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/

M1 du MPRI
2009-2010

Outline

Introduction
Bibliography

Models
Specifications
Linear Time Specifications
Branching Time Specifications

Need for formal verifications methods

Disastrous software bugs

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

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!
- Period instead of comma in FORTRAN? 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.

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

References

Bibliography
Principles of Model Checking.
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**

- MWGC, WC, MG
- C, MWG, MWC, G, MG, WC, MWGC

**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 system or Kripke structure**

**Definition:** TS

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

- \( S \): 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
- \( \text{AP} \): set of atomic propositions
- \( \ell : S \rightarrow 2^{\text{AP}} \): labelling function.

**Example: Digicode**

Every discrete system may be described with a TS.

**Transition systems with variables**

**Definition:** TSV

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

- \( \mathcal{V} \): set of (typed) variables, e.g., boolean, \([0..4], \ldots\)
- 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 \( f : D \rightarrow D \)
  - Symbolic descriptions: \( x := 0 \), \( x := (y + 1)^2 \), ...
- \( T \subseteq S \times (2^D \times \Sigma \times D^D) \times S \)
  - Symbolic descriptions: \( s \xrightarrow{y<50, \text{coin}=x\rightarrow y} s' \)
- \( I \subseteq S \times 2^D \)
  - 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: \( T' \subseteq (S \times D) \times \Sigma \times (S \times D) \)
  \[
  \frac{g, a, f}{s \xrightarrow{(s, \nu)} s'} \land \nu \models g
  \]

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

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

Example: Digicode

Only variables

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

Definition: TSV

\[
M = (V, (D_\nu)_{\nu \in V}, I, T, I, \ell)
\]

Symbolic representations with boolean functions

- \( I \) 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 functions with Binary Decision Diagrams.

Example: Boolean circuit: modulo 8 counter

\[
b'_0 = \neg b_0 \\
b'_1 = b_0 \oplus b_1 \\
b'_2 = (b_0 \land b_1) \oplus b_2
\]
Symbolic representation

Example: Logical representation

```
cpt < n
B, C

cpt ++

A

2

2

3

3

4

A

B

5

ERROR

delta_B = s = 1 \land cpt < n \land s' = 1 \land cpt' = cpt + 1
\lor s = 1 \land cpt = n \land s' = 5 \land cpt' = cpt + 1
\lor s = 2 \land s' = 3 \land cpt' = cpt
\lor s = 3 \land cpt < n \land s' = 1 \land cpt' = cpt + 1
\lor s = 3 \land cpt = n \land s' = 5 \land cpt' = cpt + 1
```
Example: Printer manager

Example: Asynchronous product
Synchronization by states: \((P, P)\) is forbidden

Synchronization by Rendez-vous

Definition: Rendez-vous
- \(!m\) sending message \(m\)
- \(?m\) receiving message \(m\)
- SOS: Structural Operational Semantics
- Local actions

\[
\begin{align*}
(s_1, s_2) \xrightarrow{a_1} (s'_1, s_2) \\
(s_1, s_2) \xrightarrow{a_2} (s_1, s'_2)
\end{align*}
\]
- Rendez-vous

\[
\begin{align*}
(s_1, s_2) \xrightarrow{!m} (s'_1, s_2) \\
(s_1, s_2) \xrightarrow{?m} (s_1, s'_2)
\end{align*}
\]
- It is a kind of synchronization by actions.
- Essential feature of process algebra.

Example: Elevator with 1 cabin, 3 doors, 3 calling devices
- \(?up\) is uncontrollable for the cabin
- \(?call_0\) is uncontrollable for the system

Example: digicode

Example: Synchronous product
Synchronization by transitions

Example: Elevator

Example: Synchronization by Rendez-vous

We should design the controller
Shared variables

Definition: Asynchronous product + shared variables

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

Semantics (SOS)

\[ \nu \models g \iff \exists s \in D \colon \nu(s) \models g \]

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

Exercise:

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

Atomicity

Example:

Initially \( x = 1 \land y = 2 \)
Program \( P_1 \):
\[ x := x + y \parallel y := x + y \]
Program \( P_2 \):
\[
\begin{align*}
\text{Load}R_1,x \parallel \text{Load}R_2,x \\
\text{Add}R_1,y \parallel \text{Add}R_2,y \parallel \\
\text{Store}R_1,x \parallel \text{Store}R_2,y
\end{align*}
\]
Assuming each instruction is atomic, what are the possible results of \( P_1 \) and \( P_2 \)?

Atomicity

Definition: Atomic statements

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

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

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

Assuming each instruction is atomic, what are the possible results of \( P_1 \) and \( P_2 \)?

