

Initiation à la vérification

Basics of Verification

<http://mpri.master.univ-paris7.fr/C-1-22.html>

Paul Gastin

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

MPRI – M1
2010 – 2011

Outline

- 1 Introduction
 - Bibliography

Models

Specifications

Linear Time Specifications

Branching Time Specifications

Need for formal verifications methods

Critical systems

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

Disastrous software bugs

Mariner 1 probe, 1962

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

- ▶ Destroyed 293 seconds after launch
- ▶ Missing hyphen in the data or program? **No!**
- ▶ Overbar missing in the mathematical specification:

\bar{R}_n : *n*th smoothed value of the time derivative of a radius.

Without the smoothing function indicated by the bar, the program treated normal minor variations of velocity as if they were serious, causing spurious corrections that sent the rocket off course.



Disastrous software bugs

Ariane 5 flight 501, 1996

See http://en.wikipedia.org/wiki/Ariane_5_Flight_501

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

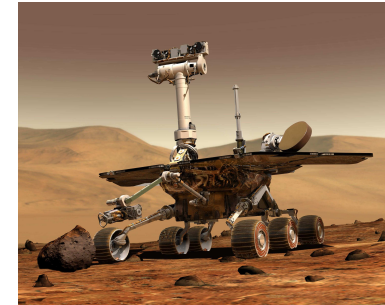


Disastrous software bugs

Spirit Rover (Mars Exploration), 2004

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

- ▶ Landed on January 4, 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 3 death by massive overdoses of radiation.
Race condition in accessing shared resources.
See <http://en.wikipedia.org/wiki/Therac-25>
- ▶ Electricity blackout, USA and Canada, 2003, 55 millions people.
Race condition in accessing shared resources.
See http://en.wikipedia.org/wiki/Northeast_Blackout_of_2003
- ▶ Pentium FDIV bug, 1994.
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

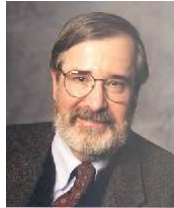
Formal verifications methods

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.

Model Checking

3 steps

- ▶ Constructing the model M (transition systems)
- ▶ Formalizing the specification φ (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, ...
- ▶ Scale current tools to cope with real-size systems. Needs for modularity, abstractions, symmetries, ...

References

Bibliography

- [1] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.
- [2] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, Ph. Schnoebelen. *Systems and Software Verification. Model-Checking Techniques and Tools*. Springer, 2001.
- [3] E.M. Clarke, O. Grumberg, D.A. Peled. *Model Checking*. MIT Press, 1999.
- [4] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer, 1991.
- [5] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer, 1995.

Outline

Introduction

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

Specifications

Linear Time Specifications

Branching Time Specifications

Constructing the model

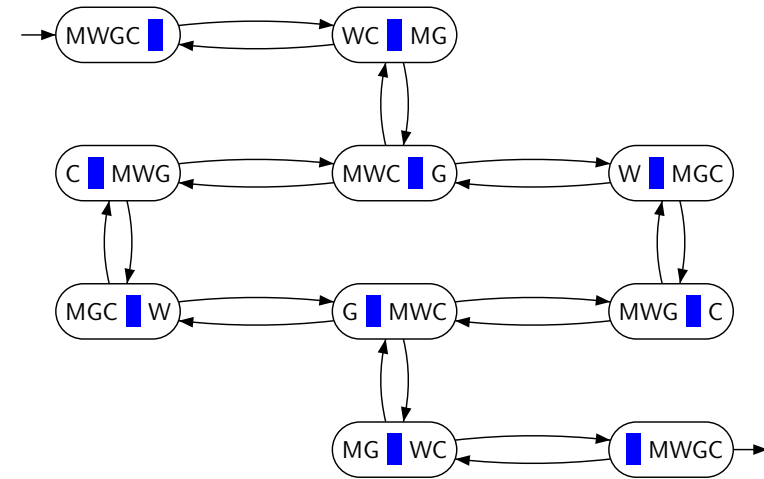
Example: Men, Wolf, Goat, Cabbage



Model = Transition system

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

Transition system



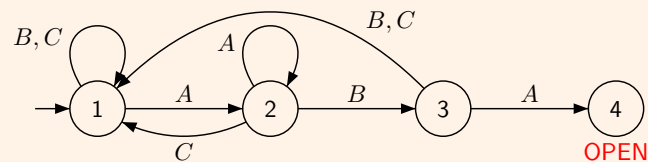
Transition system or Kripke structure

Definition: TS

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

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

Example: Digicode ABA



Every discrete system may be described with a TS.

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

- \mathcal{V} : set of (typed) variables, e.g., boolean, $[0..4]$, ...
- Each variable $v \in \mathcal{V}$ has a domain D_v (finite or infinite)
- Guard or Condition: unary predicate over $D = \prod_{v \in \mathcal{V}} D_v$
Symbolic descriptions: $x < 5, x + y = 10, \dots$
- Instruction or Update: map $f : D \rightarrow D$
Symbolic descriptions: $x := 0, x := (y + 1)^2, \dots$
- $T \subseteq S \times (2^D \times \Sigma \times D^D) \times S$
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

Transition systems with variables

Semantics: low level TS

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

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

SOS: Structural Operational Semantics

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

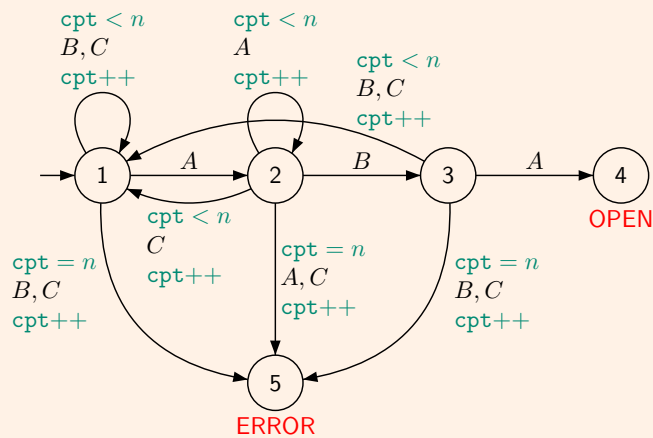
Programs = Kripke structures with variables

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

Example: GCD

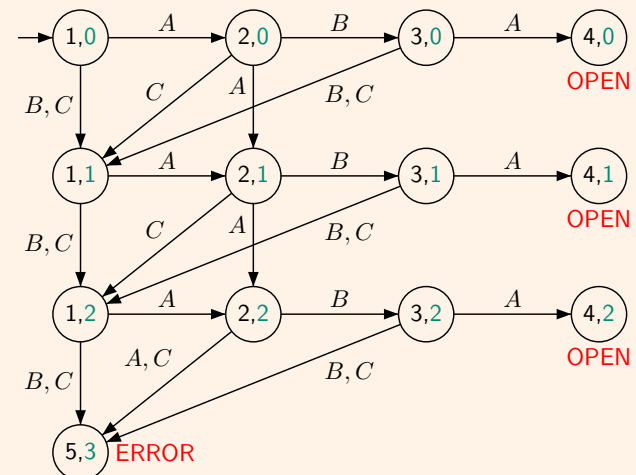
TS with variables ...

Example: Digicode



... and its semantics (n = 2)

Example: Digicode



Only variables

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

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

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

Symbolic representations with logic formulae

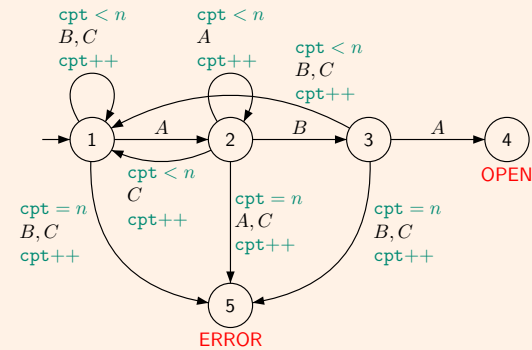
- ▶ I given by a formula $\psi(\nu)$
- ▶ T given by a formula $\varphi(\nu, \nu')$
- ▶ ν : values **before** the transition
- ▶ ν' : values **after** the transition
- ▶ Often we use boolean variables only: $D_v = \{0, 1\}$
- ▶ Concise descriptions of boolean formulae with Binary Decision Diagrams.

Example: Boolean circuit: modulo 8 counter

$$\begin{aligned} b'_0 &= \neg b_0 \\ b'_1 &= b_0 \oplus b_1 \\ b'_2 &= (b_0 \wedge b_1) \oplus b_2 \end{aligned}$$

Symbolic representation

Example: Logical representation



$$\begin{aligned} \delta_B = & s = 1 \wedge \text{cpt} < n \wedge s' = 1 \wedge \text{cpt}' = \text{cpt} + 1 \\ \vee & s = 1 \wedge \text{cpt} = n \wedge s' = 5 \wedge \text{cpt}' = \text{cpt} + 1 \\ \vee & s = 2 \wedge s' = 3 \wedge \text{cpt}' = \text{cpt} \\ \vee & s = 3 \wedge \text{cpt} < n \wedge s' = 1 \wedge \text{cpt}' = \text{cpt} + 1 \\ \vee & s = 3 \wedge \text{cpt} = n \wedge s' = 5 \wedge \text{cpt}' = \text{cpt} + 1 \end{aligned}$$

Modular description of concurrent systems

$$M = M_1 \parallel M_2 \parallel \dots \parallel M_n$$

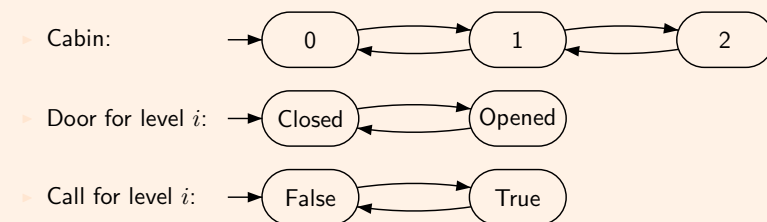
Semantics

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

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

Modular description of concurrent systems

Example: Elevator



The actual system is a **synchronized product** of all these automata. It consists of (at most) $3 \times 2^3 \times 2^3 = 192$ states.

