Department of Computer Science, University of Udine

The Safety Fragment of Temporal Logics on Infinite Sequences

Luca Geatti
luca.geatti@uniud.it
Angelo Montanari
angelo.montanari@uniud.it

July, 31st - August, 4th 2023

INTRODUCTION

Temporal Logics

Temporal logics are mathematical formalisms to reason about time.

They are extensively used in some of the main fields of Computer Science and Artificial Intelligence (AI), including, for instance, formal verification and machine learning.

Temporal Logics

Temporal logics are mathematical formalisms to reason about time.

Temporal logics are traditionally partitioned into:

- those modeling time as a *linear order* (i.e., a sequence),
- or as a *tree*

Linear Temporal Logic (LTL) is the de-facto standard for reasoning over infinite linear time.

Reference

Amir Pnueli (1977). "The temporal logic of programs". In: *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*. IEEE, pp. 46–57. DOI: 10.1109/SFCS.1977.32

Formal Verification

While simulation and testing explore *some* of the possible behaviors and scenarios of a system, leaving the question of whether the unexplored trajectories may contain the fatal bug open, formal verification conducts an *exhaustive exploration* of all possible behaviors.

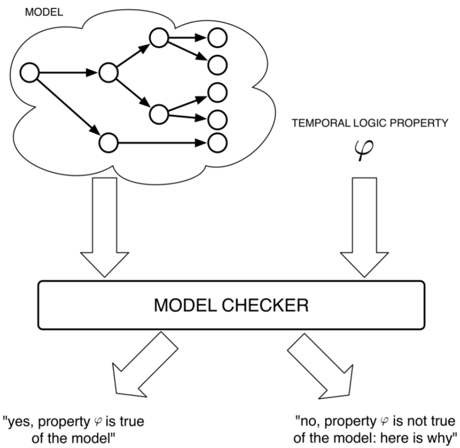
Reference

Edmund M Clarke et al. (2018). *Model checking*. MIT press

Important techniques in formal verification:

- consistency checking
- model checking
- reactive synthesis
- ...

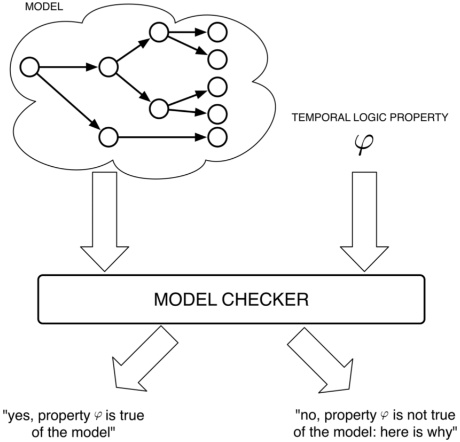
Relationship between Linear Temporal Logic and Formal Verification

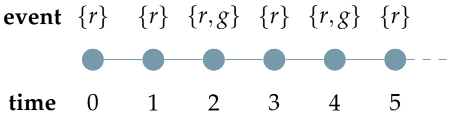


We are interested in *infinite linear time*: specification of *Reactive Systems*.

Picture taken from: Alessandro Abate et al. (2021). "Rational verification: game-theoretic verification of multi-agent systems". In: *Applied Intelligence* 51.9, pp. 6569–6584

Relationship between Linear Temporal Logic and Formal Verification





r = request
g = grant

The Safety Fragment

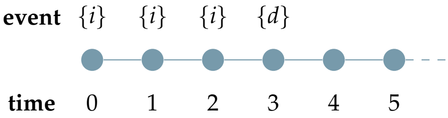
The safety fragment includes those properties stating that *"something bad never happens"*, like, for instance, a deadlock or a simultaneous access to a critical section.

The Cosafety Fragment

The cosafety fragment is the dual of the safety one. It is defined as the set of properties asking that *"something good will eventually happen"*, *e.g.*, termination of a program.

Safety

Property: *"The program never enters a deadlock"*

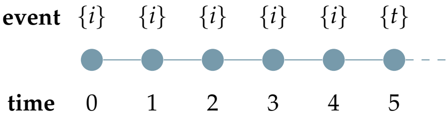


i = instruction
d = deadlock

Given a safety property, a prefix of a sequence suffices to establish whether it *does not* satisfy the property.

Cosafety

Property: *"The program terminates"*



i = instruction
t = termination

Given a cosafety property, a prefix of a sequence suffices to establish whether it *does* satisfy the property.

A crucial feature of both the safety fragment and the cosafety one is that they allow one to reason on *finite sequences* instead of infinite ones.

This feature has been exploited to design efficient techniques in formal verification:

- Model Checking
  - we can exploit (forward or background) reachability analysis, that is, reachability of an error state (*invariance checking*)
  - a counterexample is always a finite trace: often more helpful than an infinite errore trace

A crucial feature of both the safety fragment and the cosafety one is that they allow one to reason on *finite sequences* instead of infinite ones.

This feature has been exploited to design efficient techniques in formal verification:

- Monitoring
  - model checking is not always applicable (the system is too complex, some parts of the system are not observable, etc.);
  - *runtime verification* and *monitoring* are viable alternatives: we can monitor at runtime the trace generated so far by the system (such a trace is always finite);
  - we cannot monitor arbitrary properties;
  - safety and cosafety properties are monitorable

A crucial feature of both the safety fragment and the cosafety one is that they allow one to reason on *finite sequences* instead of infinite ones.

This feature has been exploited to design efficient techniques in formal verification:

- Reactive Synthesis
  - determinization can be done using classic *subset construction* instead of the complicated Safra's construction
- ...

Outline

Outline

BACKGROUND

Finite and Infinite Words

We fix a finite alphabet Σ.

Finite Words

- Modal interpretation:
- First-order interpretation:

Infinite Words

- Modal interpretation:
- First-order interpretation:

Finite and Infinite Words

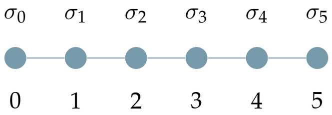
We fix a finite alphabet Σ.

Finite Words

- Modal interpretation:

$$\sigma \in \Sigma^*$$

$$\sigma = \langle \sigma_0, \ldots, \sigma_n \rangle \text{ for some } n \in \mathbb{N}$$

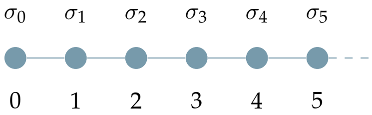


Infinite Words

- Modal interpretation:

$$\sigma \in \Sigma^\omega$$

$$\sigma = \langle \sigma_0, \sigma_1, \sigma_2, \ldots \rangle$$



Finite and Infinite Words

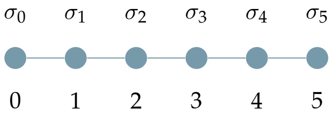
We fix a finite alphabet Σ.

Finite Words

- Modal interpretation:

$$\sigma \in \Sigma^*$$

$$\sigma = \langle \sigma_0, \ldots, \sigma_n \rangle \text{ for some } n \in \mathbb{N}$$



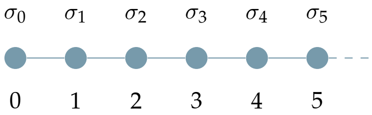
- *length* of σ: $|\sigma| = n + 1$
- word = synonym of finite word

Infinite Words

- Modal interpretation:

$$\sigma \in \Sigma^\omega$$

$$\sigma = \langle \sigma_0, \sigma_1, \sigma_2, \ldots \rangle$$



- *length* of σ: $|\sigma| = \omega$
- ω-word = synonym of infinite word

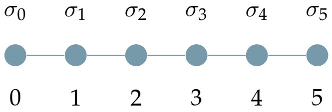
Finite and Infinite Words

We fix a finite alphabet Σ.

Finite Words

- Modal interpretation:

$$\sigma \in \Sigma^*$$
$$\sigma = \langle \sigma_0, \ldots, \sigma_n \rangle \text{ for some } n \in \mathbb{N}$$



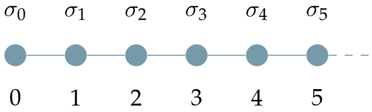
Example

$\Sigma := \{a, b\}$ and $\sigma = ababab$

Infinite Words

- Modal interpretation:

$$\sigma \in \Sigma^\omega$$
$$\sigma = \langle \sigma_0, \sigma_1, \sigma_2, \ldots \rangle$$



Example

$\Sigma := \{a, b, c\}$ and $\sigma = aaabaaacaaab \ldots$

Finite and Infinite Words

We fix a finite alphabet Σ.

Finite Words

- Modal interpretation: …
- First-order interpretation:

$$\langle D, 0, +1, <, =, \{P\}_{P \in \Sigma} \rangle$$

  - $D = [a, b]$ (for some $a, b \in \mathbb{N}$) is the *domain*
  - the constant 0, the +1 function, and the relations $<$ and $=$ have their natural interpretation
  - each P is a *unary predicate*

Infinite Words

- Modal interpretation: …
- First-order interpretation:

$$\langle \mathbb{N}, 0, +1, <, =, \{P\}_{P \in \Sigma} \rangle$$

  - $\mathbb{N}$ is the *domain*
  - the constant 0, the +1 function, and the relations $<$ and $=$ have their natural interpretation
  - each P is a *unary predicate*

Finite Words

- A *regular expression* is an expression built starting from
  - $\varnothing$: the empty set
  - ε: the word of length 0
  - a (for some $a \in \Sigma$): any word of lenght 1

  using the following operations:
  - $L_1 \cup L_2$: union
  - $L_1 \cdot L_2$: concatenation
  - $\overline{L}$: complementation
  - L^*: Kleene's star
- Example: $a^* \cdot b \cdot \Sigma^*$

- A *language* is a set of finite words.
- A *regular language* is a language that can be built using a regular expression.
- We denote with RE the set of regular languages.

Infinite Words

- Given a regular language L, we define its *ω-closure*, denoted by $(L)^\omega$, as the set of ω-words built starting from elements in L.
- A *ω-regular expression* is an expression of the form:

$$\bigcup_{i=1,\ldots,n} U_i \cdot (V_i)^\omega$$

where U_i and V_i are regular languages, for $i = 1, \ldots, n$.

Example

$(a^* \cdot b) \cdot (\Sigma)^\omega$

- A *ω-language* is a set of ω-words.
- A *ω-regular language* is a ω-language that can be built using a ω-regular expression.
- We denote with ω-RE the set of ω-regular languages.

Finite Words

- A regular expression is called *star-free* iff it is devoid of Kleene's star.
- A regular language is called *star-free* iff it can be built by a star-free regular expression.
- We call SF the set of star-free regular languages.

Example

$$\Sigma^* \cdot a \cdot \Sigma^* \cdot b \cdot \Sigma^*$$

Note that $\Sigma^* := \overline{\varnothing}$.

Infinite Words

- An ω-regular expression is called *star-free* iff it is of the form:

$$\bigcup_{i=1,\ldots,n} U_i \cdot (V_i)^\omega$$

  where U_i and V_i are star-free regular expressions, for $i = 1, \ldots, n$.
- An ω-regular language is called *star-free* iff it can be built by a star-free ω-regular expression.
- We call ω-SF the set of star-free ω-regular languages.

The *monadic second-order theory of one successor* (S1S, for short) is a fragment of second-order logic in which we fix this alphabet:

$$\underbrace{0}_{\text{constant}}, \underbrace{+1}_{\text{function}}, \underbrace{<, =}_{\substack{\text{binary}\\\text{predicates}}}, \underbrace{\{P\}_{P \in \Sigma}}_{\substack{\text{unary}\\\text{predicates}}}$$

The *monadic second-order theory of one successor* (S1S, for short) is a fragment of second-order logic in which we fix this alphabet:

$$\underbrace{0}_{\text{constant}}, \underbrace{+1}_{\text{function}}, \underbrace{<, =}_{\substack{\text{binary}\\\text{predicates}}}, \underbrace{\{P\}_{P\in\Sigma}}_{\substack{\text{unary}\\\text{predicates}}}$$

Its syntax is the following. Let $\mathcal{V} = \{x, y, z, \ldots\}$ be a set of *first-order variables*. Let $\mathcal{V}' = \{X, Y, Z, \ldots\}$ be a set of *second-order variables*.

$$\text{(terms)} \quad t \coloneqq x \mid 0 \mid t+1$$

$$\text{(formulas)} \quad \phi \coloneqq \underbrace{P(t)}_{\text{with } P\in\Sigma} \mid \underbrace{X(t)}_{\substack{\text{with } X\\\text{monadic}\\\text{variable}}} \mid t < t' \mid t = t' \mid \neg\phi \mid \phi \vee \phi \mid \underbrace{\exists x \,.\, \phi}_{\substack{\text{first-order}\\\text{quantifier}}} \mid \underbrace{\exists X \,.\, \phi}_{\substack{\text{monadic}\\\text{second-order}\\\text{quantifier}}}$$

The *monadic second-order theory of one successor* (S1S, for short) is a fragment of second-order logic in which we fix this alphabet:

$$\underbrace{0}_{\text{constant}}, \underbrace{+1}_{\text{function}}, \underbrace{<, =}_{\substack{\text{binary}\\ \text{predicates}}}, \underbrace{\{P\}_{P\in\Sigma}}_{\substack{\text{unary}\\ \text{predicates}}}$$

Semantics:

Words

$$\langle D, 0, +1, <, =, \{P\}_{P\in\Sigma}\rangle$$

ω-Words

$$\langle \mathbb{N}, 0, +1, <, =, \{P\}_{P\in\Sigma}\rangle$$

The *monadic second-order theory of one successor* (S1S, for short) is a fragment of second-order logic in which we fix this alphabet:

$$\underbrace{0}_{\text{constant}}, \underbrace{+1}_{\text{function}}, \underbrace{<, =}_{\substack{\text{binary}\\\text{predicates}}}, \underbrace{\{P\}_{P\in\Sigma}}_{\substack{\text{unary}\\\text{predicates}}}$$

- Let $\phi(x, y, z, X, Y, Z, \ldots)$ be an S1S formula with free variables $x, y, z, X, Y, Z, \ldots$ and let ρ be a variable evaluation function.
- We write $\langle D, 0, +1, <, =, \{P\}_{P\in\Sigma}\rangle, \rho \models \phi(x, y, z, X, Y, Z, \ldots)$ to denote the fact that the finite word $\langle D, 0, +1, <, =, \{P\}_{P\in\Sigma}\rangle$ *satisfies* $\phi(x, y, z, X, Y, Z, \ldots)$ under the evaluation ρ of the free variables.
- The same holds for ω-words $\langle \mathbb{N}, 0, +1, <, =, \{P\}_{P\in\Sigma}\rangle$.

Example

There exists a position in which both P_1 and P_2 hold.

$$\exists x \, . \, (P_1(x) \wedge P_2(x))$$

Example

Each position where P_1 holds is followed by a position where P_2 holds (by using $+1$ and second-order quantification).

$$\forall x \, . \, \Big(P_1(x) \rightarrow \forall X \, . \, \big(X(x) \wedge \forall y \, . \, (X(y) \rightarrow X(y+1)) \rightarrow \exists z \, . \, (X(z) \wedge P_2(z))\big)\Big)$$

- We call S1S[FO] (the *first-order* fragment of S1S) the fragment of S1S devoid of second-order quantifiers.
- We denote with $\mathsf{S1S_f}$ the logic S1S interpreted over *finite words*.

- We are interested on S1S[FO] formula $\phi(x)$ with *exactly one free variable* x.
  - x is meant to represent the initial time point.
- The *language over finite words* of $\phi(x)$, denoted with $\mathcal{L}^{<\omega}(\phi(x))$ is defined as:

$$\mathcal{L}^{<\omega}(\phi(x)) \coloneqq \left\{ \langle D, 0, +1, <, =, \{P\}_{P\in\Sigma} \rangle, x \mapsto 0 \models \phi(x) \right\}$$

- The *language over ω-words* of $\phi(x)$, denoted with $\mathcal{L}(\phi(x))$ is defined as:

$$\mathcal{L}(\phi(x)) \coloneqq \left\{ \langle D, 0, +1, <, =, \{P\}_{P\in\Sigma} \rangle, x \mapsto 0 \models \phi(x) \right\}$$

Theorem (Büchi's Theorem over ω-words)

- *For each* S1S *formula* ϕ*, the language* $\mathcal{L}(\phi)$ *is an* ω*-regular language.*
- *For each* ω*-regular language* $\mathcal{L}$*, there exists an* S1S *formula* ϕ *such that* $\mathcal{L} = \mathcal{L}(\phi)$*.*

Theorem (Büchi's Theorem over finite words)

- *For each* $\mathsf{S1S_f}$ *formula* ϕ*, the language* $\mathcal{L}^{<\omega}(\phi)$ *is a regular language.*
- *For each regular language* $\mathcal{L}$*, there exists an* $\mathsf{S1S_f}$ *formula* ϕ *such that* $\mathcal{L} = \mathcal{L}^{<\omega}(\phi)$*.*

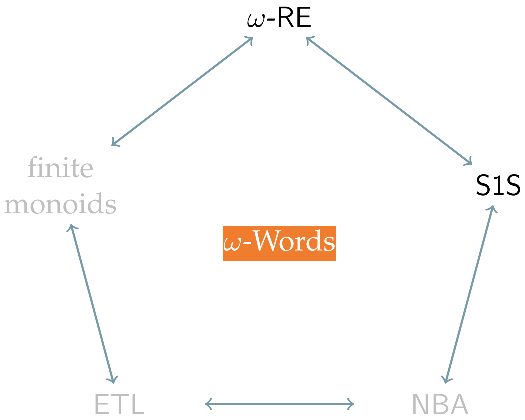
Reference:

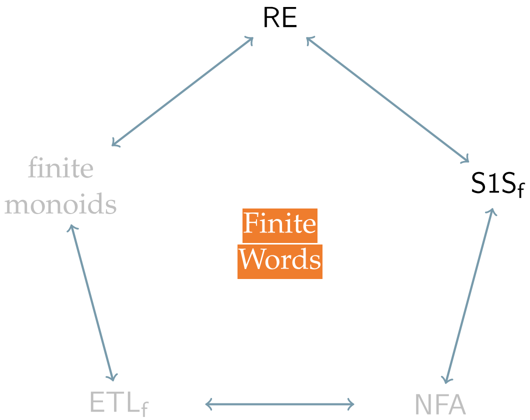
J. R. Buechi (1960). "On a decision method in restricted second-order arithmetics". In: *Proc. Internat. Congr. on Logic, Methodology and Philosophy of Science, 1960*

Reference:

Calvin C Elgot (1961). "Decision problems of finite automata design and related arithmetics". In: *Transactions of the American Mathematical Society* 98.1, pp. 21–51. DOI: 10.1090/S0002-9947-1961-0139530-9

Characterizations of ω-Regular Languages




RE
finite monoids
S1S$_f$
Finite Words
ETL$_f$
NFA

Theorem (Expressive Equivalence over ω-words)

- *For each* S1S[FO] *formula* ϕ*, the language* $\mathcal{L}(\phi)$ *is a star-free* ω*-language.*
- *For each star-free* ω*-language* $\mathcal{L}(\phi)$*, there exists an* S1S[FO] *formula* ϕ *such that* $\mathcal{L} = \mathcal{L}(\phi)$.

Theorem (Expressive Equivalence over finite words)

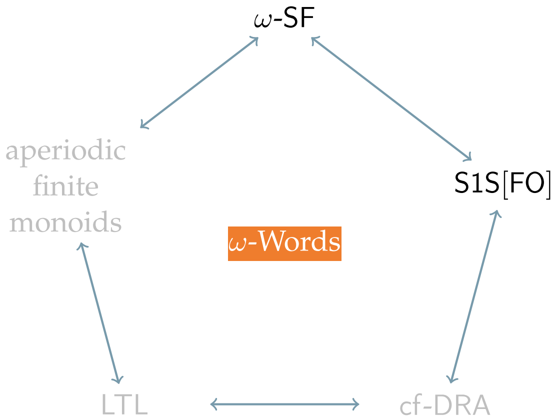
- *For each* $\mathsf{S1S[FO]_f}$ *formula* ϕ*, the language* $\mathcal{L}^{<\omega}(\phi)$ *is a star-free language.*
- *For each star-free language* $\mathcal{L}$*, there exists an* $\mathsf{S1S[FO]_f}$ *formula* ϕ *such that* $\mathcal{L} = \mathcal{L}^{<\omega}(\phi)$.

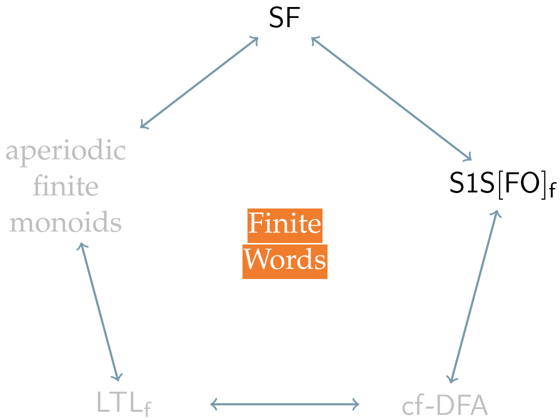
Reference:

Richard E Ladner (1977). "Application of model theoretic games to discrete linear orders and finite automata". In: *Information and Control* 33.4, pp. 281–303. DOI: 10.1016/S0019-9958(77)90443-0

Reference:

Wolfgang Thomas (1981). "A combinatorial approach to the theory of ω-automata". In: *Information and Control* 48.3, pp. 261–283. DOI: 10.1016/S0019-9958(81)90663-X

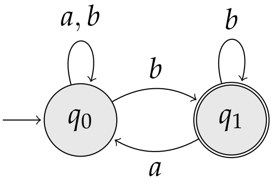

ω-SF
aperiodic
finite
monoids
S1S[FO]
ω-Words
LTL
cf-DRA


SF
aperiodic
finite
monoids
S1S[FO]f
Finite
Words
LTLf
cf-DFA

Definition (Nondeterministic Automaton)

A *nondeterministic automaton* $\mathcal{A}$ is a tuple $\langle Q, \Sigma, I, \Delta, F\rangle$ where:

- Q is the *set of states*;
- Σ is the *alphabet*;
- $I \subseteq Q$ is the set of *initial states*;
- $\Delta \subseteq Q \times \Sigma \times Q$ is the *transition relation*;
- $F \subseteq Q$ is the set of *final states*;



- $Q = \{q_0, q_1\}$;
- $\Sigma = \{a, b\}$;
- $I = \{q_0\}$;
- $\Delta = \{(q_0, a, q_0), (q_0, b, q_0), (q_0, b, q_1), (q_1, b, q_1), (q_1, a, q_0)\}$;
- $F = \{q_1\}$;

Definition (Nondeterministic Automaton)

A *nondeterministic automaton* $\mathcal{A}$ is a tuple $\langle Q, \Sigma, I, \Delta, F \rangle$ where:

- Q is the *set of states*;
- Σ is the *alphabet*;
- $I \subseteq Q$ is the set of *initial states*;
- $\Delta \subseteq Q \times \Sigma \times Q$ is the *transition relation*;
- $F \subseteq Q$ is the set of *final states*;

A nondeterministic automaton is *deterministic* iff Δ is a *function*, that is:

$$|\Delta(q, a)| = 1 \qquad \text{for each } q \in Q, a \in \Sigma$$

Let $\mathcal{A}$ be a nondeterministic automaton with alphabet Σ.

Words

- Given a word $\sigma \in \Sigma^*$ with $\sigma = \langle \sigma_0, \sigma_1, \ldots, \sigma_n \rangle$, a *run* π *of* $\mathcal{A}$ *over* σ is a finite sequence of states $\langle q_0, q_1, \ldots, q_{n+1} \rangle \in Q^*$ such that:
  - $q_0 \in I$;
  - $(q_i, \sigma_i, q_{i+1}) \in \Delta$, for each $0 \leq i \leq n$

ω-Words

- Given an ω-word $\sigma \in \Sigma^\omega$ with $\sigma = \langle \sigma_0, \sigma_1, \ldots \rangle$, a *run* π *of* $\mathcal{A}$ *over* σ is an infinite sequence of states $\langle q_0, q_1, \ldots \rangle \in Q^\omega$ such that:
  - $q_0 \in I$;
  - $(q_i, \sigma_i, q_{i+1}) \in \Delta$, for each $i \geq 0$

Let $\mathcal{A}$ be a nondeterministic automaton with alphabet Σ.

