Need for formal verifications methods

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

Disastrous software bugs

Mariner 1 probe, 1962
See http://en.wikipedia.org/wiki/Mariner_1
- Destroyed 293 seconds after launch
- Missing hyphen in the data or program? No!
- Overbar missing in the mathematical specification:
  \( \bar{R}_n \): \( n \)th \textit{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.

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.

Other well-known bugs


Disastrous software bugs

Formal verifications methods

Complementary approaches

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

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

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

Prix Turing 2007.

Model Checking

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

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

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

References


Model and abstractions

Example: Golden face

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

Model = Transition system

I States: configurations of the board: $2^9 = 512$ states
I Transitions: flipping a line/column/diagonal
I Problem: reachability

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

Model and Specification

Example: Men, Wolf, Goat, Cabbage

Model = Transition system

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

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, \([0..4], \mathbb{N}, \ldots\)
- Each variable \( v \in V \) has a domain \( D_v \) (finite or infinite).
- Guard or Condition \( g \) with semantics \([g] \subseteq D \) (unary predicate).
- Symbolic descriptions: \( x < 5, x + y = 10, \ldots \)
- Instruction or Update \( f \) with semantics \([f] : D \rightarrow D \).
- Symbolic descriptions: \( x := 0, x := (y + 1)^2, \ldots \)
- \( T \subseteq S \times (\text{Guard} \times \Sigma \times \text{Update}) \times S \)
  Symbolic descriptions:
  - \( x < 50, ?\text{coin}, x := \text{coin} + x \rightarrow s' \)
  - \( I \subseteq S \times \text{Guard} \)
  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

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

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

Definition: General product
- Components: \( M_i = (S_i, \Sigma_i, T_i, I_i, \text{AP}_i, \text{\`}_i) \)
- Product: \( M = (S, \Sigma, T, I, \text{AP}, \text{\`}) \) with
\[ S = \prod_i S_i, \quad \Sigma = \prod_i (\Sigma_i \cup \{\varepsilon\}), \quad \text{and} \quad I = \prod_i I_i \]
\[ T = \{(p_1, \ldots, p_n) \mid (a_1, \ldots, a_n) \in T_i \} \quad \text{for all} \quad i, \quad (p_i, a_i, q_i) \in T_i \quad \text{or} \quad a_i = \varepsilon \quad \text{and} \quad p_i = q_i \]
\[ \text{AP} = \bigcup_i \text{AP}_i \quad \text{and} \quad \text{\`}(p_1, \ldots, p_n) = \bigcup_i \text{\`}(p_i) \]

Synchronized products: restrictions of the general product.
Parallel compositions: 2 special cases
- Synchronous: \( \Sigma_{\text{sync}} = \prod_i \Sigma_i \)
- Asynchronous: \( \Sigma_{\text{async}} = \bigcup_i \Sigma_i \) with \( \Sigma_i = \{\varepsilon\}^{i-1} \times \Sigma_i \times \{\varepsilon\}^{n-i} \)

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

Shared variables

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)
\[ (\bar{s}, \nu) \xrightarrow{g} (\bar{s}', \nu') \quad \text{for} \quad j \neq i \]

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 \( i \): \quad // \( i \) is not a variable
loop forever
req[i] := true; turn := 1-i
wait until (turn = i or req[i-1] = 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{align*}
\text{Load}_{R_1}, x \\
\text{Add}_{R_1}, y \\
\text{Store}_{R_1}, x
\end{align*}
\parallel
\begin{align*}
\text{Load}_{R_2}, x \\
\text{Add}_{R_2}, y \\
\text{Store}_{R_2}, y
\end{align*}$

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

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
  
  \[
  \begin{align*}
  s_1 \xrightarrow{a_{s_1}} s'_1 \
  s_2 \xrightarrow{a_{s_2}} s'_2
  \end{align*}
  \]
- Rendez-vous
  
  \[
  \begin{align*}
  s_1 \xrightarrow{?m} s'_1 \land s_2 \xrightarrow{m} s'_2 \\
  \quad \text{or} \\
  s_1 \xleftarrow{?m} s'_1 \land s_2 \xleftarrow{m} s'_2
  \end{align*}
  \]
- 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

Atomicity

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

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

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

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.

\[
\begin{align*}
(s, \nu) \xrightarrow{ES} (s', \nu') \\
(s, \nu) \xrightarrow{\text{atomic}(ES)} (s', \nu')
\end{align*}
\]

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 $\in \{1, \ldots, n\}$.

send(id)

loop forever

receive(x)

if (x = id) then STOP fi

if (x > id) then send(x)
Channels

Definition: Channels

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

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

- Domain: Let \( D_m \) be the domain for a single message.
  - \( D_e = D_m^k \) size \( k \)
  - \( D_c = D_m^n \) unbounded
  - \( D_c = \{e\} \) Rendez-vous

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

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.
  - ...

Channels

Semantics: (lossy) FIFO

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

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

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

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.
**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 \((T, <)\) where \(T\) is the nonempty set of time points and \(<\) is an irreflexive transitive relation on \(T\).

**Example: Flows of time**

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

**Definition: Temporal Structures**

Let \(AP\) be a set of atoms (atomic propositions).

A temporal structure over a class \(C\) of time flows and \(AP\) is a triple \((T, <, h)\) where \((T, <)\) is a time flow in \(C\) and \(h: AP \to 2^T\) is an assignment.

If \(p \in AP\) then \(h(p) \subseteq T\) gives the time points where \(p\) holds.

---

**Linear behaviors and specifications**

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

**Definition: Runs as temporal structures**

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

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

\(\ell(\sigma) = \ell(s_0)\ell(s_1)\ell(s_2)\cdots = (\mathbb{N}, <, w)\) with \(w(i) = \ell(s_i) \in \Sigma\).

**Linear specifications** only depend on runs.

Example: The printer manager is fair.
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, but 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 $\ell(t) = (\text{dom}(t), <, h)$ where
- $<$ is the strict prefix relation over $D^*$,
- and $h(p) = \{u \in \text{dom}(t) \mid p \in \ell(t(u))\}$.