Synchronized products

Definition: General product

- Components: $M_i = (S_i, \Sigma_i, T_i, I_i, AP_i, \ell_i)$
- Product: $M = (S, \Sigma, T, I, AP, \ell)$ with
 - $S = \prod_i S_i$, $\Sigma = \prod_i (\Sigma_i \cup \{\varepsilon\})$, and $I = \prod_i I_i$
 - $T = \{(p_1, \dots, p_n) \xrightarrow{(a_1, \dots, a_n)} (q_1, \dots, q_n) \mid \text{for all } i, (p_i, a_i, q_i) \in T_i \text{ or } p_i = q_i \text{ and } a_i = \varepsilon\}$
 - $AP = \bigcup_i AP_i$ and $\ell(p_1, \dots, p_n) = \bigcup_i \ell(p_i)$

Synchronized products: restrictions of the general product.

Parallel compositions

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

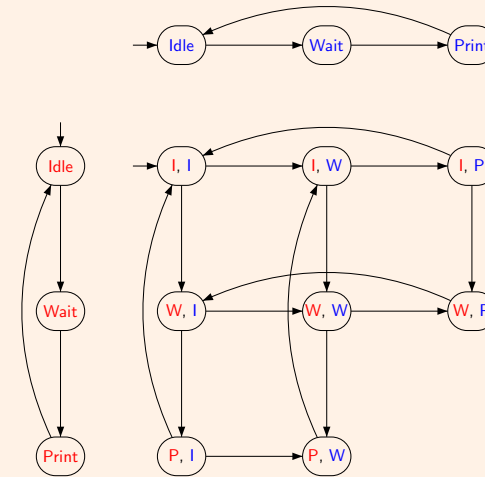
Synchronizations

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

Example: Printer manager

Example: Asynchronous product

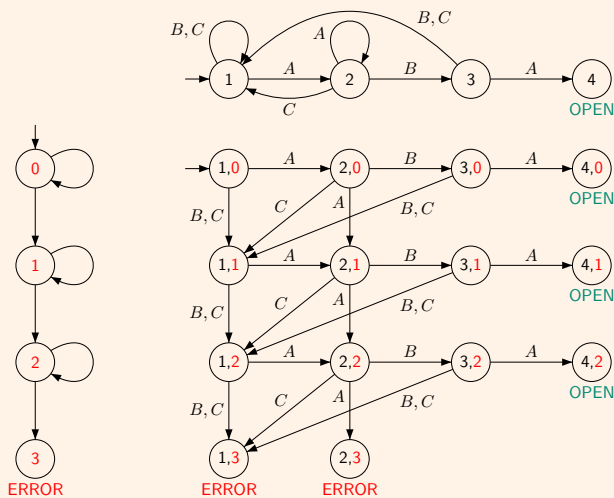
Synchronization by states: (P, P) is forbidden



Example: digicode

Example: Synchronous product

Synchronization by transitions



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

$$\text{Local actions} \quad \frac{s_1 \xrightarrow{a_1} s'_1}{(s_1, s_2) \xrightarrow{a_1} (s'_1, s_2)} \quad \frac{s_2 \xrightarrow{a_2} s'_2}{(s_1, s_2) \xrightarrow{a_2} (s_1, s'_2)}$$

$$\text{Rendez-vous} \quad \frac{s_1 \xrightarrow{!m} s'_1 \wedge s_2 \xrightarrow{?m} s'_2}{(s_1, s_2) \xrightarrow{m} (s'_1, s'_2)} \quad \frac{s_1 \xrightarrow{?m} s'_1 \wedge s_2 \xrightarrow{!m} s'_2}{(s_1, s_2) \xrightarrow{m} (s'_1, s'_2)}$$

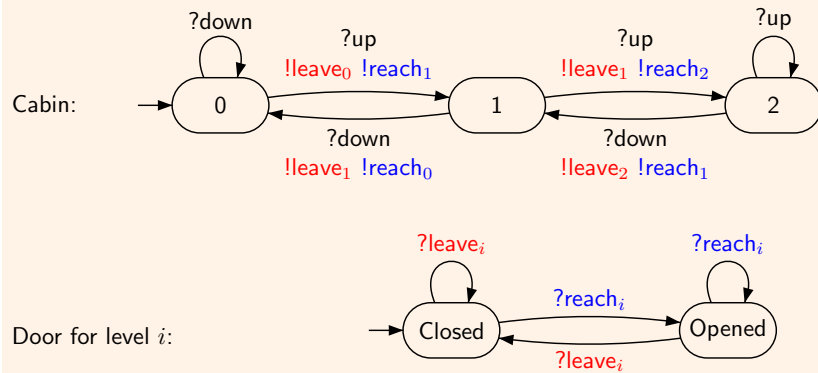
- It is a kind of synchronization by actions.
- Essential feature of process algebra.

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

- $?up$ is uncontrollable for the cabin
- $?leave_i$ is uncontrollable for door i
- $?call_0$ is uncontrollable for the system

Example: Elevator

Example: Synchronization by Rendez-vous



We should design the controller

Shared variables

Definition: Asynchronous product + shared variables

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

Semantics (SOS)
$$\frac{\nu \models g \wedge s_i \xrightarrow{g,a,f} s'_i \wedge s'_j = s_j \text{ for } j \neq i}{(\bar{s}, \nu) \xrightarrow{a} (\bar{s}', f(\nu))}$$

Example: Mutual exclusion for 2 processes satisfying

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

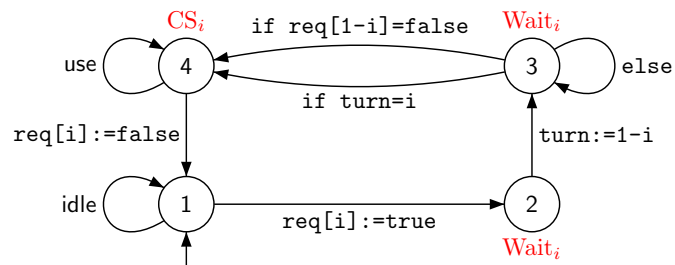
using shared variables but no synchronization mechanisms: the **atomicity** is

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

Peterson's algorithm (1981)

```

Process i:
loop forever
  req[i] := true; turn := 1-i
  wait until (turn = i or req[1-i] = false)
  Critical section
  req[i] := false
  
```



Exercise:

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

Atomicity

Example:

Initially $x = 1 \wedge y = 2$

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

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

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

Atomicity

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

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

$$ES ::= \text{skip} \mid \text{await } c \mid x := e \mid ES ; ES \mid ES \square ES \\ \mid \text{when } c \text{ do } ES \mid \text{if } c \text{ then } ES \text{ else } ES$$

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

$$\frac{(\bar{s}, \nu) \xrightarrow{ES} (\bar{s}', \nu')}{(\bar{s}, \nu) \xrightarrow{\text{atomic}(ES)} (\bar{s}', \nu')}$$

Example: Atomic statements

- ▶ $\text{atomic}(x = 0; x := 1)$ (Test and set)
- ▶ $\text{atomic}(y := y - 1; \text{await}(y = 0); y := 1)$ is equivalent to $\text{await}(y = 1)$

Channels

Example: Leader election

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

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

Channels

Definition: Channels

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

Channels

Semantics: (lossy) FIFO

$$\begin{array}{l} \text{Send} \quad \frac{s_i \xrightarrow{c!e} s'_i \wedge \nu'(c) = \nu(e) \cdot \nu(c)}{(\bar{s}, \nu) \xrightarrow{c!e} (\bar{s}', \nu')} \\ \text{Receive} \quad \frac{s_i \xrightarrow{c?x} s'_i \wedge \nu(c) = \nu'(c) \cdot \nu'(x)}{(\bar{s}, \nu) \xrightarrow{c?x} (\bar{s}', \nu')} \\ \text{Lossy send} \quad \frac{s_i \xrightarrow{c!e} s'_i}{(\bar{s}, \nu) \xrightarrow{c!e} (\bar{s}', \nu')} \end{array}$$

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.

First Order specifications

First order logic

- ▶ These specifications can be written in FO(<).
- ▶ FO(<) has a good expressive power.
... but FO(<)-formulae are not easy to write and to understand.
- ▶ FO(<) is decidable.
... but satisfiability and model checking are non elementary.

Definition: Temporal logics

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

Linear versus Branching

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

Definition: Linear specifications

Example: The printer manager is **fair**.

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

Execution sequences (runs): $\sigma = s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \dots$ with $s_i \rightarrow s_{i+1} \in T$

Two Kripke structures having the same execution sequences satisfy the same linear specifications.

Actually, linear specifications only depend on the **label** of the execution sequence

$$\ell(\sigma) = \ell(s_0) \rightarrow \ell(s_1) \rightarrow \ell(s_2) \rightarrow \dots$$

Models are words in Σ^ω with $\Sigma = 2^{AP}$.

Definition: Branching specifications

Example: Each process has the **possibility** to print first.

Such properties depend on the execution tree.

Execution tree = unfolding of the transition system

References

Bibliography

- [6] S. Demri and P. Gastin.
Specification and Verification using Temporal Logics.
In *Modern applications of automata theory*, IISc Research Monographs 2.
World Scientific, To appear.
<http://www.lsv.ens-cachan.fr/~gastin/mes-publis.php>

A large list of references is given in this paper.

Bibliography

- [7] V. Diekert and P. Gastin.
First-order definable languages.
In *Logic and Automata: History and Perspectives*, vol. 2, *Texts in Logic and Games*, pp. 261–306. Amsterdam University Press, (2008).
<http://www.lsv.ens-cachan.fr/~gastin/mes-publis.php>

A large overview of formalisms expressively equivalent to First-Order.

Some original References

- [8] J. Kamp.
Tense Logic and the Theory of Linear Order.
PhD thesis, UCLA, USA, (1968).
- [10] P. Gastin and D. Oddoux.
Fast LTL to Büchi automata translation.
In *CAV'01*, vol. 2102, *Lecture Notes in Computer Science*, pp. 53–65.
Springer, (2001).
<http://www.lsv.ens-cachan.fr/~gastin/mes-publis.php>
- [9] P. Wolper.
The tableau method for temporal logic: An overview,
Logique et Analyse. **110–111**, 119–136, (1985).
- [11] A. Sistla and E. Clarke.
The complexity of propositional linear temporal logic.
Journal of the Association for Computing Machinery. **32** (3), 733–749, (1985).