Words

Definition (NFA)

A *Nondeterministic Finite Automaton* (NFA, for short) $\langle Q, \Sigma, I, \Delta, F\rangle$ is a nondeterministic automaton in which a run $\pi := \langle \pi_0, \ldots, \pi_{n+1}\rangle \in Q^*$ is said to be *accepting* iff $\pi_{n+1} \in F$.

ω-Words

Definition (NBA)

A *Nondeterministic Büchi Automaton* (NBA, for short) $\langle Q, \Sigma, I, \Delta, F\rangle$ is a nondeterministic automaton in which a run $\pi := \langle \pi_0, \pi_1, \ldots\rangle \in Q^\omega$ is said to be *accepting* iff $\mathsf{Inf}(\pi) \cap F \neq \varnothing$.

$\mathsf{Inf}(\pi)$ is the set of states that occur infinitely often in the infinite run π.

Let $\mathcal{A}$ be a nondeterministic automaton with alphabet Σ.

Words

Definition (NFA)

A *Nondeterministic Finite Automaton* (NFA, for short) $\langle Q, \Sigma, I, \Delta, F \rangle$ is a nondeterministic automaton in which a run $\pi := \langle \pi_0, \ldots, \pi_{n+1} \rangle \in Q^*$ is said to be *accepting* iff $\pi_{n+1} \in F$.

ω-Words

Definition (NBA)

A *Nondeterministic Büchi Automaton* (NBA, for short) $\langle Q, \Sigma, I, \Delta, F \rangle$ is a nondeterministic automaton in which a run $\pi := \langle \pi_0, \pi_1, \ldots \rangle \in Q^\omega$ is said to be *accepting* iff $\mathsf{Inf}(\pi) \cap F \neq \varnothing$.

A run is accepting for a Büchi automaton iff it reaches a final state infinitely often.

Let $\mathcal{A}$ be a nondeterministic automaton with alphabet Σ.

Words

Let $\mathcal{A} = \langle Q, \Sigma, I, \Delta, F \rangle$ be an NFA.

- A word $\sigma \in \Sigma^*$ is *accepted* by $\mathcal{A}$ iff there exists at least one accepting run of $\mathcal{A}$ over σ.
- The *language of* $\mathcal{A}$, denoted as $\mathcal{L}^{<\omega}(\mathcal{A})$, is the set of words in Σ^* accepted by $\mathcal{A}$.

ω-Words

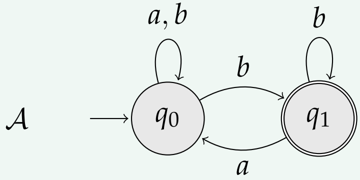
Let $\mathcal{A} = \langle Q, \Sigma, I, \Delta, F \rangle$ be an NBA.

- An ω-word $\sigma \in \Sigma^\omega$ is *accepted* by $\mathcal{A}$ iff there exists at least one accepting run of $\mathcal{A}$ over σ.
- The *language of* $\mathcal{A}$, denoted as $\mathcal{L}(\mathcal{A})$, is the set of ω-words in Σ^ω accepted by $\mathcal{A}$.

Let $\mathcal{A}$ be a nondeterministic automaton with alphabet Σ.

Words

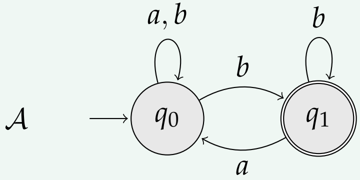
Example



$\mathcal{L}^{<\omega}(\mathcal{A}) = \{\sigma \in \Sigma^* \mid$ each a is eventually followed by b$\}$

ω-Words

Example



$\mathcal{L}(\mathcal{A}) = \{\sigma \in \Sigma^\omega \mid$ each a is eventually followed by b$\}$

An important difference

NFA

- A DFA is a deterministic NFA
- NFA are closed under *determinization*: for each NFA $\mathcal{A}$ there exists a DFA $\mathcal{A}'$ such that $\mathcal{L}^{<\omega}(\mathcal{A}) = \mathcal{L}^{<\omega}(\mathcal{A}')$.
- Subset construction.

NBA

- A DBA is a deterministic NBA
- NBA are not closed under *determinization*: there exists a NBA $\mathcal{A}$ for which all DBA $\mathcal{A}'$ are such that $\mathcal{L}(\mathcal{A}) \neq \mathcal{L}(\mathcal{A}')$.

An important difference

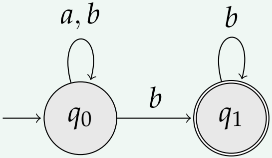
NFA

- A DFA is a deterministic NFA
- NFA are closed under *determinization*: for each NFA $\mathcal{A}$ there exists a DFA $\mathcal{A}'$ such that $\mathcal{L}^{<\omega}(\mathcal{A}) = \mathcal{L}^{<\omega}(\mathcal{A}')$.
- Subset construction.

NBA

Example

Let $\Sigma := \{a, b\}$. The language $\mathcal{L} = \{\sigma \in \Sigma^\omega \mid \exists^{<\omega} i \,.\, \sigma_i = a\}$ is not accepted by any DBA. However, it is accepted by the following NBA.



Theorem (Expressive Equivalence for NBA)

For each ω-language $\mathcal{L} \subseteq \Sigma^\omega$, it holds that:

$$\mathcal{L} \text{ is } \omega\text{-regular}$$
iff
$$\mathcal{L} = \mathcal{L}(\mathcal{A}) \text{ for some NBA } \mathcal{A}$$

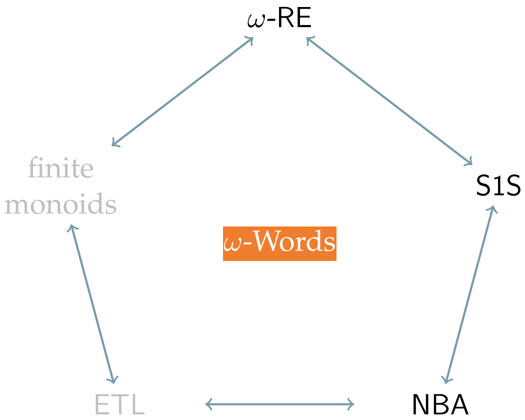
Theorem (Expressive Equivalence for NFA/DFA)

For each language $\mathcal{L} \subseteq \Sigma^*$, it holds that:

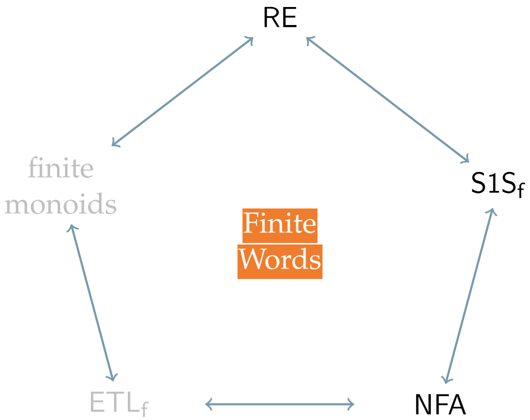
$$\mathcal{L} \text{ is regular}$$
iff
$$\mathcal{L} = \mathcal{L}^{<\omega}(\mathcal{A}) \text{ for some NFA/DFA } \mathcal{A}$$

Reference:

Robert McNaughton (1966). "Testing and generating infinite sequences by a finite automaton". In: *Information and control* 9.5, pp. 521–530. DOI: 10.1016/S0019-9958(66)80013-X


ω-RE
finite monoids
S1S
ω-Words
ETL
NBA

Characterizations of Regular Languages



Counter-free Automata over finite words

- Let $\mathcal{A} = \langle Q, \Sigma, I, \delta, F \rangle$ be a *deterministic* finite automaton (DFA).
- For each $\langle \sigma_0, \sigma_1, \dots, \sigma_n \rangle \in \Sigma^*$ and for each $q \in Q$, we define

$$\delta^*(q, \langle \sigma_0, \sigma_1, \dots, \sigma_n \rangle) = \begin{cases} \delta(q, \sigma_0) & \text{if } n = 0 \\ \delta(\delta^*(q, \langle \sigma_0, \dots, \sigma_{n-1} \rangle), \sigma_n) & \text{otherwise} \end{cases}$$

- For any word $\sigma \in \Sigma^*$ and any $i \in \mathbb{N}$, we define $(\sigma)^i$ as the word obtained from i concatenations of σ.

Definition (Nontrivial cycle)

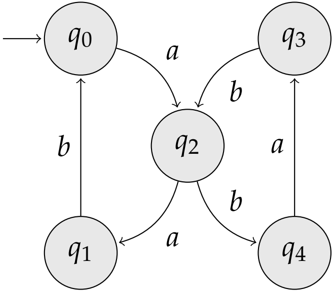
A word $\sigma \in \Sigma^*$ (with $\sigma \neq \varepsilon$) defines a *nontrivial cycle* in $\mathcal{A}$ if there exists a state $q \in Q$ such that:

- $\delta^*(q, \sigma) \neq q$
- $\delta^*(q, (\sigma)^i) = q$.

for some $i > 1$.

Definition (Counter-free DFA)

A DFA $\mathcal{A}$ is called *counter-free* if there are no words that define a nontrivial cycle. We denote this class by cf-DFA.



This automaton is *not* counter-free. The word *ab* defines the nontrivial cycle:

$$q_0 \xrightarrow{ab} q_4 \xrightarrow{ab} q_2 \xrightarrow{ab} q_0.$$

- The definition of counter-free automaton requires a *deterministic* automaton.
- NBA are not closed under *determinization*.
- We change the type of automata over ω-words which we work with.

$\Rightarrow$ Rabin Automata

Definition (DRA)

A *Deterministic Rabin Automaton* (DRA, for short) is a tuple $\langle Q, \Sigma, q_0, \delta, F \rangle$ where

$$F = \langle (A_1, B_1), \ldots, (A_n, B_n) \rangle$$

with $A_i, B_i \subseteq Q$.
A run $\pi := \langle \pi_0, \pi_1, \ldots \rangle \in Q^\omega$ is said to be *accepting* iff there exists some $i \in [1, n]$ such that

- $\mathsf{Inf}(\pi) \cap B_i \neq \varnothing$ and
- $\mathsf{Inf}(\pi) \cap A_i = \varnothing$.

Theorem

Deterministic Rabin Automata are equivalent to Nondeterministic Büchi Automata.

Definition (Counter-free DRA)

A DRA $\mathcal{A}$ is called *counter-free* if there are no words that define a nontrivial cycle. We call cf-DRA this class.

Definition (DRA)

A *Deterministic Rabin Automaton* (DRA, for short) is a tuple $\langle Q, \Sigma, q_0, \delta, F \rangle$ where

$$F = \langle (A_1, B_1), \ldots, (A_n, B_n) \rangle$$

with $A_i, B_i \subseteq Q$.
A run $\pi := \langle \pi_0, \pi_1, \ldots \rangle \in Q^\omega$ is said to be *accepting* iff there exists some $i \in [1, n]$ such that

- $\mathsf{Inf}(\pi) \cap B_i \neq \varnothing$ and
- $\mathsf{Inf}(\pi) \cap A_i = \varnothing$.

Theorem (Expressive Equivalence for cf-DRA)

For each ω-language $\mathcal{L} \subseteq \Sigma^\omega$, it holds that:

$$\mathcal{L} \text{ is star-free}$$
$$\text{iff}$$
$$\mathcal{L} = \mathcal{L}(\mathcal{A}) \text{ for some cf-DRA } \mathcal{A}$$

Theorem (Expressive Equivalence for cf-DFA)

For each language $\mathcal{L} \subseteq \Sigma^*$, it holds that:

$$\mathcal{L} \text{ is star-free}$$
$$\text{iff}$$
$$\mathcal{L} = \mathcal{L}^{<\omega}(\mathcal{A}) \text{ for some cf-DFA } \mathcal{A}$$

Reference:

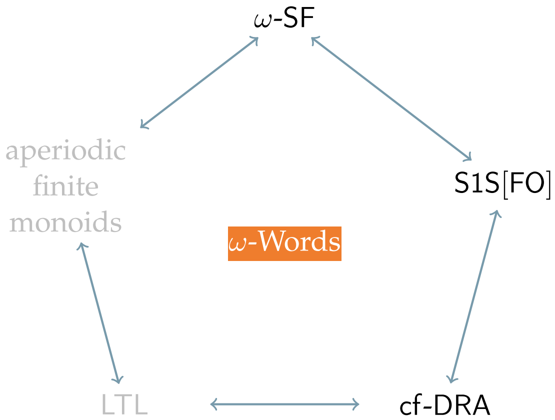
Robert McNaughton and Seymour A Papert (1971). *Counter-Free Automata (MIT research monograph no. 65)*. The MIT Press

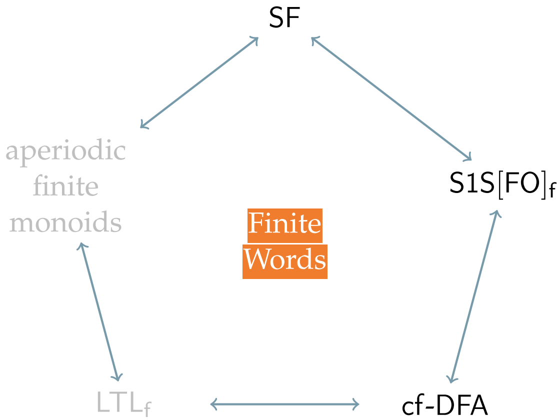
Reference:

Wolfgang Thomas (1979). "Star-free regular sets of ω-sequences". In: *Information and Control* 42.2, pp. 148–156. DOI: 10.1016/S0019-9958(79)90629-6

Reference:

Ina Schiering and Wolfgang Thomas (1996). "Counter-free automata, first-order logic, and star-free expressions extended by prefix oracles". In: *Developments in Language Theory, II (Magdeburg, 1995), Worl Sci. Publishing, River Edge, NJ,* pp. 166–175


ω-SF
aperiodic
finite
monoids
S1S[FO]
ω-Words
LTL
cf-DRA


SF
aperiodic finite monoids
S1S[FO]f
Finite Words
LTLf
cf-DFA

Temporal logic is the de-facto standard language for specifying properties of systems in *formal verification* and *artificial intelligence*.

- born in the ’50s as a tool for philosophical argumentation about time

Reference:

Arthur N Prior (2003). *Time and modality*. John Locke Lecture

- the idea of its use in formal verification can be traced back to the ’70s

Reference:

Amir Pnueli (1977). “The temporal logic of programs”. In: *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*. IEEE, pp. 46–57. DOI: 10.1109/SFCS.1977.32

Temporal logic in AI

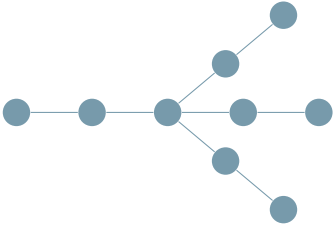
In *artificial intelligence*, when do we need to use *logic* to talk about *time*?

- automated planning
  - temporally extended goals (Bacchus and Kabanza 1998)
  - temporal planning (Fox and Long 2003)
  - timeline-based planning (Della Monica et al. 2017)
- automated synthesis (Jacobs et al. 2017)
- autonomy under uncertainty (Brafman and De Giacomo 2019)
  - specification of goals for planning over MDPs and POMDPs
- reinforcement learning (De Giacomo, Favorito, et al. 2020; Hammond et al. 2021)
  - specification of reward functions and safety conditions
- knowledge representation
  - temporal description logics (Artale, Kontchakov, et al. 2014)
- multi-agent systems
  - temporal epistemic logics (van Benthem et al. 2009)

There are many choices to be made for the representation of *time*.

Linear

Branching



There are many choices to be made for the representation of *time*.

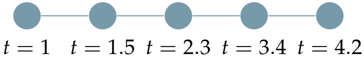
Infinite

Finite

There are many choices to be made for the representation of *time*.

Qualitative

Real-time



There are many choices to be made for the representation of *time*.

Discrete

Dense

Representing time

There are many choices to be made for the representation of *time*.

We focus here on:

- *linear*-time
- *discrete*-time
- *qualitative*-time
- *infinite*-time
    - sometimes also *finite*-time

Linear Temporal Logic with Past (LTL+P, for short) is a *modal* logic.

- introduced by Pnueli in the '70s
- interpreted over discrete, infinite *state sequences* (infinite words)
- it extends classical *propositional* logic
- temporal *operators* are used to talk about how propositions change over time

Let $\mathcal{AP} := \{p, q, r, \dots\}$ be a set of *atomic propositions*. The syntax of LTL+P is defined as follows:

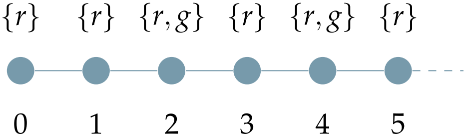
$$
\begin{aligned}
\phi := p \mid \neg\phi \mid \phi \vee \phi & \quad \text{Boolean Modalities} \\
\mid \mathsf{X}\phi \mid \phi \,\mathsf{U}\, \phi & \quad \text{Future Temporal Modalities} \\
\mid \mathsf{Y}\phi \mid \phi \,\mathsf{S}\, \phi & \quad \text{Past Temporal Modalities}
\end{aligned}
$$

where $p \in \mathcal{AP}$.

- X is called *tomorrow* (or *next*)
- U is called *until*
- Y is called *yesterday* (or *previous*)
- S is called *since*

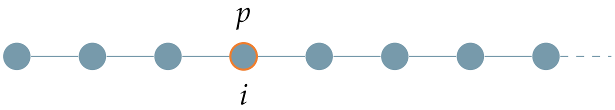
- We focus on the *infinite-time* interpretation of LTL+P.
- Given a set of atomic propositions $\mathcal{AP}$, any LTL+P formula defined over $\mathcal{AP}$ is interpreted over *infinite words* $\sigma \in (2^{\mathcal{AP}})^\omega$.
- In this context, sequences in $(2^{\mathcal{AP}})^\omega$ are also called state sequences or traces.

$$\mathcal{AP} := \{r, g\}$$



We say that σ satisfies at position i the LTL+P formula ϕ, written $\sigma, i \models \phi$, iff:

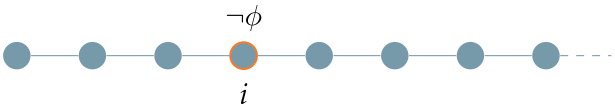
- $\sigma, i \models p$ iff $p \in \sigma_i$



p holds at position i

We say that σ satisfies at position i the LTL+P formula ϕ, written $\sigma, i \models \phi$, iff:

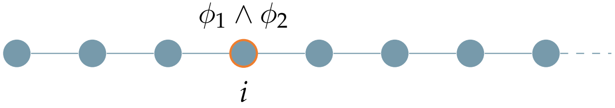
- $\sigma, i \models \neg\phi$ iff $\sigma, i \not\models \phi$



ϕ does not hold at position i

We say that σ satisfies at position i the LTL+P formula ϕ, written $\sigma, i \models \phi$, iff:

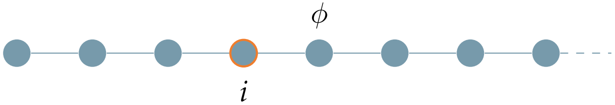
- $\sigma, i \models \phi_1 \land \phi_2$ iff $\sigma, i \models \phi_1$ and $\sigma, i \models \phi_2$



ϕ_1 and ϕ_2 hold at position i

We say that σ satisfies at position i the LTL+P formula ϕ, written $\sigma, i \models \phi$, iff:

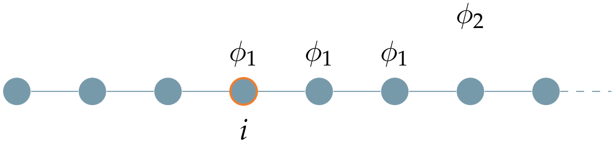
- $\sigma, i \models \mathsf{X}\phi$ iff $\sigma, i+1 \models \phi$



ϕ holds at the *next* position of i

We say that σ satisfies at position i the LTL+P formula ϕ, written $\sigma, i \models \phi$, iff:

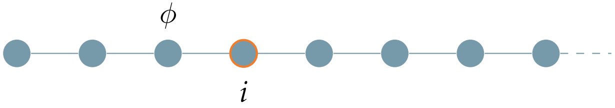
- $\sigma, i \models \phi_1 \mathsf{U} \phi_2$ iff $\exists j \geq i \,.\, \sigma, j \models \phi_2$ and $\forall i \leq k < j \,.\, \sigma, k \models \phi_1$



ϕ_1 holds *until* ϕ_2 holds

We say that σ satisfies at position i the LTL+P formula ϕ, written $\sigma, i \models \phi$, iff:

- $\sigma, i \models \mathsf{Y}\phi$ iff $i > 0$ and $\sigma, i-1 \models \phi$

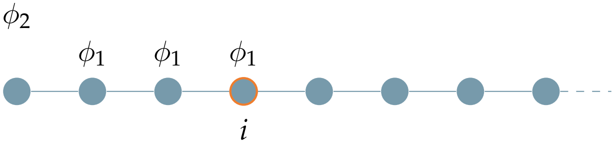


position i has a predecessor and ϕ holds at the *previous* position of i

Note: $\sigma, 0 \models \mathsf{Y}\phi$ is always false.

We say that σ satisfies at position i the LTL+P formula ϕ, written $\sigma, i \models \phi$, iff:

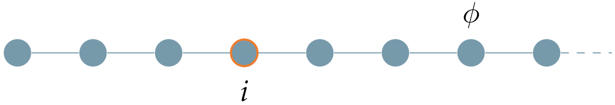
- $\sigma, i \models \phi_1 \mathsf{S} \phi_2$ iff $\exists j \leq i \,.\, \sigma, j \models \phi_2$ and $\forall j < k \leq i \,.\, \sigma, k \models \phi_1$



ϕ_1 holds *since* ϕ_2 held

Shortcuts:

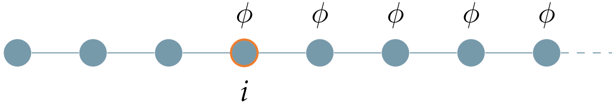
- (*eventually*) $\mathsf{F}\phi \equiv \top \,\mathsf{U}\, \phi$



ϕ will *eventually* hold

Shortcuts:

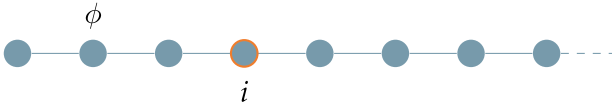
- (*globally*) $\mathsf{G}\phi \equiv \neg\mathsf{F}\neg\phi$



ϕ holds *always*

Shortcuts:

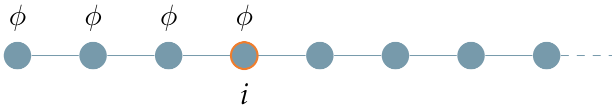
- (*once*) $\mathsf{O}\phi \equiv \top\ \mathsf{S}\ \phi$



ϕ *once* held

Shortcuts:

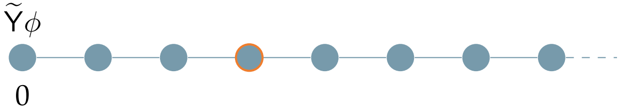
- (*historically*) $\mathsf{H}\phi \equiv \neg\mathsf{O}\neg\phi$



ϕ holds *always in the past*

Shortcuts:

- (*weak yesterday*) $\widetilde{\mathsf{Y}}\phi \equiv \neg\mathsf{Y}\neg\phi$



ϕ holds at the *previous* position of i, *if any*

Note: $\sigma, i \models \widetilde{\mathsf{Y}}\bot$ is true iff $i = 0$.

Definition (Negation Normal Form)

We define the $\mathtt{nnf}(\cdot) : \mathsf{LTL} \to \mathsf{LTL}$ (*Negation Normal Form*) function as follows:

- $\mathtt{nnf}(p) = p$
- $\mathtt{nnf}(\phi_1 \wedge \phi_2) = \mathtt{nnf}(\phi_1) \wedge \mathtt{nnf}(\phi_2)$
- $\mathtt{nnf}(\phi_1 \vee \phi_2) = \mathtt{nnf}(\phi_1) \vee \mathtt{nnf}(\phi_2)$
- $\mathtt{nnf}(\mathsf{X}\phi) = \mathsf{X}(\mathtt{nnf}(\phi))$
- $\mathtt{nnf}(\phi_1 \mathsf{U} \phi_2) = (\mathtt{nnf}(\phi_1)) \mathsf{U} (\mathtt{nnf}(\phi_2))$
- $\mathtt{nnf}(\phi_1 \mathsf{R} \phi_2) = (\mathtt{nnf}(\phi_1)) \mathsf{R} (\mathtt{nnf}(\phi_2))$

For any $\phi \in \mathsf{LTL}$, the formula $\mathtt{nnf}(\phi)$ has *negation only applied to atomic propositions*.