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

First-order vs Temporal

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

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

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

First-order Specifications

Definition: Syntax of FO(<)

Let $P, Q, \ldots$ be unary predicates twinned with atoms $p, q, \ldots$ in AP.
Let $\text{Var} = \{x, y, \ldots\}$ be first-order variables.

$$\varphi ::= \bot \mid P(x) \mid x = y \mid x < y \mid \neg \varphi \mid \varphi \lor \varphi \mid \exists \varphi$$

Definition: Semantics of FO(<)

Let $w = (\mathbb{T}, <, h)$ be a temporal structure.
Precidates $P, Q, \ldots$ twinned with $p, q, \ldots$ are interpreted as $h(p), h(q), \ldots$
Let $\nu : \text{Var} \rightarrow \mathbb{T}$ be an assignment of first-order variables to time points.

$$w, \nu \models P(x) \quad \text{if} \quad \nu(x) \in h(p)$$
$$w, \nu \models x = y \quad \text{if} \quad \nu(x) = \nu(y)$$
$$w, \nu \models x < y \quad \text{if} \quad \nu(x) < \nu(y)$$
$$w, \nu \models \exists x \varphi \quad \text{if} \quad w, \nu[x \mapsto t] \models \varphi \text{ for some } t \in \mathbb{T}$$

Previous specifications can be written in FO(<) (except the branching one).

Temporal Specifications

Definition: Syntax of TL(AP, SU, SS)

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

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

$$w, i \models p \quad \text{if} \quad i \in h(p)$$
$$w, i \models \neg \varphi \quad \text{if} \quad w, i \not\models \varphi$$
$$w, i \models \varphi \lor \psi \quad \text{if} \quad w, i \models \varphi \text{ or } w, i \models \psi$$
$$w, i \models \varphi \text{ SU } \psi \quad \text{if} \quad \exists k \ i < k \text{ and } w, k \models \psi \text{ and } \forall j (i < j < k \rightarrow w, j \models \varphi)$$
$$w, i \models \varphi \text{ SS } \psi \quad \text{if} \quad \exists k \ i > k \text{ and } w, k \models \psi \text{ and } \forall j (i > j > k \rightarrow w, j \models \varphi)$$

Previous specifications can be written in TL(AP, SU, SS) (except the branching one).
Temporal Specifications

Definition: non-strict versions of until and since
\[ \varphi U \psi \overset{\text{def}}{=} \psi \lor (\varphi \land \varphi SU \psi) \]
\[ \varphi S \psi \overset{\text{def}}{=} \psi \lor (\varphi \land \varphi SS \psi) \]
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
\[ X \varphi \overset{\text{def}}{=} \bot \land \varphi \quad \text{Next} \]
\[ Y \varphi \overset{\text{def}}{=} \bot \land SS \varphi \quad \text{Yesterday} \]
w, i \models X \varphi \quad \text{if} \quad \exists k \ i < k \text{ and } w, k \models \varphi \text{ and } \forall 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 } \forall j \ (i > j > k)\\F \varphi \overset{\text{def}}{=} \top U \varphi \quad \text{F} \]
\[ G \varphi \overset{\text{def}}{=} \neg F \neg \varphi \quad \text{G} \]
\[ P \varphi \overset{\text{def}}{=} T S \varphi \quad \text{P} \]
\[ H \varphi \overset{\text{def}}{=} \neg P \neg \varphi \quad \text{H} \]
\[ \varphi W \psi \overset{\text{def}}{=} (G \varphi) \lor (\varphi U \psi) \quad \text{Weak Until} \]
\[ \varphi R \psi \overset{\text{def}}{=} (G \psi) \lor (\psi U (\varphi \land \psi)) \quad \text{Release} \]

Temporal Specifications

Example: Specifications on the time flow \((\mathbb{N}, <)\)
- Safety: \(G \text{good} \)
- MutEx: \(\neg F(\text{crit}_1 \land \text{crit}_2) \)
- Liveness: \(GF \text{active} \)
- Response: \(G(\text{request} \rightarrow F \text{grant}) \)
- Response': \(G(\text{request} \rightarrow (\neg \text{request} SU \text{grant})) \)
- Release: \(\text{reset} R \text{ alarm} \)
- Strong fairness: \((GF \text{request}) \rightarrow (GF \text{grant}) \)
- Weak fairness: \((FG \text{request}) \rightarrow (GF \text{grant}) \)

Model checking for linear behaviors

Definition: Model checking problem
Input: A Kripke structure \(M = (S, T, I, AP, \ell)\)
A formula \(\varphi \in \text{LTL}(AP, SU, SS)\)
Question: Does \(M \models \varphi\)?
- Universal MC: \(M \models \varphi\) 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 \(M\).

\[ M \models \varphi \quad \text{iff} \quad M \not\models \neg \varphi \]

Theorem [10, Sistla, Clarke 85], [9, Lichtenstein & Pnueli 85]
The Model checking problem for LTL is PSPACE-complete.
Weaknesses of linear behaviors

Example:
\( \varphi: \) Whenever \( p \) holds, it is possible to reach a state where \( q \) holds.
\( \varphi \) cannot be checked on linear runs.
We need to consider the computation-trees.

Weaknesses of FO specifications

Example:
\( \psi: \) The system has an infinite active run, but it may always reach an inactive state.
\( \psi \) cannot be expressed in FO.

We need quantifications on runs:
\[ \psi \equiv \text{EG}(\text{Active} \land \text{EF} \neg \text{Active}) \]

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

MSO Specifications

Definition: Syntax of MSO\( (\langle \rangle) \)
Let \( P, Q, \ldots \) be unary predicates twinned with atoms \( p, q, \ldots \) in AP.
\[ \varphi ::= \bot \mid P(x) \mid x = y \mid x < y \mid x \in X \mid \neg \varphi \mid \varphi \lor \varphi \mid \exists x \varphi \mid \exists X \varphi \]
where \( x, y \) are first-order variables and \( X \) is a second-order variable.