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

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

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

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

Channels

Definition: Channels
- Declaration:
  - \( c : \text{channel \, [k]} \text{ of bool} \), size \( k \)
  - \( c : \text{channel \, [\infty]} \text{ of int} \), unbounded
  - \( c : \text{channel \, [0]} \text{ of colors} \), Rendez-vous
- Primitives:
  - \( \text{empty}(c) \)
  - \( c! e \): add the value of expression \( e \) to channel \( c \)
  - \( c? x \): read a value from \( c \) and assign it to \( x \)
- Domain: Let \( D_m \) be the domain for a single message.
  - \( D_c = D_k \), size \( k \)
  - \( D_c = D_\ast \), unbounded
  - \( D_c = \{\varepsilon\} \), Rendez-vous
- Politics: FIFO, LIFO, BAG, ...

Semantics: (lossy) FIFO

\[
\begin{align*}
\text{Send:} & \quad \frac{s_i \xrightarrow{c! e} s'_i \land \nu'(e) = \nu(e) \cdot \nu(c)}{(s, \nu) \xrightarrow{c! e} (s', \nu')} \\
\text{Receive:} & \quad \frac{s_i \xrightarrow{c? x} s'_i \land \nu(c) = \nu'(x) \cdot \nu'(c)}{(s, \nu) \xrightarrow{c? x} (s', \nu')} \\
\text{Lossy send:} & \quad \frac{s_i \xrightarrow{c! e} s'_i}{(s, \nu) \xrightarrow{c! e} (s', \nu)}
\end{align*}
\]

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 programs = transition system with variables
- Concurrent programs with shared variables
- Concurrent programs with Rendez-vous
- Concurrent programs 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.

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 $M = (S, T, I, \text{AP}, \ell)$ 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 $\Sigma = 2^{\text{AP}}$.

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.

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

$\varphi ::= \bot | p \ (p \in \text{AP}) | \neg \varphi | \varphi \lor \psi | X \varphi | \varphi U \psi$

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 $w, i \models \varphi$

$w, i \models \varphi \lor \psi$ 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:
Linear Temporal Logic (Pnueli 1977)

**Definition: Syntax:** $LTL(AP, X, U)$

$$
\varphi ::= \bot \mid p \mid p \in AP \mid \neg \varphi \mid \varphi \lor \varphi \mid X \varphi \mid \varphi U \varphi
$$

**Definition: Semantics:** $w = a_0 a_1 a_2 \cdots \in \Sigma^\omega$ with $\Sigma = 2^{AP}$ 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:**

![Example Diagram](image)

Linear Temporal Logic (Pnueli 1977)

**Definition: Macros**

- **Eventually:** $F \varphi = T U \varphi$

![Eventually Macro Diagram](image)

- **Always:** $G \varphi = \neg F \neg \varphi$

![Always Macro Diagram](image)

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

![Weak until Macro Diagram](image)

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

![Release Macro Diagram](image)

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

![Next until Macro Diagram](image)

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

Examples:

Every elevator request should be eventually satisfied.

\[ \forall i \ G(\text{Call}_i \rightarrow F(\text{atLevel}_i \land \text{openDoor}_i)) \]

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

\[ \forall i \ G(\text{Call}_i \rightarrow \neg \text{atLevel}_i \land W(\text{atLevel}_i \land \text{openDoor}_i)) \]

**Past LTL**

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

\[ w, i \models \forall \varphi \] if \( i > 0 \) and \( w, i-1 \models \varphi \)

\[ w, i \models \varphi \exists \psi \] if \( \exists k. k \leq i \) and \( w, k \models \psi \) and \( \forall j. (k < j \leq i) \rightarrow w, y \models \varphi \)

Example:

Example: LTL versus PLTL

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

Theorem (Laroussinie & Markey & Schnoebelen 2002)

PLTL may be exponentially more succinct than LTL.

**Expressivity**

**Theorem (Kamp 68)**

\[ \text{LTL}(Y, S, X, U) = \text{FO}_{\Sigma_2}(\leq) \]

Separation Theorem (Gabbay, Pnueli, Shelah & Stavi 80)

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

\[ w, k \models \varphi \iff w, k \models \bigvee_i \overline{\varphi_i} \land \overline{\psi_i} \]

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

For all \( \varphi \in \text{LTL}(Y, S, X, U) \) there exist \( \overline{\varphi} \) and \( \overline{\psi} \)

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

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 $\phi \in \text{LTL}(AP, Y, S, X, U)$
Question: Does $M \models \phi$ ?

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

$M \models \phi$ iff $M \not\models \neg \phi$

Theorem (Sistla & Clarke 85, Lichtenstein et. al 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 $\phi \in \text{LTL}(AP, Y, S, X, U)$
Question: Existence of $w \in \Sigma^\omega$ and $i \in \mathbb{N}$ such that $w, i \models \phi$.

Definition: Initial Satisfiability problem
Input: A formula $\phi \in \text{LTL}(AP, Y, S, X, U)$
Question: Existence of $w \in \Sigma^\omega$ such that $w, 0 \models \phi$.