Definition (Negation Normal Form)

We define the $\mathtt{nnf}(\cdot) : \mathsf{LTL} \to \mathsf{LTL}$ (*Negation Normal Form*) function as follows:

- $\mathtt{nnf}(\neg p) = \neg p$
- $\mathtt{nnf}(\neg\neg\phi) = \mathtt{nnf}(\phi)$
- $\mathtt{nnf}(\neg(\phi_1 \wedge \phi_2)) = \mathtt{nnf}(\neg\phi_1) \vee \mathtt{nnf}(\neg\phi_2)$
- $\mathtt{nnf}(\neg(\phi_1 \vee \phi_2)) = \mathtt{nnf}(\neg\phi_1) \wedge \mathtt{nnf}(\neg\phi_2)$
- $\mathtt{nnf}(\neg\mathsf{X}\phi) = \mathsf{X}(\mathtt{nnf}(\neg\phi))$
- $\mathtt{nnf}(\neg(\phi_1 \,\mathsf{U}\, \phi_2)) = (\mathtt{nnf}(\neg\phi_1)) \,\mathsf{R}\, (\mathtt{nnf}(\neg\phi_2))$
- $\mathtt{nnf}(\neg(\phi_1 \,\mathsf{R}\, \phi_2)) = (\mathtt{nnf}(\neg\phi_1)) \,\mathsf{U}\, (\mathtt{nnf}(\neg\phi_2))$

For any $\phi \in \mathsf{LTL}$, the formula $\mathtt{nnf}(\phi)$ has *negation only applied to atomic propositions*.

- We say that σ *satisfies* ϕ (written $\sigma \models \phi$) iff $\sigma, 0 \models \phi$.
- For any LTL+P formula ϕ, we define *the language of ϕ over infinite words* as:

$$\mathcal{L}(\phi) = \{\sigma \in (2^{\mathcal{AP}})^\omega \mid \sigma \models \phi\}$$

- We say that ϕ is satisfiable iff $\mathcal{L}(\phi) \neq \varnothing$.
- We say that ϕ is valid iff $\mathcal{L}(\phi) = (2^{\mathcal{AP}})^\omega$.

Example:

Each request (r) is eventually followed by a grant (g).

$$\mathsf{G}(r \rightarrow \mathsf{F}(g))$$

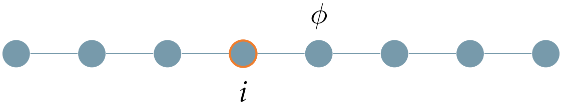
Example:

Each grant (g) is preceeded by a request (r).

$$\mathsf{G}(g \rightarrow \mathsf{O}(r)))$$

LTL+P over *finite words* is interpreted over *finite state sequences* $\sigma \in (2^{\mathcal{AP}})^+$, that is *finite, nonempty* sequences of subsets of $\mathcal{AP}$.
For the interpretation of LTL+P over finite words it suffices to consider the following cases:

- $\sigma, i \models \mathsf{X}\phi$ iff $i < |\sigma| - 1$ and $\sigma, i+1 \models \phi$

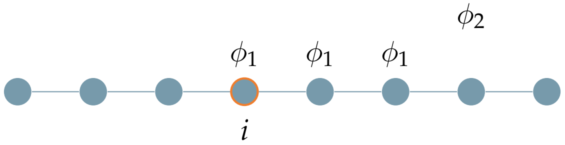


ϕ hold at the *next* position of i

Note: $\sigma, n \models \mathsf{X}\phi$ is always false when $n = |\sigma| - 1$.

LTL+P over *finite words* is interpreted over *finite state sequences* $\sigma \in (2^{\mathcal{AP}})^+$, that is *finite, nonempty* sequences of subsets of $\mathcal{AP}$.
For the interpretation of LTL+P over finite words it suffices to consider the following cases:

- $\sigma, i \models \phi_1 \mathsf{U} \phi_2$ iff $\exists i \leq j < |\sigma| \,.\, \sigma, j \models \phi_2$ and $\forall i \leq k < j \,.\, \sigma, k \models \phi_1$

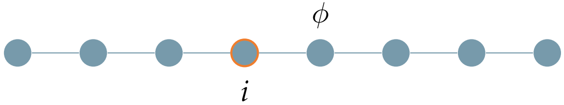


ϕ_1 holds *until* ϕ_2 holds

Shortcuts:

- (*weak tomorrow*) $\widetilde{\mathsf{X}}\phi \equiv \neg\mathsf{X}\neg\phi$

$$\sigma, i \models \widetilde{\mathsf{X}}\phi \textit{ iff } (i < |\sigma| - 1 \text{ implies } \sigma, i+1 \models \phi)$$



ϕ holds at the *next* position of i, *if any*

- **Note:** $\sigma, i \models \widetilde{\mathsf{X}}\bot$ is true iff $i = |\sigma| - 1$.
- **Note:** over *infinite traces*, X and $\widetilde{\mathsf{X}}$ coincide.

- We say that σ *satisfies* ϕ (written $\sigma \models \phi$) iff $\sigma, 0 \models \phi$.
- For any LTL+P formula ϕ, we define *the language of ϕ over finite words* as:

$$\mathcal{L}^{<\omega}(\phi) = \{\sigma \in (2^{\mathcal{AP}})^+ \mid \sigma \models \phi\}$$

Words

- We denote with $\mathsf{LTL_f{+}P}$ the set of formulas of $\mathsf{LTL{+}P}$ that we will interpret on *finite words*

ω-Words

- We denote with $\mathsf{LTL{+}P}$ the set of formulas of $\mathsf{LTL{+}P}$ that we will interpret on *infinite words*

Words

- We denote with $\mathsf{LTL}_f\mathsf{+P}$ the set of formulas of $\mathsf{LTL+P}$ that we will interpret on *finite words*
- We denote with LTL_f the set of formulas of $\mathsf{LTL}_f\mathsf{+P}$ *devoid of past temporal operators*.

ω-Words

- We denote with $\mathsf{LTL+P}$ the set of formulas of $\mathsf{LTL+P}$ that we will interpret on *infinite words*
- We denote with LTL the set of formulas of $\mathsf{LTL+P}$ *devoid of past temporal operators*.

Words

- We denote with $\mathsf{LTL_f{+}P}$ the set of formulas of $\mathsf{LTL{+}P}$ that we will interpret on *finite words*
- We denote with $\mathsf{LTL_f}$ the set of formulas of $\mathsf{LTL_f{+}P}$ *devoid of past temporal operators*.
- Given a logic $\mathbb{L}$ (*e.g.*, $\mathsf{LTL_f}$ or $\mathsf{LTL_f{+}P}$), we denote with $[\![\mathbb{L}]\!]^{<\omega} = \{\mathcal{L}^{<\omega}(\phi) \mid \phi \in \mathbb{L}\}$

ω-Words

- We denote with $\mathsf{LTL{+}P}$ the set of formulas of $\mathsf{LTL{+}P}$ that we will interpret on *infinite words*
- We denote with LTL the set of formulas of $\mathsf{LTL{+}P}$ *devoid of past temporal operators*.
- Given a logic $\mathbb{L}$ (*e.g.*, LTL or $\mathsf{LTL_f}$), we denote with $[\![\mathbb{L}]\!] = \{\mathcal{L}(\phi) \mid \phi \in \mathbb{L}\}$

Theorem

- $[\![\mathsf{LTL_f{+}P}]\!]^{<\omega} = [\![\mathsf{LTL_f}]\!]^{<\omega}$
- $[\![\mathsf{LTL{+}P}]\!] = [\![\mathsf{LTL}]\!]$

Reference:

Dov M. Gabbay et al. (1980). "On the Temporal Analysis of Fairness". In: *Conference Record of the Seventh Annual ACM Symposium on Principles of Programming Languages, Las Vegas, Nevada, USA, January 1980*. Ed. by Paul W. Abrahams, Richard J. Lipton, and Stephen R. Bourne. ACM Press, pp. 163–173. URL: https://doi.org/10.1145/567446.567462

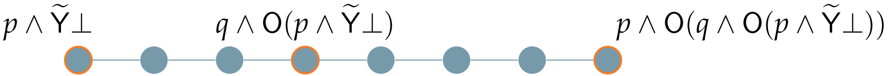
Definition (Pure-past LTL)

Pure-past LTL (pLTL, for short) is the set of LTL+P formulas *devoid* of future operators.

Example:

$p \wedge \mathsf{O}(q \wedge \mathsf{O}(p \wedge \widetilde{\mathsf{Y}}\bot))$

pLTL formulas are naturally interpreted on the *last* position of a *finite trace*.



Pure-past LTL

Theorem

$[\![\mathsf{pLTL}]\!]^{<\omega} = [\![\mathsf{LTL_f}]\!]^{<\omega}$

Reference:

Orna Lichtenstein, Amir Pnueli, and Lenore Zuck (1985). "The glory of the past". In: *Workshop on Logic of Programs*. Springer, pp. 196–218. DOI: 10.1007/3-540-15648-8_16

Reference:

Lenore Zuck (1986). "Past temporal logic". In: *Weizmann Institute of Science* 67

Linear Temporal Logic with Past

Theorem (Kamp's Theorem over ω-words)

- *For each* LTL+P *formula* ϕ*, there exists an* S1S[FO] *formula* ψ *such that* $\mathcal{L}(\phi) = \mathcal{L}(\psi)$.
- *For each* S1S[FO] *formula* ψ*, there exists an* LTL+P *formula* ϕ *such that* $\mathcal{L}(\psi) = \mathcal{L}(\phi)$.

Theorem (Kamp's Theorem over finite words)

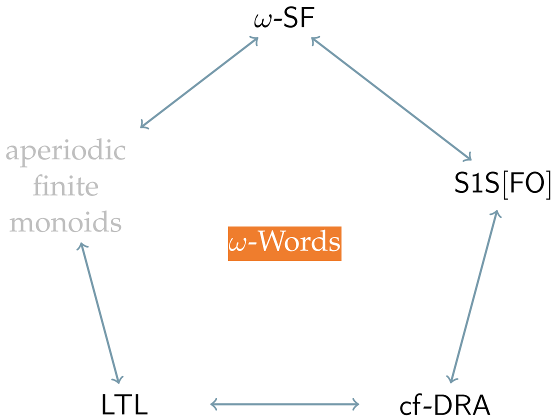
- *For each* LTL+P *formula* ϕ*, there exists an* S1S[FO] *formula* ψ *such that* $\mathcal{L}^{<\omega}(\phi) = \mathcal{L}^{<\omega}(\psi)$.
- *For each* S1S[FO] *formula* $\psi(x)$*, there exists an* LTL+P *formula* ϕ *such that* $\mathcal{L}^{<\omega}(\psi) = \mathcal{L}^{<\omega}(\phi)$.

Reference:

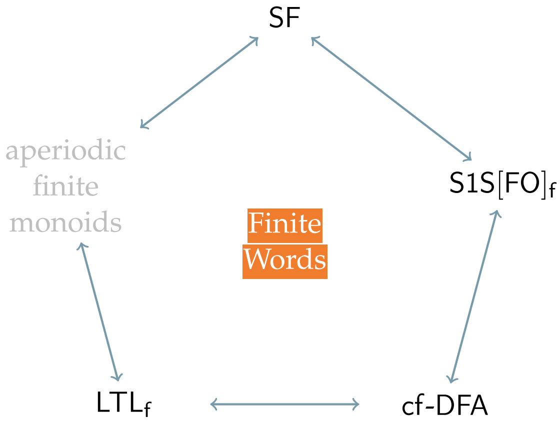
Johan Anthony Wilem Kamp (1968). “Tense logic and the theory of linear order”. In

Reference:

Dov M. Gabbay et al. (1980). “On the Temporal Analysis of Fairness”. In: *Conference Record of the Seventh Annual ACM Symposium on Principles of Programming Languages, Las Vegas, Nevada, USA, January 1980*. Ed. by Paul W. Abrahams, Richard J. Lipton, and Stephen R. Bourne. ACM Press, pp. 163–173. URL: https://doi.org/10.1145/567446.567462


ω-SF
aperiodic
finite
monoids
S1S[FO]
ω-Words
LTL
cf-DRA

Characterizations of Star-free Languages



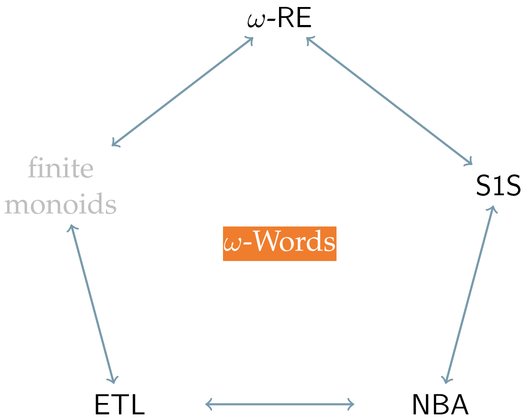
Extended Linear Temporal Logic with Past

We have seen that LTL+P captures *star-free* ω-regular languages.
In order to capture all ω-regular languages, one can consider *Extended Linear Temporal Logic* (ETL, for short).

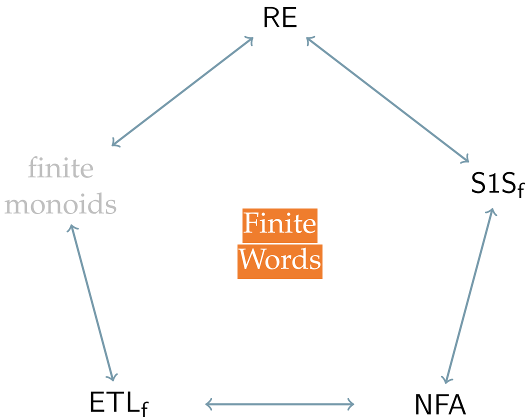
ETL = LTL + operators corresponding to *right-linear grammars*

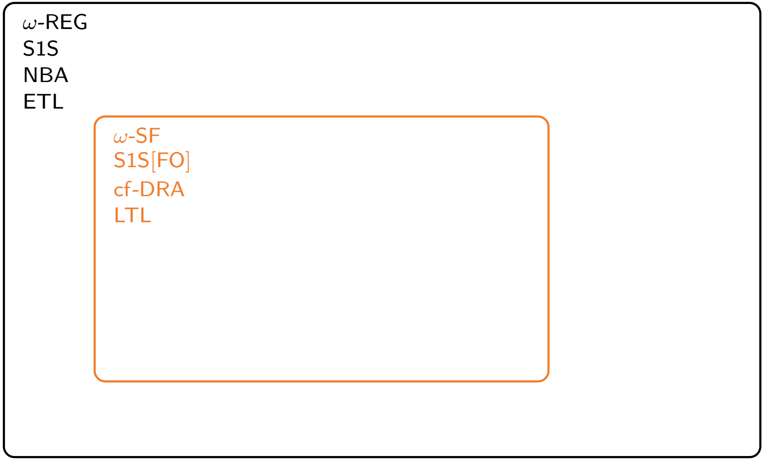
Reference:

Pierre Wolper (1983). "Temporal logic can be more expressive". In: *Information and control* 56.1-2, pp. 72–99. DOI: 10.1016/S0019-9958(83)80051-5


ω-RE
finite monoids
S1S
ω-Words
ETL
NBA

Characterizations of Regular Languages




ω-REG
S1S
NBA
ETL
ω-SF
S1S[FO]
cf-DRA
LTL

THE SAFETY FRAGMENT

OF ω-REGULAR LANGUAGES

In this part, we will mainly deal with language of *infinite words* and with logics interpreted over *infinite words*.

Informal definitions:

Safety properties express the fact that "something bad never happens".

E.g.: a deadlock or a simultaneous access to a critical section.

Any violation of a safety property is irremediable.

E.g.: once a deadlock occured, we don't have any hope to do better.

Any violation of a safety property has a finite witness.

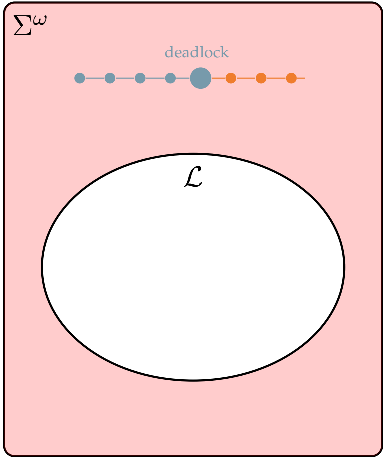
Notation:

- For any $i \in \mathbb{N}$, $\sigma_{[0,i]}$ is the prefix of σ up to position i.
- for any $\sigma \in \Sigma^*$ and for any $\sigma \in \Sigma^\omega$, $\sigma \cdot \sigma'$ is the *concatenation* of σ' to the end of σ.

Definition (Safety Property)

$\mathcal{L} \subseteq \Sigma^\omega$ is a *safety property* iff, for all $\sigma \notin \mathcal{L}$, there exists an position $i \in \mathbb{N}$ such that $\sigma_{[0,i]} \cdot \sigma' \notin \mathcal{L}$, for all $\sigma' \in \Sigma^\omega$.

$\sigma_{[0,i]}$ is called the *bad prefix* of σ.

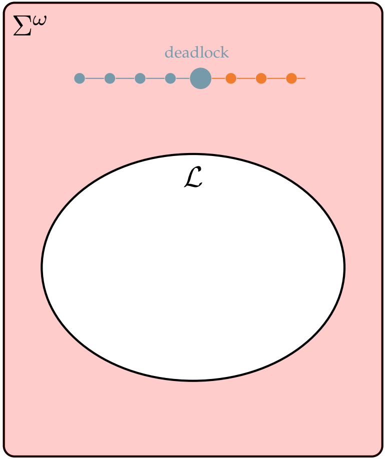


Examples

- $b \cdot (a)^\omega$ is a safety language.
- *"The set of infinite words in which each 'a' is followed by some 'b' "* is not a safety language.

- We denote with $\mathtt{bad}(\mathcal{L})$ the set of bad prefixes of $\mathcal{L}$.
- For any safety language $\mathcal{L}$, it holds that:

$$\overline{\mathcal{L}} = \mathtt{bad}(\mathcal{L}) \cdot \Sigma^\omega$$



Definition (Cosafety Property)

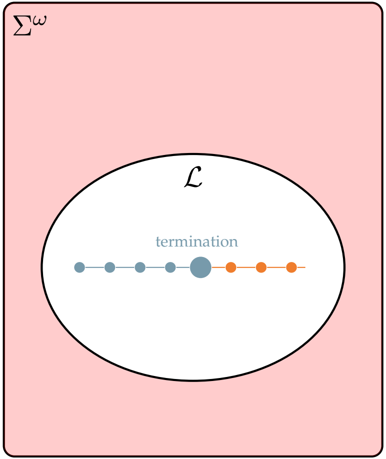
$\mathcal{L} \subseteq \Sigma^\omega$ is a *cosafety property* iff for all $\sigma \in \mathcal{L}$, there exists an position $i \in \mathbb{N}$ such that $\sigma_{[0,i]} \cdot \sigma' \in \mathcal{L}$, for all $\sigma' \in \Sigma^\omega$.

$\sigma_{[0,i]}$ is called the *good prefix* of σ.

Notation: for any $\mathcal{L} \subseteq \Sigma^\omega$, we denote with $\overline{\mathcal{L}}$ the *complement* of $\mathcal{L}$.

Property:

$\mathcal{L}$ is a cosafety property iff $\overline{\mathcal{L}}$ is a safety property.

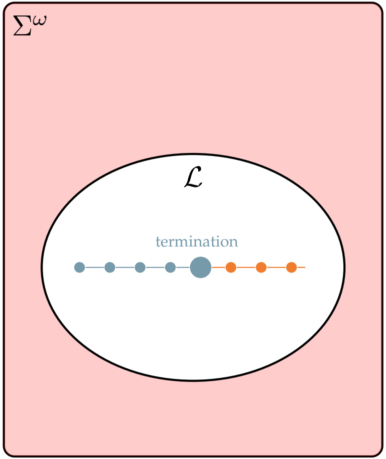


Examples

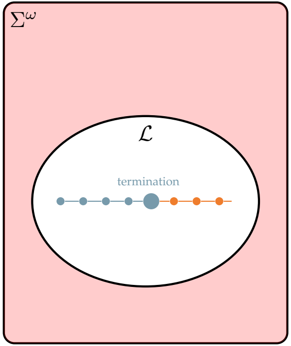
- *"The set of infinite words in which there is an 'a' that is followed by some 'b' "* is a cosafety language.
- *"The set of infinite words in which each 'a' is followed by some 'b' "* is not a cosafety language.

- We denote with $\mathtt{good}(\mathcal{L})$ the set of good prefixes of $\mathcal{L}$.
- For any cosafety language $\mathcal{L}$, it holds that:

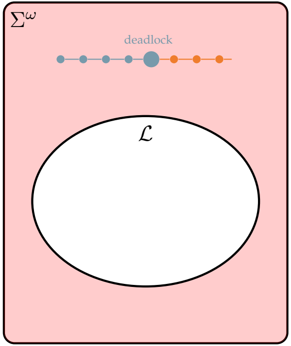
$$\mathcal{L} = \mathtt{good}(\mathcal{L}) \cdot \Sigma^\omega$$



We denote with coSAFETY the set of all cosafety ω-regular languages.



We denote with SAFETY the set of all safety ω-regular languages.



We denote with coSAFETY the set of all cosafety ω-regular languages.

ω-Regular Expressions

coSAFETY is characterized by the following type of ω-regular expressions:

$$K \cdot \Sigma^\omega$$

where $K \in$ REG.

We denote with SAFETY the set of all safety ω-regular languages.

ω-Regular Expressions

SAFETY is characterized by the following type of ω-regular expressions:

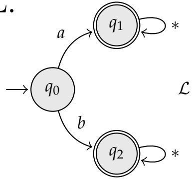
$$\overline{K \cdot \Sigma^\omega}$$

where $K \in$ REG.

We denote with coSAFETY the set of all cosafety ω-regular languages.

Automata

coSAFETY is characterized by the following type of automata: *terminal deterministic Büchi automata* (tDBA, for short), that is DBAs in which each final state has self-loop labeled with each letter in Σ.

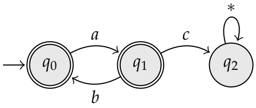


$\mathcal{L} = \{a, b\} \cdot \Sigma^\omega$

We denote with SAFETY the set of all safety ω-regular languages.

Automata

SAFETY is characterized by the following type of automata: *deterministic safety automata* (DSA, for short). Accepting condition: visit *only* final states.

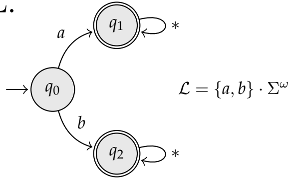


$\mathcal{L} = (ab)^\omega$

We denote with coSAFETY the set of all cosafety ω-regular languages.

Automata

coSAFETY is characterized by the following type of automata: *terminal deterministic Büchi automata* (tDBA, for short), that is DBAs in which each final state has self-loop labeled with each letter in Σ.



We denote with SAFETY the set of all safety ω-regular languages.

Automata

SAFETY is characterized by the following type of automata: *deterministic safety automata* (DSA, for short). Accepting condition: visit *only* final states.

A DSA is a DRA $\mathcal{A} = \langle Q, \Sigma, q_0, \delta, \langle (A_1, B_1), \ldots, (A_n, B_n) \rangle \rangle$ such that $B_i = Q \setminus A_i$, for each $i \in [1, n]$.

We denote with coSAFETY the set of all cosafety ω-regular languages.

S1S

To the best of our knowledge, no characterizations of coSAFETY in terms of S1S have been studied.

We denote with SAFETY the set of all safety ω-regular languages.

S1S

To the best of our knowledge, no characterizations of SAFETY in terms of S1S have been studied.

We denote with coSAFETY the set of all cosafety ω-regular languages.

