Initiation à la vérification
Basics of Verification


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

MPRI – M1
2020 – 2021
Outline

1. Introduction

Models

Temporal Specifications

Satisfiability and Model Checking

More on Temporal Specifications
Need for formal verification methods

Critical systems

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

...
Disastrous software bugs


Mariner 1 probe, 1962

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

- Destroyed 293 seconds after launch
- Missing hyphen in the data or program? No!
- Overbar missing missing in the mathematical specification:
  \( \bar{\dot{R}}_n \): \( n \)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


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

Spirit Rover (Mars Exploration), 2004


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

Other well-known bugs


Formal verifications methods

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

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

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

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

Prix Turing 2007.
Model Checking

3 steps

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

Main difficulties

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

Challenges

- Extend models and algorithms to cope with more systems. Infinite systems, parameterized systems, probabilistic systems, concurrent systems, timed systems, hybrid systems, ... See Modules 2.8 & 2.9
- Scale current tools to cope with real-size systems. Needs for modularity, abstractions, symmetries, ...
References

*Principles of Model Checking.*

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

*Model Checking.* 


*Temporal Verification of Reactive Systems: Safety.* 

*The Complexity of Temporal Logic Model Checking.* 
Outline

Introduction

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

Temporal Specifications

Satisfiability and Model Checking

More on Temporal Specifications
Example: Golden face

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

Example: Man, Wolf, Goat, Cabbage

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

**Definition: TS**

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

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

Every discrete system may be described with a TS.

**Example: Digicode ABA**
### Description Languages

**Pb:** How can we easily describe big systems?

#### Description Languages (high level)

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

#### Operational semantics

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

**Definition: TSV**

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

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

### Example: Vending machine

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

**Semantics: low level TS**

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

$$
\frac{s \xrightarrow{g,a,f} s' \land \nu \models g}{(s, \nu) \xrightarrow{a} (s', f(\nu))}
$$

**SOS: Structural Operational Semantics**

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

**Programs = Kripke structures with variables**

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

**Example: GCD**
Example: Digicode

```
cpt = 0
1
   A
   B
   C
   cpt < n
   B, C
   cpt++

2
   B
   cpt < n
   A
   cpt++

3
   A
   B
   cpt < n
   B, C
   cpt++

4
   OPEN

5
   ERROR
   B
   cpt < n
   C
   cpt++

6
   A
   B
   cpt < n
   A, C
   cpt++

7
   B
   cpt = n
   B, C
   cpt++

8
   C
   cpt = n
   B, C
   cpt++
```

```c
cpt = 0
for (int i = 0; i < n; i++) {
    if (Digicode[i] == 'A') {
        if (cptr < n && !Digicode[i + 1])
            cptr++; // Error case
        else
            cptr = n;
    } else if (Digicode[i] == 'B') {
        if (cptr < n && !Digicode[i + 1])
            cptr++; // Error case
        else
            cptr = n;
    } else if (Digicode[i] == 'C') {
        if (cptr < n && !Digicode[i + 1])
            cptr++; // Error case
        else
            cptr = n;
}
```
**Only variables**

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

**Definition: TSV**

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

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

**Symbolic representations with logic formulae**

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

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

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

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

**Semantics**

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

Atomic propositions are inherited from the local systems.

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

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

### Definition: General product

- **Components:** \( M_i = (S_i, \Sigma_i, T_i, I_i, AP_i, \ell_i) \)
- **Product:** \( M = (S, \Sigma, T, I, AP, \ell) \) with
  \[
  S = \prod_i S_i, \quad \Sigma = \prod_i (\Sigma_i \cup \{\epsilon\}), \quad \text{and} \quad I = \prod_i I_i
  \]
  \[
  T \text{ defined by } \forall i, \begin{array}{c}
  (p_i \xrightarrow{a_i} q_i) \in T_i \lor (a_i = \epsilon \land p_i = q_i) \\
  (p_1, \ldots, p_n) \xrightarrow{(a_1, \ldots, a_n)} (q_1, \ldots, q_n)
  \end{array}
  \]
- \( AP = \biguplus_i AP_i \) and \( \ell(p_1, \ldots, p_n) = \bigcup_i \ell(p_i) \)

### Synchronized products: restrictions of the general product.

**Parallel compositions:** 2 special cases

- **Synchronous:** \( \Sigma_{sync} = \prod_i \Sigma_i \)
- **Asynchronous:** \( \Sigma_{async} = \biguplus_i \Sigma_i' \) with \( \Sigma_i' = \{\epsilon\}^{i-1} \times \Sigma_i \times \{\epsilon\}^{n-i} \)

**Restrictions**

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

Definition: Asynchronous product + shared variables

\[ \bar{s} = (s_1, \ldots, s_n) \] denotes a tuple of states

\[ \nu \in D = \prod_{v \in V} D_v \] is a valuation of variables.

