Initiation à la vérification
Basics of Verification

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

MPRI – M1
2011 – 2012

Outline

1 Introduction
2 Models
3 Specifications
4 Satisfiability and Model Checking for LTL
5 Branching Time Specifications

Need for formal verifications methods

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

Disastrous software bugs

Mariner 1 probe, 1962
See http://en.wikipedia.org/wiki/Mariner_1
- Destroyed 293 seconds after launch
- Missing hyphen in the data or program? No!
- Overbar missing in the mathematical specification:
  \( \bar{H}_n \): \( n \)th smoothed value of the time derivative of a radius.
  Without the smoothing function indicated by the bar, the program treated normal minor variations of velocity as if they were serious, causing spurious corrections that sent the rocket off course.
Disastrous software bugs

Ariane 5 flight 501, 1996
See http://en.wikipedia.org/wiki/Ariane_5_Flight_501
- Destroyed 37 seconds after launch (cost: 370 millions dollars).
- Data conversion from a 64-bit floating point to 16-bit signed integer value caused a hardware exception (arithmetic overflow).
- Efficiency considerations had led to the disabling of the software handler (in Ada code) for this error trap.
- The fault occurred in the inertial reference system of Ariane 5. The software from Ariane 4 was re-used for Ariane 5 without re-testing.
- On the basis of those calculations the main computer commanded the booster nozzles, and somewhat later the main engine nozzle also, to make a large correction for an attitude deviation that had not occurred.
- The error occurred in a realignment function which was not useful for Ariane 5.

Disastrous software bugs

Spirit Rover (Mars Exploration), 2004
- Ceased communicating on January 21.
- Flash memory management anomaly:
  - Too many files on the file system
  - Resumed to working condition on February 6.

Disastrous software bugs

Other well-known bugs
- Therac-25, at least 5 death by massive overdoses of radiation.
  Race condition in accessing shared resources.
- Electricity blackout, USA and Canada, 2003, 55 millions people.
  Race condition in accessing shared resources.
  Flaw in the division algorithm, discovered by Thomas Nicely.
  See http://en.wikipedia.org/wiki/Pentium_FDIV_bug
- Needham-Schroeder, authentication protocol based on symmetric encryption. Published in 1978 by Needham and Schroeder
  Proved correct by Burrows, Abadi and Needham in 1989
  Flaw found by Lowe in 1995 (man in the middle)
  Automatically proved incorrect in 1996.
  See http://en.wikipedia.org/wiki/Needham-Schroeder_protocol

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.

References

Bibliography

Principles of Model Checking.  

Ph. Schnoebelen.  

Model Checking.  


Temporal Verification of Reactive Systems: Safety.  
**Outline**

**Introduction**

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

**Specifications**

- Satisfiability and Model Checking for LTL
- Branching Time Specifications

---

**Transition system or Kripke structure**

**Definition:** TS 

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

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

Every discrete system may be described with a TS.

**Example:** Digicode ABA

---

**Constructing the model**

**Example:** Men, Wolf, Goat, Cabbage

![Men, Wolf, Goat, Cabbage](image)

- 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.
Outline

Introduction

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

Specifications

Satisfiability and Model Checking for LTL

Branching Time Specifications

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: $T' \subseteq (S \times D) \times \Sigma \times (S \times D)$

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

SOS: Structural Operational Semantics

- $AP'$: we may use atomic propositions in $AP$ or guards in $2^D$ such as $x > 0$.

Programs = Kripke structures with variables

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

Example: GCD

Example: Digicode

Transition systems with variables

Definition: TSV

$M = (S, \Sigma, \nu, (D_v)_{v \in \nu}, T, I, AP, \ell)$

- $\nu$: set of (typed) variables, e.g., boolean, [0..4], ...
- Each variable $v \in \nu$ has a domain $D_v$ (finite or infinite)
- Guard or Condition: unary predicate over $D = \prod_{v \in \nu} D_v$
  - Symbolic descriptions: $x < 5$, $x + y = 10$, ...
- Instruction or Update: map $f : D \rightarrow D$
  - Symbolic descriptions: $x := 0$, $x := (y + 1)^2$, ...
- $T \subseteq S \times (2^D \times \Sigma \times D^D) \times S$
  - Symbolic descriptions: $s \xrightarrow{x<50, \text{coin}, x = x+\text{coin}} s'$
- $I \subseteq S \times 2^D$
  - Symbolic descriptions: $(s_0, x = 0)$

Example: Vending machine

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

TS with variables ...
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 \( \psi(v) \)
- \( T \) given by a formula \( \varphi(v, v') \)
- \( v \): values before the transition
- \( v' \): 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**

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

Outline

**Introduction**

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

**Specifications**

- Satisfiability and Model Checking for LTL
- Branching Time Specifications

**Modular description of concurrent systems**

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

**Semantics**

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

**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: \( M_i = (S_i, \Sigma_i, T_i, I_i, AP_i, \ell_i) \)
- Product: \( M = (S, \Sigma, T, I, AP, \ell) \) with

\[
S = \prod_i S_i, \quad \Sigma = \prod_i (\Sigma_i \cup \{\varepsilon\}), \quad \text{and} \quad I = \prod_i I_i
\]

\[
T = \{(p_1, \ldots, p_n) \xrightarrow{(a_1, \ldots, a_n)} (q_1, \ldots, q_n) \mid \text{for all } i, (p_i, a_i, q_i) \in T_i \text{ or } p_i = q_i \text{ and } a_i = \varepsilon\}
\]

\[
AP = \bigcup_i AP_i \text{ and } \ell(p_1, \ldots, p_n) = \bigcup_i \ell(p_i)
\]

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

**Parallel compositions: 2 special cases**

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

**Synchronizations**

- By states: \( S_{\text{sync}} \subseteq S \)
- By labels: \( \Sigma_{\text{sync}} \subseteq \Sigma \)
- By transitions: \( T_{\text{sync}} \subseteq T \)
Outline

Introduction

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

Specifications

Satisfiability and Model Checking for LTL

Branching Time Specifications

Synchronization by Rendez-vous

Synchronization by transitions is universal but too low-level.