Definition: Semantics of MSO\( (\langle \rangle) \)
Let \( w = (T, <, h) \) be a temporal structure.
An assignment \( \nu \) maps first-order variables to time points in \( T \)
and second-order variables to sets of time points.
The semantics of first-order constructs is unchanged.
\[ w, \nu \models x \in X \quad \text{if} \quad \nu(x) \in \nu(X) \]
\[ w, \nu \models \exists X \varphi \quad \text{if} \quad w, \nu[X \mapsto T] \models \varphi \quad \text{for some} \quad T \subseteq T \]
where \( \nu[X \mapsto T] \) maps \( X \) to \( T \) and keeps unchanged the other assignments.

MSO vs Temporal

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

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

Computation Tree Logic CTL\( ^* \) introduced by Emerson & Halpern (1986).
**Path formulae**

Definition: Syntax of the Computation Tree Logic $\text{CTL}^*$

$\phi := \bot \mid p \ (p \in \text{AP}) \mid \neg \phi \mid \phi \lor \overline{\phi} \mid \phi \text{ SU } \psi \mid \phi \text{ EU } \psi \mid \phi \text{ AU } \psi$

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

**Definition: Semantics of $\text{CTL}^*$**

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

Let $\sigma = s_0 s_1 s_2 \cdots$ be an infinite run of $M$.

$M, \sigma, i \models \phi$ if $p \in \ell(s_i)$

$M, \sigma, i \models \phi \text{ SU } \psi$ if $\exists k > i$, $M, \sigma, k \models \psi$ and $\forall i < j < k$, $M, \sigma, j \models \phi$

$M, \sigma, i \models \phi$ if $M, \sigma', i \models \phi$ for some infinite run $\sigma'$ such that $\sigma'[i] = \sigma[i]$

$M, \sigma, i \models \phi$ if $M, \sigma', i \models \phi$ for all infinite runs $\sigma'$ such that $\sigma'[i] = \sigma[i]$

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

**Remark:**

- $\phi \equiv \neg \psi$
- $\phi[i] = \sigma[i]$ means that future is branching but past is not.

---

**State formulae and path formulae**

Definition: State formulae

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

$M, \sigma, i \models \phi \iff M, \sigma', j \models \phi$

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

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

Example: State formulae

Atomic propositions are state formulae:

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

State formulae are closed under boolean connectives.

$[\neg \phi] = S \setminus [\phi]$  
$[\phi_1 \lor \phi_2] = [\phi_1] \cup [\phi_2]$

Formulae of the form $\phi_1 \lor \phi_2$ are state formulae, provided $\phi$ is future.

Definition: Alternative syntax

State formulae

$\phi := \bot \mid p \ (p \in \text{AP}) \mid \neg \phi \mid \phi \lor \overline{\phi} \mid \phi \text{ SU } \psi \mid \phi \text{ EU } \psi \mid \phi \text{ AU } \psi$

Path formulae

$\psi := \phi \mid \neg \psi \mid \psi \lor \psi \mid \psi \text{ SU } \psi$

---

**Model checking of $\text{CTL}^*$**

Definition: Existential and universal model checking

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

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

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

**Remark:**

$M \models \phi$ iff $I \cap [\phi] \neq \emptyset$

$M \models \psi$ iff $I \subseteq [\psi]$

$M \models \psi$ iff $M \not\models \neg $ 

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

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

Question: Does $M \models \phi$? or Does $M \models \phi$?

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

Proof later
Definition: Computation Tree Logic (CTL)

Syntax:
\[ \varphi ::= \bot \mid p \mid \neg \varphi \mid \varphi \lor \varphi \mid \text{EX} \varphi \mid \text{AX} \varphi \mid \text{E} \varphi \mid \text{A} \varphi \mid \text{E} \varphi \mid \text{A} \varphi \mid \text{E} \varphi \mid \text{A} \varphi \]

The semantics is inherited from CTL*. 

Remark: All CTL formulae are state formulae. Hence, we have a simpler semantics.

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

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

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

Example:

\[ [\text{EX} p] = \]
\[ [\text{AX} p] = \]
\[ [\text{EF} p] = \]
\[ [\text{AF} p] = \]
\[ [\text{E} q \mid \text{r}] = \]
\[ [\text{A} q \mid \text{r}] = \]
**Model checking of CTL**

**Definition: Existential and universal model checking**

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

$M \models_\exists \varphi$ if $M, s \models \varphi$ 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 $I \subseteq \llbracket \varphi \rrbracket$

$M \nvdash \forall \varphi$ iff $M \models \exists \neg \varphi$

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

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

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

**Theorem:**

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

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

---

**References**

*Principles of Model Checking.*  

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

*Model Checking.*  


*Temporal Verification of Reactive Systems: Safety.*  

*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

*Temporal logic: mathematical foundations and computational aspects.*  

On the temporal analysis of fairness.  
ACM Press.

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
    - CTL
    - Fair CTL
    - Büchi automata
    - From LTL to BA
    - LTL
    - CTL*
  - More on Temporal Specifications
**Model checking of CTL**

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

**Proof:**
Compute $[\varphi] = \{s \in S \mid M, s \models \varphi\}$ by induction on the formula.
The set $[\varphi]$ is represented by a boolean array: $L[s][\varphi]$ is $\top$ if $s \in [\varphi]$.
The labelling $\ell$ is encoded in $L$: for $p \in AP$ we have $L[s][p] = \top$ if $p \in \ell(s)$.
For each $t \in S$, the set $T^{-1}(t)$ is represented as a list.

For all $t \in S$ do for all $s \in T^{-1}(t)$ do ... od takes time $O(|T|)$.