Some original References

- [12] O. Lichtenstein and A. Pnueli.
Checking that finite state concurrent programs satisfy their linear specification.
In *ACM Symposium PoPL'85*, 97–107.
- [13] D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi.
On the temporal analysis of fairness.
In *7th Annual ACM Symposium PoPL'80*, 163–173. ACM Press.
- [14] D. Gabbay.
The declarative past and imperative future: Executable temporal logics for interactive systems.
In *Temporal Logics in Specifications, April 87*. LNCS 398, 409–448, 1989.

Outline

Introduction

Models

Specifications

4 Linear Time Specifications

- Definitions
- Main results
- Büchi automata
- From LTL to BA
- Hardness results

Branching Time Specifications

Linear Temporal Logic (Pnueli 1977)

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

$$\varphi ::= \perp \mid p \ (p \in AP) \mid \neg\varphi \mid \varphi \vee \psi \mid X\varphi \mid \varphi U \psi$$

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

| | | |
|----------------------------------|----|--|
| $w, i \models p$ | if | $p \in a_i$ |
| $w, i \models \neg\varphi$ | if | $w, i \not\models \varphi$ |
| $w, i \models \varphi \vee \psi$ | if | $w, i \models \varphi$ or $w, i \models \psi$ |
| $w, i \models X\varphi$ | if | $w, i+1 \models \varphi$ |
| $w, i \models \varphi U \psi$ | if | $\exists k. i \leq k$ and $w, k \models \psi$ and $\forall j. (i \leq j < k) \rightarrow w, j \models \varphi$ |

Example:



Linear Temporal Logic (Pnueli 1977)

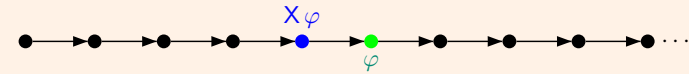
Definition: Syntax: $LTL(AP, X, U)$

$$\varphi ::= \perp \mid p \ (p \in AP) \mid \neg\varphi \mid \varphi \vee \psi \mid X\varphi \mid \varphi U \psi$$

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

| | | |
|----------------------------------|----|--|
| $w, i \models p$ | if | $p \in a_i$ |
| $w, i \models \neg\varphi$ | if | $w, i \not\models \varphi$ |
| $w, i \models \varphi \vee \psi$ | if | $w, i \models \varphi$ or $w, i \models \psi$ |
| $w, i \models X\varphi$ | if | $w, i+1 \models \varphi$ |
| $w, i \models \varphi U \psi$ | if | $\exists k. i \leq k$ and $w, k \models \psi$ and $\forall j. (i \leq j < k) \rightarrow w, j \models \varphi$ |

Example:



Linear Temporal Logic (Pnueli 1977)

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

$$\varphi ::= \perp \mid p \ (p \in AP) \mid \neg\varphi \mid \varphi \vee \psi \mid X\varphi \mid \varphi U \psi$$

Definition: Semantics: $w = a_0 a_1 a_2 \dots \in \Sigma^\omega$ with $\Sigma = 2^{AP}$ and $i \in \mathbb{N}$

$w, i \models p$ if $p \in a_i$

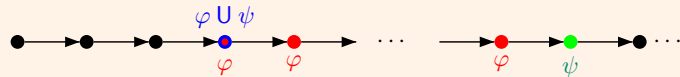
$w, i \models \neg\varphi$ if $w, i \not\models \varphi$

$w, i \models \varphi \vee \psi$ if $w, i \models \varphi$ or $w, i \models \psi$

$w, i \models X\varphi$ if $w, i+1 \models \varphi$

$w, i \models \varphi U \psi$ if $\exists k. i \leq k$ and $w, k \models \psi$ and $\forall j. (i \leq j < k) \rightarrow w, j \models \varphi$

Example:



Linear Temporal Logic (Pnueli 1977)

Definition: Macros

▶ **Eventually:** $F\varphi = \top U \varphi$



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

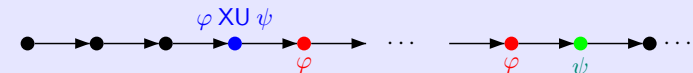


▶ **Weak until:** $\varphi W \psi = G\varphi \vee \varphi U \psi$

▶ $\neg(\varphi U \psi) = (G\neg\psi) \vee (\neg\psi U (\neg\varphi \wedge \neg\psi)) = \neg\psi W (\neg\varphi \wedge \neg\psi)$

▶ **Release:** $\varphi R \psi = \psi W (\varphi \wedge \psi) = \neg(\neg\varphi U \neg\psi)$

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



▶ $X\psi = \perp XU \psi$ and $\varphi U \psi = \psi \vee (\varphi \wedge \varphi XU \psi)$.

Linear Temporal Logic (Pnueli 1977)

Definition: Specifications:

▶ **Safety:** $G \text{ good}$

▶ **MutEx:** $\neg F(\text{crit}_1 \wedge \text{crit}_2)$

▶ **Liveness:** $G F \text{ active}$

▶ **Response:** $G(\text{request} \rightarrow F \text{ grant})$

▶ **Response':** $G(\text{request} \rightarrow X(\neg \text{request} U \text{ grant}))$

▶ **Release:** reset R alarm

▶ **Strong fairness:** $G F \text{ request} \rightarrow G F \text{ grant}$

▶ **Weak fairness:** $F G \text{ request} \rightarrow G F \text{ grant}$

Linear Temporal Logic (Pnueli 1977)

Examples:

Every elevator request should be eventually satisfied.

$$\bigwedge_i G(\text{Call}_i \rightarrow F(\text{atLevel}_i \wedge \text{openDoor}_i))$$

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

$$\bigwedge_i G(\text{Call}_i \rightarrow \neg \text{atLevel}_i W (\text{atLevel}_i \wedge \text{openDoor}_i))$$

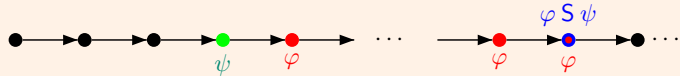
Past LTL

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

$w, i \models Y\varphi$ if $i > 0$ and $w, i-1 \models \varphi$

$w, i \models \varphi S \psi$ if $\exists k. k \leq i$ and $w, k \models \psi$ and $\forall j. (k < j \leq i) \rightarrow w, j \models \varphi$

Example:



Example: LTL versus PLTL

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

Theorem (Laroussinie & Markey & Schnoebelen 2002)

PLTL may be exponentially more succinct than LTL.

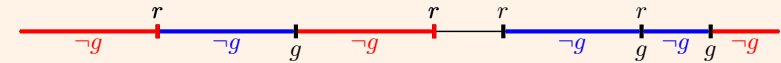
Past LTL

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

$w, i \models Y\varphi$ if $i > 0$ and $w, i-1 \models \varphi$

$w, i \models \varphi S \psi$ if $\exists k. k \leq i$ and $w, k \models \psi$ and $\forall j. (k < j \leq i) \rightarrow w, j \models \varphi$

Example:



Example: LTL versus PLTL

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

$= (\text{request} R \neg \text{grant}) \wedge G(\text{grant} \rightarrow (\text{request} \vee X(\text{request} R \neg \text{grant})))$

Theorem (Laroussinie & Markey & Schnoebelen 2002)

PLTL may be exponentially more succinct than LTL.

Expressivity

Theorem [8, Kamp 68]

$$\text{LTL}(Y, S, X, U) = \text{FO}_{\Sigma}(\leq)$$

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

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

$$w, k \models \varphi \iff w, k \models \bigvee_i \vec{\varphi}_i \wedge \vec{\varphi}_i$$

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

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

$$w, 0 \models \varphi \iff w, 0 \models \vec{\varphi}$$

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

Model checking for LTL

Definition: Model checking problem

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

Question: Does $M \models \varphi$?

- ▶ **Universal MC:** $M \models \forall \varphi$ 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 .

$$M \models \forall \varphi \iff M \not\models \exists \neg \varphi$$

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

The Model checking problem for LTL is PSPACE-complete

Satisfiability for LTL

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

Definition: Satisfiability problem

Input: A formula $\varphi \in \text{LTL}(AP, Y, S, X, U)$

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

Definition: Initial Satisfiability problem

Input: A formula $\varphi \in \text{LTL}(AP, Y, S, X, U)$

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

Remark: φ is satisfiable iff $F\varphi$ is *initially* satisfiable.

Theorem (Sistla, Clarke 85, Lichtenstein et. al 85)

The satisfiability problem for LTL is PSPACE-complete

Definition: (Initial) validity

φ is valid iff $\neg\varphi$ is **not** satisfiable.

Decision procedure for LTL

Definition: The core

From a formula $\varphi \in \text{LTL}(AP, \dots)$, construct a Büchi automaton \mathcal{A}_φ such that

$$\mathcal{L}(\mathcal{A}) = \mathcal{L}(\varphi) = \{w \in \Sigma^\omega \mid w, 0 \models \varphi\}.$$

Satisfiability (initial)

Check the Büchi automaton \mathcal{A}_φ for emptiness.

Model checking

Construct a synchronized product $\mathcal{B} = M \otimes \mathcal{A}_{\neg\varphi}$ so that the successful runs of \mathcal{B} correspond to the initial runs of M satisfying $\neg\varphi$.

Then, check \mathcal{B} for emptiness.

Theorem:

Checking Büchi automata for emptiness is NLOGSPACE-complete.

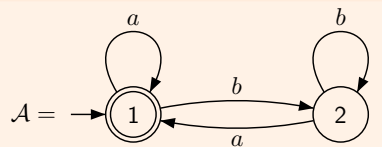
Büchi automata

Definition:

$\mathcal{A} = (Q, \Sigma, I, T, F)$ where

- ▶ Q : finite set of states
- ▶ Σ : finite set of labels
- ▶ $I \subseteq Q$: set of initial states
- ▶ $T \subseteq Q \times \Sigma \times Q$: transitions
- ▶ $F \subseteq Q$: set of accepting states (repeated, final)

Example:



$$\mathcal{L}(\mathcal{A}) = \{w \in \{a, b\}^\omega \mid |w|_a = \omega\}$$

Büchi automata for some LTL formulae

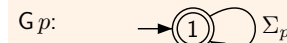
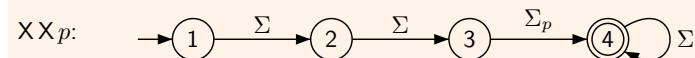
Definition:

Recall that $\Sigma = 2^{AP}$. For $\psi \in \mathbb{B}(AP)$ we let $\Sigma_\psi = \{a \in \Sigma \mid a \models \psi\}$.