Definition: Rendez-vous
- \( !m \) sending message \( m \)
- \( ?m \) receiving message \( m \)

SOS: Structural Operational Semantics

Local actions

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

Rendez-vous

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

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

Example: Elevator with 1 cabin, 3 doors, 3 calling devices
- \( \text{up} \) is uncontrollable for the cabin
- \( \text{leave} \) is uncontrollable for door \( i \)
- \( \text{call}_0 \) is uncontrollable for the system

Shared variables

Definition: Asynchronous product + shared variables

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

\[
\nu \in D = \prod_{i \in V} D_i \text{ is a valuation of variables.}
\]

Semantics (SOS)

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

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

Example: Mutual exclusion for 2 processes satisfying

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

using shared variables but no synchronization mechanisms: the atomicity is

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

Peterson’s algorithm (1981)

Process \( i \):

\[
\begin{align*}
\text{loop forever} \\
\text{req}[i] := \text{true}; \text{turn} := 1 - i \\
\text{wait until (turn = i or req[1-i] = false) } \\
\text{Critical section} \\
\text{req}[i] := \text{false}
\end{align*}
\]

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 $P_2$: \(\left(\begin{array}{c}
\text{Load} R_1, x \\
\text{Add} R_1, y \\
\text{Store} R_1, x \\
\end{array}\right) \parallel \left(\begin{array}{c}
\text{Load} R_2, x \\
\text{Add} R_2, y \\
\text{Store} R_2, y \\
\end{array}\right)\)

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

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

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

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

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

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

Example: Atomic statements

atomic($x = 0; x := 1$) (Test and set)

atomic($y := y - 1; \text{await}(y = 0); y := 1$) is equivalent to await($y = 1$)

**Channels**

Example: Leader election

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

\[\begin{align*}
\text{send}(id) \\
\text{loop forever} \\
\quad \text{receive}(x) \\
\quad \text{if } (x = id) \text{ then STOP fi} \\
\quad \text{if } (x > id) \text{ then send}(x)
\end{align*}\]

**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:
  \[
  \begin{align*}
  \text{empty}(c) \\
  c! e & \quad \text{add the value of expression } e \text{ to channel } c \\
  c! x & \quad \text{read a value from } c \text{ and assign it to variable } x
  \end{align*}
  \]

- Domain: Let $D_m$ be the domain for a single message.
  \[
  D_e = D_m^k \quad \text{size } k \\
  D_e = D_m^* \quad \text{unbounded} \\
  D_e = \{\varepsilon\} \quad \text{Rendez-vous}
  \]

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

Semantics: (lossy) FIFO

Send
\[ s_i \xrightarrow{c_{le}} s_i' \land \nu'(c) = \nu(c) \cdot \nu' \]
\[ (s, \nu) \xrightarrow{c_{le}} (s', \nu') \]

Receive
\[ s_i \xrightarrow{c_{rx}} s_i' \land \nu(c) = \nu'(c) \cdot \nu'(x) \]
\[ (s, \nu) \xrightarrow{c_{rx}} (s', \nu') \]

Lossy send
\[ s_i \xrightarrow{c_{ls}} s_i' \]
\[ (s, \nu) \xrightarrow{c_{ls}} (s', \nu) \]

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.

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

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.
- ...

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
- ...

Outline

Introduction
Models
Specifications
Definitions
Expressivity
Separation
Ehrenfeucht-Fraïssé games

Satisfiability and Model Checking for LTL

Branching Time Specifications
Some References

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 logic: mathematical foundations and computational aspects.

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

Outline

Introduction

Models

Specifications
- Definitions
- Expressivity
- Separation
- Ehrenfeucht-Fraïssé games

Satisfiability and Model Checking for LTL

Branching Time Specifications

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.
- 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 \((\mathbb{R}, <)\), \((\omega^2, <)\), ...

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 \(h : AP \rightarrow 2^T\) is an assignment.
If \(p \in AP\) then \(h(p) \subseteq T\) gives the time points where \(p\) holds.
Let $M = (S, T, I, AP, \ell)$ be a Kripke structure.

**Definition: Runs as temporal structures**

An infinite run $\sigma = s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \cdots$ with $s_i \rightarrow s_{i+1} \in T$ of $M$ 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.

---

**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.

\[
\phi ::= \bot \mid P(x) \mid x = y \mid x < y \mid \neg \phi \mid \phi \lor \psi \mid \exists x \phi
\]

**Definition: Semantics of FO(<)**

Let $w = (T, <, h)$ be a temporal structure.

Precidates $P, Q, \ldots$ twinned with $p, q, \ldots$ are interpreted as $h(p), h(q), \ldots$.

Let $\nu : \text{Var} \rightarrow T$ be an assignment of first-order variables.

\[
\begin{align*}
\nu(x) &\models P(x) & \text{if } \nu(x) \in h(p) \\
\nu(x) &\models x = y & \text{if } \nu(x) = \nu(y) \\
\nu(x) &\models x < y & \text{if } \nu(x) < h(y) \\
\nu(x) &\models \exists x \phi & \text{if } \nu(x) \models \exists \phi \text{ for some } t \in T
\end{align*}
\]

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

Previous specifications can be written in FO(<).