Temporal Logics

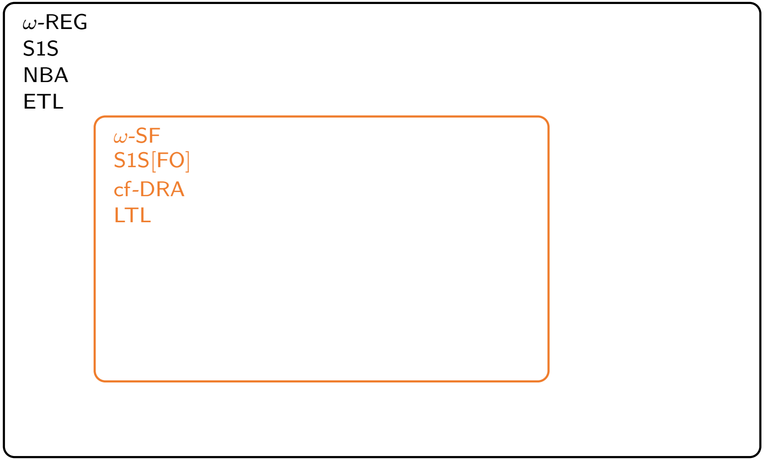
To the best of our knowledge, no characterizations of coSAFETY in terms of temporal logics have been studied.

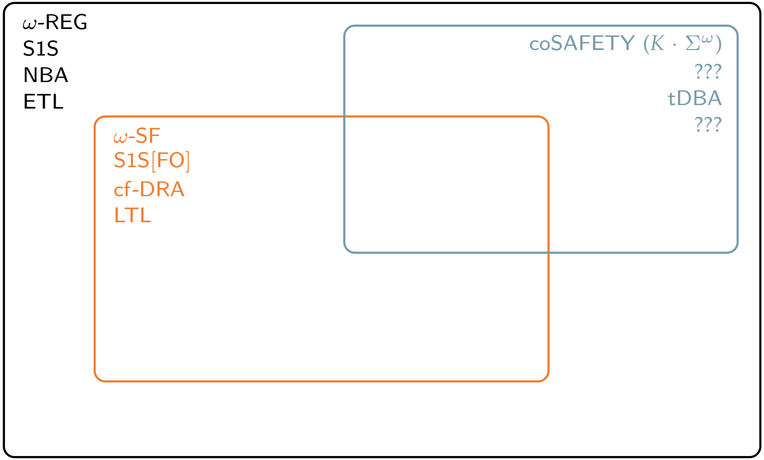
We denote with SAFETY the set of all safety ω-regular languages.

Temporal Logics

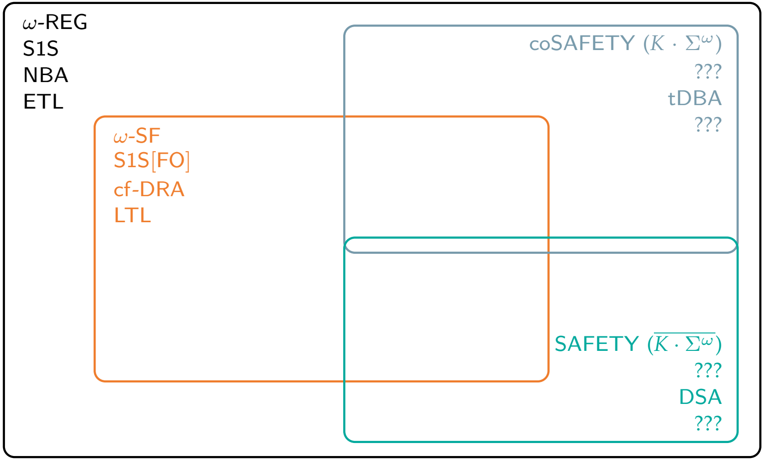
To the best of our knowledge, no characterizations of SAFETY in terms of temporal logics have been studied.

Set-theoretic view of (co)safety ω-languages





ω-REG
S1S
NBA
ETL
coSAFETY $(K \cdot \Sigma^\omega)$
???
tDBA
???
ω-SF
S1S[FO]
cf-DRA
LTL


ω-REG
S1S
NBA
ETL
coSAFETY ($K \cdot \Sigma^\omega$)
???
tDBA
???
ω-SF
S1S[FO]
cf-DRA
LTL
SAFETY ($\overline{K \cdot \Sigma^\omega}$)
???
DSA
???

Informal definitions:

> *In a liveness property, no partial execution is irremediable.*

E.g.: *"each request is eventually follows by a grant"* is a liveness property.

Definition (Liveness Property)

$\mathcal{L} \subseteq \Sigma^\omega$ is a *liveness property* iff, for all $\sigma \in \Sigma^*$, there exists a $\sigma' \in \Sigma^\omega$ such that $\sigma \cdot \sigma' \in \mathcal{L}$.

Examples:

- *"The set of infinite words in which each 'a' is followed by some 'b' "* is a liveness language.
- $b \cdot (a)^\omega$ is not a liveness language.

Theorem (Alpern & Schneider (1987))

Each ω-regular property is the intersection of a safety property and a liveness property.

Reference:

Bowen Alpern and Fred B. Schneider (1987). "Recognizing Safety and Liveness". In: *Distributed Comput.* 2.3, pp. 117–126. DOI: 10.1007/BF01782772. URL: https://doi.org/10.1007/BF01782772

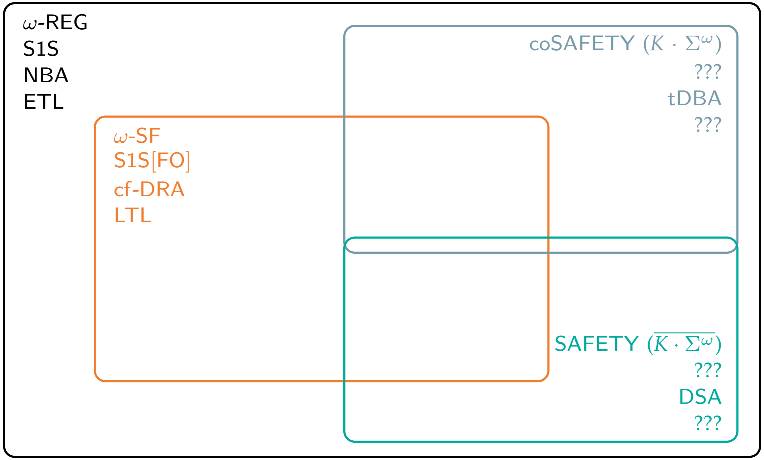
Theorem (Alpern & Schneider (1987))

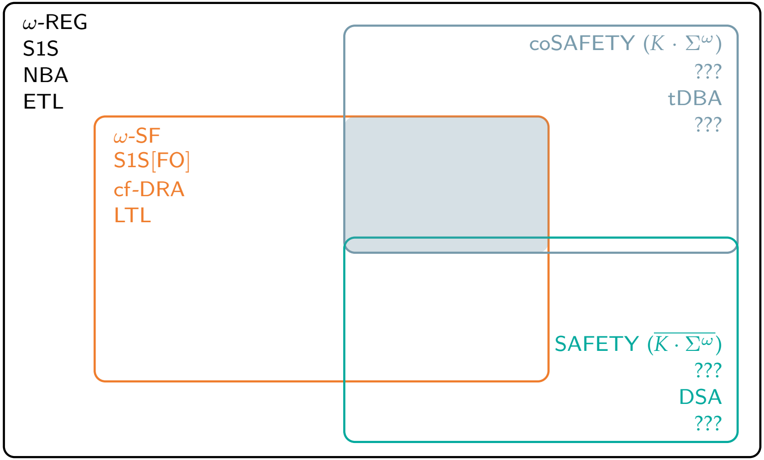
Each ω-regular property is the intersection of a safety property and a liveness property.

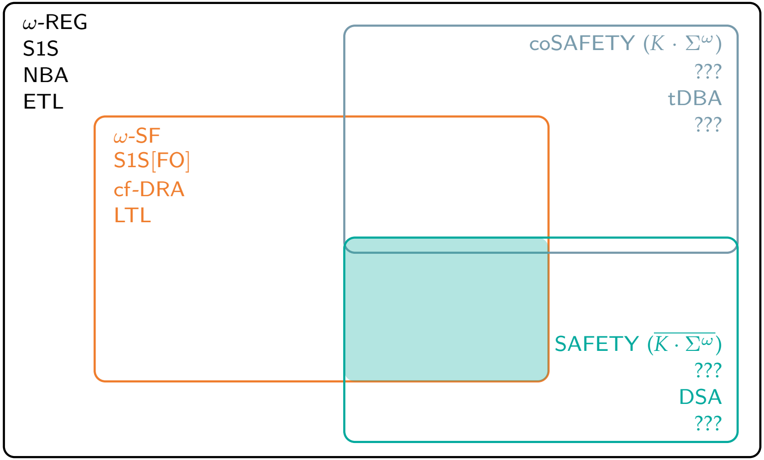
This decomposition can be performed effectively:

Given a NBA $\mathcal{A}$, there is an algorithm to build two NBA $\mathcal{A}_s$ and $\mathcal{A}_l$ such that:

- $\mathcal{L}(\mathcal{A}_s)$ is safety;
- $\mathcal{L}(\mathcal{A}_l)$ is liveness;
- $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_s) \cap \mathcal{L}(\mathcal{A}_l)$.


ω-REG
S1S
NBA
ETL
coSAFETY $(K \cdot \Sigma^\omega)$
???
tDBA
???
ω-SF
S1S[FO]
cf-DRA
LTL
SAFETY $(\overline{K \cdot \Sigma^\omega})$
???
DSA
???


ω-REG
S1S
NBA
ETL
coSAFETY $(K \cdot \Sigma^\omega)$
???
tDBA
???
ω-SF
S1S[FO]
cf-DRA
LTL
SAFETY $(\overline{K \cdot \Sigma^\omega})$
???
DSA
???


ω-REG
S1S
NBA
ETL
coSAFETY $(K \cdot \Sigma^\omega)$
???
tDBA
???
ω-SF
S1S[FO]
cf-DRA
LTL
SAFETY $(\overline{K \cdot \Sigma^\omega})$
???
DSA
???

THE SAFETY FRAGMENT OF LTL

AND ITS THEORETICAL FEATURES

Definition

The *cosafety fragment of* LTL is the set of languages in this set:

$$[\![\mathsf{LTL}]\!] \cap \mathsf{coSAFETY}$$

We will see four characterizations in terms of:

- regular expressions
- first-order logic
- automata
- temporal logic

Definition

The *cosafety fragment of* LTL is the set of languages in this set:

$$[\![\mathsf{LTL}]\!] \cap \mathsf{coSAFETY}$$

ω-regular expressions

$\mathsf{SF} \cdot \Sigma^\omega = \{K \cdot \Sigma^\omega \mid K \in \mathsf{SF}\}$

- the "SF " part corresponds to LTL
- the "$\cdot\Sigma^\omega$" part corresponds to being a cosafety fragment

Ina Schiering and Wolfgang Thomas (1996). “Counter-free automata, first-order logic, and star-free expressions extended by prefix oracles”. In: *Developments in Language Theory, II (Magdeburg, 1995), Worl Sci. Publishing, River Edge, NJ*, pp. 166–175

Definition

The *cosafety fragment of* LTL is the set of languages in this set:

$$[\![\mathsf{LTL}]\!] \cap \mathsf{coSAFETY}$$

First-order logic

We define coSafety-FO as the fragment of S1S[FO] in which quantifiers are bounded as follows:

- $\exists y \,.\, (x < y \land \ldots)$
- $\forall y \,.\, (x < y < z \to \ldots)$

Alessandro Cimatti et al. (2022). "A first-order logic characterisation of safety and co-safety languages". In: *Foundations of Software Science and Computation Structures - 25th International Conference, FOSSACS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings*. Ed. by Patricia Bouyer and Lutz Schröder. Vol. 13242. Lecture Notes in Computer Science. Springer, pp. 244–263. DOI: 10.1007/978-3-030-99253-8_13. URL: https://doi.org/10.1007/978-3-030-99253-8%5C_13

Definition

The *cosafety fragment of* LTL is the set of languages in this set:

$$[\![\mathsf{LTL}]\!] \cap \mathsf{coSAFETY}$$

First-order logic

Example

$\phi(x) := \exists y \,.\, (x < y \land P(y) \land \forall z \,.\, (x < z < y \rightarrow Q(z)))$

Definition

The *cosafety fragment of* LTL is the set of languages in this set:

$$[\![\mathsf{LTL}]\!] \cap \mathsf{coSAFETY}$$

First-order logic

- the "first-order" part corresponds to LTL
- the "bounded quantifiers" part corresponds to being a cosafety fragment

The cosafety fragment of LTL

Definition

The *cosafety fragment of* LTL is the set of languages in this set:

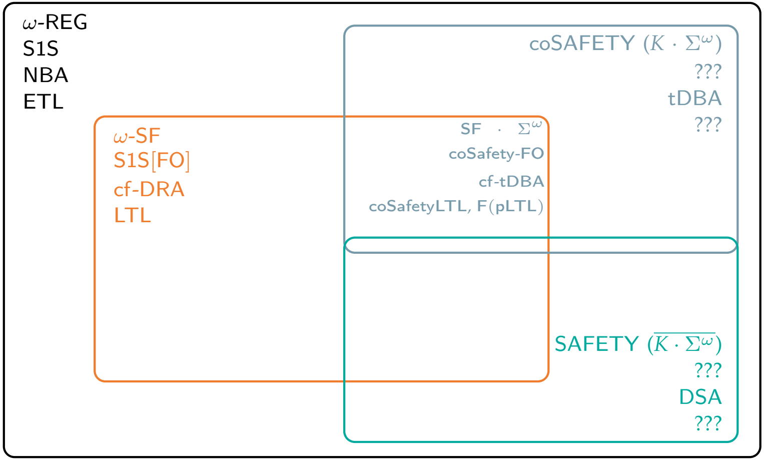
$$[\![\mathsf{LTL}]\!] \cap \mathsf{coSAFETY}$$

Automata

cf-tDBA = counter-free terminal DBA

- the "counter-free" part corresponds to LTL
- the "terminal" part corresponds to being a cosafety fragment

Ina Schiering and Wolfgang Thomas (1996). "Counter-free automata, first-order logic, and star-free expressions extended by prefix oracles". In: *Developments in Language Theory, II (Magdeburg, 1995), Worl Sci. Publishing, River Edge, NJ*, pp. 166–175


ω-REG
S1S
NBA
ETL
coSAFETY $(K \cdot \Sigma^\omega)$
???
tDBA
???
ω-SF
S1S[FO]
cf-DRA
LTL
SF · Σ^ω
coSafety-FO
cf-tDBA
coSafetyLTL, F(pLTL)
SAFETY $(\overline{K \cdot \Sigma^\omega})$
???
DSA
???

The cosafety fragment of LTL

We say that a temporal logic $\mathbb{L}$ is *cosafety* iff, for any $\phi \in \mathbb{L}$, $\mathcal{L}(\phi)$ is *cosafety*.

coSafetyLTL

Definition

$\phi := p \mid \neg p \mid \phi \vee \phi \mid \phi \wedge \phi \mid \mathsf{X}\phi \mid \mathsf{F}\phi \mid \phi\, \mathsf{U}\, \phi$

Example:

$p\, \mathsf{U}\, q$

F(pLTL)

Definition

$\phi := \mathsf{F}(\alpha)$, where $\alpha \in \mathsf{pLTL}$, that is α is a pure-past LTL formula.

Example:

$\mathsf{F}(q \wedge \widetilde{\mathsf{Y}}\mathsf{H}p)$

F(pLTL) is the canonical form of coSafetyLTL.

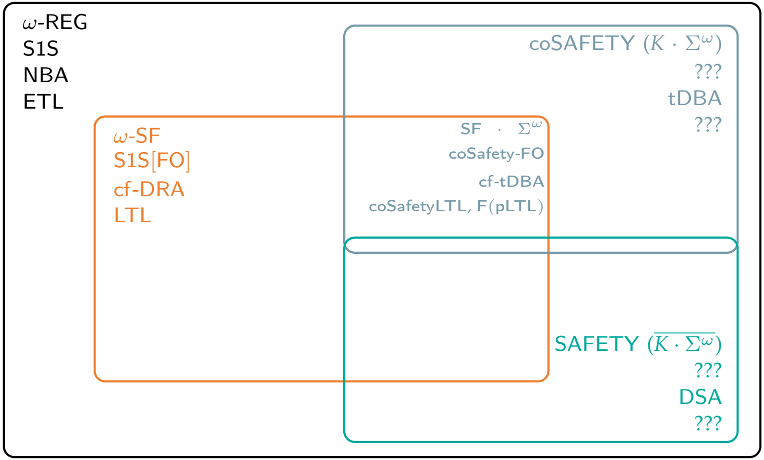
The cosafety fragment of LTL

Theorem

- coSafetyLTL *and* F(pLTL) *are expressively equivalent.*
- coSafetyLTL *and* F(pLTL) *are expressively complete w.r.t.* $[\![\mathsf{LTL}]\!] \cap$ coSAFETY.

Reference:

Edward Y. Chang, Zohar Manna, and Amir Pnueli (1992). "Characterization of Temporal Property Classes". In: *Proceedings of the 19th International Colloquium on Automata, Languages and Programming*. Ed. by Werner Kuich. Vol. 623. Lecture Notes in Computer Science. Springer, pp. 474–486. DOI: 10.1007/3-540-55719-9_97


ω-REG
S1S
NBA
ETL
coSAFETY $(K \cdot \Sigma^\omega)$
???
tDBA
???
ω-SF
S1S[FO]
cf-DRA
LTL
SF · Σ^ω
coSafety-FO
cf-tDBA
coSafetyLTL, F(pLTL)
SAFETY $(\overline{K \cdot \Sigma^\omega})$
???
DSA
???

Proposition

$[\![\mathsf{coSafetyLTL}]\!]^{<\omega} \subsetneq [\![\mathsf{LTL}]\!]^{<\omega}$

Proof.

- It is simple to prove that, for all $\phi \in \mathsf{coSafetyLTL}$, $\mathcal{L}^{<\omega}(\phi) = \mathcal{L}^{<\omega}(\phi) \cdot \Sigma^*$. In particular, $|\mathcal{L}^{<\omega}(\phi)| = \omega$ for all $\phi \in \mathsf{coSafetyLTL}$.
- In $\mathsf{LTL_f}$ we can use the *weak tomorrow* operator to hook the last position of a finite word.

$$\psi := p \wedge \widetilde{\mathsf{X}}\bot$$

The formula ψ is such that $|\mathcal{L}^{<\omega}(\psi)| = 1$. Therefore, it can't be expressed in coSafetyLTL over finite words.

□

Proposition

$[\![\mathsf{coSafetyLTL}]\!]^{<\omega} \subsetneq [\![\mathsf{LTL}]\!]^{<\omega}$

Proposition

$[\![\mathsf{coSafetyLTL}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega} = [\![\mathsf{LTL}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega}$

Reference:

Alessandro Cimatti et al. (2022). "A first-order logic characterisation of safety and co-safety languages". In: *Foundations of Software Science and Computation Structures - 25th International Conference, FOSSACS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings*. Ed. by Patricia Bouyer and Lutz Schröder. Vol. 13242. Lecture Notes in Computer Science. Springer, pp. 244–263. DOI: 10.1007/978-3-030-99253-8_13. URL: https://doi.org/10.1007/978-3-030-99253-8%5C_13

$$[\![\mathsf{LTL}]\!] \cap \mathsf{coSAFETY}$$

$$=$$

$$[\![\mathsf{coSafetyLTL}]\!]$$

$$=$$

$$[\![\mathsf{F}(\mathsf{pLTL})]\!]$$

$$=$$

$$[\![\mathsf{coSafetyLTL}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega}$$

$$=$$

$$[\![\mathsf{LTL}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega}$$

The safety fragment of LTL

Definition

The *safety fragment of* LTL is the set of languages in this set:

$$[\![\mathsf{LTL}]\!] \cap \mathsf{SAFETY}$$

We will see four characterizations in terms of:

- regular expressions
- first-order logic
- automata
- temporal logic

Definition

The *safety fragment of* LTL is the set of languages in this set:

$$[\![\mathsf{LTL}]\!] \cap \mathsf{SAFETY}$$

ω-regular expressions

$\overline{\mathsf{SF} \cdot \Sigma^\omega} = \{\overline{K \cdot \Sigma^\omega} \mid K \in \mathsf{SF}\}$

- the "SF " part corresponds to LTL
- the "$\overline{\cdot \Sigma^\omega}$" part corresponds to being a safety fragment

---

Ina Schiering and Wolfgang Thomas (1996). "Counter-free automata, first-order logic, and star-free expressions extended by prefix oracles". In: *Developments in Language Theory, II (Magdeburg, 1995), Worl Sci. Publishing, River Edge, NJ*, pp. 166–175

The safety fragment of LTL

Definition

The *safety fragment of* LTL is the set of languages in this set:

$$[\![\mathsf{LTL}]\!] \cap \mathsf{SAFETY}$$

First-order logic

We define Safety-FO as the fragment of S1S[FO] in which quantifiers are bounded as follows:

- $\exists y \,.\, (x < y < z \wedge \ldots)$
- $\forall y \,.\, (x < y \rightarrow \ldots)$

Alessandro Cimatti et al. (2022). "A first-order logic characterisation of safety and co-safety languages". In: *Foundations of Software Science and Computation Structures - 25th International Conference, FOSSACS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings*. Ed. by Patricia Bouyer and Lutz Schröder. Vol. 13242. Lecture Notes in Computer Science. Springer, pp. 244–263. DOI: 10.1007/978-3-030-99253-8_13. URL: https://doi.org/10.1007/978-3-030-99253-8%5C_13

The safety fragment of LTL

Definition

The *safety fragment of* LTL is the set of languages in this set:

$$[\![\mathsf{LTL}]\!] \cap \mathsf{SAFETY}$$

First-order logic

Example

$\phi(x) := \forall y \,.\, ((x < y \wedge G(y)) \rightarrow \exists z \,.\, (x < z < y \wedge R(z)))$

Definition

The *safety fragment of* LTL is the set of languages in this set:

$$[\![\mathsf{LTL}]\!] \cap \mathsf{SAFETY}$$

First-order logic

- the "first-order" part corresponds to LTL
- the "bounded quantifiers" part corresponds to being a safety fragment

Definition

The *safety fragment of* LTL is the set of languages in this set:

$$[\![\mathsf{LTL}]\!] \cap \mathsf{SAFETY}$$

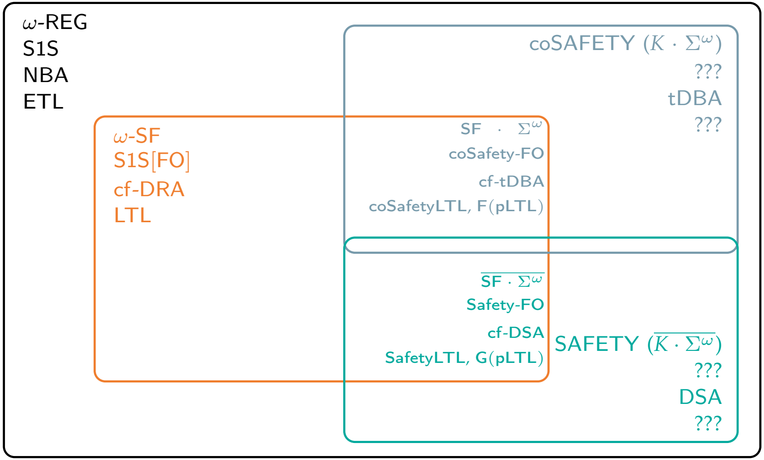
Automata

cf-DSA = counter-free DSA

- the "counter-free" part corresponds to LTL
- the "DSA " part corresponds to being a safety fragment

Ina Schiering and Wolfgang Thomas (1996). “Counter-free automata, first-order logic, and star-free expressions extended by prefix oracles”. In: *Developments in Language Theory, II (Magdeburg, 1995), Worl Sci. Publishing, River Edge, NJ*, pp. 166–175

Set-theoretic view of the (co)safety fragment of LTL



The safety fragment of LTL

We say that a temporal logic $\mathbb{L}$ is *safety* iff, for any $\phi \in \mathbb{L}$, $\mathcal{L}(\phi)$ is *safety*.