Theorem (Sistla & Clarke 85, Lichtenstein et. al 85)
The satisfiability problem for LTL is PSPACE-complete

Definition: (Initial) validity
$\phi$ is valid if $\neg \phi$ is not satisfiable.

Decision procedure for LTL

Definition: The core
From a formula $\phi \in \text{LTL}(AP, \ldots)$, construct a Büchi automaton $A_\phi$ such that
$L(A) = L(\phi) = \{ w \in \Sigma^\omega \mid w, 0 \models \phi \}$.

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

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

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:

$L(A) = \{ w \in \{ a, b \}^\omega \mid w|_a = \omega \}$
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 $\Sigma_{\neg p} = \Sigma \setminus \Sigma_p$
- $\Sigma_{p \land q} = \Sigma_p \cap \Sigma_q$ and $\Sigma_{p \lor q} = \Sigma_p \cup \Sigma_q$
- $\Sigma_{p \land \neg q} = \Sigma_p \setminus \Sigma_q$

Examples:
- $F p$: $\Sigma_1 \rightarrow_\Sigma \Sigma_2$ or $\Sigma_1 \rightarrow_\Sigma \Sigma_{\neg p} \rightarrow_\Sigma \Sigma_2$
- $X X p$: $\Sigma_1 \rightarrow_\Sigma \Sigma_2 \rightarrow_\Sigma \Sigma_3 \rightarrow_\Sigma \Sigma_4 \rightarrow_\Sigma \Sigma$
- $G p$: $\Sigma_1 \rightarrow_\Sigma \Sigma_p$

Büchi automata are closed under union, intersection, complement.
- Union: trivial
- Intersection: easy (exercice)
- Complement: hard

Let $\varphi = F((p \land X^n \neg p) \lor (\neg p \land X^n p))$
- $\Sigma_0 \rightarrow_\Sigma \Sigma_p \rightarrow_\Sigma \Sigma_1 \rightarrow_\Sigma \Sigma_2 \rightarrow_\Sigma \cdots \rightarrow_\Sigma \Sigma_{n-1} \rightarrow_\Sigma \Sigma_{p \lor q}$
- $\Sigma_{\neg p}$

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 \( \varphi \in \text{LTL}(AP, X, U) \) ... 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 \( F_i \).

Definition: acceptance on transitions
\[ A = (Q, \Sigma, I, T_1, \ldots, T_n) \text{ with } T_i \subseteq T. \]
An infinite run \( \sigma \) is successful if it uses infinitely many transitions from each \( T_i \).

GBA to BA

Proof: Synchronized product with \( B \)

Transitions:
\[ t = s_1 \xrightarrow{a} s'_1 \in A \land s_2 \xrightarrow{a} s'_2 \in B \]
Accepting states: \( Q \times \{n\} \)

Negative normal form

Definition: Syntax (\( p \in AP \))
\[ \varphi ::= T | \bot | p | \neg p | \varphi \lor \varphi | \varphi \land \varphi | X \varphi | \varphi U \varphi | \varphi R \varphi \]

Proposition: Any formula can be transformed in NNF
\[ \neg (\varphi \lor \psi) \equiv (\neg \varphi) \land (\neg \psi) \]
\[ \neg (\varphi \land \psi) \equiv (\neg \varphi) \lor (\neg \psi) \]
\[ \neg (\varphi U \psi) \equiv (\neg \varphi) R (\neg \psi) \]
\[ \neg (\varphi R \psi) \equiv (\neg \varphi) U (\neg \psi) \]
\[ \neg X \varphi \equiv X \neg \varphi \]
\[ \neg \neg \varphi \equiv \varphi \]
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

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

From LTL to BA (See [6])

Definition:
- \(Z \subseteq \text{NNF}\) is consistent if \(\bot \notin Z\) and \(\{p, \neg p\} \notin Z\) for all \(p \in \text{AP}\).
- For \(Z \subseteq \text{NNF}\), we define \(\land Z = \land_{p \in Z} \psi\).
  Note that \(\land \emptyset = \top\) if and if \(Z\) is inconsistent then \(\land Z \equiv \bot\).

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

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 \(L(A_\varphi) = \{u \in \Sigma^* | u, 0 \models \land Z\}\)
  where \(A_\varphi^Z\) is \(A_\varphi\) using \(Z\) as unique initial state.