---

**Branching behaviors and specifications**

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

**Definition: Run-trees as temporal structures**

**Run-tree** = unfolding of the transition system.

Let $D$ be a finite set with $|D|$ the outdegree of the transition relation $T$.

**Unordered tree** $t : D^* \rightarrow \Sigma$ (partial map).

Associated temporal structure $(\text{dom}(t), <, h)$ where $<$ is the strict prefix relation over $D^*$ and $h(p) = \{i \in \text{dom}(t) \mid p \in t(i)\}$.

Example: Each process has the possibility to print first.

---

**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.
Temporal Specifications

**Definition: Syntax of TL(AP, S, U)**

\[ \varphi ::= \bot | p (p \in AP) | \neg \varphi | \varphi \lor \varphi | \varphi \land \varphi | G \varphi | H \varphi | \varphi U \varphi | \varphi S \varphi | X \varphi | Y \varphi \]

**Definition: Semantics**

\[ w, i |= p \quad \text{if} \quad i \in \text{h}(p) \]

\[ w, i |= F \varphi \quad \text{if} \quad \exists j < i \text{ and } w, j |= \varphi \]

\[ w, i |= G \varphi \quad \text{if} \quad \forall j < i \rightarrow w, j |= \varphi \]

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

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

\[ w, i |= P \varphi \quad \text{if} \quad \exists j > i \text{ and } w, j |= \varphi \]

\[ w, i |= H \varphi \quad \text{if} \quad \forall j > i \rightarrow w, j |= \varphi \]

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

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

Previous specifications can be written in TL(AP).

Temporal Specifications

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

- **Safety:** G' good
- **MutEx:** \(\neg F'(\text{crit}_1 \land \text{crit}_2)\)
- **Liveness:** GF active
- **Response:** G'(request \(\rightarrow\) F grant)
- **Response':** G'(request \(\rightarrow\) (\(\neg\)request U grant))
- **Release:** reset R alarm
- **Strong fairness:** GF request \(\rightarrow\) GF grant
- **Weak fairness:** FG request \(\rightarrow\) GF grant

Outline

**Introduction**

**Models**

- **Specifications**
  - Definitions
  - Expressivity
  - Separation
  - Ehrenfeucht-Fraïssé games

**Satisfiability and Model Checking for LTL**

**Branching Time Specifications**
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.
In Temporal Logics in Specifications, April 87.

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

First-order definable languages.
In Logic and Automata: History and Perspectives, vol. 2, Texts in Logic and
Overview of formalisms expressively equivalent to First-Order for words.
http://www.lev.ens-cachan.fr/~gastin/mes-publis.php

Temporal Specifications

Proposition: For discrete linear time flows \((T, <)\)

\[
\begin{align*}
F\varphi &= X F' \varphi \\
G\varphi &= X G' \varphi \\
\varphi U \psi &= X (\varphi U' \psi) \\
\neg X \varphi &= X \neg \varphi \lor \neg X T \\
\neg (\varphi U \psi) &= (G \neg \psi) \lor (\neg \psi U (\neg \varphi \land \neg \psi)) \\
&= \neg \psi W (\neg \varphi \land \neg \psi) \\
&= \neg \varphi R \neg \psi
\end{align*}
\]

Definition: discrete linear time flows

A linear time flow \((T, <)\) is discrete if \(F T \rightarrow X T\) and \(P T \rightarrow Y T\) are valid formulae.

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

Remark:

Elegant algebraic proof of \(TL(AP, S, U) = FO_{AP}(<)\) over \((\mathbb{N}, <)\) due to Wilke 98.

Expressivity

Definition: Equivalence

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

Two formulae \(\varphi, \psi \in TL(AP, S, U)\) 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 \(\varphi \in TL(AP, S, U)\) and \(\psi(x) \in FO_{AP}(<)\) 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 \(w \models \psi(t)\).

Remark: \(LTL(AP, S, U) \subseteq FO_{AP}(<)\)

\(\forall \varphi \in TL(AP, S, U), \exists \psi(x) \in FO_{AP}(<)\) such that \(\varphi\) and \(\psi(x)\) are equivalent.

Theorem: Expressive completeness [6, Kamp 68]

For complete linear time flows,

\(TL(AP, S, U) = FO_{AP}(<)\)

Definition: complete linear time flows

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.

Remark:

Elegant algebraic proof of \(TL(AP, U) = FO_{AP}(<)\) over \((\mathbb{N}, <)\) due to Wilke 98.
**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

\[
\exists k \ i < k \ \\
\ \wedge \exists j \ (i < j < k \wedge w, j \models \neg \varphi) \\
\ \wedge \exists j \ (i < j < k \wedge \forall \ell (i < \ell < j \rightarrow w, \ell \models \varphi)) \\
\ \wedge \forall j \left( i < j < k \rightarrow \left[ \exists k' \ (j < k' \wedge \forall j' (i < j' < k' \rightarrow w, j' \models \varphi)) \right] \right)
\]

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

**Theorem:** [9, Gabbay, Hodkinson, Reynolds]

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

**Exercise: Isolated gaps**

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

**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 \( h(t) = h'(t) \)
- \( w, w' \) agree on the past of \( t \) if \( h(s) = h'(s) \) for all \( s < t \)
- \( w, w' \) agree on the future of \( t \) if \( h(s) = h'(s) \) for all \( s > t \)

**Definition: Pure formulae**

Let \( C \) be a class of time flows. A formula \( \varphi \) over some logic \( C \) is pure past (resp. pure present, pure future) over \( C \) if for all temporal structures \( w = (T, <, h) \) and \( w' = (T', <, h') \) over \( C \) and all time points \( t \in T \) such that \( w, w' \) agree on the past of \( t \) (resp. on \( t \), on the future of \( t \)) we have

\[
w, t \models \varphi \quad \text{iff} \quad w', t \models \varphi
\]
Separation

Definition: Separation

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

Theorem: [7, Gabbay, Pnueli, Shelah & Stavi 80]

TL(AP, S, U) is separable over discrete and complete linear orders.