For instance, for $p, q \in AP$,

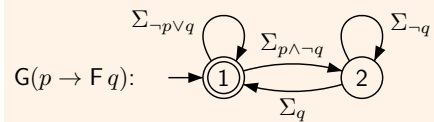
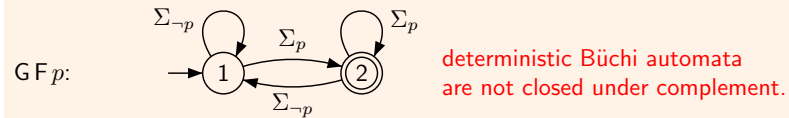
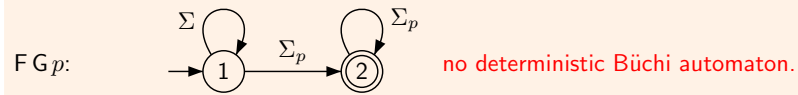
- ▶ $\Sigma_p = \{a \in \Sigma \mid p \in a\}$ and $\Sigma_{\neg p} = \Sigma \setminus \Sigma_p$
- ▶ $\Sigma_{p \wedge q} = \Sigma_p \cap \Sigma_q$ and $\Sigma_{p \vee q} = \Sigma_p \cup \Sigma_q$
- ▶ $\Sigma_{p \wedge \neg q} = \Sigma_p \setminus \Sigma_q$...

Examples:



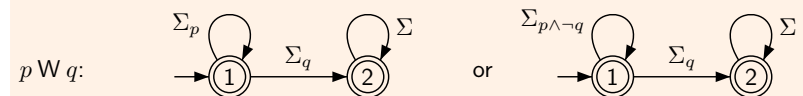
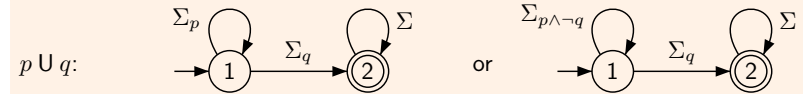
Büchi automata for some LTL formulae

Examples:



Büchi automata for some LTL formulae

Examples:



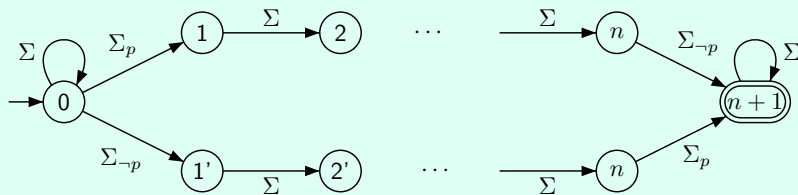
Büchi automata

Properties

Büchi automata are closed under union, intersection, complement.

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

Let $\varphi = F((p \wedge X^n \neg p) \vee (\neg p \wedge X^n p))$



Any non deterministic Büchi automaton for $\neg\varphi$ has at least 2^n states.

Büchi automata

Exercise:

Given Büchi automata for φ and ψ ,

- Construct a Büchi automaton for $X\varphi$ (trivial)
- Construct a Büchi automaton for $\varphi \cup \psi$

This gives an inductive construction of \mathcal{A}_φ from $\varphi \in \text{LTL}(AP, X, U) \dots$

\dots but the size of \mathcal{A}_φ might be non-elementary in the size of φ .

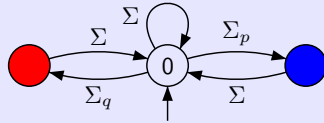
Generalized Büchi automata

Definition: acceptance on states

$\mathcal{A} = (Q, \Sigma, I, T, F_1, \dots, F_n)$ with $F_i \subseteq Q$.

An infinite run σ is successful if it visits infinitely often each F_i .

$GFp \wedge GFq$:

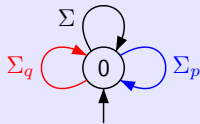


Definition: acceptance on transitions

$\mathcal{A} = (Q, \Sigma, I, T, T_1, \dots, T_n)$ with $T_i \subseteq T$.

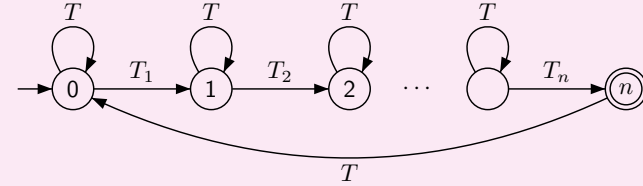
An infinite run σ is successful if it uses infinitely many transitions from each T_i .

$GFp \wedge GFq$:



GBA to BA

Proof: Synchronized product with \mathcal{B}



Transitions:
$$\frac{t = s_1 \xrightarrow{a} s'_1 \in \mathcal{A} \wedge s_2 \xrightarrow{t} s'_2 \in \mathcal{B}}{(s_1, s_2) \xrightarrow{a} (s'_1, s'_2)}$$

Accepting states: $Q \times \{n\}$

Negative normal form

Definition: Syntax ($p \in AP$)

$\varphi ::= \top \mid \perp \mid p \mid \neg p \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid X\varphi \mid \varphi U \psi \mid \varphi R \psi$

Proposition: Any formula can be transformed in NNF

$\neg(\varphi \vee \psi) \equiv (\neg\varphi) \wedge (\neg\psi)$ $\neg(\varphi \wedge \psi) \equiv (\neg\varphi) \vee (\neg\psi)$
 $\neg(\varphi U \psi) \equiv (\neg\varphi) R (\neg\psi)$ $\neg(\varphi R \psi) \equiv (\neg\varphi) U (\neg\psi)$
 $\neg X\varphi \equiv X\neg\varphi$ $\neg\neg\varphi \equiv \varphi$

This does not increase the number of Temporal subformulae.

Temporal formulae

Definition: Temporal formulae

- ▶ literals
- ▶ formulae with outermost connective X, U or R.

Reducing the number of temporal subformulae

$(X\varphi) \wedge (X\psi) \equiv X(\varphi \wedge \psi)$ $(X\varphi) U (X\psi) \equiv X(\varphi U \psi)$
 $(\varphi R \psi_1) \wedge (\varphi R \psi_2) \equiv \varphi R (\psi_1 \wedge \psi_2)$ $(\varphi_1 R \psi) \vee (\varphi_2 R \psi) \equiv (\varphi_1 \vee \varphi_2) R \psi$
 $(G\varphi) \wedge (G\psi) \equiv G(\varphi \wedge \psi)$ $GF\varphi \vee GF\psi \equiv GF(\varphi \vee \psi)$

From LTL to BA [6, Demri & Gastin 10]

Definition:

- $Z \subseteq \text{NNF}$ is **consistent** if $\perp \notin Z$ and $\{p, \neg p\} \not\subseteq Z$ for all $p \in \text{AP}$.
- For $Z \subseteq \text{NNF}$, we define $\bigwedge Z = \bigwedge_{\psi \in Z} \psi$.
- Note that $\bigwedge \emptyset = \top$ and if Z is inconsistent then $\bigwedge Z \equiv \perp$.

Intuition for the BA $\mathcal{A}_\varphi = (Q, \Sigma, I, T, (T_\alpha)_{\alpha \in U(\varphi)})$

Let $\varphi \in \text{NNF}$ be a formula.

- $\text{sub}(\varphi)$ is the set of **sub-formulae** of φ .
- $U(\varphi)$ the set of **until** sub-formulae of φ .
- We construct a BA \mathcal{A}_φ with $Q = 2^{\text{sub}(\varphi)}$ and $I = \{\varphi\}$.
- A state $Z \subseteq \text{sub}(\varphi)$ is a **set of obligations**.
- If $Z \subseteq \text{sub}(\varphi)$, we want $\mathcal{L}(\mathcal{A}_\varphi^Z) = \{u \in \Sigma^\omega \mid u, 0 \models \bigwedge Z\}$ where \mathcal{A}_φ^Z is \mathcal{A}_φ using Z as unique initial state.

Reduced formulae

Definition: Reduced formulae

- A formula is **reduced** if it is a literal (p or $\neg p$) or a next-formula ($X\beta$).
- $Z \subseteq \text{NNF}$ is **reduced** if all formulae in Z are reduced,

For $Z \subseteq \text{NNF}$ **consistent and reduced**, we define

- $\text{next}(Z) = \{\alpha \mid X\alpha \in Z\}$
- $\Sigma_Z = \bigcap_{p \in Z} \Sigma_p \cap \bigcap_{\neg p \in Z} \Sigma_{\neg p}$

Lemma: Next step

Let $Z \subseteq \text{NNF}$ be **consistent and reduced**.
Let $u = a_0 a_1 a_2 \dots \in \Sigma^\omega$ and $n \geq 0$. Then

$$u, n \models \bigwedge Z \quad \text{iff} \quad u, n+1 \models \bigwedge \text{next}(Z) \text{ and } a_n \in \Sigma_Z$$

- \mathcal{A}_φ will have transitions $Z \xrightarrow{\Sigma_Z} \text{next}(Z)$.
- Note that $\emptyset \xrightarrow{\Sigma} \emptyset$.
- Problem:** $\text{next}(Z)$ is not reduced in general (it may even be inconsistent).

Reduction rules

Definition: Reduction of obligations to literals and next-formulae

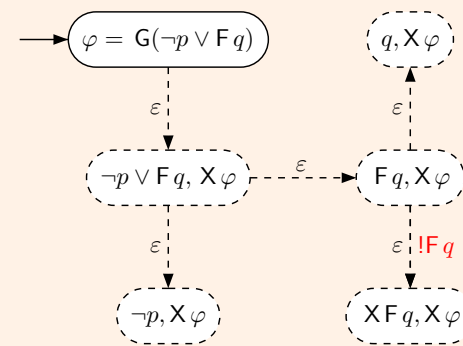
Let $Y \subseteq \text{NNF}$ and let $\psi \in Y$ **maximal not reduced**.