Reduced formulae

Definition: Reduced formulae
- A formula is reduced if it is a literal \((p\text{ or } \neg p)\) 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
- \(\text{next}(Z) = \{\alpha | \alpha \in Z\}\)
- \(\Sigma_Z = \bigcap_{p \in Z} \Sigma_p \cap \bigcap_{\neg p \in Z} \Sigma_{\neg p}\)

Lemma: Next step
Let \(Z \subseteq \text{NNF}\) be consistent and reduced.
Let \(a = a_0a_1a_2 \cdots \in \Sigma_\omega\) and \(n \geq 0\). Then
\[u, n \models \land Z \quad \text{iff} \quad u, n + 1 \models \land \text{next}(Z) \quad \text{and} \quad a_n \in \Sigma_Z\]

- \(A_\varphi\) 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).

Reduction rules

Definition: Reduction of obligations to literals and next-formulae
Let \(Y \subseteq \text{NNF}\) and let \(\psi \in Y\) maximal not reduced.

- If \(\psi = \psi_1 \land \psi_2\): \(Y \xrightarrow{\psi} Y \setminus \{\psi\} \cup \{\psi_1, \psi_2\}\)
- If \(\psi = \psi_1 \lor \psi_2\): \(Y \xrightarrow{\psi} Y \setminus \{\psi\} \cup \{\psi_1\}\)
- If \(\psi = \psi_1 R \psi_2\): \(Y \xrightarrow{\psi} Y \setminus \{\psi\} \cup \{\psi_1, \psi_2\}\)
- If \(\psi = G \psi_2\): \(Y \xrightarrow{\psi} Y \setminus \{\psi\} \cup \{\psi_2, X\psi\}\)
- If \(\psi = \psi_1 U \psi_2\): \(Y \xrightarrow{\psi} Y \setminus \{\psi\} \cup \{\psi_2\}\)
- If \(\psi = \psi_1 U \psi_2\): \(Y \xrightarrow{\psi} Y \setminus \{\psi\} \cup \{\psi_2, X\psi\}\)
- If \(\psi = F \psi_2\): \(Y \xrightarrow{\psi} Y \setminus \{\psi\} \cup \{\psi, X\psi\}\)

Note the mark \(\downarrow\) on the second transitions for \(U\) and \(F\).
**Reduction rules**

Example: \( \varphi = G(p \rightarrow Fq) \)

\[
\varphi = G(\neg p \lor Fq) \\
q, X \varphi \\
(\neg p \lor Fq, X \varphi) \\
\varepsilon \\
Fq, X \varphi \\
\varepsilon \\
X Fq, X \varphi \\
(\neg p, X \varphi)
\]

State = set of obligations.
Reduce obligations to literals and next-formulae.
Note again the mark \(!Fq\) on the last edge.

**Automaton \( A_{\varphi} \)**

Definition: Automaton \( A_{\varphi} \)

- States: \( Q = 2^{\text{sub}(\varphi)} \), \( I = \{ \varphi \} \)
- Transitions: \( T = \{ Y \triangleleft a \rightarrow \text{next}(Z) \mid Y \in Q, a \in \Sigma_Z \text{ and } Z \in \text{Red}(Y) \} \)
- Acceptance: \( T_\alpha = \{ Y \triangleleft a \rightarrow \text{next}(Z) \mid Y \in Q, a \in \Sigma_Z \text{ and } Z \in \text{Red}_\alpha(Y) \} \)

for each \( \alpha \in U(\varphi) \).

**Automaton \( A_{\varphi} \)**

Example: \( \varphi = G(p \rightarrow Fq) \)

\[
\varphi = G(\neg p \lor Fq) \\
q, X \varphi \\
(\neg p \lor Fq, X \varphi) \\
\varepsilon \\
Fq, X \varphi \\
\varepsilon \\
X Fq, X \varphi \\
(\neg p, X \varphi)
\]

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, \( \forall \psi \in \text{sub}(\varphi), \forall n \geq 0, \forall Y_n \xrightarrow{Y} Z \) with \( a_n \in \Sigma_Z, Y_{n+1} = \text{next}(Z) \)
\( \psi \in Y \implies u, n \models \psi \)

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

---

**Satisfiability and Model Checking**

**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 \( M = (S, T, I, AP, \ell) \) 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 \( M \otimes A_{\neg \varphi} \):

**Transitions:**
\[
\begin{align*}
\Delta &\xrightarrow{s \xrightarrow{Y \ell(s)} Y' \in A_{\neg \varphi}} (s, Y) \xrightarrow{(s', Y')} (s', Y')
\end{align*}
\]

Acceptance conditions: inherited from \( A_{\neg \varphi} \).
Check \( M \otimes A_{\neg \varphi} \) for emptiness.

---

**On the fly simplifications \( A_\varphi \)**

**Built-in:** reduction of a maximal formula.

**Definition:** Additional reduction rules

If \( \bigwedge Y \equiv \bigwedge Y' \) then we may use \( Y \xrightarrow{\ell} Y' \).