---

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

<table>
<thead>
<tr>
<th>Case</th>
<th>Description</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varphi = \neg \varphi_1$</td>
<td>semantics($\varphi_1$)</td>
<td>$O(</td>
</tr>
<tr>
<td>$\varphi = \varphi_1 \lor \varphi_2$</td>
<td>semantics($\varphi_1$); semantics($\varphi_2$)</td>
<td>$O(</td>
</tr>
<tr>
<td>$\varphi = E \varphi_1$</td>
<td>semantics($\varphi_1$)</td>
<td>$O(</td>
</tr>
<tr>
<td>$\varphi = A \varphi_1$</td>
<td>semantics($\varphi_1$)</td>
<td>$O(</td>
</tr>
</tbody>
</table>

**Invariant:**
- $[\varphi_2] \subseteq Z \subseteq [E \varphi_1 \lor \varphi_2]$ and $[\varphi_1] \cap T^{-1}(Z \setminus L) \subseteq Z$
- $[\varphi_1] \cap T^{-1}(Z \setminus L) \subseteq Z$

**Z is only used to make the invariant clear. It can be replaced by $[\varphi]$.
Complexity of CTL

**Definition:** SAT(CTL)

**Input:** A formula $\varphi \in CTL$

**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.

---

**fair CTL**

**Definition:** Syntax of fair-CTL

$\varphi ::= \bot \mid p \ (p \in AP) \mid \neg \varphi \mid \varphi \vee \varphi \mid E_f \varphi \mid A_f \varphi \mid E_f \varphi \ U \varphi \mid A_f \varphi \ U \varphi$

**Definition:** Semantics as a fragment ofCTL$^*$

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

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

where

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

**Lemma:** CTL$^f$ cannot be expressed in CTL

**fair CTL**

**Proof:** CTL$^f$ cannot be expressed in CTL

Consider the Kripke structure $M_k$ defined by:

- $M_k, 2k \models EF p$ but $M_k, 2k - 2 \not\models EF p$
- If $\varphi \in CTL$ and $|\varphi| \leq m \leq k$ then $M_k, 2k \models \varphi$ iff $M_k, 2m \models \varphi$
- $M_k, 2k - 1 \models \varphi$ iff $M_k, 2m - 1 \models \varphi$

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

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

**Proof:** Computation of $\text{Fair} = \{ s \in S \mid M, s \models E_f T \}$
Compute the SCC of $M$ with Tarjan’s algorithm (in time $O(|M|)$).
Let $S'$ be the union of the (non trivial) SCCs which intersect each $F_i$.
Then, $\text{Fair}$ is the set of states that can reach $S'$.
Note that reachability can be computed in linear time.

---

Some References

Checking that finite state concurrent programs satisfy their linear specification.
In ACM Symposium PoPL’85, 97–107.

The tableau method for temporal logic: An overview,

The complexity of propositional linear temporal logic.

Fast LTL to Büchi automata translation.

*Specification and Verification using Temporal Logics.*
In Modern applications of automata theory, IISc Research Monographs 2.

---

Büchi automata

**Definition:**
A Büchi automaton (BA) is a tuple $A = (Q, \Sigma, I, T, F)$ where
- $Q$: finite set of states
- $\Sigma$: finite set of labels
- $I \subseteq Q$: set of initial states
- $T \subseteq Q \times \Sigma \times Q$: set of transitions (non-deterministic)
- $F \subseteq Q$: set of accepting (repeated, final) states

**Run:** $\rho = a_0, a_0, q_1, a_1, q_2, a_2, q_3, \ldots$ with $(q_i, a_i, q_{i+1}) \in T$ for all $i \geq 0$.
$\rho$ is **accepting** if $q_0 \in I$ and $q_i \in F$ for infinitely many $i$’s.

$$L(A) = \{ a_0a_1a_2 \cdots \in \Sigma^* \mid \exists \rho = a_0, a_0, q_1, a_1, q_2, a_2, q_3, \ldots \text{ accepting run} \}$$

An language $L \subseteq \Sigma^*$ is $\omega$-regular if it can be accepted by some Büchi automaton.
Examples:

Infinitely many a’s:

Finitely many a’s:

Whenever a then later b:

Properties

Büchi automata are closed under union, intersection, complement.

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

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

Any non-deterministic Büchi automaton for $\Sigma^* \setminus L$ has at least $2^n$ states.

Exercises:

1. Construct a BA for $L(\varphi)$ where $\varphi$ is the FO$_{\Sigma^2}(\prec)$ sentence
   
   $$(\forall x, (P_a(x) \rightarrow \exists y > x, P_a(y))) \rightarrow (\forall x, (P_b(x) \rightarrow \exists y > x, P_b(y)))$$

2. Given BA for $L_1 \subseteq \Sigma^*$ and $L_2 \subseteq \Sigma^*$, construct BA for
   
   $\text{next}(L_1) = \Sigma : L_1$
   
   $\text{until}(L_1, L_2) = \{uv \in \Sigma^* | u \in \Sigma^+ \land v \in L_2 \land u''v \in L_1 \text{ for all } u', u'' \in \Sigma^+ \text{ with } u = u'u''\}$

Definition: acceptance on states or on transitions

$A = (Q, \Sigma, I, T, F_1, \ldots, F_n)$ with $F_1 \subseteq Q$.

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

$A = (Q, \Sigma, I, T, T_1, \ldots, T_n)$ with $T_i \subseteq T$.

An infinite run $\sigma$ is successful if it uses infinitely many transitions from each $T_i$.

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

Theorem:

1. GBA and BA have the same expressive power.
2. Checking whether a BA or GBA has an accepting run is NLOGSPACE-complete.
Büchi automata with output

Definition: SBT: Synchronous (letter to letter) Büchi transducer

Let $A$ and $B$ be two alphabets.
A synchronous Büchi transducer from $A$ to $B$ is a tuple $A = (Q, A, I, T, F, \mu)$ where $(Q, A, I, T, F)$ is a Büchi automaton (input) and $\mu : T \to B$ is the output function.