Semantics (SOS)

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

Example: Mutual exclusion for 2 processes satisfying

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

Using shared variables but without further restrictions: the atomicity is

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

Process $i$: // $i$ is not a variable
loop forever
  req[$i$] := true; turn := 1-$i$
  wait until (turn = $i$ or req[1-$i$] = false)
Critical section
  req[$i$] := false

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

Example:

Initially $x = 1 \land y = 2$

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

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

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

**Definition:** Atomic statements: `atomic(ES)`

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

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

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

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

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

**Example:** Atomic statements

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

Restriction on transitions is universal but too low-level.

**Definition: Rendez-vous**

- \(!m\) sending message \(m\)
- \(?m\) receiving message \(m\)
- **SOS: Structural Operational Semantics**

Local actions

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

Rendez-vous

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

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

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

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

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

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

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

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

- Domain: Let \( D_m \) be the domain for a single message.
  \[ D_c = D_{m}^{\leq k} \quad \text{size } k \]
  \[ D_c = D_{m}^{*} \quad \text{unbounded} \]
  \[ D_c = \{ \varepsilon \} \quad \text{Rendez-vous} \]

- Politics: FIFO, LIFO, BAG, . . .
Semantics: (lossy) FIFO

**Send**
\[
\frac{s_i \xrightarrow{c!e} s'_i \land \nu'(c) = \nu(e) \cdot \nu(c)}{(\bar{s}, \nu) \xrightarrow{c!e} (\bar{s}', \nu')}
\]

**Receive**
\[
\frac{s_i \xrightarrow{c?x} s'_i \land \nu(c) = \nu'(c) \cdot \nu'(x)}{(\bar{s}, \nu) \xrightarrow{c?e} (\bar{s}', \nu')}
\]

**Lossy send**
\[
\frac{s_i \xrightarrow{c!e} s'_i}{(\bar{s}, \nu) \xrightarrow{c!e} (\bar{s}', \nu)}
\]

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

Exercises:

1. Implement a FIFO channel using rendez-vous with an intermediary process.
2. Give the semantics of a LIFO channel.
3. Model the alternating bit protocol (ABP) using a lossy FIFO channel.
   
   *Fairness assumption:* For each channel, if infinitely many messages are sent, then infinitely many messages are delivered.
High-level descriptions

Summary

- Sequential program = transition system with variables
- Concurrent program with shared variables
- Concurrent program with Rendez-vous
- Concurrent program with FIFO communication
- Petri net
- ...

Models: expressivity versus decidability

Remark: (Un)decidability

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

Remark: Some infinite state models are decidable

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

Introduction

Models

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

Satisfiability and Model Checking

More on Temporal Specifications
Static and dynamic properties

Example: Static properties

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

Example: Dynamic properties

Every elevator request should be eventually granted.

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

Definition: Flows of time

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

Example: Flows of time

- \((\{0, \ldots, n\}, <)\): Finite runs of sequential systems.
- \((\mathbb{N}, <)\): Infinite runs of sequential systems.
- \((\mathbb{R}, <)\): Runs of real-time sequential systems.
- Trees: Finite or infinite run-trees of sequential systems.
- Mazurkiewicz traces: Runs of distributed systems (rendez-vous).
- Message sequence charts: Runs of distributed systems (FIFO).
- And also \((\mathbb{Z}, <)\) or \((\mathbb{Q}, <)\) or \((\omega^2, <)\), . . .

Definition: Temporal Structures

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

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

**Definition: Runs as temporal structures**

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

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

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

**Linear specifications** only depend on runs.

Example: The printer manager is starvation free.

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

**Remark:**

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

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

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

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

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

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

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

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

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

Definition: Syntax of FO(AP, <)
Let \( \text{Var} = \{x, y, \ldots\} \) be first-order variables.