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

- If \( \psi = \psi_1 \lor \psi_2 \) and \( \psi_1 \in Y \lor \psi_2 \in Y \): \( Y \xrightarrow{\ell} Y \setminus \{\psi\} \)
- If \( \psi = \psi_1 U \psi_2 \) and \( \psi_2 \in Y \): \( Y \xrightarrow{\ell} Y \setminus \{\psi\} \)
- If \( \psi = \psi_1 R \psi_2 \) and \( \psi_1 \in Y \): \( Y \xrightarrow{\ell} Y \setminus \{\psi\} \cup \{\psi_2\} \)
On the fly simplifications $A_\varphi$

**Definition:** Merging equivalent states

Let $A = (Q, \Sigma, I, T_1, T_2, \ldots, T_n)$ and $s_1, s_2 \in Q$.

We can merge $s_1$ and $s_2$ if they have the same outgoing transitions:

$$\forall a \in \Sigma, \forall s_i \in Q, \quad (s_1, a, s) \in T_i \iff (s_2, a, s) \in T_i$$

and

$$\forall a \in \Sigma, \forall s_i \in Q, \quad (s_1, a, s) \in T_i \iff (s_2, a, s) \in T_i \quad \text{for all } 1 \leq i \leq n.$$

**Remark:** Sufficient condition

Two states $Y, Y'$ of $A_\varphi$ have the same outgoing transition if

$$\text{Red}(Y) = \text{Red}(Y')$$

and

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

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

Without merging states $A_\varphi$ has 4 states.

These 4 states have the same outgoing transitions.

The simplified automaton has only one state.

---

**MC$^3(X, U) \leq_P \text{SAT}(X, U)$ (Sistla & Clarke 85)**

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

Introduce new atomic propositions: $AP' = \{a_t \mid s \in S\}$

Define $AP'' = AP \cup AP'$

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

Define

$$\psi_M = \left( \bigvee_{a \in I} a_t \right) \land G \left( \bigvee_{a \in I} \left( a_t \land \bigwedge_{p \in I} \neg a_t \land \bigwedge_{p \in I} p \land \bigwedge_{p \in I} \neg p \land \bigvee_{t \in T(s)} X a_t \right) \right)$$

We have $w \models \psi_M$ iff $\pi(w) = \ell(\sigma)$ for some initial infinite run $\sigma$ of $M$.

Therefore, $M \models \varphi$ iff $\psi_M \land \varphi$ is satisfiable

$M \models \varphi$ iff $\psi_M \land \neg \varphi$ is not satisfiable

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

---

**QBF Quantified Boolean Formulae**

**Definition:** QBF

**Input:** A formula $\gamma = Q_1 x_1 \cdots Q_n x_n \gamma'$ with $\gamma' = \bigwedge_{1 \leq i \leq n} a_{ij}$

$Q_i \in \{\forall, \exists\}$ 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 $v = v_1 \cdots v_n \in \{0, 1\}^\omega$.

We write $v[i]$ for the prefix of length $i$.

Let $V \subseteq \{0, 1\}^\omega$ 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 $\forall v \in V, \forall 1 \leq i \leq n \text{ s.t. } Q_i = \forall, \exists v' \in V \text{ s.t. } v[i - 1] = v'[i - 1] \text{ and } \{v_i, v'_i\} = \{0, 1\}$.

**Proposition:** $\gamma$ is valid if $\exists V \subseteq \{0, 1\}^\omega$ s.t. $V$ is nonempty valid and closed

---

**Other constructions**

- Tableau construction. See for instance [8]
  + Easy definition, easy proof of correctness
  + Works both for future and past modalities
  - Inefficient without optimizations
- Using Very Weak Alternating Automata [7].
  + Very efficient
  - Only for future modalities
- The domain is still very active.
- See other references in [6].

Fast LTL to Buechi 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,
QBF \leq_p MC^3(U) \quad (Sistla & Clarke 85)

Let $\gamma = Q_1x_1 \cdots Q_nx_n \land \bigvee_{1 \leq i \leq m} a_{ij}$ with $Q_i \in \{\forall, \exists\}$ and $a_{ij}$ literals.

Consider the KS $M$:

We can extend Define by induction Let Final step: the infinite run. We say that Assume Proof: If Then, Let Consider the KS defined by $\psi$ is valid i.e., $\exists \sigma \in \{0, 1\}^\omega$ be nonempty, valid and closed. First ingredient: extension of a run. Assume $\tau = e_0 \overset{\tau}{\rightarrow} f_m$ 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 $\tau' = \tau \overset{\tau'}{\rightarrow} e_0 \overset{\tau'}{\rightarrow} e_0 \overset{\tau'}{\rightarrow} f_m$ with $v'^\tau = v'$ and $\tau', 0 \models \psi$. We say that $\tau'$ is the extension of $\tau$ wrt. $i$. Second step: the sequence of indices for the extensions. Let $1 \leq i_k < \cdots < i_1 \leq n$ be the indices of universal quantifications ($Q_{i_k} = \forall$). Define by induction $w_1 = i_1$ and if $k < l$, $w_{k+1} = w_k i_{k+1} w_k$. Let $w = (w_1)^\omega$. Final step: the infinite run. Let $\nu \in V \neq \emptyset$ and let $\tau = e_0 \overset{\tau}{\rightarrow} f_m$ 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: $\sigma, 0 \models \psi \land \varphi$. Complex of LTL