- | | |
|------------------------------------|---|
| If $\psi = \psi_1 \wedge \psi_2$: | $Y \xrightarrow{\varepsilon} (Y \setminus \{\psi\}) \cup \{\psi_1, \psi_2\}$ |
| If $\psi = \psi_1 \vee \psi_2$: | $Y \xrightarrow{\varepsilon} (Y \setminus \{\psi\}) \cup \{\psi_1\}$ $Y \xrightarrow{\varepsilon} (Y \setminus \{\psi\}) \cup \{\psi_2\}$ |
| If $\psi = \psi_1 R \psi_2$: | $Y \xrightarrow{\varepsilon} (Y \setminus \{\psi\}) \cup \{\psi_1, \psi_2\}$ $Y \xrightarrow{\varepsilon} (Y \setminus \{\psi\}) \cup \{\psi_2, X\psi\}$ |
| If $\psi = G \psi_2$: | $Y \xrightarrow{\varepsilon} (Y \setminus \{\psi\}) \cup \{\psi_2, X\psi\}$ |
| If $\psi = \psi_1 U \psi_2$: | $Y \xrightarrow{\varepsilon} (Y \setminus \{\psi\}) \cup \{\psi_2\}$ $Y \xrightarrow{\varepsilon, !\psi} (Y \setminus \{\psi\}) \cup \{\psi_1, X\psi\}$ |
| If $\psi = F \psi_2$: | $Y \xrightarrow{\varepsilon} (Y \setminus \{\psi\}) \cup \{\psi_2\}$ $Y \xrightarrow{\varepsilon, !\psi} (Y \setminus \{\psi\}) \cup \{X\psi\}$ |

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

Reduction rules

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



State = set of obligations.

Reduce obligations to literals and next-formulae.

Note again the mark $!Fq$ on the last edge

Reduction

Lemma:

- if there is only one rule $Y \xrightarrow{\varepsilon} Y_1$ then $\bigwedge Y \equiv \bigwedge Y_1$
- if there are two rules $Y \xrightarrow{\varepsilon} Y_1$ and $Y \xrightarrow{\varepsilon} Y_2$ then $\bigwedge Y \equiv \bigwedge Y_1 \vee \bigwedge Y_2$

Definition:

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

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

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

Lemma: Soundness

- Let $Y \subseteq \text{NNF}$, then $\bigwedge Y \equiv \bigvee_{Z \in \text{Red}(Y)} \bigwedge Z$
- Let $u = a_0 a_1 a_2 \dots \in \Sigma^\omega$ and $n \geq 0$ with $u, n \models \bigwedge Y$. Then, $\exists Z \in \text{Red}(Y)$ such that $u, n \models \bigwedge Z$ and $Z \in \text{Red}_\alpha(Y)$ for all $\alpha = \alpha_1 \cup \alpha_2 \in \text{U}(\varphi)$ such that $u, n \models \alpha_2$.

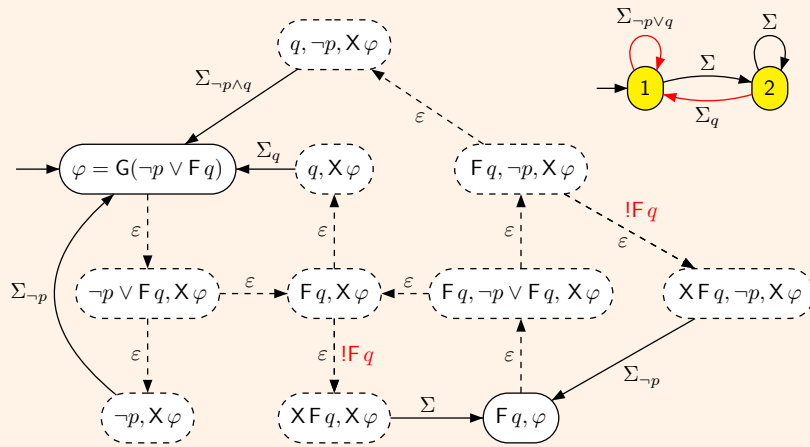
Automaton \mathcal{A}_φ

Definition: Automaton \mathcal{A}_φ

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

Automaton \mathcal{A}_φ

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



Transition = check literals and move forward.
Simplification

Correctness of \mathcal{A}_φ

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

Lemma:

Let $\rho = Y_0 \xrightarrow{a_0} Y_1 \xrightarrow{a_1} Y_2 \dots$ be an accepting run of \mathcal{A}_φ on $u = a_0 a_1 a_2 \dots \in \Sigma^\omega$.

Then, for all $\psi \in \text{sub}(\varphi)$ and $n \geq 0$, for all reduction path $Y_n \xrightarrow{\varepsilon_*} Y \xrightarrow{\varepsilon_*} Z$ with $a_n \in \Sigma_Z$ and $Y_{n+1} = \text{next}(Z)$,

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

Corollary: $\mathcal{L}(\mathcal{A}_\varphi) \subseteq \mathcal{L}(\varphi)$

$\mathcal{L}(\varphi) \subseteq \mathcal{L}(\mathcal{A}_\varphi)$

Proof:

Let $u = a_0 a_1 a_2 \dots \in \Sigma^\omega$ be such that $u, 0 \models \varphi$. By induction, we build a run

$$\rho = Y_0 \xrightarrow{a_0} Y_1 \xrightarrow{a_1} Y_2 \dots$$

We start with $Y_0 = \{\varphi\}$. Assume that $u, n \models \bigwedge Y_n$ for some $n \geq 0$. By Lemma [Soundness], there is $Z_n \in \text{Red}(Y_n)$ such that $u, n \models \bigwedge Z_n$ and for all until subformulae $\alpha = \alpha_1 \cup \alpha_2 \in \text{U}(\varphi)$, if $u, n \models \alpha_2$ then $Z_n \in \text{Red}_\alpha(Y_n)$. Then we define $Y_{n+1} = \text{next}(Z_n)$. Since $u, n \models \bigwedge Z_n$, Lemma [Next Step] implies $a_n \in \Sigma_{Z_n}$ and $u, n+1 \models \bigwedge Y_{n+1}$. Therefore, ρ is a run for u in \mathcal{A}_φ .

It remains to show that ρ is successful. By definition, it starts from the initial state $\{\varphi\}$. Now let $\alpha = \alpha_1 \cup \alpha_2 \in \text{U}(\varphi)$. Assume there exists $N \geq 0$ such that $Y_n \xrightarrow{a_n} Y_{n+1} \notin T_\alpha$ for all $n \geq N$. Then $Z_n \notin \text{Red}_\alpha(Y_n)$ for all $n \geq N$ and we deduce that $u, n \not\models \alpha_2$ for all $n \geq N$. But, since $Z_N \in \text{Red}_\alpha(Y_N)$, the formula α has been reduced using an ε -transition marked $\text{!}\alpha$ along the path from Y_N to Z_N . Therefore, $X\alpha \in Z_N$ and $\alpha \in Y_{N+1}$. By construction of the run we have $u, N+1 \models \bigwedge Y_{N+1}$. Hence, $u, N+1 \models \alpha$, a contradiction with $u, n \not\models \alpha_2$ for all $n \geq N$. Consequently, the run ρ is successful and u is accepted by \mathcal{A}_φ .

$\mathcal{L}(\mathcal{A}_\varphi) \subseteq \mathcal{L}(\varphi)$

Lemma:

Let $\rho = Y_0 \xrightarrow{a_0} Y_1 \xrightarrow{a_1} Y_2 \dots$ be an accepting run of \mathcal{A}_φ on $u = a_0 a_1 a_2 \dots \in \Sigma^\omega$.

Then, for all $\psi \in \text{sub}(\varphi)$ and $n \geq 0$,

for all reduction path $Y_n \xrightarrow{\varepsilon} Y' \xrightarrow{\varepsilon} Z$ with $a_n \in \Sigma_Z$ and $Y_{n+1} = \text{next}(Z)$,

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

Proof: by induction on ψ

- $\psi = \top$. The result is trivial.
- $\psi = p \in \text{AP}(\varphi)$. Since p is reduced, we have $p \in Z$ and it follows $\Sigma_Z \subseteq \Sigma_p$. Therefore, $p \in a_n$ and $u, n \models p$. The proof is similar if $\psi = \neg p$ for some $p \in \text{AP}(\varphi)$.
- $\psi = X\psi_1$. Then $\psi \in Z$ and $\psi_1 \in Y_{n+1}$. By induction we obtain $u, n+1 \models \psi_1$ and we deduce $u, n \models X\psi_1 = \psi$.
- $\psi = \psi_1 \wedge \psi_2$. Along the path $Y \xrightarrow{\varepsilon} Z$ the formula ψ must be reduced so $Y \xrightarrow{\varepsilon} Y' \xrightarrow{\varepsilon} Z$ with $\psi_1, \psi_2 \in Y'$. By induction, we obtain $u, n \models \psi_1$ and $u, n \models \psi_2$. Hence, $u, n \models \psi$. The proof is similar for $\psi = \psi_1 \vee \psi_2$.

$\mathcal{L}(\mathcal{A}_\varphi) \subseteq \mathcal{L}(\varphi)$

Proof:

• $\psi = \psi_1 \cup \psi_2$. Along the path $Y \xrightarrow{\varepsilon} Z$ the formula ψ must be reduced so $Y \xrightarrow{\varepsilon} Y' \xrightarrow{\varepsilon} Y'' \xrightarrow{\varepsilon} Z$ with either $Y'' = Y' \setminus \{\psi\} \cup \{\psi_2\}$ or $Y'' = Y' \setminus \{\psi\} \cup \{\psi_1, X\psi\}$. In the first case, we obtain by induction $u, n \models \psi_2$ and therefore $u, n \models \psi$. In the second case, we obtain by induction $u, n \models \psi_1$. Since $X\psi$ is reduced we get $X\psi \in Z$ and $\psi \in \text{next}(Z) = Y_{n+1}$.

