, s \models E_f G \varphi \iff M, s \models E \varphi U S' \).

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