It computes the relation

$$[A] = \{(u, v) \in A^* \times B^* \mid \exists \rho = q_0, a_0, q_1, a_1, q_2, a_2, q_3, \ldots \text{ accepting run with } u = a_0a_1a_2 \cdots \text{ and } v = \mu(q_0, a_0, q_1)\mu(q_1, a_1, q_2)\mu(q_2, a_2, q_3) \cdots \}$$

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

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

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

![Diagram of left shift]

Composition of Büchi transducers

Definition: Composition

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

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

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

iff

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

is an SGBT from $A$ to $C$.

When the transducers define functions, we also denote the composition by $A' \circ A$.

Proposition: Composition

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

Product of Büchi transducers

Definition: Product

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

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

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

iff

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

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

Proposition: Product

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

1. We have $[A \times A'] = \{(u, v, v') \mid (u, v) \in [A] \text{ and } (u, v') \in [A']\}$.
2. If $(Q, A, I, T, (F_i)_i)$ and $(Q', A', I', T', (F'_j)_j)$ are unambiguous then $(Q \times Q', A \times I', T', (F_i \times Q'_j)_i, (Q \times F'_j)_j)$ is also unambiguous, and, $\forall u \in A^*$ we have $[A \times A'](u) = ([A](u), [A']([u]))$.

Subalphabets of $\Sigma = 2^{AP}$

Definition: For a propositional formula $\xi$ over $AP$, we let $\Sigma_\xi = \{a \in \Sigma \mid a \models \xi\}$.

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

Notation:

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

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

Example: $G(p \rightarrow F q)$
Semantics of LTL with sequential functions

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

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

Define $\llbracket \varphi \rrbracket : \Sigma^* \rightarrow B^*$ by $\llbracket \varphi \rrbracket (u) = b_0 b_1 b_2 \cdots$ with $b_i = \begin{cases} 1 & \text{if } u, i \models \varphi \\ 0 & \text{otherwise} \end{cases}$.

Example:

$\llbracket [p \text{ SU } q] \rrbracket (\text{true}; \{q\}; \{p\}; \{p, q\}; \{p\}; \{p, q\}; \{p\}; \{p, q\}) = 1001110110$

$\llbracket [X p] \rrbracket (\text{true}; \{q\}; \{p\}; \{p\}; \{p, q\}; \{p\}; \{p, q\}) = 0101100110$

$\llbracket [F p] \rrbracket (\text{true}; \{q\}; \{p\}; \{p\}; \{p, q\}; \{p\}; \{p, q\}) = 1111111110$

The aim is to compute $\llbracket \varphi \rrbracket$ with Büchi transducers.

Synchronous Büchi transducer for $p \text{ SU } q$

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

Lemma: The input BA is unambiguous (prophetic)

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

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

Hence, the SBT computes $\llbracket p \text{ SU } q \rrbracket$.

Special cases of Until: Future and Next

Example: $F q = \top \text{ SU } q$ and $X q = \bot \text{ SU } q$

Exercise: Give SBT’s for the following formulae:

$p \text{ U } q$, $F q$, $G q$, $G q$, $p \text{ R } q$, $p \text{ R } q$, $p \text{ SS } q$, $p \text{ S } q$, $G(p \rightarrow F q)$.

From LTL to Büchi automata

Definition: SBT for LTL modalities

$A_T$ from $\Sigma$ to $B = \{0, 1\}$:

$A_p$ from $\Sigma$ to $B = \{0, 1\}$:

$A_\neg$ from $B$ to $B$:

$A_V$ from $B^2$ to $B$:

$A_A$ from $B^2$ to $B$:  

$p \text{ U } q$, $F q$, $G q$, $G q$, $p \text{ R } q$, $p \text{ R } q$, $p \text{ SS } q$, $p \text{ S } q$, $G(p \rightarrow F q)$. 
**From LTL to Büchi automata**

**Definition: SBT for LTL modalities (cont.)**

1. \(A_{SU}\) from \(\mathbb{B}^2\) to \(\mathbb{B}\):
   - Unambiguous
   - Prophetic
   - Deterministic

![Diagram](image)

**Useful simplifications**

Reducing the number of temporal subformulae

\[(X \varphi) \land (X \psi) \equiv X(\varphi \land \psi)\]
\[(X \varphi) \land (X \psi) \equiv X(\varphi \land \psi)\]
\[(G \varphi) \land (G \psi) \equiv G(\varphi \land \psi)\]
\[(G \varphi) \land (G \psi) \equiv G(\varphi \land \psi)\]
\[(\varphi_1 \land \varphi_2) \land (\varphi_3 \land \varphi_4) \equiv (\varphi_1 \land \varphi_2) \land (\varphi_3 \land \varphi_4)\]
\[(\varphi_1 \land \varphi_2) \land (\varphi_3 \land \varphi_4) \equiv (\varphi_1 \land \varphi_2) \land (\varphi_3 \land \varphi_4)\]

Merging equivalent states

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

We can merge \(s_1\) and \(s_2\) if they have the same outgoing transitions:

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

and \(s_1, a, s) \in T_i \iff (s_2, a, s) \in T_i\) for all \(1 \leq i \leq n\).

**From LTL to Büchi automata**

**Definition: Translation from LTL to SGBT**

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

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

**Theorem: Correctness of the translation**

For each \(\xi \in \text{LTL}(\text{AP}, \text{SU}, \text{SS})\), we have \([A_\xi] = [\xi]\) and \(A_\xi\) is unambiguous.

Moreover, the number of states of \(A_\xi\) is at most \(2^{|\xi|_{SS}} \cdot 3^{|\xi|_{SU}}\).

The number of acceptance conditions is \(|\xi|_{SS}\).

where \(|\xi|_{SS}\) (resp. \(|\xi|_{SU}\)) is the number of SS (resp. SU) occurring in \(\xi\).