$\neg (N, <)$ is the unique (up to isomorphism) discrete and complete linear order with a first point and no last point.

$\neg (Z, <)$ 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 $(Z, <)$.

Theorem: Gabbay, Reynolds, see [10]

TL(AP, S, U) is separable over $(\mathbb{R}, <)$.

Separation and Expressivity

Theorem: [8, 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 $F$ and $P$.

Then, $L$ is separable over $C$ iff it is expressively complete over $C$.

Initial equivalence

Definition: Initial Equivalence

Let $C$ be a class of time flows having a minimum (denoted 0). Two formulae $\varphi, \psi \in TL(AP, S, U)$ 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 $\varphi \in TL(AP, S, U)$ and $\psi(x) \in FO_{AP}(<)$ 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 $\varphi \in TL(AP, S, U)$ there exists $\psi \in TL(AP, U)$ such that $\varphi$ and $\psi$ are initially equivalent over $(N, <)$.

Initial equivalence

Example: TL(AP, S, U) versus TL(AP, U)

$$G'(\text{grant } \rightarrow \neg\text{grant } S \text{ request})$$

is initially equivalent to

$$(\text{request } R' \neg\text{grant}) \land G(\text{grant } \rightarrow (\text{request } \lor (\text{request } R \neg\text{grant})))$$

Theorem: (Laroussinie & Markey & Schnoebelen 2002)

TL(AP, S, U) may be exponentially more succinct than TL(AP, U) over $(N, <)$.
Outline

Introduction

Models

Specifications
- Definitions
- Expressivity
- Separation
  - Ehrenfeucht-Fraïssé games

Satisfiability and Model Checking for LTL

Branching Time Specifications

Some References

Finite automata, formal logic, and circuit complexity.

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

Temporal depth

Definition: Temporal depth of \( \varphi \in \text{TL}(\mathcal{AP}, \mathcal{S}, \mathcal{U}) \)

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

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

\( k \)-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 \( \text{TL}(\mathcal{AP}, \mathcal{S}, \mathcal{U}) \) of temporal depth at most \( k \).

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

Example:
Let \( a = \{p\} \) and \( b = \{q\} \). Let \( w_0 = bababaaba \) and \( w_1 = baabaabaab \).

\[
\begin{align*}
(w_0, 3) &\equiv_0 (w_1, 4) \\
(w_0, 3) &\equiv_1 (w_1, 4) \\
(w_0, 3) &\equiv_1 (w_1, 6)
\end{align*}
\]

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

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 $i_1 \in T_1$.
A configuration is a tuple $(w_0, i_0, w_1, i_1)$ 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, S, U): Until or Since (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 $(w_0, i_0) \not\equiv_0 (w_1, i_1)$ is reached.
Otherwise, duplicator wins.

Winning strategy

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 $(w_0, i_0) \sim_k (w_1, i_1)$.

Spoiler has a winning strategy in the $k$-round EF-game starting from $(w_0, i_0, w_1, i_1)$ if she can win all plays starting from this configuration.

Example:
Let $a = \{p\}$, $b = \{q\}$, $c = \{r\}$. Let $w_0 = aaabbc$ and $w_1 = aababc$.

$(w_0, 0) \sim_1 (w_1, 0)$
$(w_0, 0) \not\sim_2 (w_1, 0)$

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

Until and Since moves

Definition: (Strict) Until move
- Spoiler chooses $e \in \{0, 1\}$ and $k_e \in \mathbb{T}_e$ such that $i_k < k_e$.
- Duplicator chooses $k_{1-e} \in \mathbb{T}_{1-e}$ such that $i_{1-e} < k_{1-e}$.
- Spoiler wins if there is no such $k_{1-e}$.
  - Either spoiler chooses $(k_0, k_1)$ as next configuration of the EF-game, or the move continues as follows
  - Spoiler chooses $j_{1-e} \in \mathbb{T}_{1-e}$ with $i_{1-e} < j_{1-e} < k_{1-e}$.
  - Duplicator chooses $j_e \in \mathbb{T}_e$ with $i_e < j_e < k_e$.
  - Spoiler wins if there is no such $j_e$.
  - The next configuration is $(j_0, j_1)$.

Similar definition for the (strict) Since move.

EF-games for TL(AP, S, U)

Lemma: Determinacy
The $k$-round EF-game for TL(AP, S, U) 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, $aaabbc, 0 \models p \cup (q \cup r)$ but $aababc, 0 \not\models p \cup (q \cup r)$.
Hence, $p \cup (q \cup r)$ 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, S, U).
Moves for Future and Past modalities

**Definition:** (Strict) Future move

- Spoiler chooses \( e \in \{0, 1\} \) and \( j_e \in T_\varepsilon \) such that \( i_e < j_e \).
- Duplicator chooses \( j_{1-e} \in T_{1-\varepsilon} \) such that \( i_{1-e} < j_{1-e} \).

Spoiler wins if there is no such \( j_{1-e} \).

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

Similar definition for (strict) Past move.

**Example:**

\( p \cup q \) is not expressible in \( TL(AP, P, F) \) 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 Future and Past moves.

Non-strict Until and Since moves

**Definition:** non-strict Until move

- Spoiler chooses \( e \in \{0, 1\} \) and \( k_e \in T_\varepsilon \) such that \( i_e \leq k_e \).
- Duplicator chooses \( k_{1-e} \in T_{1-\varepsilon} \) such that \( i_{1-e} \leq k_{1-e} \).
  