Theorem: Complexity of LTL
The following problems are PSPACE-complete:
- SAT(LTL(X, U, Y, S)), MC^0(LTL(X, U, Y, S)), MC^3(LTL(X, U, Y, S))
- SAT(LTL(X, F)), MC^0(LTL(X, F)), MC^3(LTL(X, F))
- SAT(LTL(U)), MC^0(LTL(U)), MC^3(LTL(U))
- The restriction of the above problems to a unique propositional variable

The following problems are NP-complete:
- SAT(LTL(F)), MC^3(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:

\[ M_1: \quad \begin{array}{c|c|c|c}
1 & p, q & 2 & p \\
2 & p & 3 & q \\
4 & & & \\
\end{array} \]
\[ M_2: \quad \begin{array}{c|c|c|c}
1 & p, q & 2 & p \\
2' & p & 3 & q \\
4 & & & \\
\end{array} \]

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

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

We need quantifications on runs:

\( \varphi = AG(p \rightarrow EF q) \)

- \( E \): for some infinite run
- \( A \): for all infinite runs

CTL* (Emerson & Halpern 86)

Definition: Syntax of the Computation Tree Logic CTL*

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

Definition: Semantics:

Let \( M = (S, T, I, AP, t) \) 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:

\( A \varphi \equiv \neg E \neg \varphi \)
PSPACE-hardness: follows from

Proof:
The model checking problem for
Theorem
Path formulae
State formulae
Definition: Alternative syntax
Formulae of the form
Example: State formulae
If \( \phi \)
Definition: State formulae
State formulae are closed under boolean connectives.
Complexity of \( \text{CTL}^* \)
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 \( \text{LTL} \)-model checking by inductive eliminations of path quantifications.

\[
\begin{align*}
\phi \in \text{CTL}^* & \text{ is a state formula if } \forall M, \sigma, \sigma', i, j \text{ such that } \sigma(i) = \sigma'(j) \text{ we have } \\
M, \sigma, i \models \phi & \iff M, \sigma', j \models \phi \\
\text{If } \phi \text{ is a state formula and } M = (S, T, I, AP, \ell) \text{, define } \\
[\phi]^M &= \{ s \in S \mid M, s \models \phi \}
\end{align*}
\]

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
Path formulae

\[
\begin{align*}
\varphi & ::= \bot \mid [p \in \text{AP}) \mid \neg \varphi \mid \varphi \lor \varphi \mid E \varphi \mid A \varphi \\
\psi & ::= \varphi \mid \neg \psi \mid \psi \lor \psi \mid X \psi \mid \psi U \psi
\end{align*}
\]

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.

\[
M \models \exists \varphi \text{ if } M, \sigma, 0 \models \varphi \text{ for some initial infinite run } \sigma \text{ of } M. \\
M \models \forall \varphi \text{ if } M, \sigma, 0 \models \varphi \text{ for all initial infinite run } \sigma \text{ of } M.
\]

Remark:
\[
\begin{align*}
M \models \exists \varphi & \iff I \cap [E \varphi] \neq \emptyset \\
M \models \forall \varphi & \iff I \subseteq [A \varphi] \\
M \models \forall \varphi & \iff M \not\models \neg \varphi
\end{align*}
\]

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 \( M \models \exists \varphi \) ? \quad \text{or} \quad \text{Does } M \models \forall \varphi ?

\[
\text{MC}_{\text{CTL}^*}^\forall \text{ in PSPACE}
\]

Proof:
For \( Q \in \{\exists, \forall\} \), let \( \text{MC}_{\text{CTL}^*}^Q(M, s, \varphi) \) be the function which computes in polynomial space whether \( M, s \models_Q \varphi \).

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

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

If \( E, A \) do not occur in \( \varphi \) then return \( \text{MC}_{\text{CTL}^*}^Q(M, s, \varphi) \) fi
Let \( Q \varphi \) be a subformula of \( \varphi \) with \( \varphi \in \text{LTL} \) and \( Q \in \{E, A\} \)
Let \( p_Q \psi \) be a new propositional variable
Define \( \ell': S \rightarrow 2^{AP} \) with \( AP' = AP \cup \{ p_Q \psi \} \) by
\[
\ell'(t) \cap AP = \ell(t) \text{ and } p_Q \psi \in \ell'(t) \text{ iff } \text{MC}_{\text{CTL}^*}^Q(M, t, \psi)
\]
Let \( M' = (S, T, I, AP', \ell') \)
Let \( \varphi' = \varphi[p_Q \psi/Q \psi] \) be obtained from \( \varphi \) by replacing each \( Q \psi \) by \( p_Q \psi \)
Return \( \text{MC}_{\text{CTL}^*}^Q(M', s, \varphi') \)
Satisfiability for $\text{CTL}^*$

Definition: $\text{SAT}(\text{CTL}^*)$
Input: A formula $\varphi \in \text{CTL}^*$
Question: Existence of a model $M$ and a run $\sigma$ such that $M, \sigma, 0 \models \varphi$?