**Remark:**

- If a subformula \(\varphi\) occurs serveral time in \(\xi\), we only need one copy of \(A_\varphi\).
- We may also use automata for other modalities: \(A_X, A_U, \ldots\)

**Other constructions**

- Tableau construction. See for instance [15, Wolper 85]
  - + : Easy definition, easy proof of correctness
  - - : Inefficient without strong optimizations
- Using **Very Weak Alternating Automata** [16, Gastin & Oddoux 01].
  - + : Very efficient
  - - : Only for future modalities
  - Online tool: [http://www.lsv.ens-cachan.fr/~gastin/ltl2ba/](http://www.lsv.ens-cachan.fr/~gastin/ltl2ba/)
- Using reduction rules [6, Demri & Gastin 10].
  - + : Efficient and produces small automata
  - + : Can be used by hand on real examples
  - - : Only for future modalities
- The domain is still very active.
**Satisfiability for LTL over \((\mathbb{N}, <)\)**

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

**Definition:** Satisfiability problem

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

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

**Definition:** Initial Satisfiability problem

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

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

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

**Definition:** (Initial) validity

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

**Theorem** [10, Sistla, Clarke 85], [9, Lichtenstein & Pnueli 85]

The satisfiability problem for LTL is PSPACE-complete.

---

**Model checking for LTL**

**Definition:** Model checking problem

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

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

**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 \(M\).

**Theorem** [10, Sistla, Clarke 85], [9, Lichtenstein & Pnueli 85]

The Model checking problem for LTL is PSPACE-complete.

---

**QBF Quantified Boolean Formulae**

**Definition:** QBF

**Input:** A formula \(\gamma = Q_1x_1 \cdots Q_nx_n \gamma'\) with \(\gamma' = \bigwedge_{1 \leq i \leq m} \bigvee_{1 \leq j \leq k_i} a_{ij}\)

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

**Question:** Is \(\gamma\) valid?

**Definition:**

An assignment of the variables \(\{x_1, \ldots, x_n\}\) is a word \(v = v_1 \cdots v_n \in \{0,1\}^n\).

We write \(v[i]\) for the prefix of length \(i\).

Let \(V \subseteq \{0,1\}^n\) be a set of assignments.

- \(V\) is valid (for \(\gamma'\)) if \(v \models \gamma'\) for all \(v \in V\).
- \(V\) is closed (for \(\gamma\)) if \(\forall v \in V, \forall i \leq i \leq n\) s.t. \(Q_i = \forall\),

\[
\exists v' \in V \text{ s.t. } v[i - 1] = v'[i - 1] \text{ and } v'_i = 1 - v_i.
\]

**Proposition:**

\(\gamma\) is valid if \(\exists V \subseteq \{0,1\}^n\) s.t. \(V\) is nonempty valid and closed.
QBF $\leq_P \text{MC}^3(U)$ [10, Sistla & Clarke 85]
Let $\gamma = Q_1 x_1 \cdots Q_n x_n \bigwedge_{1 \leq i \leq m} \bigvee_{1 \leq j \leq k_i} a_{ij}$ with $Q_i \in \{\forall, \exists\}$ and $a_{ij}$ literals.
Consider the KS $M$: 

Let $e_i \to s_i$ if $a_{ij} = x_k$ and $\psi_i = \bigwedge_{i \neq j} \psi_{ij}$.
Let $\varphi_i = G(e_{i+1} \to (\neg a_{i+1} \cup x_i) \wedge (\neg a_{i+1} \cup x_i))$ and $\varphi = \bigwedge_{i > 0} \varphi_i$.
Then, $\gamma$ is valid iff $M \models \psi \land \varphi$.

Complexity of LTL

Theorem: Complexity of LTL
The following problems are PSPACE-complete:
- SAT(LTL(SU, SS)), MC$^3$(LTL(SU, SS)), MC$^3$(LTL(SU, SS))
- SAT(LTL(X, F)), MC$^3$(LTL(X, F)), MC$^3$(LTL(X, F))
- SAT(LTL(U)), MC$^3$(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))

Proof:
For $\psi \in \text{LTL}$, let $\text{MC}_\text{LTL}^3(M, t, \psi)$ be the function which computes in polynomial space whether $M, t \models \psi$, i.e., if $M, t \models E \psi$.
Let $M = (S, T, I, \text{AP}, t)$ be a Kripke structure, $s \in S$ and $\varphi \in \text{CTL}^*$.
Replacing $A \psi$ by $\neg E \neg \psi$ we assume $\varphi$ only contains the existential path quantifier.
$\text{MC}_\text{CTL}^3(M, s, \varphi)$

if $E$ does not occur in $\varphi$ then return $\text{MC}_\text{LTL}^3(M, s, \varphi)$
Let $E \psi$ be a subformula of $\varphi$ with $\psi \in \text{LTL}$
Let $e_\psi$ be a new atomic proposition
Define $\ell' : S \to 2^{\text{AP}'}$ with $\text{AP}' = \text{AP} \cup \{e_\psi\}$ by
$\ell'(t) \cap \text{AP} = \ell(t)$ and $e_\psi \in \ell'(t)$ if $\text{MC}^3_{\text{LTL}}(M, t, \psi)$ (iff $M, t \models E \psi$)
Let $M' = (S, T, I, \text{AP}', \ell')$
Let $\varphi' = \varphi[e_\psi / E \psi]$ be obtained from $\varphi$ by replacing each $E \psi$ by $e_\psi$
Return $\text{MC}^3_{\text{CTL}^*}(M', s, \varphi')$
**Satisfiability for CTL\(^*\)**

**Definition:** SAT(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 CTL\(^*\) is 2-EXPTIME-complete

---

**Expressivity**

**Definition:** Equivalence

Let \( C \) be a class of time flows.

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

\[
  w, t \models \varphi \iff w, t \models \psi
\]

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

\[
  w, t \models \varphi \iff w, x \mapsto t \models \psi
\]

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

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

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

---

**Outline**

Introduction

Models

Temporal Specifications

Satisfiability and Model Checking

---

**Expressivity**

Definition: complete linear time flows

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

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

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

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

**Theorem:** Expressive completeness [11, Kamp 68]

For complete linear time flows,

\( \text{TL}(\text{AP}, \text{SU}, \text{SS}) = \text{FO}_{\text{AP}}(<) \)

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

See also Diekert-Gastin [17]: \( \text{TL} = \text{FO} = \text{SF} = \text{AP} = \text{CFBA} = \text{VWAA} \).