SafetyLTL

Definition

$\phi := p \mid \neg p \mid \phi \vee \phi \mid \phi \wedge \phi \mid \mathsf{X}\phi \mid \mathsf{G}\phi \mid \phi \mathbin{\mathsf{R}} \phi$

Example:

$\mathsf{G}(r \rightarrow \mathsf{X}\mathsf{X}g)$

G(pLTL)

Definition

$\phi := \mathsf{G}(\alpha)$, where $\alpha \in$ pLTL, that is α is a pure-past LTL formula.

Example:

$\mathsf{G}(\widetilde{\mathsf{Y}}\widetilde{\mathsf{Y}}r \rightarrow g)$

G(pLTL) is the canonical form of SafetyLTL.

The cosafety fragment of LTL

Proposition

- $\phi \in \mathsf{SafetyLTL}$ *iff* $\texttt{nnf}(\neg\phi) \in \mathsf{coSafetyLTL}$
- $\phi \in \mathsf{G}(\mathsf{pLTL})$ *iff* $\texttt{nnf}(\neg\phi) \in \mathsf{F}(\mathsf{pLTL})$

The cosafety fragment of LTL

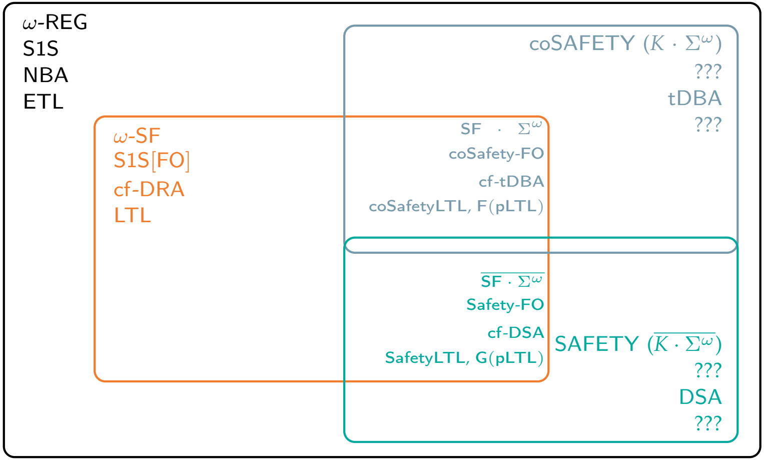
Theorem

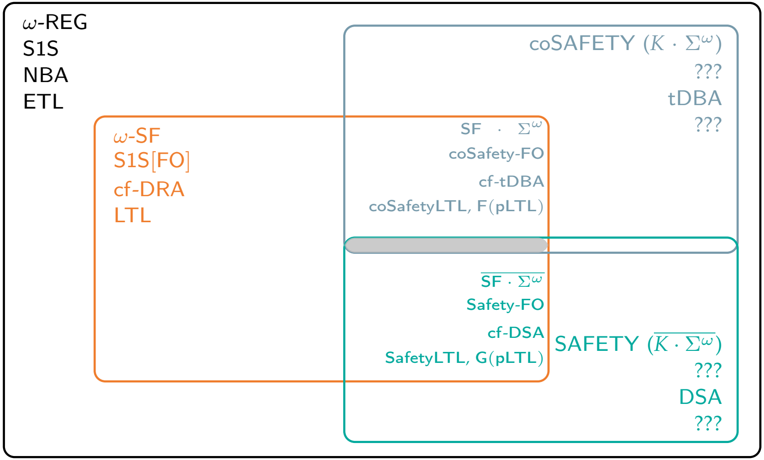
- SafetyLTL *and* G(pLTL) *are expressively equivalent.*
- SafetyLTL *and* G(pLTL) *are expressively complete w.r.t.* $[\![\mathsf{LTL}]\!] \cap \mathsf{SAFETY}$.

Reference:

Edward Y. Chang, Zohar Manna, and Amir Pnueli (1992). "Characterization of Temporal Property Classes". In: *Proceedings of the 19th International Colloquium on Automata, Languages and Programming*. Ed. by Werner Kuich. Vol. 623. Lecture Notes in Computer Science. Springer, pp. 474–486. DOI: 10.1007/3-540-55719-9_97

Set-theoretic view of (co)safety ω-languages





ω-REG
S1S
NBA
ETL
ω-SF
S1S[FO]
cf-DRA
LTL
coSAFETY $(K \cdot \Sigma^\omega)$
???
tDBA
???
$\mathsf{SF} \cdot \Sigma^\omega$
coSafety-FO
cf-tDBA
coSafetyLTL, F(pLTL)
$\overline{\mathsf{SF} \cdot \Sigma^\omega}$
Safety-FO
cf-DSA
SafetyLTL, G(pLTL)
SAFETY $(\overline{K \cdot \Sigma^\omega})$
???
DSA
???

LTL ∩ coSAFETY ∩ SAFETY

- We denote with $\mathbb{B}$ the set of Boolean formulas.
- We denote with $\mathsf{LTL}[\mathsf{X}]$ the set of LTL formulas in which the only temporal operator that is used is the *tomorrow* (X).

Proposition

- $\mathbb{B} \subseteq \mathsf{LTL} \cap \mathsf{coSAFETY} \cap \mathsf{SAFETY}$
- $\mathsf{LTL}[\mathsf{X}] \subseteq \mathsf{LTL} \cap \mathsf{coSAFETY} \cap \mathsf{SAFETY}$

Other safety and cosafety fragments

Cosafety

- We denote with $\mathsf{LTL}[\mathsf{X}, \mathsf{F}]$ the set of coSafetyLTL formulas in which the only temporal operators that are used are the *tomorrow* (X) and the *eventually* (F).
- Clearly, $\mathsf{LTL}[\mathsf{X}, \mathsf{F}]$ is a cosafety logic, but it is strictly less expressive than coSafetyLTL.

Proposition

$[\![\mathsf{LTL}[\mathsf{X}, \mathsf{F}]]\!] \subsetneq [\![\mathsf{coSafetyLTL}]\!]$

E.g. $p \,\mathsf{U}\, q$ is not definable in $\mathsf{LTL}[\mathsf{X}, \mathsf{F}]$.

Safety

- We denote with $\mathsf{LTL}[\mathsf{X}, \mathsf{G}]$ the set of SafetyLTL formulas in which the only temporal operators that are used are the *tomorrow* (X) and the *globally* (G).
- Clearly, $\mathsf{LTL}[\mathsf{X}, \mathsf{G}]$ is a safety logic, but it is strictly less expressive than SafetyLTL.

Proposition

$[\![\mathsf{LTL}[\mathsf{X}, \mathsf{G}]]\!] \subsetneq [\![\mathsf{SafetyLTL}]\!]$

E.g. $p \,\mathsf{R}\, q$ is not definable in $\mathsf{LTL}[\mathsf{X}, \mathsf{G}]$.

The Temporal Hierarchy

Reactivity

$\bigcup_{n=1}^{\infty}\left(\bigwedge_{i=1}^{n}(\mathsf{GF}\alpha_i \vee \mathsf{FG}\beta_i) \right)$

Liveness

$\mathsf{GF}\alpha$

Persistence

$\mathsf{FG}\alpha$

Obligation

$\bigcup_{n=1}^{\infty}\left(\bigwedge_{i=1}^{n}(\mathsf{G}\alpha_i \vee \mathsf{F}\beta_i) \right)$

Safety

$\mathsf{G}\alpha$

Cosafety

$\mathsf{F}\alpha$

Legend:

- $\alpha, \alpha_i, \beta, \beta_i$ are pure-past LTL formulas (pLTL)
- $\rightarrow$ denotes set inclusion

Theorem

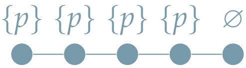
Reactivity $= [\![\mathsf{LTL}]\!]$

Zohar Manna and Amir Pnueli (1990). "A hierarchy of temporal properties (invited paper, 1989)". In: *Proceedings of the 9th annual ACM symposium on Principles of distributed computing*, pp. 377–410. DOI: 10.1145/93385.93442

Kupferman and Vardi's Classification of the safety properties of LTL

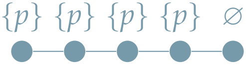
Classification of Safety Properties
by Kupferman and Vardi

Consider the formula $\mathsf{G}(p)$. The following trace is a *bad prefix*:



Recall that $\sigma \in \Sigma^*$ is a bad prefix for a language $\mathcal{L}$ iff $\sigma \cdot \sigma' \notin \mathcal{L}$, for all $\sigma' \in \Sigma^\omega$.

Consider the formula $\mathsf{G}(p)$. The following trace is a *bad prefix*:

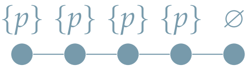


Recall that $\sigma \in \Sigma^*$ is a bad prefix for a language $\mathcal{L}$ iff $\sigma \cdot \sigma' \notin \mathcal{L}$, for all $\sigma' \in \Sigma^\omega$.
Consider now the formula $\mathsf{G}(p \vee (\mathsf{X}q \wedge \mathsf{X}\neg q))$.

- it is equivalent to $\mathsf{G}(p)$
- therefore, it is a safety formula
- its set of *bad prefixes* is the same as the one of $\mathsf{G}(p)$

Classification of Safety Properties
by Kupferman and Vardi

Consider the formula $\mathsf{G}(p)$. The following trace is a *bad prefix*:



Recall that $\sigma \in \Sigma^*$ is a bad prefix for a language $\mathcal{L}$ iff $\sigma \cdot \sigma' \notin \mathcal{L}$, for all $\sigma' \in \Sigma^\omega$. Consider now the formula $\mathsf{G}(p \vee (\mathsf{X}q \wedge \mathsf{X}\neg q))$.

- it is equivalent to $\mathsf{G}(p)$
- therefore, it is a safety formula
- its set of *bad prefixes* is the same as the one of $\mathsf{G}(p)$

Nevertheless, the previous prefix does *not* tell the whole story about the violation of $\mathsf{G}(p \vee (\mathsf{X}q \wedge \mathsf{X}\neg q))$. In fact:

- Negation of the above formula:

$$\mathsf{F}(\neg p \wedge (\mathsf{X}\neg q \vee \mathsf{X}q))$$

- Any violation depends on the fact that at certain point:
  - p is false and
  - in the *next* state q or $\neg q$ holds. (*this is always true*)
- In the previous prefix, the point in which $\neg p$ holds does *not* have a successor:
  - the prefix is *not informative*

Classification of Safety Properties
by Kupferman and Vardi

Consider the formula $\mathsf{G}(p)$. The following trace is a *bad prefix*:

$$\{p\}\ \{p\}\ \{p\}\ \{p\}\ \varnothing$$

Recall that $\sigma \in \Sigma^*$ is a bad prefix for a language $\mathcal{L}$ iff $\sigma \cdot \sigma' \notin \mathcal{L}$, for all $\sigma' \in \Sigma^\omega$. Consider now the formula $\mathsf{G}(p \vee (\mathsf{X}q \wedge \mathsf{X}\neg q))$.

- it is equivalent to $\mathsf{G}(p)$
- therefore, it is a safety formula
- its set of *bad prefixes* is the same as the one of $\mathsf{G}(p)$

Nevertheless, the previous prefix does *not* tell the whole story about the violation of $\mathsf{G}(p \vee (\mathsf{X}q \wedge \mathsf{X}\neg q))$. In fact:

- Negation of the above formula:

  $$\mathsf{F}(\neg p \wedge (\mathsf{X}\neg q \vee \mathsf{X}q))$$

- This prefix is *informative* for the formula:

  $$\{p\}\ \{p\}\ \{p\}\ \{p\}\ \varnothing\ \varnothing$$

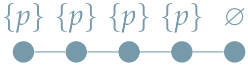
Classification of Safety Properties
by Kupferman and Vardi

- Consider the specification:

$$\mathsf{G}(p \lor (\mathsf{X}q \land \phi \land \mathsf{X}\neg q))$$

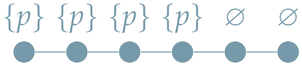
  where ϕ is a very complex Boolean formula.

- If the user is given the prefix



  then it is very hard for him/her to notice that the specification contains a redundant part ($\mathsf{X}q \land \mathsf{X}\neg q$).

- If instead the user is given this prefix



  then he/she
  - notice that the state in which $\neg p$ holds has a successor
  - inspect the parts of the specification that talk about the successor state ($\mathsf{X}q \land \mathsf{X}\neg q$)
  - notice that they are *redundant*
  - and finally remove them.

Classification of Safety Properties
by Kupferman and Vardi

- This intuition of a prefix that *"tells the whole story"* is the base for a classification of safety properties in three distinct safety levels.
- This intuition is formalized by defining the notion of *informative prefix*
  - it is based on the semantics of LTL over finite traces

Reference:

Orna Kupferman and Moshe Y Vardi (2001). “Model checking of safety properties”. In: *Formal Methods in System Design* 19.3, pp. 291–314. DOI: 10.1023/A:1011254632723

Usage:

- Detect the cause of unconsistent specifications:
  - e.g.: in formulas like $\mathsf{G}(p \vee (\mathsf{X}q \wedge \phi \wedge \mathsf{X}\neg q))$, the cause of unconsistency may not be easy to notice by the user, especially in more complicated examples
- Efficient automata construction
  - The automaton that recognizes all and only the informative prefixes of a formula is *exponentially smaller* than the automaton recognizing all and only the bad prefixes.
  - $\Rightarrow$ Efficient algorithms for model checking

Reference:

Orna Kupferman and Moshe Y Vardi (2001). "Model checking of safety properties". In: *Formal Methods in System Design* 19.3, pp. 291–314. DOI: 10.1023/A:1011254632723

Recall that $\mathtt{nnf}(\psi)$ is the *negation normal form* of ψ, that is, a formula equivalent to ψ but with negations only applied to atomic propositions.

We define a new semantics for LTL interpreted over finite traces, that we denote with $\models_{\mathrm{KV}}$.

- $\sigma, i \models_{\mathrm{KV}} p$ iff $p \in \sigma_i$
- $\sigma, i \models_{\mathrm{KV}} \phi_1 \vee \phi_2$ iff $\sigma, i \models_{\mathrm{KV}} \phi_1$ or $\sigma, i \models_{\mathrm{KV}} \phi_2$
- $\sigma, i \models_{\mathrm{KV}} \phi_1 \wedge \phi_2$ iff $\sigma, i \models_{\mathrm{KV}} \phi_1$ and $\sigma, i \models_{\mathrm{KV}} \phi_2$
- $\sigma, i \models_{\mathrm{KV}} \mathsf{X}\phi$ iff $i + 1 < |\sigma|$ and $\sigma, i+1 \models_{\mathrm{KV}} \phi$

Recall that $\mathtt{nnf}(\psi)$ is the *negation normal form* of ψ, that is, a formula equivalent to ψ but with negations only applied to atomic propositions.

We define a new semantics for LTL interpreted over finite traces, that we denote with $\models_{\mathrm{KV}}$.

- $\sigma, i \models_{\mathrm{KV}} p$ iff $p \in \sigma_i$
- $\sigma, i \models_{\mathrm{KV}} \phi_1 \vee \phi_2$ iff $\sigma, i \models_{\mathrm{KV}} \phi_1$ or $\sigma, i \models_{\mathrm{KV}} \phi_2$
- $\sigma, i \models_{\mathrm{KV}} \phi_1 \wedge \phi_2$ iff $\sigma, i \models_{\mathrm{KV}} \phi_1$ and $\sigma, i \models_{\mathrm{KV}} \phi_2$
- $\sigma, i \models_{\mathrm{KV}} \mathsf{X}\phi$ iff $i + 1 < |\sigma|$ and $\sigma, i + 1 \models_{\mathrm{KV}} \phi$
- $\sigma, i \models_{\mathrm{KV}} \mathsf{F}\phi$ iff $\exists i \leq j < |\sigma|$ and $\sigma, j \models_{\mathrm{KV}} \phi$
- $\sigma, i \models_{\mathrm{KV}} \mathsf{G}\phi$ is always false

Recall that $\mathtt{nnf}(\psi)$ is the *negation normal form* of ψ, that is, a formula equivalent to ψ but with negations only applied to atomic propositions.

We define a new semantics for LTL interpreted over finite traces, that we denote with $\models_{\mathrm{KV}}$.

- $\sigma, i \models_{\mathrm{KV}} p$ iff $p \in \sigma_i$
- $\sigma, i \models_{\mathrm{KV}} \phi_1 \vee \phi_2$ iff $\sigma, i \models_{\mathrm{KV}} \phi_1$ or $\sigma, i \models_{\mathrm{KV}} \phi_2$
- $\sigma, i \models_{\mathrm{KV}} \phi_1 \wedge \phi_2$ iff $\sigma, i \models_{\mathrm{KV}} \phi_1$ and $\sigma, i \models_{\mathrm{KV}} \phi_2$
- $\sigma, i \models_{\mathrm{KV}} \mathsf{X}\phi$ iff $i+1 < |\sigma|$ and $\sigma, i+1 \models_{\mathrm{KV}} \phi$
- $\sigma, i \models_{\mathrm{KV}} \phi_1 \mathsf{U} \phi_2$ iff $\exists i \leq j < |\sigma| \,.\, \sigma, j \models_{\mathrm{KV}} \phi_2$ and $\forall i \leq k < j \,.\, \sigma, k \models_{\mathrm{KV}} \phi_1$
- $\sigma, i \models_{\mathrm{KV}} \phi_1 \mathsf{R} \phi_2$ iff $\exists i \leq j \leq |\sigma| \,.\, \sigma, j \models_{\mathrm{KV}} \phi_1$ and $\forall i \leq k < j \,.\, \sigma, k \models_{\mathrm{KV}} \phi_2$

Intuition:

If $\sigma \models_{KV} \mathtt{nnf}(\neg\phi)$, then σ carries all the information to violate ϕ over infinite traces.

Remark

The definition of $\models_{KV}$ is exactly the one used in *Bounded Model Checking* for defining the truth of an LTL formula over a finite trace.

Reference:

Armin Biere et al. (2003). "Bounded model checking". In: *Adv. Comput.* 58, pp. 117–148. DOI: 10.1016/S0065-2458(03)58003-2. URL: https://doi.org/10.1016/S0065-2458(03)58003-2

Definition (Informative Prefix)

Let ϕ be an LTL formula over $\mathcal{AP}$ and let $\sigma \in (2^{\mathcal{AP}})^+$ be a finite trace over $2^{\mathcal{AP}}$.

σ is an *informative prefix for* ϕ
iff
$\sigma \models_{\text{KV}} \texttt{nnf}(\neg\phi)$

Note: in the original paper by Kupferman and Vardi, informative prefixes are defined using a mapping L. This is equivalent to our definition.

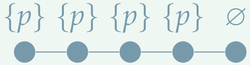
Definition (Informative Prefix)

Let ϕ be an LTL formula over $\mathcal{AP}$ and let $\sigma \in (2^{\mathcal{AP}})^+$ be a finite trace over $2^{\mathcal{AP}}$.

σ is an *informative prefix for* ϕ
iff
$\sigma \models_{\text{KV}} \texttt{nnf}(\neg\phi)$

Example:

This prefix is *informative for* $\mathsf{G}(p)$.



$\texttt{nnf}(\neg\mathsf{G}(p)) := \mathsf{F}(\neg p)$

Classification of Safety Properties
by Kupferman and Vardi

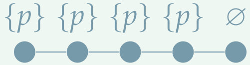
Definition (Informative Prefix)

Let ϕ be an LTL formula over $\mathcal{AP}$ and let $\sigma \in (2^{\mathcal{AP}})^+$ be a finite trace over $2^{\mathcal{AP}}$.

σ is an *informative prefix for* ϕ
iff
$\sigma \models_{KV} \mathtt{nnf}(\neg\phi)$

Example:

This prefix is *<u>not</u> informative for* $\phi := \mathsf{G}(p \vee (\mathsf{X}q \wedge \mathsf{X}\neg q))$.



$\mathtt{nnf}(\neg\phi) := \mathsf{F}(\neg p \wedge (\mathsf{X}\neg q \vee \mathsf{X}q))$

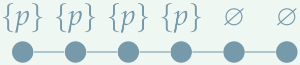
Definition (Informative Prefix)

Let ϕ be an LTL formula over $\mathcal{AP}$ and let $\sigma \in (2^{\mathcal{AP}})^+$ be a finite trace over $2^{\mathcal{AP}}$.

σ is an *informative prefix for* ϕ
iff
$\sigma \models_{\mathrm{KV}} \mathtt{nnf}(\neg\phi)$

Example:

This prefix is *informative for* $\phi := \mathsf{G}(p \vee (\mathsf{X}q \wedge \mathsf{X}\neg q))$.



$\mathtt{nnf}(\neg\phi) := \mathsf{F}(\neg p \wedge (\mathsf{X}\neg q \vee \mathsf{X}q))$

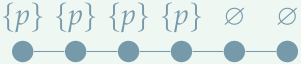
Definition (Informative Prefix)

Let ϕ be an LTL formula over $\mathcal{AP}$ and let $\sigma \in (2^{\mathcal{AP}})^+$ be a finite trace over $2^{\mathcal{AP}}$.

σ is an *informative prefix for* ϕ
iff
$\sigma \models_{\mathrm{KV}} \mathtt{nnf}(\neg\phi)$

Example:

This prefix is *not informative for* $\phi := \big(\mathsf{G}(q \vee \mathsf{FG}p) \wedge \mathsf{G}(r \vee \mathsf{FG}\neg p)\big) \vee \mathsf{G}q \vee \mathsf{G}r$.



$\mathtt{nnf}(\neg\phi) := \big(\mathsf{F}(\neg q \wedge \mathsf{GF}\neg p) \vee \mathsf{F}(\neg r \wedge \mathsf{GF}p)\big) \wedge \mathsf{F}\neg q \wedge \mathsf{F}\neg r$

Definition (Informative Prefix)

Let ϕ be an LTL formula over $\mathcal{AP}$ and let $\sigma \in (2^{\mathcal{AP}})^+$ be a finite trace over $2^{\mathcal{AP}}$.

σ is an *informative prefix for* ϕ
iff
$\sigma \models_{\mathrm{KV}} \mathtt{nnf}(\neg\phi)$

Example:

This prefix is *not informative for* $\phi := (\mathsf{G}(q \vee \mathsf{FG}p) \wedge \mathsf{G}(r \vee \mathsf{FG}\neg p)) \vee \mathsf{G}q \vee \mathsf{G}r$.

$\mathsf{G}(\dots)$ is always false under $\models_{\mathrm{KV}}$: no prefix is informative for ϕ

$\mathtt{nnf}(\neg\phi) := (\mathsf{F}(\neg q \wedge \mathsf{GF}\neg p) \vee \mathsf{F}(\neg r \wedge \mathsf{GF}p)) \wedge \mathsf{F}\neg q \wedge \mathsf{F}\neg r$

Definition (Informative Prefix)

Let ϕ be an LTL formula over $\mathcal{AP}$ and let $\sigma \in (2^{\mathcal{AP}})^+$ be a finite trace over $2^{\mathcal{AP}}$.

σ is an *informative prefix for* ϕ
iff
$\sigma \models_{\mathrm{KV}} \mathtt{nnf}(\neg\phi)$

Remark:

Given σ and ϕ, checking whether $\sigma \models_{\mathrm{KV}} \phi$ can be done in time $\mathcal{O}(|\sigma| \cdot |\phi|)$.

Let ϕ be any LTL formula such that $\mathcal{L}(\phi)$ is a safety language. The definition of informative prefix is used to classify such formulas ϕ into three types:

1. intentionally safe
2. accidentally safe
3. pathologically safe

Classification of Safety Properties
by Kupferman and Vardi

Let ϕ be any LTL formula such that $\mathcal{L}(\phi)$ is a safety language. The definition of informative prefix is used to classify such formulas ϕ into three types:

1. intentionally safe

ϕ is intentionally safe iff *all* bad prefixes are informative.

For example:

- the formula $\mathsf{G}(p)$ is intentionally safe.
- the formula $\mathsf{G}(p \vee (\mathsf{X}q \wedge \mathsf{X}\neg q))$ is *not* intentionally safe, because $\langle \{p\}, \{p\}, \{p\}, \{p\}, \varnothing \rangle$ is a bad prefix but it is not informative.

2. accidentally safe
3. pathologically safe

Let ϕ be any LTL formula such that $\mathcal{L}(\phi)$ is a safety language. The definition of informative prefix is used to classify such formulas ϕ into three types:

1. intentionally safe
2. accidentally safe

ϕ is accidentally safe iff *(i)* not all the bad prefixes of ψ are informative, but *(ii)* every $\sigma \in (2^{AP})^\omega$ that violates ϕ has an informative bad prefix.