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

---

$\text{CTL}$ (Clarke & Emerson 81)

Definition: Computation Tree Logic ($\text{CTL}$)

Syntax:

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

The semantics is inherited from $\text{CTL}^*$.

Remark: All $\text{CTL}$ formulae are state formulae

Examples: Macros
- $\text{EF}\varphi = \text{E}\top \text{U} \varphi$ and $\text{AF}\varphi = \text{A}\top \text{U} \varphi$
- $\text{EG}\varphi = \neg \text{AF}\neg \varphi$ and $\text{AG}\varphi = \neg \text{EF}\neg \varphi$
- $\text{AG}(\text{req} \rightarrow \text{EF grant})$
- $\text{AG}(\text{req} \rightarrow \text{AF grant})$

---

$\text{CTL}$ (Clarke & Emerson 81)

Definition: Semantics
All $\text{CTL}$-formulae are state formulae. Hence, we have a simpler semantics.

Let $M = (S,T,I,\text{AP},\ell)$ be a Kripke structure without deadlocks and let $s \in S$.

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

---

Example:

\begin{itemize}
  \item $\text{EX} p = \{1, 2, 3, 5, 6\}$
  \item $\text{AX} p = \{3, 6\}$
  \item $\text{EF} p = \{1, 2, 3, 4, 5, 6, 7, 8\}$
  \item $\text{AF} p = \{2, 3, 5, 6, 7\}$
  \item $\text{E} q \text{ U} r = \{1, 2, 3, 4, 5, 6\}$
  \item $\text{A} q \text{ U} r = \{2, 3, 4, 5, 6\}$
\end{itemize}
Remark: Equivalent formulae

- \( AX \varphi = \neg EX \neg \varphi \)
- \( \neg (\varphi U \psi) = G \neg \psi \lor (\neg \psi U (\neg \varphi \land \neg \psi)) \)
- \( A \varphi U \psi = \neg E \neg \psi \land E \neg \psi U (\neg \varphi \land \neg \psi) \)
- \( AG(req \rightarrow F grant) = AG(req \rightarrow AF grant) \)

- \( AFG \varphi = AGAF \varphi \)
- \( EG \varphi \neq EGF \varphi \)
- \( AF \varphi \neq AG \varphi \)
- \( EX \varphi \neq GX \varphi \)

\[ \varphi \]

Model checking of CTL

Definition: Existential and universal model checking

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

- \( M \models_{3} \varphi \) 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:

- \( M \models_{3} \varphi \) if \( I \cap \lbrack \varphi \rbrack \neq \emptyset \)
- \( M \models_{\forall} \varphi \) if \( I \subseteq \lbrack \varphi \rbrack \)
- \( M \models_{\exists} \neg \varphi \) if \( M \neq M \models_{3} \neg \varphi \)

Definition: Model checking problems \( MC_{CTL}^{\forall} \) and \( MC_{CTL}^{3} \)

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

Question: Does \( M \models_{\forall} \varphi \)? or Does \( M \models_{3} \varphi \)?

--

Theorem

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

Proof:

Compute \( \lbrack \varphi \rbrack \) by induction on the formula.

The set \( \lbrack \varphi \rbrack \) is represented by a boolean array.

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

\[
\begin{aligned}
case \varphi = p \in AP & \quad \lbrack \varphi \rbrack := \{ s \in S \mid p \in \ell(s) \} \quad O(|S|) \\
case \varphi = \neg \varphi_1 & \quad \lbrack \varphi \rbrack := S \setminus \lbrack \varphi_1 \rbrack \quad O(|S|) \\
case \varphi = \varphi_1 \lor \varphi_2 & \quad \text{semantics}(\varphi_1); \text{semantics}(\varphi_2) \\
& \quad \lbrack \varphi \rbrack := \lbrack \varphi_1 \rbrack \cup \lbrack \varphi_2 \rbrack \quad O(|S|) \\
\end{aligned}
\]
Model checking of CTL