**Example:**

\[
  \psi(x) = \neg P_a(x) \wedge \neg P_b(x) \wedge \forall y \forall z \left( P_a(y) \wedge P_b(z) \wedge y < z \rightarrow \right.
  
  \left. \exists v y < v < z \wedge \left( P_a(v) \wedge x < y \right) \lor P_b(v) \wedge z < x \lor P_a(v) \wedge y < x < z \right)
\]
**Stavi connectives: Time flows with gaps**

**Definition: Stavi Until:** \( \mathcal{U} \)

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

\[
\exists k \ i < k \\
\land \exists j (i < j < k \land \forall \ell (i < \ell < j \rightarrow \ell \models \varphi)) \\
\land \forall j [i < j < k \rightarrow \left( \exists k' (j < k' \land \forall j' (i < j' < k' \rightarrow \ell \models \varphi)) \lor \exists \ell (i < \ell < j \land \ell \models \neg \varphi) \right)]
\]

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

**Example:**

Let \( w = (\mathbb{R} \setminus \{0\}, <, h) \) with \( h(p) = \mathbb{R}_- \) and \( h(q) = \mathbb{R}_+ \).

Then, \( w, -1 \models p \mathcal{U} q \) but \( w, -1 \not\models p \mathcal{U} q \).

**Theorem:** [13, Gabbay, Hodkinson, Reynolds]

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

**Temporal depth**

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

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

**Lemma:**

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

**Exercise: Isolated gaps**

Let \( \varphi_p = p \mathcal{SU} p \land SF \neg p \land \neg (p \mathcal{SU} \neg p) \land \neg (p \mathcal{SU} \neg (p \mathcal{SU} \neg p)) \).

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

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

Let \( \psi_{p,q} = \varphi_p \land (q \mathcal{V} \varphi_p) \mathcal{SU} (q \mathcal{V} \neg p) \).

Show that \( \psi_{p,q} \) is equivalent to \( p \mathcal{U} q \) over the time flow \( (\mathbb{R} \setminus \{0\}, <) \).

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

**k-equivalence**

**Definition:**

Let \( w_0 = (T_0, <, h_0) \) and \( w_1 = (T_1, <, h_1) \) be two temporal structures.

Let \( i_0 \in T_0 \) and \( i_1 \in T_1 \). Let \( k \in \mathbb{N} \).

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

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

**Example:**

Let \( a = \{p\} \) and \( b = \{q\} \). Let \( w_0 = \text{baabaababa}a \) and \( w_1 = \text{baababa}a \).  

\[
\begin{align*}
(w_0, 3) \equiv_1 (w_1, 4) \\
(w_0, 3) \equiv_1 (w_1, 4) \\
(w_0, 3) \equiv_1 (w_1, 6)
\end{align*}
\]

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

The EF-game has two players: **Spoiler (Player I)** and **Duplicator (Player II)**. The game board consists of 2 temporal structures:

- \( w_0 = (T_0, \prec, h_0) \) and \( w_1 = (T_1, \prec, h_1) \).

There are two tokens, one on each structure: \( i_0 \in T_0 \) and \( i_1 \in T_1 \).

**Configuration** is a tuple \((i_0, i_1)\) or simply \((i_0, i_1)\) if the game board is understood. Let \( k\in \mathbb{N} \).

The \( k \)-round EF-game from a configuration proceeds with (at most) \( k \) moves. There are 2 available moves for TL(AP, SU, SS): **SU-move or SS-move** (see below).

**Spoiler** chooses which move is played in each round.

- **Spoiler wins** if
  - Either duplicator cannot answer during a move (see below).
  - Or a configuration such that \((w_0, i_0) \not\sim_0 (w_1, i_1)\) is reached.

Otherwise, duplicator wins.

**Definition: Winning strategy**

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

**Example:**

Let \( a = \{p\}, b = \{q\}, c = \{r\} \). Let \( w_0 = aaaaabb \) and \( w_1 = aaababc \).

- \((w_0, 0) \sim (w_1, 0)\)
- \((w_0, 0) \not\sim_2 (w_1, 0)\)

Here, \( T_0 = T_1 = \{0, 1, 2, \ldots, 5\} \).

---

**Strict Until and Since moves**

**Definition: SU-move**

- **Spoiler** chooses \( e \in \{0, 1\} \) and \( k_e \in \mathbb{T}_{k_e} \) such that \( i_{k_e} < k_{k_e} \).
- **Duplicator** chooses \( k_{1-k} \in \mathbb{T}_{1-k} \) such that \( i_{1-k} < k_{1-k} \).

**Spoiler wins** if there is no such \( k_{1-k} \).

- Either spoiler chooses \((k_0, k_1)\) as next configuration of the EF-game, or the move continues as follows.
- **Spoiler** chooses \( j_{1-k} \in \mathbb{T}_{1-k} \) with \( i_{1-k} < j_{1-k} < k_{1-k} \).
- **Duplicator** chooses \( j_e \in \mathbb{T}_{e} \) with \( i_k < j_k < k_e \).

**Spoiler wins** if there is no such \( j_k \).

The next configuration is \((j_0, j_1)\).

Similar definition for the SS-move.

---

**EF-games for TL(AP, SU, SS)**

**Lemma: Determinacy**

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

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

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

\[(w_0, i_0) \sim_k (w_1, i_1) \iff (w_0, i_0) \equiv_k (w_1, i_1)\]

**Example:**

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

Then, \( aaaaabc, 0 = p \ SU (q \ SU r) \) but \( aaaaabc, 0 \not\models p \ SU (q \ SU r) \).

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

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

**Exercise:**

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

Definition: SF-move
- Spoiler chooses \( \varepsilon \in \{0, 1\} \) and \( j_\varepsilon \in T_\varepsilon \) such that \( i_\varepsilon < j_\varepsilon \).
- Duplicator chooses \( j_{1-\varepsilon} \in T_{1-\varepsilon} \) such that \( i_{1-\varepsilon} < j_{1-\varepsilon} \).
  