For example:

- $\mathsf{G}(p \vee (\mathsf{X}q \wedge \mathsf{X}\neg q))$ is accidentally safe: $\langle \{p\}, \{p\}, \{p\}, \{p\}, \varnothing \rangle$ is a bad prefix but it is not informative. However, every infinite trace violating the formula has an informative prefix of type $\{p\}^* \cdot \varnothing \cdot \varnothing$.

3. pathologically safe

Let ϕ be any LTL formula such that $\mathcal{L}(\phi)$ is a safety language. The definition of informative prefix is used to classify such formulas ϕ into three types:

1. intentionally safe
2. accidentally safe
3. pathologically safe

ϕ is pathologically safe iff there is a $\sigma \in (2^{\mathcal{AP}})^\omega$ that violates ϕ and has no informative bad prefixes.

For example:

- $(\mathsf{G}(q \vee \mathsf{FG}p) \wedge \mathsf{G}(r \vee \mathsf{FG}\neg p)) \vee \mathsf{G}q \vee \mathsf{G}r$
  - the computation $\varnothing^\omega$ violates the formula

    $$\varnothing^\omega \models (\mathsf{F}(\neg q \wedge \mathsf{GF}\neg p) \vee \mathsf{F}(\neg r \wedge \mathsf{GF}p)) \wedge \mathsf{F}(\neg q) \wedge \mathsf{F}(\neg r)$$

  - but each of its prefixes σ is *not informative* because $\sigma \not\models_{KV} (\mathsf{F}(\neg q \wedge \mathsf{GF}\neg p) \vee \mathsf{F}(\neg r \wedge \mathsf{GF}p)) \wedge \mathsf{F}(\neg q) \wedge \mathsf{F}(\neg r)$, but no finite prefix is such.

Classification of Safety Properties
by Kupferman and Vardi

Let ϕ be any LTL formula such that $\mathcal{L}(\phi)$ is a safety language. The definition of informative prefix is used to classify such formulas ϕ into three types:

1. intentionally safe
2. accidentally safe
3. pathologically safe

Formulas that are accidentally safe or pathologically safe are *needlessly complicated*:

- They contain a redundancy that can be eliminated.
- If a user wrote a pathologically safe formula, then probably he/she didn't mean to write a safety formula.
- This classification helps in detecting unconsistent or redundant specifications.

Theorem

For any formula ϕ of SafetyLTL*, it holds that ϕ is either intentionally or accidentally safe.*

Proof.

- By the duality between SafetyLTL and coSafetyLTL, we have that $\mathtt{nnf}(\neg\phi)$ is a formula of coSafetyLTL and is equivalent to ϕ. Let $\psi := \mathtt{nnf}(\neg\phi)$.
- Let $\sigma = \langle \sigma_0, \sigma_1, \ldots \rangle$ be an infinite trace that satisfies ψ, that is $\sigma \models \psi$.
- Since, by definition of coSafetyLTL, ψ contains only X and U as temporal operators, there exists a furthermost time point i such that $\sigma_{[0,i]} \models \psi$ (under finite traces semantics).

Theorem

For any formula ϕ of SafetyLTL, *it holds that ϕ is either intentionally or accidentally safe.*

Proof.

- Since on the operators X and U the definitions of $\models$ and $\models_{KV}$ coincide, we have also that $\sigma_{[0,i]} \models_{KV} \psi$. Therefore, by definition, $\sigma_{[0,i]}$ is an *informative prefix*.
- It follows that every infinite trace that violates ϕ has an informative prefix, thus ϕ is either intentionally or accidentally safe. □

As we will see, this classification is exploited for having efficient verification algorithms.

- An automaton that recognizes only the bad prefixes that are *informative* can be built exponentially more efficiently than the automaton for *all* the bad prefixes.
- Moreover, in practice, almost all the benefits that one can obtain from an automaton for the bad prefixes can also be obtained from an automaton for the *informative* bad prefixes.
  - for example, we can perform *model checking* algorithms considering only the informative bad prefixes
  - since there may be bad prefixes that are not informative but may become informative if extended, *minimality* of counterexamples is the only thing that is sacrified when dealing with informative bad prefixes.

RECOGNIZING SAFETY

Algorithms & Complexity

Outline

In this part, we will answer to these questions:

- Can we effectively determine whether a NBA recognizes a safety property? If so, with which complexity?
- Can we effectively determine whether a LTL formula recognizes a safety property? If so, with which complexity?
- How complex is building the *automaton* for the set of *bad prefixes* of a safety ω-regular language?

Theorem (Alpern & Schneider (1987), Sistla (1994))

Given a NBA $\mathcal{A}$*, checking whether* $\mathcal{L}(\mathcal{A})$ *is safety is can be performed effectively.*

References:

- Bowen Alpern and Fred B. Schneider (1987). "Recognizing Safety and Liveness". In: *Distributed Comput.* 2.3, pp. 117–126. DOI: 10.1007/BF01782772. URL: https://doi.org/10.1007/BF01782772
- A Prasad Sistla (1994). "Safety, liveness and fairness in temporal logic". In: *Formal Aspects of Computing* 6.5, pp. 495–511. DOI: 10.1007/BF01211865

Theorem (Alpern & Schneider (1987), Sistla (1994))

Given a NBA $\mathcal{A}$, *checking whether* $\mathcal{L}(\mathcal{A})$ *is safety is can be performed effectively.*

We prove this theorem.

Definition (Reduced NBA)

A NBA $\mathcal{A} = \langle Q, \Sigma, I, \Delta, F \rangle$ is *reduced* (rNBA, for short) iff from every state in Q there exists a path (of length at least 1) reaching a final state in F.

- Every NBA $\mathcal{A}$ can be turned into rNBA $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$, by removing the states (and its incoming transitions) from which no final state is reachable.
  - **Important:** this can add *undefined* transitions
- This can be done in *time* linear in $|Q|$ and in *space* nondeterministic logarithmic in $|Q|$ (Savitch's Theorem).

Definition (Closure of a rNBA)

Given a rNBA $\mathcal{A} = \langle Q, \Sigma, I, \Delta, F \rangle$, we define the *closure of* $\mathcal{A}$, denoted with $\mathtt{cl}(\mathcal{A})$, as the automaton $\mathtt{cl}(\mathcal{A}) = \langle Q, \Sigma, I, \Delta, Q \rangle$.

- We will use the automaton $\mathtt{cl}(\mathcal{A})$ to determine whether $\mathcal{L}(\mathcal{A})$ is a safety property.
- **Important:** the automaton $\mathtt{cl}(\mathcal{A})$ rejects a word in Σ^ω only by attempting an *undefined transition*.

Theorem

For any rNBA $\mathcal{A}$, *it holds that* $\mathcal{L}(\mathcal{A})$ *is a safety property iff* $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathtt{cl}(\mathcal{A}))$.

Proof.

($\Rightarrow$)

- Suppose that $\mathcal{L}(\mathcal{A})$ is a safety property. We show that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathtt{cl}(\mathcal{A}))$.
- $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathtt{cl}(\mathcal{A}))$: trivial, because $\mathtt{cl}(\mathcal{A})$ is obtained from $\mathcal{A}$ by making all states as accepting.

Theorem

For any rNBA $\mathcal{A}$, *it holds that* $\mathcal{L}(\mathcal{A})$ *is a safety property iff* $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathtt{cl}(\mathcal{A}))$.

Proof.

$(\Rightarrow)$

- We show that $\mathcal{L}(\mathtt{cl}(\mathcal{A})) \subseteq \mathcal{L}(\mathcal{A})$. We first show that, for any $\sigma \in \mathcal{L}(\mathtt{cl}(\mathcal{A}))$, it holds that:

$$\forall i \geq 0 \,.\, \exists \sigma' \in \Sigma^\omega \,.\, \sigma_{[0,i]} \cdot \sigma' \in \mathcal{L}(\mathcal{A})$$

Theorem

For any rNBA $\mathcal{A}$, *it holds that* $\mathcal{L}(\mathcal{A})$ *is a safety property iff* $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\texttt{cl}(\mathcal{A}))$.

Proof.

($\Rightarrow$)

- Let $\sigma \in \mathcal{L}(\texttt{cl}(\mathcal{A}))$. Choose *any* prefix $\sigma_{[0,i]}$ and let q_i be any of the states reached by $\mathcal{A}$ after reading $\sigma_{[0,i]}$.
- Since $\mathcal{A}$ is *reduced*, there exists a final state q_{f_1} reachable from q_i when $\mathcal{A}$ reads some $\beta_0 \in \Sigma^*$.
- Similarly, since $\mathcal{A}$ is *reduced*, there exists a final state q_{f_2} reachable from q_{f_1} when $\mathcal{A}$ reads some $\beta_1 \in \Sigma^*$.
- ... and so on and so forth ...

Theorem

For any rNBA $\mathcal{A}$, *it holds that* $\mathcal{L}(\mathcal{A})$ *is a safety property iff* $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathtt{cl}(\mathcal{A}))$.

Proof.

($\Rightarrow$)

- Let $\beta = \beta_0 \cdot \beta_1 \cdot \ldots$. Since, by construction, $\sigma_{[0,i]} \cdot \beta$ induces $\mathcal{A}$ to visit final state *infinitely often*, the word $\sigma_{[0,i]} \cdot \beta$ belongs to $\mathcal{L}(\mathcal{A})$.
- We have proved that, for any $\sigma \in \mathcal{L}(\mathtt{cl}(\mathcal{A}))$, it holds that:

$$\forall i \geq 0 \,.\, \exists \sigma' \in \Sigma^\omega \,.\, \sigma_{[0,i]} \cdot \sigma' \in \mathcal{L}(\mathcal{A})$$

Theorem

For any rNBA $\mathcal{A}$, *it holds that* $\mathcal{L}(\mathcal{A})$ *is a safety property iff* $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathtt{cl}(\mathcal{A}))$.

Proof.

($\Rightarrow$)

- Since by hypothesis $\mathcal{L}(\mathcal{A})$ is a safety property, for all $\sigma \in \Sigma^\omega$, we have that,

$$\sigma \notin \mathcal{L}(\mathcal{A}) \leftrightarrow \exists i \geq 0 \, . \, \forall \sigma' \in \Sigma^\omega \, . \, \sigma_{[0,i]} \cdot \sigma' \notin \mathcal{L}(\mathcal{A})$$

- Since before we proved that the rightmost part of the above equation is false for any $\sigma \in \mathcal{L}(\mathtt{cl}(\mathcal{A}))$, we have that $\sigma \in \mathcal{L}(\mathcal{A})$.

Theorem

For any rNBA $\mathcal{A}$, *it holds that* $\mathcal{L}(\mathcal{A})$ *is a safety property iff* $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathtt{cl}(\mathcal{A}))$.

Proof.

($\Leftarrow$)

- Suppose that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathtt{cl}(\mathcal{A}))$.
- We prove that, for all $\sigma \in \Sigma^\omega$, it holds:

$$\sigma \notin \mathcal{L}(\mathcal{A}) \leftrightarrow \exists i \geq 0 \,.\, \forall \sigma' \in \Sigma^\omega \,.\, \sigma_{[0,i]} \cdot \sigma' \notin \mathcal{L}(\mathcal{A})$$

Theorem

For any rNBA $\mathcal{A}$, *it holds that* $\mathcal{L}(\mathcal{A})$ *is a safety property iff* $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathtt{cl}(\mathcal{A}))$.

Proof.

$(\Leftarrow)$

- The right-to-left direction

$$\forall \sigma \in \Sigma^\omega \ . \qquad \big(\sigma \notin \mathcal{L}(\mathcal{A}) \leftarrow \exists i \geq 0 \ . \ \forall \sigma' \in \Sigma^\omega \ . \ \sigma_{[0,i]} \cdot \sigma' \notin \mathcal{L}(\mathcal{A})\big)$$

holds for every language: it suffices to take $\sigma' := \sigma_{[i+1,\infty)}$.

Theorem

For any rNBA $\mathcal{A}$, *it holds that* $\mathcal{L}(\mathcal{A})$ *is a safety property iff* $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathtt{cl}(\mathcal{A}))$.

Proof.

$(\Leftarrow)$

- We prove the left-to-right direction:

$$\forall \sigma \in \Sigma^\omega \ . \qquad \big(\sigma \notin \mathcal{L}(\mathcal{A}) \rightarrow \exists i \geq 0 \ . \ \forall \sigma' \in \Sigma^\omega \ . \ \sigma_{[0,i]} \cdot \sigma' \notin \mathcal{L}(\mathcal{A})\big)$$

- Since by hypothesis $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathtt{cl}(\mathcal{A}))$, it is equivalent to prove:

$$\forall \sigma \in \Sigma^\omega \ . \qquad \big(\sigma \notin \mathcal{L}(\mathtt{cl}(\mathcal{A})) \rightarrow \exists i \geq 0 \ . \ \forall \sigma' \in \Sigma^\omega \ . \ \sigma_{[0,i]} \cdot \sigma' \notin \mathcal{L}(\mathtt{cl}(\mathcal{A}))\big)$$

Theorem

For any rNBA $\mathcal{A}$, *it holds that* $\mathcal{L}(\mathcal{A})$ *is a safety property iff* $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathtt{cl}(\mathcal{A}))$.

Proof.

$(\Leftarrow)$

- $\forall \sigma \in \Sigma^\omega \;.\; (\sigma \notin \mathcal{L}(\mathtt{cl}(\mathcal{A})) \rightarrow \exists i \geq 0 \;.\; \forall \sigma' \in \Sigma^\omega \;.\; \sigma_{[0,i]} \cdot \sigma' \notin \mathcal{L}(\mathtt{cl}(\mathcal{A})))$
- Suppose $\sigma \notin \mathcal{L}(\mathtt{cl}(\mathcal{A}))$. Thus the automaton $\mathtt{cl}(\mathcal{A})$ *rejects* σ.
- Since by hypothesis $\mathtt{cl}(\mathcal{A})$ is a *reduced* Büchi automatom, $\mathtt{cl}(\mathcal{A})$ can reject σ only by attempting an *undefined* transition.

Theorem

For any rNBA $\mathcal{A}$, *it holds that* $\mathcal{L}(\mathcal{A})$ *is a safety property iff* $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathtt{cl}(\mathcal{A}))$.

Proof.

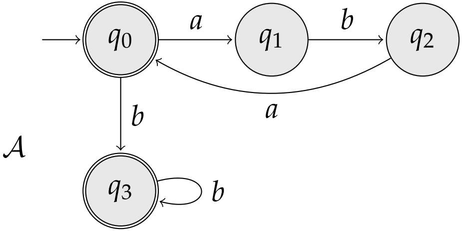
$(\Leftarrow)$

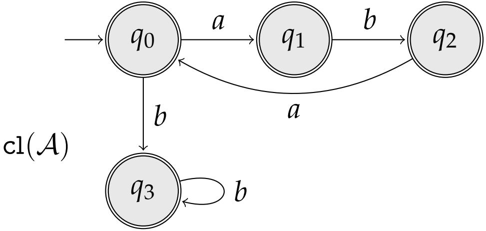
- $\forall \sigma \in \Sigma^\omega \; . \qquad (\sigma \notin \mathcal{L}(\mathtt{cl}(\mathcal{A})) \rightarrow \exists i \geq 0 \; . \; \forall \sigma' \in \Sigma^\omega \; . \; \sigma_{[0,i]} \cdot \sigma' \notin \mathcal{L}(\mathtt{cl}(\mathcal{A})))$
- Let i be the position of σ after which $\mathtt{cl}(\mathcal{A})$ takes the undefined transition.
- Clearly, it holds that:

$$\forall \sigma' \in \Sigma^\omega \; . \; \sigma_{[0,i]} \cdot \sigma' \notin \mathcal{L}(\mathtt{cl}(\mathcal{A}))$$

- Thus $\mathtt{cl}(\mathcal{A})$ (and $\mathcal{A}$ as well) specify a safety property. □

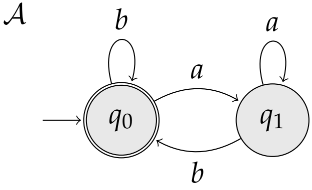
$\Sigma = \{a, b\}$, $\mathcal{L} = (a \cdot b \cdot a)^\omega \cup (a \cdot b \cdot a)^* \cdot b^\omega$

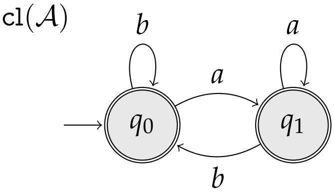




The language $\mathcal{L}$ is *safety* because $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathtt{cl}(\mathcal{A}))$.

$\Sigma = \{a, b\}$, $\mathcal{L} = \{\sigma \in \Sigma^\omega \mid$ each 'a' is eventually followed by 'b'$\}$





The language $\mathcal{L}$ is <u>*not*</u> safety because $\mathcal{L}(\mathcal{A}) \neq \mathcal{L}(\mathtt{cl}(\mathcal{A}))$.

- $a^\omega \in \mathcal{L}(\mathtt{cl}(\mathcal{A}))$ but $a^\omega \notin \mathcal{L}(\mathcal{A})$

Complexity of the procedure

Checking whether $\mathcal{L}(\mathtt{cl}(\mathcal{A})) = \mathcal{L}(\mathcal{A})$ is done by checking whether:

$$\mathcal{L}(\mathtt{cl}(\mathcal{A})) \subseteq \mathcal{L}(\mathcal{A}) \ \wedge \ \mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathtt{cl}(\mathcal{A}))$$

which in turn is equivalent to check whether:

$$\mathcal{L}(\mathtt{cl}(\mathcal{A})) \cap \overline{\mathcal{L}(\mathcal{A})} = \varnothing \ \wedge \ \mathcal{L}(\mathcal{A}) \cap \overline{\mathcal{L}(\mathtt{cl}(\mathcal{A}))} = \varnothing$$

- Complementation of NBA is needed.
- Complexity of Büchi complementation (n = number of states):
  - upper bound: $\mathcal{O}(0.96n)^n$
  - lower bound: $\Omega(0.76n)^n$
- Sven Schewe (2009). "Büchi Complementation Made Tight". In: *26th International Symposium on Theoretical Aspects of Computer Science, STACS 2009, February 26-28, 2009, Freiburg, Germany, Proceedings*. Ed. by Susanne Albers and Jean-Yves Marion. Vol. 3. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany, pp. 661–672. DOI: 10.4230/LIPIcs.STACS.2009.1854. URL: https://doi.org/10.4230/LIPIcs.STACS.2009.1854

Complexity of the procedure

Checking whether $\mathcal{L}(\mathtt{cl}(\mathcal{A})) = \mathcal{L}(\mathcal{A})$ is done by checking whether:

$$\mathcal{L}(\mathtt{cl}(\mathcal{A})) \subseteq \mathcal{L}(\mathcal{A}) \ \wedge \ \mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathtt{cl}(\mathcal{A}))$$

which in turn is equivalent to check whether:

$$\mathcal{L}(\mathtt{cl}(\mathcal{A})) \cap \overline{\mathcal{L}(\mathcal{A})} = \varnothing \ \wedge \ \mathcal{L}(\mathcal{A}) \cap \overline{\mathcal{L}(\mathtt{cl}(\mathcal{A}))} = \varnothing$$

- The emptiness check can be performed *on-the-fly* during the construction of the automata.
- Total Complexity: polynomial space (PSPACE)

Complexity of the problem

Theorem

The set of NBA *recognizing safety properties is* PSPACE.

Open Question:

Is PSPACE-complete?

Complexity for the deterministic case

Theorem

Given a DBA $\mathcal{A}$ *with* n *states, checking whether* $\mathcal{L}(\mathcal{A})$ *is safety can be done in time polynomial in* n*.*

Proof.

$\mathcal{L}(\mathsf{cl}(\mathcal{A})) = \mathcal{L}(\mathcal{A})$ iff $\mathcal{L}(\mathsf{cl}(\mathcal{A})) \cap \overline{\mathcal{L}(\mathcal{A})} = \varnothing \;\wedge\; \mathcal{L}(\mathcal{A}) \cap \overline{\mathcal{L}(\mathsf{cl}(\mathcal{A}))} = \varnothing$.

- Complementation of DBA is straightforward: swap final states with nonfinal states.
- Intersection can be done in polynomial time in n.
- Emptiness can be checked in (nondeterministic) logarithmic space in n: Savitch's Theorem.

$\square$

Theorem (Sistla (1994))

The set of LTL *formulas* ϕ *such that* $\mathcal{L}(\phi)$ *is safety is* PSPACE-*complete.*

Or equivalently: Given a LTL formula ϕ, the problem of establishing whether $\mathcal{L}(\phi)$ is safety is PSPACE-complete.

Theorem

For any LTL *formula* ϕ *(with* $n = |\phi|$*) over the set of atomic propositions* $\mathcal{AP}$ *there exists a* NBA $\mathcal{A}_\phi$ *over the alphabet* $2^{\mathcal{AP}}$ *such that:*

- $\mathcal{L}(\phi) = \mathcal{L}(\mathcal{A}_\phi)$
- $|\mathcal{A}_\phi| \in 2^{\mathcal{O}(n)}$

Reference

Moshe Y Vardi and Pierre Wolper (1986). "An automata-theoretic approach to automatic program verification". In: *Proceedings of the First Symposium on Logic in Computer Science*. IEEE Computer Society, pp. 322–331

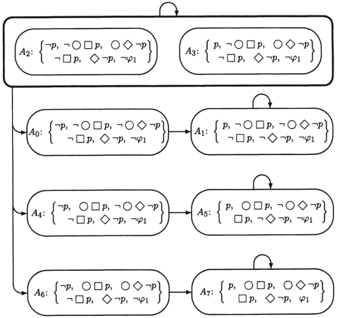
Reference

Moshe Y Vardi (1996). "An automata-theoretic approach to linear temporal logic". In: *Logics for concurrency*. Springer, pp. 238–266

Theorem

For any LTL *formula* ϕ *(with* $n = |\phi|$*) over the set of atomic propositions* $\mathcal{AP}$ *there exists a* NBA $\mathcal{A}_\phi$ *over the alphabet* $2^{\mathcal{AP}}$ *such that:*

- $\mathcal{L}(\phi) = \mathcal{L}(\mathcal{A}_\phi)$
- $|\mathcal{A}_\phi| \in 2^{\mathcal{O}(n)}$



Picture taken from

Zohar Manna and Amir Pnueli (1995). *Temporal verification of reactive systems - safety*. Springer. ISBN: 978-0-387-94459-3

Theorem (Sistla (1994))

The set of LTL *formulas* ϕ *such that* $\mathcal{L}(\phi)$ *is safety is* PSPACE-*complete.*

Or equivalently: Given a LTL formula ϕ, the problem of establishing whether $\mathcal{L}(\phi)$ is safety is PSPACE-complete.

Theorem (Sistla (1994))

The set of LTL *formulas* ϕ *such that* $\mathcal{L}(\phi)$ *is safety is* PSPACE*-complete.*

Proof.

- Let $\phi \in \mathsf{LTL}$.
- We can effectively build a NBA $\mathcal{A}_\phi$ such that $\mathcal{L}(\mathcal{A}_\phi) = \mathcal{L}(\phi)$ and $|\mathcal{A}_\phi| = 2^{\mathcal{O}(n)}$.
- In *space polynomial in n*, we can turn $\mathcal{A}_\phi$ into an equivalent rNBA $\mathcal{A}'_\phi$.
- Let $\mathtt{cl}(\mathcal{A}'_\phi)$ be its *closure*.
- $\mathcal{L}(\phi)$ is safety iff:
  - $\mathcal{L}(\mathcal{A}'_\phi) \subseteq \mathcal{L}(\mathtt{cl}(\mathcal{A}'_\phi))$ and
  - $\mathcal{L}(\mathtt{cl}(\mathcal{A}'_\phi)) \subseteq \mathcal{L}(\mathcal{A}'_\phi)$

Since the 1st point is always true, it suffices to prove the second.

Theorem (Sistla (1994))

The set of LTL *formulas* ϕ *such that* $\mathcal{L}(\phi)$ *is safety is* PSPACE*-complete.*

Proof.