Definition: procedure semantics(ψ)

\[
\begin{align*}
\text{case } & \varphi = E_{\varphi_1} \lor \varphi_2 & \mathcal{O}(|S| + |T|) \\
\text{semantics(φ_1); semantics(φ_2)} & L := \{φ_2\} & \text{// the set } L \text{ is the "todo" list} \\
\text{semantics(φ_2)} & Z := \emptyset & \text{// the set } Z \text{ is the "done" list} \\
\text{while } & L \neq \emptyset & \\
\text{Invariant: } & [φ_2] \cup (\{s\} \cap T^{-1}(Z)) \subseteq Z \cup L \subseteq [E \varphi_1 \lor \varphi_2] & \mathcal{O}(1) \\
\text{take } & t \in L; L := L \setminus \{t\}; Z := Z \cup \{t\} & \\
\text{for all } & s \in T^{-1}(t) & |T| \text{ times} \\
\text{if } & s \in [φ_2] \setminus (Z \cup L) \text{ then } L := L \cup \{s\} & \\
[φ] := Z & \text{// } Z \text{ is only used to make the invariant clear} & \\
\end{align*}
\]

Model checking of CTL

Definition: procedure semantics(ψ)

\[
\begin{align*}
\text{case } & \varphi = A_{\varphi_1} \lor \varphi_2 & \mathcal{O}(|S| + |T|) \\
\text{semantics(φ_1); semantics(φ_2)} & L := \{φ_2\} & \text{// the set } L \text{ is the "todo" list} \\
\text{semantics(φ_2)} & Z := \emptyset & \text{// the set } Z \text{ is the "done" list} \\
\text{while } & L \neq \emptyset & \\
\text{Invariant: } & \forall s \in S. c[s] = |T(s) \setminus Z| \text{ and} &
\{φ_2\} \cup (\{s\} \cap (s \in S \mid T(s) \subseteq Z)) \subseteq Z \cup L \subseteq [A \varphi_1 \lor \varphi_2] & \mathcal{O}(1) \\
\text{take } & t \in L; L := L \setminus \{t\}; Z := Z \cup \{t\} & \\
\text{for all } & s \in T^{-1}(t) & |T| \text{ times} \\
\text{c[s]} := c[s] & |T| \text{ times} \\
\text{if } & c[s] = 0 \land s \in [φ_2] \setminus (Z \cup L) \text{ then } L := L \cup \{s\} & \\
[φ] := Z & \text{// } Z \text{ is only used to make the invariant clear} & \\
\end{align*}
\]
Complexity of CTL

Definition: SAT(CTL)
Input: A formula $\phi \in \text{CTL}$
Question: Existence of a model $M$ and a state $s$ such that $M, s \models \phi$?

Theorem: Complexity
- The model checking problem for CTL is PTIME-complete.
- The satisfiability problem for CTL is EXPTIME-complete.

fair CTL

Definition: Syntax of fair-CTL
$\phi ::= \bot \mid p \ (p \in \text{AP}) \mid \neg \phi \mid \phi \lor \psi \mid E_f X \phi \mid A_f X \phi \mid E_f \phi U \psi \mid A_f \phi U \psi$

Definition: Semantics as a fragment of CTL*
Let $M = (S, T, I, \ell, F_1, \ldots, F_n)$ be a fair Kripke structure.

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

where $\text{fair} = \bigwedge_i GF F_i$

Lemma: CTL$_f$ cannot be expressed in CTL

fair CTL

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: $\bigwedge_i \neg GF \text{CS}_i$

Definition: Fair Kripke structure
$M = (S, T, I, \text{AP}, \ell, F_1, \ldots, F_n)$ with $F_i \subseteq S$.

An infinite run $\sigma$ is fair if it visits infinitely often each $F_i$

Proof: CTL$_f$ cannot be expressed in CTL
Consider the Kripke structure $M_k$ defined by:

- $M_k, 2k \models EGF p$ but $M_k, 2k - 2 \not\models EGF p$
- If $\phi \in \text{CTL}$ and $|\phi| \leq m \leq k$ then
  $M_k, 2k \models \phi$ iff $M_k, 2m \models \phi$
  $M_k, 2k - 1 \models \phi$ iff $M_k, 2m - 1 \models \phi$

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 Fair**
Let $\text{Fair} = \{s \in S \mid M, s \models E_f \top\}$
Then, Fair is the set of states that can reach $S'$.

**Proof: Computation of SCC**
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: Reductions
$E_f X \varphi = EX(\text{Fair} \land \varphi)$ and $E_f \varphi U \psi = E \varphi U (\text{Fair} \land \psi)$
It remains to deal with $A_f \varphi U \psi$.
Recall that $A_f \varphi U \psi = \neg EG \neg \psi \land \neg E \neg \psi U (\neg \varphi \land \neg \psi)$
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 $\llbracket \varphi \rrbracket$.
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 $M, s \models E \varphi U S'$ iff $M_{\varphi}, s \models EF S'$.
This is again a reachability problem which can be solved in linear time.