  Either spoiler chooses \((k_0, k_1)\) as new configuration of the EF-game, or the move continues as follows.

- Spoiler chooses \( j_{1-e} \in T_{1-\varepsilon} \) with \( i_{1-e} \leq j_{1-e} < k_{1-e} \).
- Duplicator chooses \( j_e \in T_\varepsilon \) with \( i_e \leq j_e < k_e \).

Spoiler wins if there is no such \( j_e \).

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

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

Similar definition for the non-strict Since move.

**Exercise:**

1. Show that strict until is not expressible in \( TL(AP, S', U') \) over \((\mathbb{R}, \prec)\).
2. Show that strict until is not expressible in \( TL(AP, S', U') \) over \((\mathbb{N}, \prec)\).

Moves for Next and Yesterday modalities

**Notation:** \( i \prec j \equiv i < j \land \exists k (i < k < j) \).

**Definition:** Next move

- Spoiler chooses \( e \in \{0, 1\} \) and \( j_e \in T_\varepsilon \) such that \( i_e \prec j_e \).
- Duplicator chooses \( j_{1-e} \in T_{1-\varepsilon} \) such that \( i_{1-e} \prec j_{1-e} \).

Spoiler wins if there is no such \( j_{1-e} \).

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

Similar definition for Yesterday move.

**Exercise:**

Show that \( p \cup q \) is not expressible in \( TL(AP, Y, P, X, F) \) over linear flows of time.

Outline

**Introduction**

**Models**

**Specifications**

- Satisfiability and Model Checking for LTL
  - Büchi automata
    - From LTL to BA
  - Decidability and Complexity

**Branching Time Specifications**
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.
In CAV’01, vol. 2102, Lecture Notes in Computer Science, pp. 53–65.
http://www.lsv.ens-cachan.fr/~gastin/mes-publis.php

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

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: $\rho = q_0, a_0, q_1, a_1, q_2, a_2, q_3, \ldots$ 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.

$L(A) = \{a_0 a_1 a_2 \ldots \in \Sigma^\omega \mid \exists \rho = q_0, a_0, q_1, a_1, q_2, a_2, q_3, \ldots \text{ accepting run} \}$

A language $L \subseteq \Sigma^\omega$ is ω-regular if it can be accepted by some Büchi automaton.

Outline

Introduction
Models
Specifications

- Satisfiability and Model Checking for LTL
  - Büchi automata
    - From LTL to BA
      - Decidability and Complexity
Branching Time Specifications

Büchi automata

Examples:

Infinitely many $a$’s:

Finitely many $a$’s:

Whenever $a$ then later $b$: 
Büchi automata

Properties

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

Let \( L = \Sigma^*(a\Sigma^* b \cup b\Sigma^* a)\Sigma^* \).

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

Example: Left shift with a/b

Büchi automata

Theorem: Büchi

Let \( L \subseteq \Sigma^w \) be a language. The following are equivalent:

- \( L \) is \( \omega \)-regular
- \( L \) is \( \omega \)-rational, i.e., \( L \) is a finite union of languages of the form \( L_1 \cdot L_2^* \) where \( L_1, L_2 \subseteq \Sigma^+ \) are rational.
- \( L \) is MSO-definable, i.e., there is a sentence \( \varphi \in \text{MSO}_\Sigma(\leq_\Sigma(\prec)) \) such that \( L = \mathcal{L}(\varphi) = \{ w \in \Sigma^* \mid w \models \varphi \} \).

Exercises:

1. Construct a BA for \( \mathcal{L}(\varphi) \) where \( \varphi \) is the FO\( \Sigma(\prec) \) sentence
   \[
   (\forall x, (P_u(x) \rightarrow \exists y > x, P_u(y))) \rightarrow (\forall x, (P_u(x) \rightarrow \exists y > x, P_u(y)))
   \]

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

Generalized Büchi automata

Definition: acceptance on states or on transitions

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

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

\( A = (Q, I, T, T_1, \ldots, T_n) \) 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

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 \( \mu : T \rightarrow B \) is the output function.

It computes the relation

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

If \((Q,A,I,T,F)\) is unambiguous then \([A] : A^\omega \rightarrow B^\omega \) is a (partial) function.

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

Example: Left shift with \( A = B = \{ a, b \} \)
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 $A \cdot A' = (Q \times Q', A, I \times I', T''_i, (F_i \times Q'), (Q \times F'_j)_j, \mu'')$ is defined by:

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

iff

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

$A \cdot A'$ 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 $[A \cdot A'] = [A] \cdot [A']$.
2. If $(Q, A, I, T, (F_i)_i)$ and $(Q', B, I', T', (F'_j)_j)$ are unambiguous then $(Q \times Q', A, I \times I', T''_i, (F_i \times Q'), (Q \times F'_j)_j)$ is also unambiguous.

Then, $\forall u \in A^\omega$ we have $[A' \circ A](u) = [A']([A](u))$.

Outline

Introduction
Models
Specifications
- Satisfiability and Model Checking for LTL
  - Büchi automata
  - From LTL to BA
Decidability and Complexity
Branching Time Specifications

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

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

iff

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

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

Proposition: Product
We identify $(B \times C)^\omega$ with $B^\omega \times C^\omega$.
1. We have $[A \times A'] = \{(u, v, v') | (u, v) \in [A] \text{ and } (u, v') \in [A']\}$.
2. If $(Q, A, I, T, (F_i)_i)$ and $(Q', A', I', T', (F'_j)_j)$ are unambiguous then $(Q \times Q', A, I \times I', T''_i, (F_i \times Q'), (Q \times F'_j)_j)$ is also unambiguous.