- $\mathcal{L}(\mathtt{cl}(\mathcal{A}'_\phi)) \subseteq \mathcal{L}(\mathcal{A}'_\phi)$ is equivalent to $\mathcal{L}(\mathtt{cl}(\mathcal{A}'_\phi)) \cap \overline{\mathcal{L}(\mathcal{A}'_\phi)}$
- ... but instead of complementing $\mathcal{A}'_\phi$ (which is difficult) we complement the formula ϕ (which has a trivial, constant complexity)
- We can effectively build a NBA $\mathcal{A}_{\neg\phi}$ such that $\mathcal{L}(\mathcal{A}_{\neg\phi}) = \mathcal{L}(\neg\phi)$ and $|\,\mathcal{A}_{\neg\phi}\,| = 2^{\mathcal{O}(n)}$.
- We have that $\mathcal{L}(\mathcal{A}_{\neg\phi}) = \overline{\mathcal{L}(\mathcal{A}'_\phi)}$.

Theorem (Sistla (1994))

The set of LTL *formulas* ϕ *such that* $\mathcal{L}(\phi)$ *is safety is* PSPACE*-complete.*

Proof.

- $\mathcal{L}(\phi)$ is safety iff $\mathcal{L}(\mathtt{cl}(\mathcal{A}'_\phi)) \cap \mathcal{L}(\mathcal{A}_{\neg\phi}) = \varnothing$.
- Check *emptiness* of $\mathtt{cl}(\mathcal{A}'_\phi) \times \mathcal{A}_{\neg\phi}$:
  - $\mathtt{cl}(\mathcal{A}'_\phi) \times \mathcal{A}_{\neg\phi}$ is of size $2^{\mathcal{O}(n)}$
  - Emptiness: nondeterministic *logarithmic* space in the number of states of the automaton.
  - It can be performed *on-the-fly* during the construction of $\mathtt{cl}(\mathcal{A}'_\phi) \times \mathcal{A}_{\neg\phi}$.
  - Total Complexity: Polynomial Space (PSPACE)

Theorem (Sistla (1994))

The set of LTL *formulas* ϕ *such that* $\mathcal{L}(\phi)$ *is safety is* PSPACE*-complete.*

Proof.

- We prove that the problem is PSPACE-hard.
- Reduction from the LTL validity problem, which is PSPACE-complete.
- Let $\phi \in$ LTL over the atomic propositions $\mathcal{AP}$ and let $p \notin \mathcal{AP}$ a *fresh* proposition.
- It holds that: ϕ is *valid* iff $\mathcal{L}(\phi \vee \mathsf{F}p)$ is *safety*.

Theorem (Sistla (1994))

The set of LTL *formulas* ϕ *such that* $\mathcal{L}(\phi)$ *is safety is* PSPACE*-complete.*

Proof.

- We prove: if ϕ is *valid* then $\mathcal{L}(\phi \vee \mathsf{F}p)$ is *safety*.
- Suppose that ϕ is valid.
- Then $\phi \vee \mathsf{F}p$ is equivalent to $\top$, that is $\mathcal{L}(\phi \vee \mathsf{F}p) = (2^{\mathcal{AP}})^\omega$.
- Clearly, $(2^{\mathcal{AP}})^\omega$ is a *safety* language, because every violation (there are none) is irremediable.

Theorem (Sistla (1994))

The set of LTL *formulas* ϕ *such that* $\mathcal{L}(\phi)$ *is safety is* PSPACE*-complete.*

Proof.

- We prove: if $\mathcal{L}(\phi \vee \mathsf{F}p)$ is *safety* then ϕ is *valid*.
- Suppose there exists a violation of $\mathcal{L}(\phi \vee \mathsf{F}p)$, that is a trace $\sigma \in (2^{\mathcal{AP} \cup \{p\}})^\omega$ such that $\sigma \models \neg\phi \wedge \mathsf{G}\neg p$.
- Since by hypothesis $\mathcal{L}(\phi \vee \mathsf{F}p)$ is *safety*, this violation must be *irremediable*, that is $\exists i \geq 0 \,.\, \forall \sigma' \,.\, \sigma_{[0,i]} \cdot \sigma' \models \neg\phi \wedge \mathsf{G}\neg p$.
- Because $\sigma_{[0,i]} \cdot \sigma'$ has also to satisfy $\mathsf{G}\neg p$, there exists no such i.
- This means that there are no violations of $\phi \vee \mathsf{F}p$ (this formula is valid).
- Since p doesn't occur in ϕ, this means that ϕ is valid.

DETECTING BAD PREFIXES

Algorithms & Complexity

For problems like *model checking* and *reactive synthesis*, given a safety property:

- one doesn't want to build a NBA
- but rather to reason on finite words and to build a DFA.

In particular, we consider the automaton over finite words for the set of bad prefixes.

Reasoning over finite words is simpler than reasoning over infinite words.

Task:

Given a NBA $\mathcal{A}$, to give an algorithm for building the automaton recognizing exactly the set of bad prefixes of $\mathcal{L}(\mathcal{A})$ and to analyze its complexity.

For problems like *model checking* and *reactive synthesis*, given a safety property:

- one doesn't want to build a NBA
- but rather to reason on finite words and to build a DFA.

In particular, we consider the automaton over finite words for the set of bad prefixes.

Reasoning over finite words is simpler than reasoning over infinite words.

Reference:

Orna Kupferman and Moshe Y Vardi (2001). "Model checking of safety properties". In: *Formal Methods in System Design* 19.3, pp. 291–314. DOI: 10.1023/A:1011254632723

Definition (Safety Property)

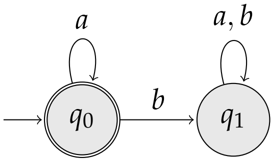
$\mathcal{L} \subseteq \Sigma^\omega$ is a *safety property* iff, for all $\sigma \notin \mathcal{L}$, there exists an position $i \in \mathbb{N}$ such that $\sigma_{[0,i]} \cdot \sigma' \notin \mathcal{L}$, for all $\sigma' \in \Sigma^\omega$.

- $\sigma_{[0,i]}$ is called the *bad prefix* of σ.
- We denote with $\mathtt{bad}(\mathcal{L})$ the set of bad prefixes of $\mathcal{L}$.
- $\mathtt{bad}(\mathcal{L})$ is a language of finite words, that is $\mathtt{bad}(\mathcal{L}) \subseteq \Sigma^*$.

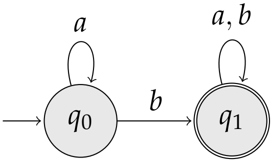
The Deterministic Case

If $\mathcal{A}$ is a DBA (Deterministic Büchi Automaton), then building the automaton for $\mathtt{bad}(\mathcal{L}(\mathcal{A}))$ is straightforward

- nondeterministic polynomial space and linear time.



$\mathcal{A}$

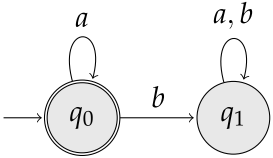


$\mathcal{A}_{bad}$

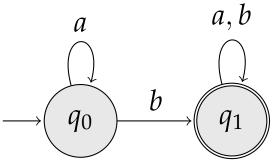
The Deterministic Case

If $\mathcal{A}$ is a DBA (Deterministic Büchi Automaton), then building the automaton for $\texttt{bad}(\mathcal{L}(\mathcal{A}))$ is straightforward

- Given a set of states S of $\mathcal{A}$, we denote with $\mathcal{A}^S$ the automaton obtained from $\mathcal{A}$ by defining the set of initial states to be S.
- Let $\mathcal{A}_{bad}$ be the DFA obtained from $\mathcal{A}$ by defining a state q to be *final* iff $\mathcal{A}^{\{q\}}$ recognizes the empty set.
- It holds that $\mathcal{L}(\mathcal{A}_{bad}) = \texttt{bad}(\mathcal{L}(\mathcal{A}))$.



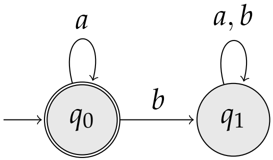
$\mathcal{A}$



$\mathcal{A}_{bad}$

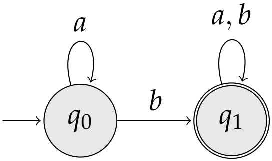
The Deterministic Case

- $\mathcal{L}(\mathcal{A}) = a^\omega$



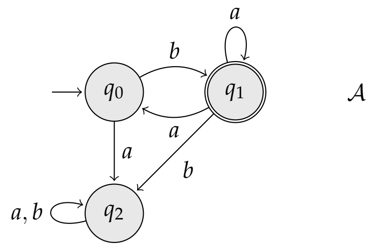
$\mathcal{A}$

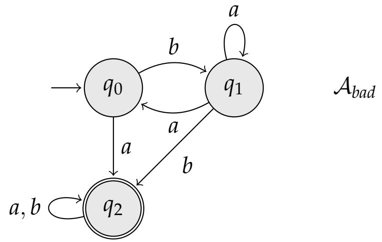
- $\texttt{bad}(\mathcal{L}(\mathcal{A})) = a^* \cdot b \cdot \Sigma^*$



$\mathcal{A}_{bad}$

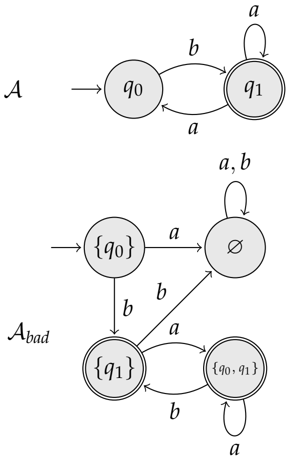
- The nondeterministic case is more involved.
- The previous algorithm for the deterministic case does not work in the nondeterministic case.
- Counterexample:
  - $\mathcal{L}(\mathcal{A}) = b \cdot a^\omega \cup (b \cdot a \cdot a^*)^\omega \cup (b \cdot a \cdot a^*)^* \cdot a^\omega$
  - The automaton $\mathcal{A}_{bad}$ recognizes also the word "*bab*" which is not a bad prefix.
- We need another way to build $\mathcal{A}_{bad}$.





- Let $\mathcal{A} = \langle Q, \Sigma, I, \Delta, F \rangle$ be NBA.
- We define $\mathcal{A}_{bad}$ as the DFA $\langle 2^Q, \Sigma, q_0', \delta', F' \rangle$ such that:
  - $q_0' := I$
  - for every $S \in 2^Q$ and every $\sigma \in \Sigma$, $\delta(S, \sigma) := \bigcup_{q \in S} \delta(q, \sigma)$.
  - $F := \{S \in 2^Q \mid \mathcal{L}(\mathcal{A}^S) = \varnothing\}$.
- Complexity: $|\mathcal{A}_{bad}| \in 2^{\mathcal{O}(n)}$ where $n = |Q|$.

The detection of bad prefixes with a nondeterministic Büchi automaton has the flavor of determinization.



- Let $\mathcal{A} = \langle Q, \Sigma, I, \Delta, F \rangle$ be NBA.
- We define $\mathcal{A}_{bad}$ as the DFA $\langle 2^Q, \Sigma, q'_0, \delta', F' \rangle$ such that:
  - $q'_0 := I$
  - for every $S \in 2^Q$ and every $\sigma \in \Sigma$, $\delta(S, \sigma) := \bigcup_{q \in S} \delta(q, \sigma)$.
  - $F := \{S \in 2^Q \mid \mathcal{L}(\mathcal{A}^S) = \varnothing\}$.
- Complexity: $|\mathcal{A}_{bad}| \in 2^{\mathcal{O}(n)}$ where $n = |Q|$.

The detection of bad prefixes with a nondeterministic Büchi automaton has the flavor of determinization.

This is a *lowerbound*.

- There exists an NFA $\mathcal{A}$ with n states such that
  - all states are accepting
  - its complement $\overline{\mathcal{A}}$ has $2^{\Theta(n)}$ states.
- Let $\mathcal{A}'$ be the NBA obtained by considering $\mathcal{A}$ as a Büchi automaton.
- Since both $\mathcal{A}$ and $\mathcal{A}'$ can reject a word only by attempting an undefined transition, it holds that $\mathsf{bad}(\mathcal{A}') = \overline{\mathcal{A}}$.
- It follows that the automaton for $\mathsf{bad}(\mathcal{A})$ has $2^{\Theta(n)}$ states.

An analogous result holds for the cosafety case.

Theorem

Given a NBA $\mathcal{A}$ *with* n *states such that* $\mathcal{L}(\mathcal{A})$ *is cosafety, the size of an automaton for* $\texttt{good}(\mathcal{A})$ *is* $2^{\Theta(n)}$.

Detecting bad prefixing of an LTL formula recognizing a safety language is doubly exponential.

Theorem

Given an LTL *formula* ϕ *such that* $\mathcal{L}(\phi)$ *is safety and* $|\phi| = n$, *the size of an automaton for* $\texttt{bad}(\mathcal{L}(\phi))$ *is* $2^{2^{\mathcal{O}(n)}}$ *and* $2^{2^{\Omega(\sqrt{n})}}$.

An analogous result holds for the cosafety case.

Theorem

Given a NBA $\mathcal{A}$ *with* n *states such that* $\mathcal{L}(\mathcal{A})$ *is cosafety, the size of an automaton for* $\mathtt{good}(\mathcal{A})$ *is* $2^{\Theta(n)}$.

Reference:

Orna Kupferman and Moshe Y Vardi (2001). "Model checking of safety properties". In: *Formal Methods in System Design* 19.3, pp. 291–314. DOI: 10.1023/A:1011254632723

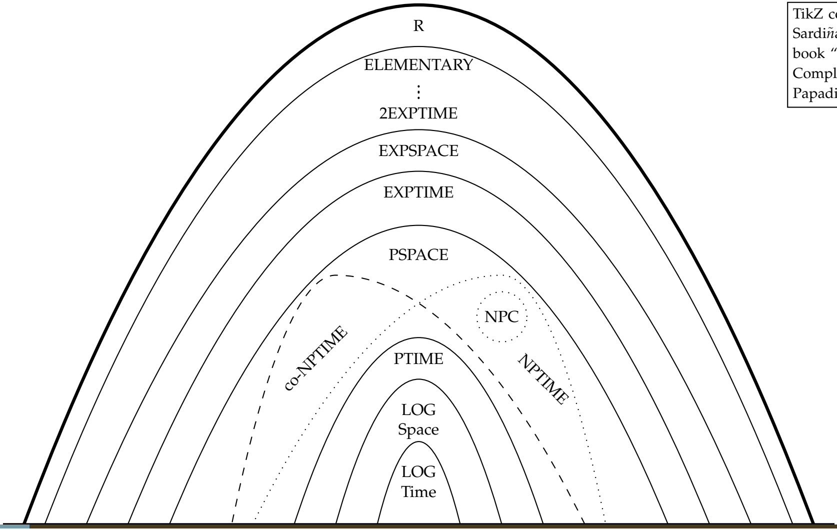
ALGORITHMS & COMPLEXITY

for the safety fragment of LTL

Outline

- Efficient algorithms and theoretical complexity for the problems of:
  - satisfiability
  - model checking
    - symbolic algorithms
    - exploiting the Kupferman & Vardi's classification (informative prefixes)
  - reactive synthesis

Recap of complexity classes



TikZ code by Sebastian Sardiña, based on the book “Computational Complexity” by C. H. Papadimitriou

SATISFIABILITY

of (co)safety fragments of LTL

The satisfiability problem

Let $\mathbb{L}$ be a temporal logic over infinite sequences.

Definition

Given a formula ϕ of $\mathbb{L}$, we say that ϕ is satisfiable iff $\mathcal{L}(\phi) \neq \varnothing$.

The satisfiability problem of $\mathbb{L}$ is the problem of checking whether a given input formula ϕ is satisfiable.

The satisfiability problem of LTL (LTL-SAT) is PSPACE-complete.

- same for LTL+P

Reference:

A Prasad Sistla and Edmund M Clarke (1985). "The complexity of propositional linear temporal logics". In: *Journal of the ACM (JACM)* 32.3, pp. 733–749. DOI: 10.1145/3828.3837

Classic Algorithm

Given $\phi \in$ LTL+P of size n,

- build an NBA $\mathcal{A}$ such that:
  - $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\phi)$
  - $|\mathcal{A}| \in 2^{\mathcal{O}(n)}$
- check the (non)emptiness of $\mathcal{A}$, *i.e.*, the existence of a state q such that:
  - $q_0 \rightsquigarrow q$
  - $q \rightsquigarrow q$

The satisfiability problem of LTL (LTL-SAT) is PSPACE-complete.

- same for LTL+P

Reference:

A Prasad Sistla and Edmund M Clarke (1985). "The complexity of propositional linear temporal logics". In: *Journal of the ACM (JACM)* 32.3, pp. 733–749. DOI: 10.1145/3828.3837

Classic Algorithm

Complexity:

- nonemptiness = reachability, thus nondeterministic logarithmic space
- can be done on-the-fly while building the NBA
- Total: nondeterministic polynomial space (PSPACE)

The satisfiability problem of $\mathsf{LTL_f}$ ($\mathsf{LTL_f}$-SAT) is PSPACE-complete.

Reference:

Giuseppe De Giacomo and Moshe Y. Vardi (2013). “Linear Temporal Logic and Linear Dynamic Logic on Finite Traces”. In: *Proceedings of the 23rd International Joint Conference on Artificial Intelligence*. Ed. by Francesca Rossi. IJCAI/AAAI, pp. 854–860

Classic Algorithm

Complexity:

- nonemptiness = reachability, thus nondeterministic logarithmic space
- can be done on-the-fly while building the NBA
- Total: nondeterministic polynomial space (PSPACE)

Theorem

The satisfiability problem for the logics SafetyLTL, G(pLTL), coSafetyLTL, F(pLTL), *and* LTL[X, G] *is* PSPACE-*complete.*

Reference:

Alessandro Artale, Luca Geatti, et al. (2023b). "Complexity of Safety and coSafety Fragments of Linear Temporal Logic". In: *Proc. of the 36th AAAI Conf. on Artificial Intelligence*. AAAI Press

Theorem

The satisfiability problem for the logics SafetyLTL, G(pLTL), coSafetyLTL, F(pLTL), *and* LTL[X, G] *is* PSPACE-*complete.*

Proof.

- membership: from LTL-SAT
- hardness: reduction from $\mathsf{LTL_f}$-SAT

□

Theorem

The satisfiability problem for the logics SafetyLTL, G(pLTL), coSafetyLTL, F(pLTL), *and* LTL[X, G] *is* PSPACE*-complete.*

The restriction to (co)safety fragments, i.e., *the restriction on reasoning over finite traces, does not change the worst-case complexity of the satisfiability problem.*

Theorem

The satisfiability problem for $\mathsf{LTL}[\mathsf{X}, \mathsf{F}]$ *is* NP*-complete.*

Lemma (Small model property)

For any $\phi \in \mathsf{LTL}[\mathsf{X}, \mathsf{F}]$*, it holds that* ϕ *is satisfiable iff there exists a trace* σ *such that:*

- $\sigma \models \phi$
- $|\sigma| \leq |\phi|$

Theorem

The satisfiability problem for LTL[X, F] *is* NP*-complete.*

Proof.

- membership: nondeterministic algorithm
  - guess an $n \leq |\phi|$ and the assignments for the first n states of a candidate trace σ
  - check whether $\sigma \cdot (2^{\mathcal{AP}})^{\omega} \models \phi$
  - if at least one candidate model is indeed a correct model, terminate with SAT; otherwise terminate with UNSAT.
- hardness: from Boolean satisfiability

□

Theorem

The satisfiability problem for $\mathsf{LTL}[\mathsf{X}, \mathsf{F}]$ *is* NP*-complete.*

Reference:

A Prasad Sistla and Edmund M Clarke (1985). “The complexity of propositional linear temporal logics”. In: *Journal of the ACM (JACM)* 32.3, pp. 733–749. DOI: 10.1145/3828.3837

Logics	Problems		
	satisfiability	model checking	realizability
coSafetyLTL	PSPACE-c	???	2EXPTIME-c
F(pLTL)	PSPACE-c	???	EXPTIME-c
LTL[X, F]	NP-c	???	EXPTIME-c

Logics	Problems		
	satisfiability	model checking	realizability
SafetyLTL	PSPACE-c	???	2EXPTIME-c
G(pLTL)	PSPACE-c	???	EXPTIME-c
LTL[$\widetilde{\mathsf{X}}$, G]	PSPACE-c	???	EXPTIME-c

MODEL CHECKING

for safety fragments of LTL

- Automatic formal verification techniques: great progress in the last decades.
- Big chip or software companies have integrated them in their development or quality assurance process.
- Intel: FDIV bug, error in the floating point division instruction on some Intel®Pentium® processors.
  - it costed ≈ US $475 million;
  - big investment in formal verification.

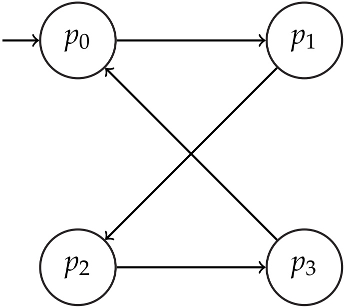
The most used formal verification technique is Model Checking (MC, for short).

- the system to verify is modeled as a finite-state machine (*i.e.*, Kripke structure) and the specification is expressed by means of a temporal logic formula;
- distinctive features:
  - fully automatic;
  - exhaustive;
  - it generates a counterexample trace if the specification does not hold.

Definition (Kripke structure)

A Kripke structure is a tuple $M = \langle \mathcal{AP}, Q, I, T, L \rangle$ where:

- $\mathcal{AP}$ is a finite alphabet,
- Q is the finite set of states,
- $I \subseteq Q$ is the set of initial states,
- $T \subseteq Q \times Q$ is a *complete* transition relation, and
- $L : Q \to 2^{\mathcal{AP}}$ is the labeling function that assigns to each state the set of atoms in $\mathcal{AP}$ that are true in that state.



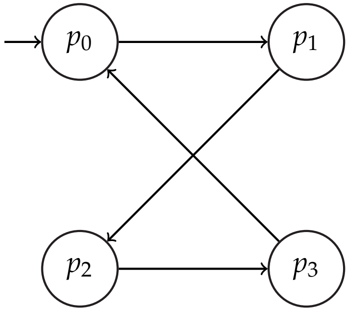
Definition (Model Checking of LTL)

Given:

- a Kripke structure $M = \langle \mathcal{AP}, Q, I, T, L \rangle$
- an initial state $s \in I$ of M
- an LTL formula ϕ over the set of atomic propositions $\mathcal{AP}$

we write $M, s \models \mathsf{A}\phi$ iff *all* paths of M starting from s are models of ϕ.

A is the *"for all paths"* operator of CTL.

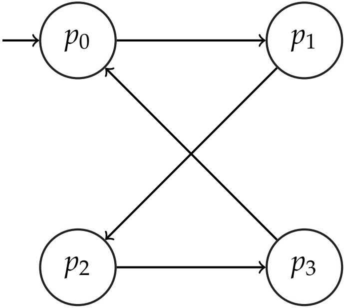


Definition (Model Checking of LTL)

The model checking problem of LTL (LTL-MC) is the problem of establishing whether $M, s \models \mathsf{A}\phi$.

Example:

- $M, s \models \mathsf{GF}(p_0)$
- $M, s \not\models \mathsf{FG}(p_0)$

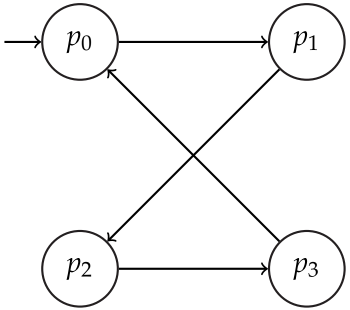


Theorem

The LTL-*MC is* PSPACE-*complete.*

Reference:

A Prasad Sistla and Edmund M Clarke (1985). "The complexity of propositional linear temporal logics". In: *Journal of the ACM (JACM)* 32.3, pp. 733–749. DOI: 10.1145/3828.3837



Classical approach

In order to decide if $M, s \models \phi$:

1. Build the Büchi automaton $\mathcal{A}_M$ that accepts all and only the words corresponding to computations of M;
2. Build the Büchi automaton $\mathcal{A}_{\neg\phi}$ that accepts all and only the words corresponding to models of $\neg\phi$;
3. Check the *(non)emptiness* of the product automaton $\mathcal{A}_M \times \mathcal{A}_{\neg\phi}$.
   - $L(\mathcal{A}_M \times \mathcal{A}_{\neg\phi}) \neq \varnothing \Leftrightarrow M, s \not\models \phi$
   - MC=universal problem
   - EMPTINESS= existential problem

Classical approach

- Emptiness of $\mathcal{A}_M \times \mathcal{A}_{\neg\phi}$:
  - $\exists q \,.\, q_0 \rightsquigarrow q \wedge q \rightsquigarrow q$
- Checking the existence of a *fair cycle* in M
- IMPORTANT: in practice, this is much more difficult than simply the reachability of a state q.