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

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

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

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

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

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

First-order logic

- \( \text{FO}(<) \) has a good expressive power
  - ... but \( \text{FO}(<) \)-formulae are not easy to write and to understand.
- \( \text{FO}(<) \) is decidable
  - ... but satisfiability and model checking are non elementary.

Temporal logics

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

Linear Temporal Logic (LTL) over \((\mathbb{N}, <, \text{AP})\) introduced by Pnueli (1977) as a convenient specification language for verification of systems.
**Temporal Specifications**

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

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

**Definition: Semantics:** $w = (\mathbb{T}, <, \lambda)$ temporal structure and $i \in \mathbb{T}$

- $w, i \models p$ if $p \in \lambda(i)$
- $w, i \models \neg \varphi$ if $w, i \not\models \varphi$
- $w, i \models \varphi \lor \psi$ if $w, i \models \varphi$ or $w, i \models \psi$

- $w, i \models \varphi \text{SU} \psi$ if $\exists k \ i < k$ and $w, k \models \psi$ and $\forall j \ (i < j < k \rightarrow w, j \models \varphi)$

- $w, i \models \varphi \text{SS} \psi$ if $\exists k \ i > k$ and $w, k \models \psi$ and $\forall j \ (i > j > k \rightarrow w, j \models \varphi)$

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

**Theorem:** $\text{TL} \subseteq \text{FO}^3$

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

**Definition: non-strict versions of until and since**

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

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

\[
w, i \models \varphi S \psi \quad \text{if} \quad \exists k \ i \geq k \ \text{and} \ w, k \models \psi \ \text{and} \ \forall j \ (i \geq j > k \rightarrow w, j \models \varphi)
\]

**Definition: Derived modalities**

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

\[
w, i \models X \varphi \quad \text{if} \quad \exists k \ i < k \ \text{and} \ w, k \models \varphi \ \text{and} \ \neg \exists j \ (i < j < k)
\]

\[
w, i \models Y \varphi \quad \text{if} \quad \exists k \ i > k \ \text{and} \ w, k \models \varphi \ \text{and} \ \neg \exists j \ (i > j > k)
\]

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

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

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

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

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

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

- **Safety:** \(G\) good
- **MutEx:** \(\neg F(crit_1 \land crit_2)\)
- **Liveness:** \(GF\) active
- **Response:** \(G(request \rightarrow F\ grant)\)
- **Response’:** \(G(request \rightarrow (\neg request SU grant))\)
- **Release:** reset \(R\) alarm
- **Strong fairness:** \((GF\ request) \rightarrow (GF\ grant)\)
- **Weak fairness:** \((FG\ request) \rightarrow (GF grant)\)
- **Stability:** \(G \neg p \lor (\neg p U G\ p)\)
Discrete linear time flows

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

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

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

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

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

\[
\begin{align*}
\phi \mathbb{S} \psi &\equiv X(\phi \mathbb{U} \psi) & \neg X \phi &\equiv \neg X \mathbb{T} \lor X \neg \phi \\
\phi \mathbb{S} \mathbb{S} \psi &\equiv Y(\phi \mathbb{S} \psi) & \neg Y \phi &\equiv \neg Y \mathbb{T} \lor Y \neg \phi \\
\neg(\phi \mathbb{U} \psi) &\equiv (G \neg \psi) \lor (\neg \psi \mathbb{U} (\neg \phi \land \neg \psi)) & \equiv \neg \psi \mathbb{W} (\neg \phi \land \neg \psi) &\equiv \neg \phi \mathbb{R} \neg \psi
\end{align*}
\]

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

\(\neg(\phi \mathbb{U} \psi)\) does not imply \(\neg \phi \mathbb{R} \neg \psi\).

For instance, \(w = (\mathbb{T}, <, \ell)\) with \(\mathbb{T} = \{0\} \cup \left\{\frac{1}{n} \mid n \in \mathbb{N}\right\}\) with \(\ell(0) = \{p\}\), \(\ell(\frac{1}{2n}) = \{p\}\) and \(\ell(\frac{1}{2n+1}) = \{q\}\). Then, \(w, 0 \models \neg(p \mathbb{U} q)\) and \(w, 0 \not\models \neg p \mathbb{R} \neg q\).
Model checking for linear behaviors

Definition: Model checking problem

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

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

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

\[ M \models \forall \varphi \iff M \not\models \exists \neg \varphi \]

Theorem [11, Sistla, Clarke 85], [10, Lichtenstein & Pnueli 85]
The Model checking problem for LTL is PSPACE-complete.