Then, $\forall u \in A^\omega$ we have $[A \times A'](u) = ([A](u), [A'](u))$.

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

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

Notation:
In automata, $p \xrightarrow{\Sigma_{\xi}} q$ stands for the set of transitions $\{p\} \times \Sigma_{\xi} \times \{q\}$.
To simplify the pictures, we use $p \xrightarrow{\xi} q$ instead of $p \xrightarrow{\Sigma_{\xi}} q$.

Example:
Semantics of LTL with sequential functions

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

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

Define $[\varphi] : \Sigma^\omega \to \Sigma^\omega$ by $[\varphi](u) = b_0b_1b_2 \cdots$ with $b_i = \begin{cases} 1 & \text{if } u_i \models \varphi \\ 0 & \text{otherwise} \end{cases}$.

Example:

$[p \text{ U } q](\emptyset) = 1001110110^\omega$

$[Xp](\emptyset) = 0101100110^\omega$

$[Fp](\emptyset) = 111111110^\omega$

The aim is to compute $[\varphi]$ with Büchi transducers.

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

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

Lemma: The input BA is prophetic

For all $u = a_0a_1a_2 \cdots \in \Sigma^\omega$, there is a unique accepting run $\rho = q_0, a_0, q_1, a_1, q_2, a_2, q_3, \ldots$ of $A$ on $u$.

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

Special cases of Until: Future and Next

Example: $Fq = \top \text{ U } q$ and $Xq = \bot \text{ U } q$

Exercise: Give SBT’s for the following formulae:

$p U q, Fq, Gq, G^*q, p R q, p R^* q, p S q, p S^* q, G(p \to F q)$.

From LTL to Büchi automata

Definition: SBT for LTL modalities

- $A_{\top}$ from $\Sigma$ to $\Sigma = \{0, 1\}$:
- $A_{p}$ from $\Sigma$ to $\Sigma = \{0, 1\}$:
- $A_{\neg}$ from $B$ to $\Sigma$:
- $A_\vee$ from $B^2$ to $\Sigma$:
- $A_\wedge$ from $B^2$ to $\Sigma$:
From LTL to Büchi automata

Definition: SBT for LTL modalities (cont.)

- \( A_U \) from \( \mathbb{B}^2 \) to \( \mathbb{B} \):
  - 0, 1/1
  - 1, 1/1
  - 0, 0/1
  - 1, 0/1
  - 0, 1/0
  - 1, 1/0
  - 0, 0/0
  - 1, 0/0

- \( A_S \) from \( \mathbb{B}^2 \) to \( \mathbb{B} \):
  - 0, 0/0
  - 1, 0/0
  - 0, 0/1
  - 1, 1/1

Useful simplifications

Reducing the number of temporal subformulae

- \((X\varphi) \land (X\psi) \equiv X(\varphi \land \psi)\)
- \((G\varphi) \land (G\psi) \equiv G(\varphi \land \psi)\)
- \((\varphi_1 \land \varphi_2) U \psi \equiv (\varphi_1 \land \varphi_2) U \psi\)
- \((\varphi \psi_1) V (\varphi \psi_2) \equiv \varphi (\psi_1 V \psi_2)\)

Merging equivalent states

Let \( A = (Q, \Sigma, I, T_1, \ldots, T_n) \) 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 \( \xi \in \text{LTL}(\text{AP}, S, U) \), we define inductively an SGBT \( A_\xi \) as follows:

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

Theorem: Correctness of the translation

For each \( \xi \in \text{LTL}(\text{AP}, S, U) \), we have \( |A_\xi| = |\xi| \).
Moreover, the number of states of \( A_\xi \) is at most \( 2^{|\xi|} \cdot 3^{|\xi|} \)
where \(|\xi|_S\) (resp. \(|\xi|_U\)) is the number of \( S\) (resp. \( U\)) occurring in \( \xi \).

Remark:
- If a subformula \( \varphi \) occurs several 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 [13, Wolper 85]
  + : Easy definition, easy proof of correctness
  + : Works both for future and past modalities
  - : Inefficient without strong optimizations
- Using Very Weak Alternating Automata [15, Gastin & Oddoux 01].
  + : Very efficient
  - : Only for future modalities
  Online tool: http://www.lsv.ens-cachan.fr/~gastin/ltl2ba/
- Using reduction rules [16, 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.
**Outline**

- Introduction
- Models
- Specifications
  - Satisfiability and Model Checking for LTL
    - Büchi automata
    - From LTL to BA
  - Decidability and Complexity
- Branching Time Specifications

---

**Model checking for LTL**

**Definition**: Model checking problem

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

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

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

**Theorem** [14, Sistla, Clarke 85], [12, Lichtenstein & Pnueli 85]

The Model checking problem for LTL is PSPACE-complete.

---

**Satisfiability for LTL over \((\mathbb{N}, <)\)**

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

**Definition**: Satisfiability problem

- **Input**: A formula \( \varphi \in \text{LTL}(AP, S, U) \)
- **Question**: Existence of \( w \in \Sigma^\omega \) and \( i \in \mathbb{N} \) such that \( w, i \models \varphi \).

**Definition**: Initial Satisfiability problem

- **Input**: A formula \( \varphi \in \text{LTL}(AP, S, U) \)
- **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** [14, Sistla, Clarke 85], [12, Lichtenstein & Pnueli 85]

The satisfiability problem for LTL is PSPACE-complete.

---

**MC^2(U) \leq_P SAT(U)** [14, Sistla & Clarke 85]

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

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

Define \( AP^* = AP \cup AP_S \)

\( \Sigma' = 2^{AP^*} \)

\( \pi : \Sigma^\omega \to \Sigma' \) by \( \pi(a) = a \cap AP \).