Let $k > n$ be minimal such that $Y_k \xrightarrow{a_k} Y_{k+1} \in T_\psi$ (such a value k exists since ρ is accepting). We first show by induction that $u, i \models \psi_1$ and $\psi \in Y_{i+1}$ for all $n \leq i < k$. Recall that $u, n \models \psi_1$ and $\psi \in Y_{n+1}$. So let $n < i < k$ be such that $\psi \in Y_i$. Let $Z' \in \text{Red}(Y_i)$ be such that $a_i \in \Sigma_{Z'}$ and $Y_{i+1} = \text{next}(Z')$. Since k is minimal we know that $Z' \notin \text{Red}_\psi(Y_i)$. Hence, along any reduction path from Y_i to Z' we must use a step $Y' \xrightarrow{\varepsilon} Y' \setminus \{\psi\} \cup \{\psi_1, X\psi\}$. By induction on the formula we obtain $u, i \models \psi_1$. Also, since $X\psi$ is reduced, we have $X\psi \in Z'$ and $\psi \in \text{next}(Z') = Y_{i+1}$.

Second, we show that $u, k \models \psi_2$. Since $Y_k \xrightarrow{a_k} Y_{k+1} \in T_\psi$, we find some $Z' \in \text{Red}_\psi(Y_k)$ such that $a_k \in \Sigma_{Z'}$ and $Y_{k+1} = \text{next}(Z')$. Since $\psi \in Y_k$, along some reduction path from Y_k to Z' we use a step $Y' \xrightarrow{\varepsilon} Y' \setminus \{\psi\} \cup \{\psi_2\}$. By induction we obtain $u, k \models \psi_2$. Finally, we have shown $u, n \models \psi_1 \cup \psi_2 = \psi$.

$\mathcal{L}(\mathcal{A}_\varphi) \subseteq \mathcal{L}(\varphi)$

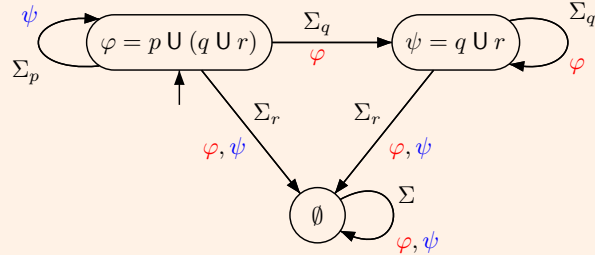
Proof:

• $\psi = \psi_1 \text{ R } \psi_2$. Along the path $Y \xrightarrow{\varepsilon} Z$ the formula ψ must be reduced so $Y \xrightarrow{\varepsilon} Y' \xrightarrow{\varepsilon} Y'' \xrightarrow{\varepsilon} Z$ with either $Y'' = Y' \setminus \{\psi\} \cup \{\psi_1, \psi_2\}$ or $Y'' = Y' \setminus \{\psi\} \cup \{\psi_2, X\psi\}$. In the first case, we obtain by induction $u, n \models \psi_1$ and $u, n \models \psi_2$. Hence, $u, n \models \psi$ and we are done. In the second case, we obtain by induction $u, n \models \psi_2$ and we get also $\psi \in Y_{n+1}$. Continuing with the same reasoning, we deduce easily that either $u, n \models G\psi_2$ or $u, n \models \psi_2 \cup (\psi_1 \wedge \psi_2)$.

Example with two until sub-formulae

Example: Nested until: $\varphi = p \text{ U } \psi$ with $\psi = q \text{ U } r$

$$\begin{aligned} \text{Red}(\{\varphi\}) &= \{\{p, X\varphi\}, \{q, X\psi\}, \{r\}\} & \text{Red}(\{\psi\}) &= \{\{q, X\psi\}, \{r\}\} \\ \text{Red}_\varphi(\{\varphi\}) &= \{\{q, X\psi\}, \{r\}\} & \text{Red}_\varphi(\{\psi\}) &= \{\{q, X\psi\}, \{r\}\} \\ \text{Red}_\psi(\{\varphi\}) &= \{\{p, X\varphi\}, \{r\}\} & \text{Red}_\psi(\{\psi\}) &= \{\{r\}\} \end{aligned}$$



Satisfiability and Model Checking

Corollary: PSPACE upper bound for satisfiability and model checking

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

Proof:

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

$$\text{Transitions: } \frac{s \rightarrow s' \in M \quad \wedge \quad Y \xrightarrow{\ell(s)} Y' \in \mathcal{A}_{\neg\varphi}}{(s, Y) \xrightarrow{\ell(s)} (s', Y')}$$

Initial states: $I \times \{\{\neg\varphi\}\}$.

Acceptance conditions: inherited from $\mathcal{A}_{\neg\varphi}$.

Check $M \otimes \mathcal{A}_{\neg\varphi}$ for emptiness.

On the fly simplifications \mathcal{A}_φ

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

Definition: Additional reduction rules

If $\bigwedge Y \equiv \bigwedge Y'$ then we may use $Y \xrightarrow{\varepsilon} Y'$.

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

$$\begin{aligned} \text{If } \psi &= \psi_1 \vee \psi_2 \text{ and } \psi_1 \in Y \text{ or } \psi_2 \in Y: & Y &\xrightarrow{\varepsilon} Y \setminus \{\psi\} \\ \text{If } \psi &= \psi_1 \text{ U } \psi_2 \text{ and } \psi_2 \in Y: & Y &\xrightarrow{\varepsilon} Y \setminus \{\psi\} \\ \text{If } \psi &= \psi_1 \text{ R } \psi_2 \text{ and } \psi_1 \in Y: & Y &\xrightarrow{\varepsilon} Y \setminus \{\psi\} \cup \{\psi_2\} \end{aligned}$$

On the fly simplifications \mathcal{A}_φ

Definition: Merging equivalent states

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

We can merge s_1 and s_2 if they have the same outgoing transitions:
 $\forall a \in \Sigma, \forall s \in Q,$

$$\begin{aligned} (s_1, a, s) \in T &\iff (s_2, a, s) \in T \\ \text{and } (s_1, a, s) \in T_i &\iff (s_2, a, s) \in T_i \quad \text{for all } 1 \leq i \leq n. \end{aligned}$$

Remark: Sufficient condition

Two states Y, Y' of \mathcal{A}_φ have the same outgoing transition if

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

Example: Let $\varphi = \text{GF } p \wedge \text{GF } q$.

Without merging states \mathcal{A}_φ has 4 states.

These 4 states have the same outgoing transitions.

The simplified automaton has only one state.

Other constructions

- ▶ Tableau construction. See for instance [9, Wolper 85]
 - + : Easy definition, easy proof of correctness
 - + : Works both for future and past modalities
 - : Inefficient without optimizations
- ▶ Using **Very Weak Alternating Automata** [10, Gastin & Oddoux 01].
 - + : Very efficient
 - : Only for future modalities
 Online tool: <http://www.lsv.ens-cachan.fr/~gastin/ltl2ba/>
- ▶ The domain is still very active.
- ▶ See other references in [6, Demri & Gastin 10].

$MC^{\exists}(X, U) \leq_P SAT(X, U)$ [11, Sistla & Clarke 85]

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

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

Define $AP' = AP \uplus AP_S$ $\Sigma' = 2^{AP'}$ $\pi : \Sigma'^{\omega} \rightarrow \Sigma^{\omega}$ by $\pi(a) = a \cap AP$.

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

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

$$\psi_M = \left(\bigvee_{s \in I} at_s \right) \wedge G \left(\bigvee_{s \in S} \left(at_s \wedge \bigwedge_{t \neq s} \neg at_t \wedge \bigwedge_{p \in \ell(s)} p \wedge \bigwedge_{p \notin \ell(s)} \neg p \wedge \bigvee_{t \in T(s)} X at_t \right) \right)$$

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

Therefore, $M \models_{\exists} \varphi$ iff $\psi_M \wedge \varphi$ is satisfiable
 $M \models_{\forall} \varphi$ iff $\psi_M \wedge \neg \varphi$ is not satisfiable

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

QBF Quantified Boolean Formulae

Definition: QBF

Input: A formula $\gamma = Q_1 x_1 \dots Q_n x_n \gamma'$ with $\gamma' = \bigwedge_{1 \leq i \leq m} \bigvee_{1 \leq j \leq k_i} a_{ij}$
 $Q_i \in \{\forall, \exists\}$ and $a_{ij} \in \{x_1, \neg x_1, \dots, x_n, \neg x_n\}$.

Question: Is γ valid?

Definition:

An assignment of the variables $\{x_1, \dots, x_n\}$ is a word $v = v_1 \dots v_n \in \{0, 1\}^n$.
 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 γ') if $v \models \gamma'$ for all $v \in V$,
- ▶ V is **closed** (for γ) if $\forall v \in V, \forall 1 \leq i \leq n$ s.t. $Q_i = \forall$,
 $\exists v' \in V$ s.t. $v[i-1] = v'[i-1]$ and $\{v_i, v'_i\} = \{0, 1\}$.

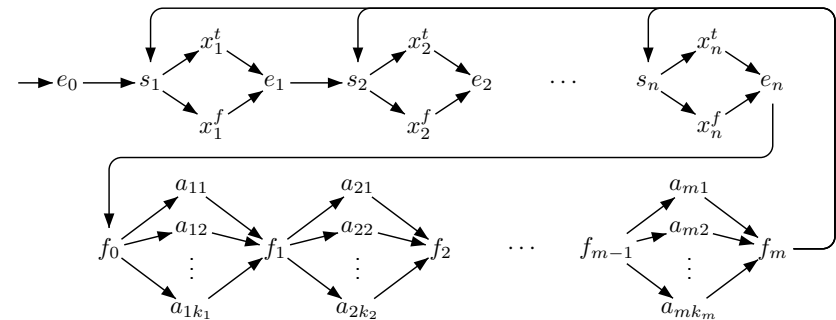
Proposition:

γ is valid iff $\exists V \subseteq \{0, 1\}^n$ s.t. V is nonempty valid and closed

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

Let $\gamma = Q_1 x_1 \dots Q_n x_n \bigwedge_{1 \leq i \leq m} \bigvee_{1 \leq j \leq k_i} a_{ij}$ with $Q_i \in \{\forall, \exists\}$ and a_{ij} literals.

Consider the KS M :



Let $\psi_{ij} = \begin{cases} G(x_k^f \rightarrow s_k R \neg a_{ij}) & \text{if } a_{ij} = x_k \\ G(x_k^t \rightarrow s_k R \neg a_{ij}) & \text{if } a_{ij} = \neg x_k \end{cases}$ and $\psi = \bigwedge_{i,j} \psi_{ij}$.