Proof later
Weaknesses of linear behaviors

Example:

\[ \varphi: \text{Whenever } p \text{ holds, it is possible to reach a state where } q \text{ holds.} \]

\[ \varphi \text{ cannot be checked on linear behaviors.} \]

We need to consider the computation-trees.

Remark: FO definable on the computation tree

\[ \forall x \left( p(x) \rightarrow \exists y \left( x < y \land q(y) \right) \right) \]
Weaknesses of FO specifications

Example:

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

$$\psi$$ cannot be expressed in FO.

We need quantifications on runs: $$\psi = \text{EG}(\text{Active} \land \text{EF } \lnot\text{Active})$$

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

### Definition: Syntax of MSO(AP, <)

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

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

### Definition: Semantics of MSO(AP, <)

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

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

The semantics of first-order constructs is unchanged.

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

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

MSO logic

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

We need a temporal logic

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

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

$$\varphi ::= \bot | p \ (p \in \text{AP}) | \neg \varphi | \varphi \lor \varphi | \varphi \text{ SU } \varphi | E \varphi | A \varphi$$

We may also add the past modality SS. Two implicit free variables.

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

Let $M = (S, T, I, \text{AP}, \ell)$ be a Kripke structure (encodes the computation tree $T$). Let $\sigma = s_0s_1s_2 \cdots$ be an infinite run of $M$ (infinite branch of $T$). $i \in \mathbb{N}$ (current position in the run $\sigma$).

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

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

Remark:

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

- **EF** $\varphi$: $\varphi$ is possible (FO-definable on CT)
- **AG** $\varphi$: $\varphi$ is an invariant (FO-definable on CT)
- **AF** $\varphi$: $\varphi$ is unavoidable (not FO-definable on CT)
- **EG** $\varphi$: $\varphi$ holds globally along some path (not FO-definable on CT)

Remark: Some equivalences

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

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

For each $\varphi \in \text{CTL}^*(\text{AP}, \text{SU})$ we can construct an equivalent formula with two free variables $\tilde{\varphi}(X, x) \in \text{MSO}(\text{AP}, <)$. For all computation tree $T$, infinite branch $B$ of $T$ and position $i$ in $B$, we have $T, B, i \models \varphi$ iff $T, X \mapsto B, x \mapsto i \models \tilde{\varphi}$. 
**Definition: Existential and universal model checking**

Let $M = (S, T, I, AP, ℓ)$ be a Kripke structure and $ϕ ∈ CTL^*$ a formula.

$M |=_∃ ϕ$ if $M, σ, 0 |= ϕ$ for some initial infinite run $σ$ of $M$.

$M |=_∀ ϕ$ if $M, σ, 0 |= ϕ$ for all initial infinite runs $σ$ of $M$.

**Remark:** $M |=_∀ ϕ$ iff $M ⊭_∃ ¬ϕ$.

**Remark:** Often, formulas start with $E$ or $A$ and if $M$ has a single initial state, we do not need to distinguish between $|=_∃$ and $|=_∀$.

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

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

**Question:** Does $M |=_∀ ϕ$? or Does $M |=_∃ ϕ$?

**Theorem:**

The model checking problem for $CTL^*$ is PSPACE-complete. **Proof later**
**State formulae and path formulae**

**Definition: State formulae**

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

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

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

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

and

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

**Example: State formulae**

Atomic propositions are state formulae:

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

State formulae are closed under boolean connectives.

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

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

**Remark:**

\( M \models \exists \varphi \text{ iff } I \cap [E \varphi] \neq \emptyset \)
\( M \models \forall \varphi \text{ iff } I \subseteq [A \varphi] \)

**Definition: Alternative syntax**

State formulae

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

Path formulae

\( \psi ::= \varphi \mid \neg \psi \mid \psi \lor \psi \mid \psi \text{ SU } \psi \)
**Definition: Computation Tree Logic CTL(AP, X, U)**

Syntax:

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

The semantics is inherited from CTL*.

Remark: \( E \varphi \lor \varphi \) is not FO-definable on the computation tree.