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

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

\[ \psi_M = \left( \bigvee_{s \in S} at_s \right) \land G' \left( \bigvee_{s \in S} \bigwedge_{s \neq t} at_t \land \bigwedge_{p \in E(s)} p \land \bigwedge_{q \in E(t)} \neg p \land \bigvee_{t \in T(s)} X at_t \right) \]

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

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

\( \psi_M \models \varphi \) if \( \psi_M \land \neg \varphi \) is not satisfiable

**Remark**: we also have \( MC^2(X, F') \leq_P SAT(X, F') \).
The following problems are PSPACE-complete:

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

The following problems are NP-complete:

- SAT(LTL(F')), MC^\exists(LTL(F'))

\[ QBF \leq_P MC^\exists(U') \]  
Let \( \gamma = Q_1 x_1 \cdots Q_n x_n \bigwedge_{1 \leq m \leq n} \bigvee_{1 \leq j \leq k_i} a_{ij} \) with \( Q_i \in \{ \forall, \exists \} \) and \( a_{ij} \) literals.

Consider the KS \( M \):

\[ \begin{align*}
V_0 &\rightarrow V_1 \\
V_1 &\rightarrow V_2 \\
&\vdots \\
V_n &\rightarrow V_{n+1}
\end{align*} \]

Let \( \psi_{ij} = \left\{ \begin{array}{ll}
G'(x_k^j \rightarrow s_k R' \neg a_{ij}) & \text{if } a_{ij} = x_k \\
G'(x_k^j \rightarrow s_k R' \neg a_{ij}) & \text{if } a_{ij} = \neg x_k
\end{array} \right. \) and \( \psi = \bigwedge_{i,j} \psi_{ij} \).

Let \( \varphi_j = G'(s_{j-1} \rightarrow (\neg s_{j-1} U' x_j^j) \land (\neg s_{j-1} U' x_j^j)) \) and \( \varphi = \bigwedge_j \varphi_j \).

Then, \( \gamma \) is valid iff \( M \models_{\exists} \psi \land \varphi \).
Possibility is not expressible in LTL

Example:

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

We need quantifications on runs: 
\( \phi = AG(p \rightarrow EF q) \)
- E: for some infinite run
- A: for all infinite runs

Outline

Introduction
Models
Specifications

Satisfiability and Model Checking for LTL

- Branching Time Specifications
  - CTL*
    - CTL
    - Fair CTL

CTL* (Emerson & Halpern 86)

Definition: Syntax of the Computation Tree Logic CTL*

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

In this chapter, temporal modalities U, F, G, ... are non-strict.
We may also add past modalities Y and y.

Definition: Semantics of CTL*

Let \( M = (S, T, I, AP, \ell) \) be a Kripke structure.
Let \( \sigma = s_0 s_1 s_2 \ldots \) be an infinite run of \( M \).

\[
\begin{align*}
M, \sigma, i & \models E \phi \quad \text{if} \quad M, \sigma', i \models \phi \quad \text{for some infinite run } \sigma' \text{ such that } \sigma'[i] = \sigma[i] \\
M, \sigma, i & \models A \phi \quad \text{if} \quad M, \sigma', i \models \phi \quad \text{for all infinite runs } \sigma' \text{ such that } \sigma'[i] = \sigma[i]
\end{align*}
\]

where \( \sigma[i] = s_0 \ldots s_i \).

Remark:

- \( A \phi \equiv \neg E \neg \phi \)
- \( \sigma'[i] = \sigma[i] \) means that future is branching but past is not.

Example: Some specifications

- \( EF \phi: \phi \) is possible
- \( AG \phi: \phi \) is an invariant
- \( AF \phi: \phi \) is unavoidable
- \( EG \phi: \phi \) holds globally along some path
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

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

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.

Definition: Alternative syntax
State formulae
\[ \varphi ::= \bot \mid p \mid \neg \varphi \mid \varphi \lor \varphi \mid E \varphi \mid A \varphi \]
Path formulae
\[ \psi ::= \varphi \mid \neg \psi \mid \psi \lor \psi \mid X \psi \mid \psi U \psi \]

Complexity of \( \text{CTL}^* \)

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

Theorem
The model checking problem for \( \text{CTL}^* \) is PSPACE-complete

Proof:
PSPACE-hardness: follows from \( \text{LTL} \subseteq \text{CTL}^* \).
PSPACE-easiness: reduction to LTL-model checking by inductive eliminations of path quantifications.

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

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.

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

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

Definition: Model checking problems \( \text{MC}^\forall_{\text{CTL}^*} \) and \( \text{MC}^\exists_{\text{CTL}^*} \)
Input:
A Kripke structure \( M = (S, T, I, \text{AP}, \ell) \) and a formula \( \varphi \in \text{CTL}^* \)
Question:
Does \( M \models \exists \varphi \) ?

\( \text{MC}^\exists_{\text{CTL}^*} \) in PSPACE

Proof:
For \( \psi \in \text{LTL} \), let \( \text{MC}^\exists_{\text{LTL}}(M, t, \psi) \) be the function which computes in polynomial space whether \( M, t \models \psi \), i.e., if \( M, t \models E \psi \).

Let \( M = (S, T, I, \text{AP}, \ell) \) 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.