Let $\varphi_j = G(e_{j-1} \rightarrow (\neg s_{j-1} U x_j^t) \wedge (\neg s_{j-1} U x_j^f))$ and $\varphi = \bigwedge_{j|Q_j=\forall} \varphi_j$.

Then, γ is valid iff $M \models_{\exists} \psi \wedge \varphi$.

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

Proof: If $M \models_{\exists} \psi \wedge \varphi$ then γ is valid

Each finite path $\tau = e_0 \xrightarrow{*} f_m$ in M defines a valuation v^τ by:

$$v_k^\tau = \begin{cases} 1 & \text{if } \tau, |\tau| \models \neg s_k \text{ S } x_k^t \\ 0 & \text{if } \tau, |\tau| \models \neg s_k \text{ S } x_k^f \end{cases}$$

Let σ be an initial infinite path of M s.t. $\sigma, 0 \models \psi \wedge \varphi$.

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

Claim: V is nonempty, valid and closed.

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

Proof: If γ is valid then $M \models_{\exists} \psi \wedge \varphi$

Let $V \subseteq \{0, 1\}^n$ be nonempty, valid and closed.

First ingredient: extension of a run.

Assume $\tau = e_0 \xrightarrow{*} f_m$ satisfies $v^\tau \in V$ and $\tau, 0 \models \psi$.

Let $1 \leq i \leq n$ with $Q_i = \forall$.

Let $v' \in V$ s.t. $v'[i-1] = v[i-1]$ and $\{v_i, v'_i\} = \{0, 1\}$.

We can extend τ in $\tau' = \tau \rightarrow s_i \xrightarrow{*} e_n \rightarrow f_0 \xrightarrow{*} f_m$ with $v^{\tau'} = v'$ and $\tau', 0 \models \psi$.

We say that τ' is an extension of τ wrt. i .

Second step: the sequence of indices for the extensions.

Let $1 \leq i_\ell < \dots < i_1 \leq n$ be the indices of universal quantifications ($Q_{i_j} = \forall$).

Define by induction $w_1 = i_1$ and if $k < \ell$, $w_{k+1} = w_k i_{k+1} w_k$. Let $w = (w_\ell 1)^\omega$.

Final step: the infinite run.

Let $v \in V \neq \emptyset$ and let $\tau = e_0 \xrightarrow{*} f_m$ with $v^\tau \in V$ and $\tau, 0 \models \psi$.

We build an infinite run σ by extending τ inductively wrt. the sequence of indices defined by w .

Claim: $\sigma, 0 \models \psi \wedge \varphi$.

Complexity of LTL

Theorem: Complexity of LTL

The following problems are PSPACE-complete:

- ▶ SAT(LTL(X, U, Y, S)), MC $^{\forall}$ (LTL(X, U, Y, S)), MC $^{\exists}$ (LTL(X, U, Y, S))
- ▶ SAT(LTL(X, F)), MC $^{\forall}$ (LTL(X, F)), MC $^{\exists}$ (LTL(X, F))
- ▶ SAT(LTL(U)), MC $^{\forall}$ (LTL(U)), MC $^{\exists}$ (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))

Outline

Introduction

Models

Specifications

Linear Time Specifications

5 Branching Time Specifications

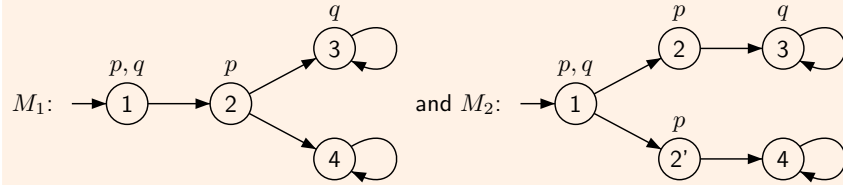
- CTL*
- CTL
- Fair CTL

Possibility is not expressible in LTL

Example:

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

Consider the two models:



$M_1 \models \varphi$ but $M_2 \not\models \varphi$
 M_1 and M_2 satisfy the same LTL formulae.

We need quantifications on runs: $\varphi = AG(p \rightarrow EFq)$

- ▶ E: for some infinite run
- ▶ A: for all infinite runs

CTL* (Emerson & Halpern 86)

Definition: Syntax of the Computation Tree Logic CTL*

$$\varphi ::= \perp \mid p \ (p \in AP) \mid \neg\varphi \mid \varphi \vee \varphi \mid X\varphi \mid \varphi U \varphi \mid E\varphi \mid A\varphi$$

Definition: Semantics:

Let $M = (S, T, I, AP, \ell)$ be a Kripke structure and σ an infinite run of M .

$$M, \sigma, i \models E\varphi \text{ if } M, \sigma', 0 \models \varphi \text{ for some infinite run } \sigma' \text{ such that } \sigma'(0) = \sigma(i)$$

$$M, \sigma, i \models A\varphi \text{ if } M, \sigma', 0 \models \varphi \text{ for all infinite runs } \sigma' \text{ such that } \sigma'(0) = \sigma(i)$$

Example: Some specifications

- ▶ EF φ : φ is possible
- ▶ AG φ : φ is an invariant
- ▶ AF φ : φ is unavoidable
- ▶ EG φ : φ holds globally along some path

Remark:

$$A\varphi \equiv \neg E\neg\varphi$$

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 φ is a state formula and $M = (S, T, I, AP, \ell)$, define

$$\llbracket \varphi \rrbracket^M = \{s \in S \mid M, s \models \varphi\}$$

Example: State formulae

Formulae of the form p or $E\varphi$ or $A\varphi$ are state formulae.
 State formulae are closed under boolean connectives.

$$\llbracket p \rrbracket = \{s \in S \mid p \in \ell(s)\} \quad \llbracket \neg\varphi \rrbracket = S \setminus \llbracket \varphi \rrbracket \quad \llbracket \varphi_1 \vee \varphi_2 \rrbracket = \llbracket \varphi_1 \rrbracket \cup \llbracket \varphi_2 \rrbracket$$

Definition: Alternative syntax

State formulae $\varphi ::= \perp \mid p \ (p \in AP) \mid \neg\varphi \mid \varphi \vee \varphi \mid E\psi \mid A\psi$
 Path formulae $\psi ::= \varphi \mid \neg\psi \mid \psi \vee \psi \mid X\psi \mid \psi U \psi$

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 $M, \sigma, 0 \models \varphi$ for some initial infinite run σ of M .
 $M \models_{\forall} \varphi$ if $M, \sigma, 0 \models \varphi$ for all initial infinite run σ of M .

Remark:

- $M \models_{\exists} \varphi$ iff $I \cap \llbracket E\varphi \rrbracket \neq \emptyset$
- $M \models_{\forall} \varphi$ iff $I \subseteq \llbracket A\varphi \rrbracket$
- $M \models_{\forall} \varphi$ iff $M \not\models_{\exists} \neg\varphi$

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

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

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

Complexity of CTL*

Definition: Syntax of the Computation Tree Logic CTL*

$$\varphi ::= \perp \mid p \ (p \in AP) \mid \neg\varphi \mid \varphi \vee \varphi \mid X\varphi \mid \varphi \cup \varphi \mid E\varphi \mid A\varphi$$

Theorem

The model checking problem for CTL* is PSPACE-complete

Proof:

PSPACE-hardness: follows from $LTL \subseteq CTL^*$.

PSPACE-easiness: reduction to LTL-model checking by inductive eliminations of path quantifications.

$MC_{CTL^*}^{\forall}$ in PSPACE

Proof:

For $Q \in \{\exists, \forall\}$ and $\psi \in LTL$, let $MC_{LTL}^Q(M, t, \psi)$ be the function which computes in polynomial space whether $M, t \models_Q \psi$, i.e., if $M, t \models Q\psi$.

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

$MC_{CTL^*}^{\forall}(M, s, \varphi)$

If E, A do not occur in φ then return $MC_{LTL}^{\forall}(M, s, \varphi)$ fi

Let $Q\psi$ be a subformula of φ with $\psi \in LTL$ and $Q \in \{E, A\}$

Let $p_{Q\psi}$ be a new propositional variable

Define $\ell' : S \rightarrow 2^{AP'}$ with $AP' = AP \uplus \{p_{Q\psi}\}$ by

$$\ell'(t) \cap AP = \ell(t) \text{ and } p_{Q\psi} \in \ell'(t) \text{ iff } MC_{LTL}^Q(M, t, \psi)$$

Let $M' = (S, T, I, AP', \ell')$

Let $\varphi' = \varphi[p_{Q\psi}/Q\psi]$ be obtained from φ by replacing each $Q\psi$ by $p_{Q\psi}$

Return $MC_{CTL^*}^{\forall}(M', s, \varphi')$

Satisfiability for CTL*

Definition: SAT(CTL*)

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

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

Theorem

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

CTL (Clarke & Emerson 81)

Definition: Computation Tree Logic (CTL)

Syntax:

$$\varphi ::= \perp \mid p \ (p \in AP) \mid \neg\varphi \mid \varphi \vee \varphi \mid EX\varphi \mid AX\varphi \mid E\varphi \cup \varphi \mid A\varphi \cup \varphi$$

The semantics is inherited from CTL*.