  Spoiler wins if there is no such \( j_{1-\varepsilon} \).
  
  The new configuration is \((j_0, j_1)\).

Similar definition for the SP-move.

Example:
\( p \) SU \( q \) is not expressible in \( TL(AP, SP, SF) \) over linear flows of time.
Let \( a = \emptyset, b = \{p\} \) and \( c = \{q\} \).
Let \( w_0 = (abc)^n a (abc)^n \) and \( w_1 = (abc)^n (abc)^n \).
If \( n > k \) then, starting from \((w_0, 3n, w_1, 3n)\), duplicator has a winning strategy in the \( k \)-round EF-game using SF-moves and SP-moves.

Non-strict Until and Since moves

Definition: U-move
- Spoiler chooses \( \varepsilon \in \{0, 1\} \) and \( k_\varepsilon \in T_\varepsilon \) such that \( i_\varepsilon \leq k_\varepsilon \).
- Duplicator chooses \( k_{1-\varepsilon} \in T_{1-\varepsilon} \) such that \( i_{1-\varepsilon} \leq k_{1-\varepsilon} \).
  
  Either spoiler chooses \((k_0, k_1)\) as new configuration of the EF-game, or the move continues as follows
  
  - Spoiler chooses \( j_{1-\varepsilon} \in T_{1-\varepsilon} \) with \( i_{1-\varepsilon} \leq j_{1-\varepsilon} < k_{1-\varepsilon} \).
  - Duplicator chooses \( j_\varepsilon \in T_\varepsilon \) with \( i_\varepsilon \leq j_\varepsilon < k_\varepsilon \).
  
  Spoiler wins if there is no such \( j_\varepsilon \).
  
  The new configuration is \((j_0, j_1)\).

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

Similar definition for the S-move.

Exercise:
1. Show that SU is not expressible in TL(AP, S, U) over \((R, <)\).
2. Show that SU is not expressible in TL(AP, S, U) over \((N, <)\).

Moves for Next and Yesterday modalities

Notation: \( i \preceq j \iff i < j \land \exists k (i < k < j) \).

Definition: X-move
- Spoiler chooses \( \varepsilon \in \{0, 1\} \) and \( j_\varepsilon \in T_\varepsilon \) such that \( i_\varepsilon \leq j_\varepsilon \).
- Duplicator chooses \( j_{1-\varepsilon} \in T_{1-\varepsilon} \) such that \( i_{1-\varepsilon} \leq j_{1-\varepsilon} \).
  
  Spoiler wins if there is no such \( j_{1-\varepsilon} \).
  
  The new configuration is \((j_0, j_1)\).

Similar definition for the Y-move.

Exercise:
Show that \( p \) SU \( q \) is not expressible in \( TL(AP, Y, SP, X, SF) \) over linear time flows.

Semantic Separation

Definition:
Let \( w = (T, <, h) \) and \( w' = (T, <, h') \) be temporal structures over the same time flow, and let \( t \in T \) be a time point.
- \( w, w' \) agree on \( t \) if \( \ell(t) = \ell'(t) \)
- \( w, w' \) agree on the past of \( t \) if \( \ell(s) = \ell'(s) \) for all \( s < t \)
- \( w, w' \) agree on the future of \( t \) if \( \ell(s) = \ell'(s) \) for all \( s > t \)

Recall: \( h: AP \rightarrow 2^T \) and we let \( \ell(t) = \{p \in AP \mid t \in h(p)\} \).

Definition: Pure formulae and separation
Let \( C \) be a class of time flows. A formula \( \varphi \) over some logic \( \mathcal{L} \) is pure past (resp. pure present, pure future) over \( C \) if
\[
 w, t \models \varphi \iff w', t \models \varphi
\]
for all temporal structures \( w = (T, <, h) \) and \( w' = (T, <, h') \) over \( C \) and all time points \( t \in T \) such that
\[
 w, w' \text{ agree on the past of } t \text{ (resp. on } t, \text{ on the future of } t). \]

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

**Definition: Syntactically pure formulae and separation**

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

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

**Example:**
The formulae $\varphi_1 = \text{SF}(q \land \text{SP} \, p)$ and $\varphi_2 = \text{SF}(q \land \neg \text{SP} \, \neg p)$ are not separated but there are equivalent syntactically separated formulae.

**Remark: Syntax versus semantic**

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

---

**Separation**

**Theorem:** [8, Gabbay, Pnueli, Shelah & Stavi 80]

$\text{TL}(\text{AP}, \text{SU}, \text{SS})$ is syntactically separable over discrete and complete linear orders.

**Definition: Discrete linear order**

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

- $(\mathbb{N}, <)$ is the unique (up to isomorphism) discrete and complete linear order with a first point and no last point.
- $(\mathbb{Z}, <)$ is the unique (up to isomorphism) discrete and complete linear order with no first point and no last point.
- Any discrete and complete linear order is isomorphic to a sub-flow of $(\mathbb{Z}, <)$.

**Theorem:** Gabbay, Reynolds, see [7]

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

---

**Initial equivalence**

**Definition: Initial Equivalence**

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

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

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

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

**Corollary:** of the separation theorem

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

---

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

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

is initially equivalent to

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

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

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

Separation and Expressivity

Theorem: [12, Gabbay 89] (already stated by Gabbay in 81)

Let \( C \) be a class of linear time flows.
Let \( L \) be a temporal logic able to express SF and SP.

Then, \( L \) is separable over \( C \) iff it is expressively complete for \( \mathbf{FO}_{\mathbf{AP}}(<) \) over \( C \).

Exercise: Checking semantically pure

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

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

Question: Is the formula \( \varphi \) semantically pure future?

Some References

Tense Logic and the Theory of Linear Order.

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

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

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

Separation — Past, Present and Future.

Great survey on separation properties.

Some References

Temporal logic: mathematical foundations and computational aspects.

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

Finite automata, formal logic, and circuit complexity.

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