\( \text{MC}^\exists_{\text{CTL}^*}(M, s, \varphi) \)
If \( E \) does not occur in \( \varphi \) then return \( \text{MC}^\exists_{\text{LTL}}(M, s, \varphi) \)
Let \( E \psi \) be a subformula of \( \varphi \) with \( \psi \in \text{LTL} \)
Let \( e \psi \) be a new propositional variable
Define \( \ell' : S \to 2^{\text{AP}'} \) with \( \text{AP}' = \text{AP} \cup \{ e \psi \} \) by
\[ \ell'(t) \cap \text{AP} = \ell(t) \] and \( e \psi \in \ell'(t) \) iff \( \text{MC}^\exists_{\text{LTL}}(M, t, \psi) \)
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 \( \text{MC}^\exists_{\text{CTL}^*}(M', s, \varphi') \)
Satisfiability for $\text{CTL}^*$

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

**Input:** A formula $\varphi \in \text{CTL}^*$

**Question:** Existence of a model $M$ and a run $\sigma$ such that $M, \sigma, 0 \models \varphi$?

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

---

**Outline**

Introduction

Models

Specifications

Satisfiability and Model Checking for LTL

- Branching Time Specifications
  - $\text{CTL}^*$
  - $\text{CTL}$
  - Fair $\text{CTL}$

---

**CTL (Clarke & Emerson 81)**

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

**Syntax:**

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

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

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

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

**Examples:** Macros

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

---

**CTL (Clarke & Emerson 81)**

**Definition:** Semantics

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

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

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

- $M \models \exists \varphi$ if $I \cap \{s \mid M, s \models \varphi\} \neq \emptyset$.
- $M \models \forall \varphi$ if $I \subseteq \{s \mid M, s \models \varphi\}$.
- $M \models \forall \varphi$ if $M, s \not\models \exists \varphi$.

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

- **Input:** A Kripke structure $M = (S, T, I, AP, \ell)$ and a formula $\varphi \in \text{CTL}$
- **Question:** Does $M \models \forall \varphi$? or Does $M \models \exists \varphi$?

### CTL (Clarke & Emerson 81)

**Remark: Equivalent formulae**

- $AX \varphi = \neg EX \neg \varphi$.
- $\neg (\varphi U \psi) = G \neg \psi V (\neg \psi U (\neg \varphi \land \neg \psi))$.
- $A \varphi U \psi = \neg EG \neg \psi \land \neg E (\neg \psi U (\neg \varphi \land \neg \psi))$.
- $AG(req \rightarrow F \text{grant}) = AG(req \rightarrow AF \text{grant})$.

- $AG F \varphi = AG AF \varphi$.
- $EF G \varphi = EF EG \varphi$.
- $EG EF \varphi \neq EG F \varphi$.
- $AF AG \varphi \neq AF G \varphi$.
- $EG EX \varphi \neq EG G \varphi$.

### 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 $M \models \exists \varphi$ 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: $L[s][\varphi] = T$ if $s \in [\varphi]$.

The labelling $\ell$ is encoded in $L$: for $p \in AP$ we have $L[s][p] = T$ if $p \in \ell(s)$. 

### CTL (Clarke & Emerson 81)
Theorem: Complexity

- The model checking problem for CTL is PTIME-complete.
- The satisfiability problem for CTL is EXPTIME-complete.
Outline

Introduction
Models
Specifications
Satisfiability and Model Checking for LTL

Branching Time Specifications

CTL∗
CTL
Fair CTL

fair CTL

Definition: Syntax of fair-CTL

ϕ ::= ⊥ | p (p ∈ AP) | ¬ϕ | ϕ ∨ ϕ | E_fϕ | A_fϕ | E_f U ϕ | A_f U ϕ

Definition: Semantics as a fragment of CTL∗

Let M = (S, T, I, AP, ℓ, F₁, ..., F_n) be a fair Kripke structure.

Then, E_fϕ = E(fair ∧ ϕ) and A_fϕ = A(fair → ϕ)

where fair = \bigwedge_i GF_i

Lemma: CTL_f cannot be expressed in CTL

fair CTL

Example: Fairness

Only fair runs are of interest

Each process is enabled infinitely often: \( \bigwedge_i GF_{run_i} \)

No process stays ultimately in the critical section: \( \bigwedge_i \neg GFCS_i = \bigwedge_i GF \neg CS_i \)

Definition: Fair Kripke structure

M = (S, T, I, AP, ℓ, F₁, ..., F_n) with F_i ⊆ S.

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

Proof: CTL_f cannot be expressed in CTL

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

- \( M_k, 2k \models EG F p \) but \( M_k, 2k - 2 \not\models EG F p \)
- If \( ϕ \in CTL \) and \( |ϕ| \leq m \leq k \) then
  \( M_k, 2k \models ϕ \) iff \( M_k, 2m \models ϕ \)
  \( M_k, 2k - 1 \models ϕ \) iff \( M_k, 2m - 1 \models ϕ \)

If the fairness condition is \( ℓ^{-1}(p) \) then \( E_f T \) cannot be expressed in CTL.
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.

Proof: Reductions
$E_f X \varphi = E(X(\text{Fair} \land \varphi))$ and $E_f \varphi U \psi = E \varphi U (\text{Fair} \land \psi)$
It remains to deal with $A_f \varphi U \psi$.
We have $A_f \varphi U \psi = E_f \neg \psi \land A_f (\neg \psi U (\neg \varphi \land \neg \psi))$
Hence, we only need to compute the semantics of $E_f G \varphi$.

Proof: Computation of $E_f G \varphi$
Let $M_\varphi$ be the restriction of $M$ to $[\varphi]_f$.
Compute the SCC of $M_\varphi$ with Tarjan’s algorithm (in linear time).
Let $S'$ be the union of the (non trivial) SCCs of $M_\varphi$ which intersect each $F_i$.
Then, $M, s \models E_f G \varphi$ iff $M, s \models E \varphi U S'$ iff $M_\varphi, s \models E F S'$.
This is again a reachability problem which can be solved in linear time.