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

$$\llbracket \varphi \rrbracket^M = \{s \in S \mid M, s \models \varphi\}$$

Examples: Macros

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

CTL (Clarke & Emerson 81)

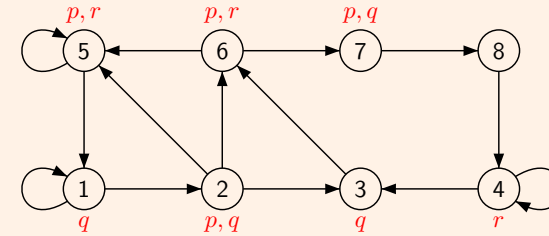
Definition: Semantics

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

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

CTL (Clarke & Emerson 81)

Example:



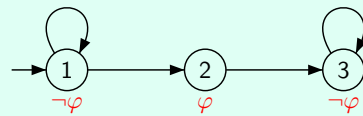
- $\llbracket \text{EX } p \rrbracket = \{1, 2, 3, 5, 6\}$
- $\llbracket \text{AX } p \rrbracket = \{3, 6\}$
- $\llbracket \text{EF } p \rrbracket = \{1, 2, 3, 4, 5, 6, 7, 8\}$
- $\llbracket \text{AF } p \rrbracket = \{2, 3, 5, 6, 7\}$
- $\llbracket \text{E } q \text{ U } r \rrbracket = \{1, 2, 3, 4, 5, 6\}$
- $\llbracket \text{A } q \text{ U } r \rrbracket = \{2, 3, 4, 5, 6\}$

CTL (Clarke & Emerson 81)

Remark: Equivalent formulae

- $\text{AX } \varphi = \neg \text{EX } \neg \varphi$,
- $\neg(\varphi \text{ U } \psi) = \text{G } \neg \psi \vee (\neg \psi \text{ U } (\neg \varphi \wedge \neg \psi))$
- $\text{A } \varphi \text{ U } \psi = \neg \text{EG } \neg \psi \wedge \neg \text{E } \neg \psi \text{ U } (\neg \varphi \wedge \neg \psi)$
- $\text{AG}(\text{req} \rightarrow \text{F grant}) = \text{AG}(\text{req} \rightarrow \text{AF grant})$
- $\text{AGF } \varphi = \text{AG AF } \varphi$
- $\text{EFG } \varphi = \text{EFEG } \varphi$
- $\text{EGEF } \varphi \neq \text{EGF } \varphi$
- $\text{AFAG } \varphi \neq \text{AFG } \varphi$
- $\text{EGEX } \varphi \neq \text{EGX } \varphi$

infinitely often
ultimately



Model checking of CTL

Definition: Existential and universal model checking

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

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

Remark:

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

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

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

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

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 $M \models \varphi$ is decidable in time $\mathcal{O}(|M| \cdot |\varphi|)$

Proof:

Compute $\llbracket \varphi \rrbracket = \{s \in S \mid M, s \models \varphi\}$ by induction on the formula.

The set $\llbracket \varphi \rrbracket$ is represented by a boolean array: $L[s][\varphi] = \top$ if $s \in \llbracket \varphi \rrbracket$.

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

Model checking of CTL

Definition: procedure semantics(φ)

case $\varphi = \neg\varphi_1$
 semantics(φ_1)
 $\llbracket \varphi \rrbracket := S \setminus \llbracket \varphi_1 \rrbracket$ $\mathcal{O}(|S|)$

case $\varphi = \varphi_1 \vee \varphi_2$
 semantics(φ_1); semantics(φ_2)
 $\llbracket \varphi \rrbracket := \llbracket \varphi_1 \rrbracket \cup \llbracket \varphi_2 \rrbracket$ $\mathcal{O}(|S|)$

case $\varphi = EX\varphi_1$
 semantics(φ_1)
 $\llbracket \varphi \rrbracket := \emptyset$ $\mathcal{O}(|S|)$
 for all $(s, t) \in T$ do if $t \in \llbracket \varphi_1 \rrbracket$ then $\llbracket \varphi \rrbracket := \llbracket \varphi \rrbracket \cup \{s\}$ $\mathcal{O}(|T|)$

case $\varphi = AX\varphi_1$
 semantics(φ_1)
 $\llbracket \varphi \rrbracket := S$ $\mathcal{O}(|S|)$
 for all $(s, t) \in T$ do if $t \notin \llbracket \varphi_1 \rrbracket$ then $\llbracket \varphi \rrbracket := \llbracket \varphi \rrbracket \setminus \{s\}$ $\mathcal{O}(|T|)$

Model checking of CTL

Definition: procedure semantics(φ)

case $\varphi = E\varphi_1 U \varphi_2$ $\mathcal{O}(|S| + |T|)$
 semantics(φ_1); semantics(φ_2)
 $L := \llbracket \varphi_2 \rrbracket$ // the set L is the "todo" list $\mathcal{O}(|S|)$
 $Z := \emptyset$ // the set Z is the "done" list $\mathcal{O}(|S|)$
 while $L \neq \emptyset$ do $|S|$ times
 Invariant: $\llbracket \varphi_2 \rrbracket \cup (\llbracket \varphi_1 \rrbracket \cap T^{-1}(Z)) \subseteq Z \cup L \subseteq \llbracket E\varphi_1 U \varphi_2 \rrbracket$
 take $t \in L$; $L := L \setminus \{t\}$; $Z := Z \cup \{t\}$ $\mathcal{O}(1)$
 for all $s \in T^{-1}(t)$ do $|T|$ times
 if $s \in \llbracket \varphi_1 \rrbracket \setminus (Z \cup L)$ then $L := L \cup \{s\}$
 $\llbracket \varphi \rrbracket := Z$

Z is only used to make the invariant clear.
 $Z \cup L$ can be replaced by $\llbracket \varphi \rrbracket$.

Model checking of CTL

Definition: procedure semantics(φ)

Replacing $Z \cup L$ by $\llbracket \varphi \rrbracket$

case $\varphi = E\varphi_1 U \varphi_2$ $\mathcal{O}(|S| + |T|)$
 semantics(φ_1); semantics(φ_2)
 $L := \llbracket \varphi_2 \rrbracket$ // the set L is implemented with a list $\mathcal{O}(|S|)$
 $\llbracket \varphi \rrbracket := \llbracket \varphi_2 \rrbracket$ $\mathcal{O}(|S|)$
 while $L \neq \emptyset$ do $|S|$ times
 take $t \in L$; $L := L \setminus \{t\}$ $\mathcal{O}(1)$
 for all $s \in T^{-1}(t)$ do $|T|$ times
 if $s \in \llbracket \varphi_1 \rrbracket \setminus \llbracket \varphi \rrbracket$ then $L := L \cup \{s\}$; $\llbracket \varphi \rrbracket := \llbracket \varphi \rrbracket \cup \{s\}$ $\mathcal{O}(1)$

Model checking of CTL

Definition: procedure semantics(φ)

```

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

```

Z is only used to make the invariant clear.
 $Z \cup L$ can be replaced by $\llbracket \varphi \rrbracket$.

Model checking of CTL

Definition: procedure semantics(φ)

Replacing $Z \cup L$ by $\llbracket \varphi \rrbracket$

```

case  $\varphi = A\varphi_1 \cup \varphi_2$   $\mathcal{O}(|S| + |T|)$ 
  semantics( $\varphi_1$ ); semantics( $\varphi_2$ )
   $L := \llbracket \varphi_2 \rrbracket$  // the set  $L$  is implemented with a list  $\mathcal{O}(|S|)$ 
   $\llbracket \varphi \rrbracket := \llbracket \varphi_2 \rrbracket$   $\mathcal{O}(|S|)$ 
  for all  $s \in S$  do  $c[s] := |T(s)|$   $\mathcal{O}(|S|)$ 
  while  $L \neq \emptyset$  do  $|S|$  times
    take  $t \in L; L := L \setminus \{t\}$   $\mathcal{O}(1)$ 
    for all  $s \in T^{-1}(t)$  do  $|T|$  times
       $c[s] := c[s] - 1$   $\mathcal{O}(1)$ 
      if  $c[s] = 0 \wedge s \in \llbracket \varphi_1 \rrbracket \setminus \llbracket \varphi \rrbracket$  then  $\mathcal{O}(1)$ 
         $L := L \cup \{s\}; \llbracket \varphi \rrbracket := \llbracket \varphi \rrbracket \cup \{s\}$   $\mathcal{O}(1)$ 

```

Complexity of CTL

Definition: SAT(CTL)

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

Question: Existence of a model M and a state s such that $M, s \models \varphi$?

Theorem: Complexity

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

fairness

Example: Fairness

Only fair runs are of interest

- ▶ Each process is enabled infinitely often: $\bigwedge_i \text{GF run}_i$
- ▶ No process stays ultimately in the critical section: $\bigwedge_i \neg \text{FG CS}_i = \bigwedge_i \text{GF } \neg \text{CS}_i$

Definition: Fair Kripke structure

$M = (S, T, I, \text{AP}, \ell, F_1, \dots, F_n)$ with $F_i \subseteq S$.

An infinite run σ is **fair** if it visits infinitely often each F_i

fair CTL

Definition: Syntax of fair-CTL

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

Definition: Semantics as a fragment of CTL*

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

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

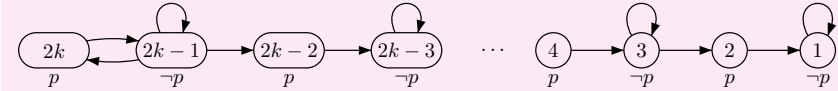
where $\text{fair} = \bigwedge_i G F F_i$

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:



• $M_k, 2k \models EGF p$ but $M_k, 2k-2 \not\models EGF p$

• 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 CTL_f

Theorem

The model checking problem for CTL_f is decidable in time $\mathcal{O}(|M| \cdot |\varphi|)$

Proof: Computation of $\text{Fair} = \{s \in S \mid M, s \models E_f \top\}$

Compute the SCC of M with **Tarjan's algorithm** (in time $\mathcal{O}(|M|)$).

Let S' be the union of the (non trivial) SCCs which intersect each F_i .

Then, Fair is the set of states that can reach S' .

Note that **reachability** can be computed in linear time.

Model checking of CTL_f

Proof: Reductions

$E_f X \varphi = EX(\text{Fair} \wedge \varphi)$ and $E_f \varphi U \psi = E \varphi U (\text{Fair} \wedge \psi)$

It remains to deal with $A_f \varphi U \psi$.

Recall that $A \varphi U \psi = \neg EG \neg\psi \wedge \neg E \neg\psi U (\neg\varphi \wedge \neg\psi)$

This formula also holds for fair quantifications A_f and E_f .

Hence, we only need to compute the semantics of $E_f G \varphi$.

Proof: Computation of $E_f G \varphi$

Let M_φ be the restriction of M to $\llbracket \varphi \rrbracket_f$.

Compute the SCC of M_φ with **Tarjan's algorithm** (in linear time).

Let S' be the union of the (non trivial) SCCs of M_φ which intersect each F_i .

Then, $M, s \models E_f G \varphi$ iff $M, s \models E \varphi U S'$ iff $M_\varphi, s \models EF S'$.

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