Remark: All CTL formulae are state formulae

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

**Examples: Macros**

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

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

$$M, s \models p \quad \text{if} \quad p \in \ell(s)$$

$$M, s \models \text{EX} \varphi \quad \text{if} \quad \exists s \rightarrow s' \text{ with } M, s' \models \varphi$$

$$M, s \models \text{AX} \varphi \quad \text{if} \quad \forall s \rightarrow s' \text{ we have } M, s' \models \varphi$$

$$M, s \models \text{E} \varphi \mathbf{U} \psi \quad \text{if} \quad \exists s = s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \cdots s_k \text{ finite path, with}$$
$$\quad M, s_k \models \psi \text{ and } M, s_j \models \varphi \text{ for all } 0 \leq j < k$$

$$M, s \models \text{A} \varphi \mathbf{U} \psi \quad \text{if} \quad \forall s = s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \cdots \text{ infinite paths, } \exists k \geq 0 \text{ with}$$
$$\quad M, s_k \models \psi \text{ and } M, s_j \models \varphi \text{ for all } 0 \leq j < k$$

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

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

NB. Here models are computation trees.
Example:

\[
\begin{align*}
\text{CTL} & \quad \text{(Clarke & Emerson 81)} \\
\text{Example:} & \\
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
5 & 6 & 7 & 8 \\
q & p, q & q & r
\end{array} & \\
\begin{array}{cccc}
p, r & p, r & p, q & \\
5 & 6 & 7 & 8 \\
q & p, q & q & r
\end{array} & \\
\begin{array}{cccc}
\text{[EX } p\text{]} & = & \left\{ 1, 2, 3, 5, 6 \right\} & \\
\text{[AX } p\text{]} & = & \left\{ 3, 6 \right\} & \\
\text{[EF } p\text{]} & = & \left\{ 1, 2, 3, 4, 5, 6, 7, 8 \right\} & \\
\text{[AF } p\text{]} & = & \left\{ 2, 3, 4, 5, 6, 7 \right\} & \\
\text{[E } q \cup r\text{]} & = & \left\{ 1, 2, 3, 4, 5, 6 \right\} & \\
\text{[A } q \cup r\text{]} & = & \left\{ 2, 3, 4, 5 \right\} & \\
\end{array}
\end{align*}
\]
Remark: Equivalent formulae

- \( AX \varphi \equiv \neg EX \neg \varphi, \) assuming no deadlocks
- \( \neg(\varphi U \psi) \equiv G \neg \psi \lor (\neg \psi U (\neg \varphi \land \neg \psi)) \)
- \( A \varphi U \psi \equiv \neg EG \neg \psi \land \neg E(\neg \psi U (\neg \varphi \land \neg \psi)) \)
- \( AG(req \to F\ grant) \equiv AG(req \to AF\ grant) \)
- \( A GF \varphi \equiv AG AF \varphi \)
- \( EF G \varphi \equiv EF EG \varphi \)
- \( EG AF \varphi \implies EG F \varphi \implies EG EF \varphi \)
  but \( M_1 \vDash EG F \varphi, M_1 \nvDash EG AF \varphi \) and \( M_2 \vDash EG EF \varphi, M_2 \nvDash EG F \varphi. \)
- \( EG AF \varphi \neq E GF \varphi \neq EG EF \varphi \)
- \( AF EG \varphi \neq AF G \varphi \neq AF AG \varphi \)
- \( EG EX \varphi \neq EG X \varphi \neq EG AX \varphi \)
Model checking of **CTL**

**Definition: Existential and universal model checking**

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

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

**Remark:**

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

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

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

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

**Theorem:**

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

The model checking problem $M \models \exists \varphi$ is decidable in time $O(|M| \cdot |\varphi|)$.
References

Principles of Model Checking. 


Model Checking. 


Temporal Verification of Reactive Systems: Safety. 

The Complexity of Temporal Logic Model Checking. 
References

*Specification and Verification using Temporal Logics.*  
http://www.lsv.ens-cachan.fr/~gastin/mes-publis.php

*Temporal logic: mathematical foundations and computational aspects.*  

On the temporal analysis of fairness.  

Checking that finite state concurrent programs satisfy their linear specification.  

The complexity of propositional linear temporal logic.  
Outline

Introduction

Models

Temporal Specifications

Satisfiability and Model Checking

More on Temporal Specifications