Invariance checking

- Invariance checking: it is defined as LTL model checking of a formula of the form $\mathsf{G}(\phi)$ where ϕ is a Boolean formula.

  Does ϕ hold in (at least) every reachable state of M?

- Given $M = \langle \mathcal{AP}, Q, I, T, L \rangle$ and a Boolean formula ϕ over the variables $\mathcal{AP}$

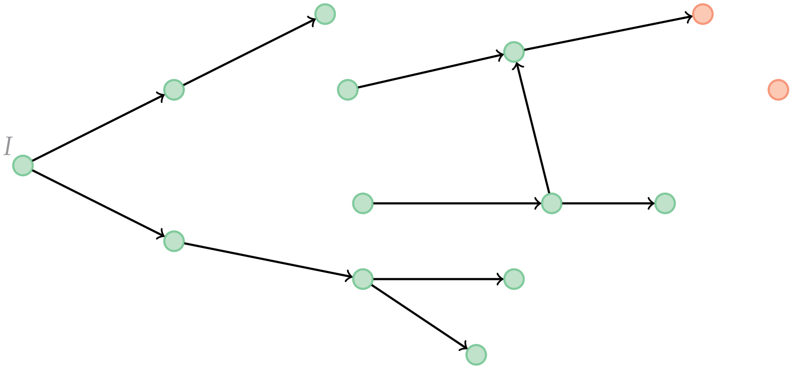
  find a state in which $\neg\phi$ holds or establish its nonexistence.

  - *it is a reachability problem*
  - if ϕ holds in every reachable state of M, then ϕ is invariant in M
  - otherwise, there is a *finite trace* as counterexample:

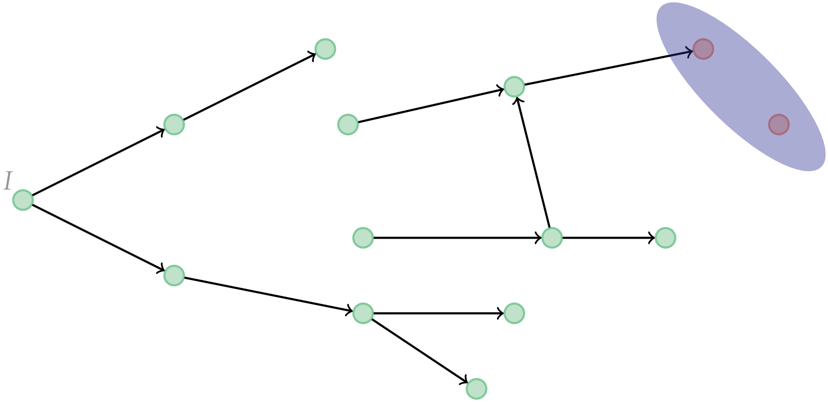
  $$\langle s_0, s_1, \ldots, s_n \rangle$$

  such that $s_i \models \phi$ for any $i < n$ and $s_n \not\models \phi$.

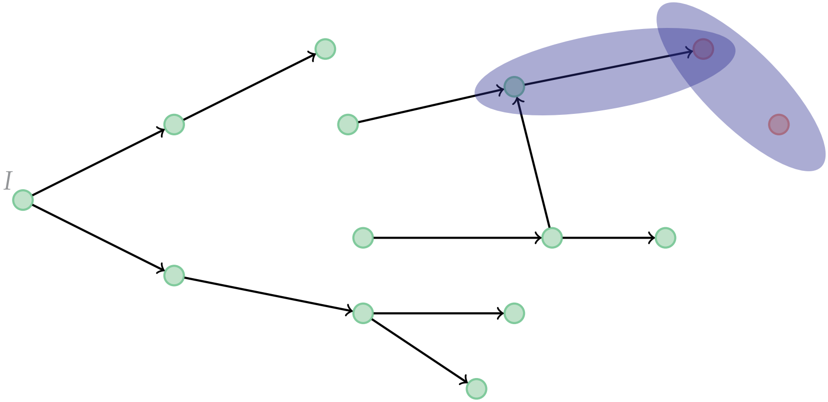
Standard Algorithm for Invariance Checking



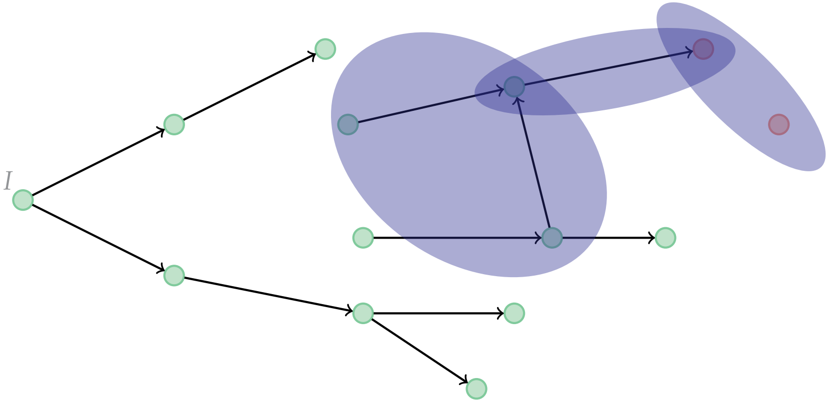
Standard Algorithm for Invariance Checking



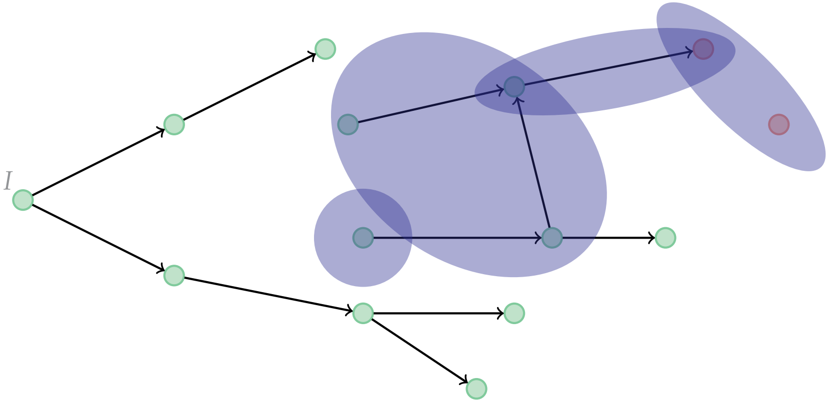
Standard Algorithm for Invariance Checking



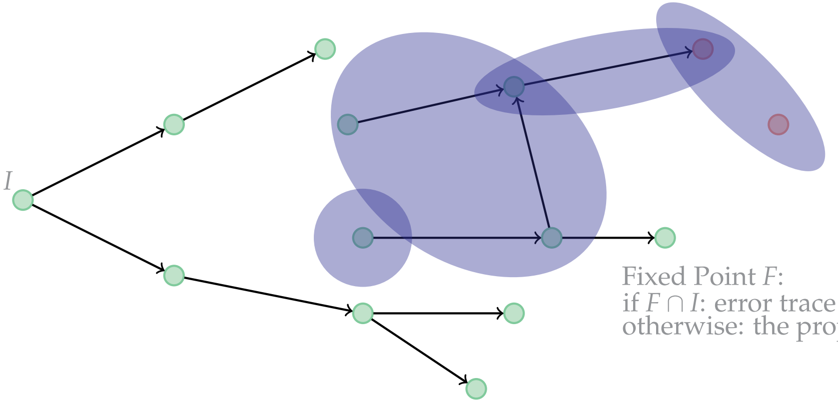
Standard Algorithm for Invariance Checking



Standard Algorithm for Invariance Checking

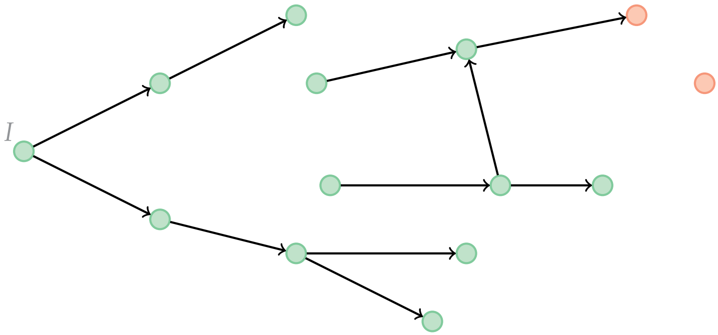


Standard Algorithm for Invariance Checking

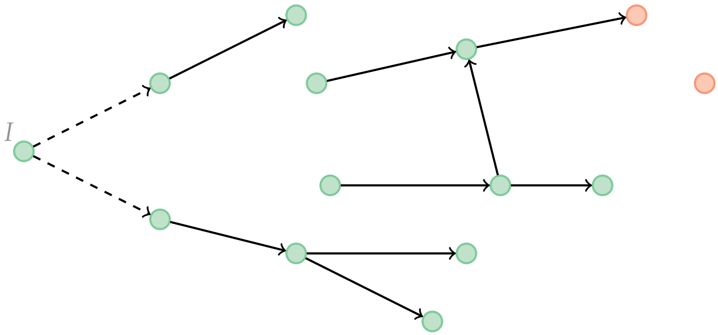


Fixed Point F:
if $F \cap I$: error trace
otherwise: the property is true in M

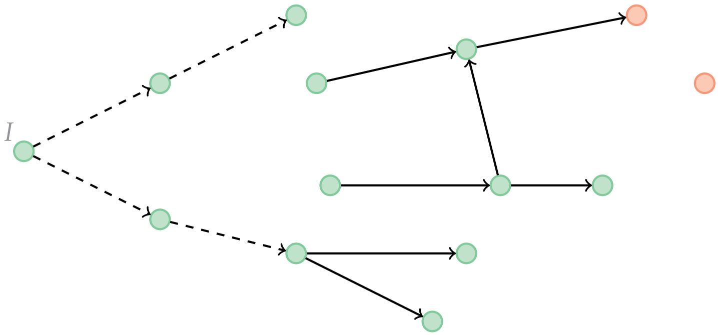
Standard Algorithm for Invariance Checking



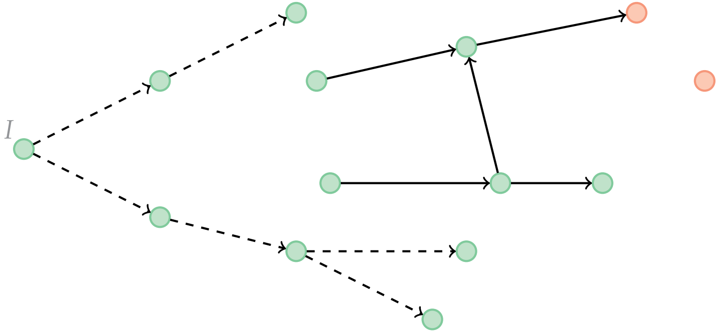
Standard Algorithm for Invariance Checking



Standard Algorithm for Invariance Checking



Standard Algorithm for Invariance Checking



Symbolic Model Checking

- The previous algorithms belongs to the class of explicit-state model checking algorithms:
  - the Kripke Structure M is represented as a set of memory locations, pointers ecc...
- MC suffers from the state-space explosion problem: the number of states of

$$M = M_1 \times M_2 \times \cdots \times M_n$$

  is exponential in n;
- the size of system that could be verified by explicit model checkers was restricted to $\approx 10^6$ states.
- Solution: Symbolic Model Checking

Symbolic Transition Systems

Consider a (explicit) Kripke structure $\mathcal{M} = (S, I, T, L)$.

- Symbolic Finite-state transition system $\mathcal{M} = (\bar{i}, \bar{x}, I, T)$
  - $\bar{i}$ is a set of input variables;
  - $\bar{x}$ is a set of state variables;
  - $I(\bar{x})$ is the formula for initial states;
  - $T(\bar{x}, \bar{i}, \bar{x}')$ is the formula for the transition relation;

Symbolic Transition Systems

Three main techniques have been proposed:

- BDD-based symbolic model checking
  - kind of *compressed truth tables*
- partial order reduction
- SAT-based symbolic model checking, aka *Bounded Model Checking*.

They allowed for the verification of systems with $> 10^{120}$ states.

- substantially larger than the number of atoms in the observable universe (around 10^{80})

Symbolic Invariance Checking

The problem of invariance checking is thoroughly studied in symbolic model checking.

- IC3 is arguably the state-of-the-art algorithm for symbolic invariance checking
- outstanding performance

Reference:

Aaron R Bradley (2011). “SAT-based model checking without unrolling”. In: *International Workshop on Verification, Model Checking, and Abstract Interpretation*. Springer, pp. 70–87

Classical Approach

Let M be Kripke structure, s an initial state of M, and ϕ be an LTL formula such that $\mathcal{L}(\phi)$ is *safety*.

- Objective: efficient algorithms for model checking of safety properties ($M, s \models \mathsf{A}\,\phi$)
  - exploiting the reduction from infinite to *finite* trace
  - exploiting efficient backends for *symbolic invariance checking*

Classical Approach

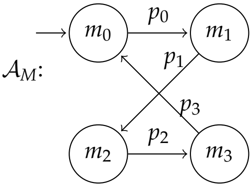
Let M be Kripke structure, s an initial state of M, and ϕ be an LTL formula such that $\mathcal{L}(\phi)$ is *safety*.

1. Build the *automaton over finite words* (DFA) $\mathcal{A}_{bad}$ for the bad prefixes of $\mathcal{L}(\phi)$.
2. Build the product $\mathcal{A}_M \times \mathcal{A}_{bad}$.
3. Check the reachability of a final state in $\mathcal{A}_M \times \mathcal{A}_{bad}$
   - or equivalently that the property *"the current state is not final"* is *invariant*

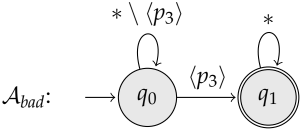
$$\mathsf{G}(\neg \mathit{final})$$

4. Output:
   - if found: there is a counterexample to ϕ
   - otherwise: ϕ holds in M

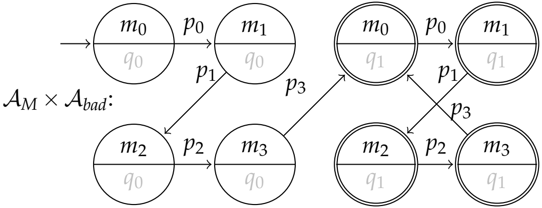
- Kripke Structure M:



- Automaton for the bad prefixes of $\mathsf{G}(p_0 \vee p_1 \vee p_2)$:



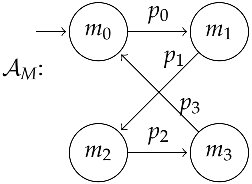
We denote with $\langle p_3 \rangle$ all the subsets of $\{p_0, p_1, p_2, p_3\}$ that contain the proposition p_3.



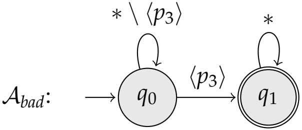
- We reduced the problem $M, s \models \mathsf{A}\,\mathsf{G}(p_0 \vee p_1 \vee p_2)$ to checking whether: (*reachability*)

$$\mathcal{A}_M \times \mathcal{A}_{bad} \models \mathsf{G}(\neg q_0)$$

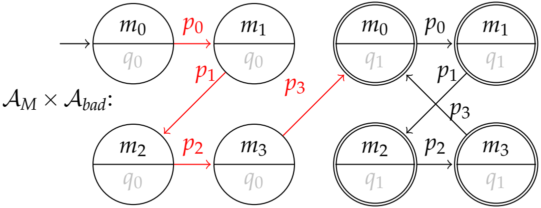
- Kripke Structure M:



- Automaton for the bad prefixes of $\mathsf{G}(p_0 \vee p_1 \vee p_2)$:



We denote with $\langle p_3 \rangle$ all the subsets of $\{p_0, p_1, p_2, p_3\}$ that contain the proposition p_3.



- We reduced the problem $M, s \models \mathsf{A}\,\mathsf{G}(p_0 \vee p_1 \vee p_2)$ to checking whether: (*reachability*)

$$\mathcal{A}_M \times \mathcal{A}_{bad} \models \mathsf{G}(\neg q_0)$$

- The property does not hold: counterexample trace

- Problem: the automaton for the bad prefixes is *doubly exponential* in the size of the formula, in the worst case:

$$|\phi| = n \;\rightarrow\; |\,\mathcal{A}_{bad}\,| \in 2^{2^{\mathcal{O}(n)}}$$

  This can become easily impractical.
- Solution: we *relax* the fact that the automaton has to recognize *all* bad prefixes.

Definition (Fine Automata)

Given a safety language $\mathcal{L}$, a DFA $\mathcal{A}$ is *fine for* $\mathcal{L}$ iff it accepts *at least one* bad prefix for each violation of $\mathcal{L}$, *i.e.*: $\forall \sigma \notin \mathcal{L} \,.\, \exists i \geq 0 \,.\, \sigma_{[0,i]} \in \mathcal{L}(\mathcal{A})$.

Theorem

For every LTL *formula* ϕ *such that* $\mathcal{L}(\phi)$ *is safety, there exists a* NFA $\mathcal{A}$ *that is fine for* $\mathcal{L}(\phi)$ *and* $|\mathcal{A}| \in 2^{\mathcal{O}(n)}$

Reference:

Orna Kupferman and Moshe Y Vardi (2001). "Model checking of safety properties". In: *Formal Methods in System Design* 19.3, pp. 291–314. DOI: 10.1023/A:1011254632723

Theorem

For every LTL *formula* ϕ *such that* $\mathcal{L}(\phi)$ *is safety, there exists a* NFA $\mathcal{A}$ *that is fine for* $\mathcal{L}(\phi)$ *and* $|\mathcal{A}| \in 2^{\mathcal{O}(n)}$

Pros:

- it is exponentially smaller than $\mathcal{A}_{bad}$
- it is built using *alternating automata*

Cons:

- we sacrify *minimality*
    - this may be good for model checking
    - less good for *monitoring*
- it is *nondeterministic* (differently from $\mathcal{A}_{bad}$):
    - ok for model checking
    - not ok for *reactive synthesis*

- (Symbolic) Invariance Checking: very efficient algorithms
- Some algorithms for LTL model checking leverage this efficiency:
  - LTL-MC $\rightsquigarrow$ invariance checking
- K-Liveness

Reference:

Koen Claessen and Niklas Sörensson (2012). “A liveness checking algorithm that counts”. In: *2012 Formal Methods in Computer-Aided Design (FMCAD)*. IEEE, pp. 52–59

Objectives:

1. Solve LTL-MC

$$M, s \models \mathsf{A}\,\phi$$

where ϕ is an LTL formula.

2. Reduction to a sequence of invariance checking problems.

Solution:

- To *count* and *bound* the number of times the product automaton $\mathcal{A}_M \times \mathcal{A}_{\neg\phi}$ visits a final state of $\mathcal{A}_{\neg\phi}$.

Objectives:

1. Solve LTL-MC

$$M, s \models \mathsf{A}\,\phi$$

where ϕ is an LTL formula.

2. Reduction to a sequence of invariance checking problems.

Solution:

- To *count* and *bound* the number of times the product automaton $\mathcal{A}_M \times \mathcal{A}_{\neg\phi}$ visits a final state of $\mathcal{A}_{\neg\phi}$.

Main idea:

- Let $\mathcal{A}_{\neg\phi}$ be a NBA for $\neg\phi$.
- $M, s \models \mathsf{A}\,\phi$ iff the language of $\mathcal{A}_M \times \mathcal{A}_{\neg\phi}$ is *empty*
- ... iff each computation of $\mathcal{A}_M \times \mathcal{A}_{\neg\phi}$ visits a final state of $\mathcal{A}_{\neg\phi}$ a *finite number of times*

This number is clearly *bounded* above by the number of states of $\mathcal{A}_M \times \mathcal{A}_{\neg\phi}$, *i.e.*, $|M| \cdot |\mathcal{A}_{\neg\phi}|$.

- K-Liveness proceeds *incrementally*, checking whether $\mathcal{A}_M \times \mathcal{A}_{\neg\phi}$ visits a final state K times for $K = 1, 2, 3, \ldots$
- Methodology: use a *counter*
  - K-counter $\mathcal{A}_K$ = automaton that stays in its state q_f iff the computation has visited *less than* K times a final state of $\mathcal{A}_{\neg\phi}$
- Each subproblem is of the form:

$$\mathcal{A}_M \times \mathcal{A}_{\neg\phi} \times \mathcal{A}_K, s \models \mathsf{A}\,\mathsf{G}(q_f)$$

It is an *invariance checking problem*.

Main idea:

- Let $\mathcal{A}_{\neg\phi}$ be a NBA for $\neg\phi$.
- $M, s \models \mathsf{A}\,\phi$ iff the language of $\mathcal{A}_M \times \mathcal{A}_{\neg\phi}$ is *empty*
- …iff each computation of $\mathcal{A}_M \times \mathcal{A}_{\neg\phi}$ visits a final state of $\mathcal{A}_{\neg\phi}$ a *finite number of times*

This number is clearly *bounded* above by the number of states of $\mathcal{A}_M \times \mathcal{A}_{\neg\phi}$, *i.e.*, $|M| \cdot |\mathcal{A}_{\neg\phi}|$.

- K-Liveness proceeds *incrementally*, checking whether $\mathcal{A}_M \times \mathcal{A}_{\neg\phi}$ visits a final state K times for $K = 1, 2, 3, \ldots$
- Methodology: use a *counter*
  - K-counter $\mathcal{A}_K$ = automaton that stays in its state q_f iff the computation has visited *less than* K times a final state of $\mathcal{A}_{\neg\phi}$
- Each subproblem is of the form:

$$\mathcal{A}_M \times \mathcal{A}_{\neg\phi} \times \mathcal{A}_K, s \models \mathsf{A}\,\mathsf{G}(q_f)$$

It is an *invariance checking problem*.

Termination:

- if $M, s \models \mathsf{A}\,\phi$, there exists a K for which $\mathcal{A}_M \times \mathcal{A}_{\neg\phi}$ visits final states at most K times.
- if $M, s \not\models \mathsf{A}\,\phi$, the algorithms increments K until the upper bound: it then stops.

Implementation:

- K-Liveness is implemented in the nuXmv model checker.

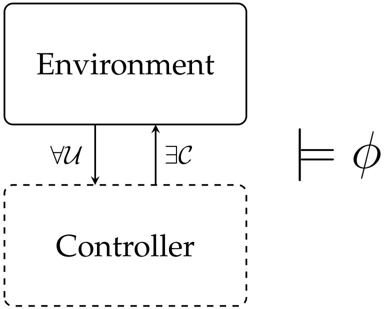
Roberto Cavada et al. (2014). "The nuXmv symbolic model checker". In: *International Conference on Computer Aided Verification (CAV)*. Springer, pp. 334–342. DOI: 10.1007/s10009-006-0001-2

Logics	Problems		
	satisfiability	model checking	realizability
coSafetyLTL	PSPACE-c	???	2EXPTIME-c
F(pLTL)	PSPACE-c	???	EXPTIME-c
LTL[X, F]	NP-c	???	EXPTIME-c

Logics	Problems		
	satisfiability	model checking	realizability
SafetyLTL	PSPACE-c	???	2EXPTIME-c
G(pLTL)	PSPACE-c	???	EXPTIME-c
LTL[$\widetilde{X}$, G]	PSPACE-c	???	EXPTIME-c

REACTIVE SYNTHESIS

from safety fragments of LTL



1. What are realizability and reactive synthesis?
   - model-based design: all the effort on the quality of the specification
   - culmination of declarative programming
2. Complexity:
   - for S1S: non-elementary
   - for LTL: 2EXPTIME-complete.

Definition (Strategy)

Let $\Sigma = \mathcal{C} \cup \mathcal{U}$ be an alphabet partitioned into the set of controllable variables $\mathcal{C}$ and the set of uncontrollable ones $\mathcal{U}$, such that $\mathcal{C} \cap \mathcal{U} = \varnothing$. A *strategy for Controller* is a function

$$g : (2^{\mathcal{U}})^{+} \to 2^{\mathcal{C}}$$

that, given the sequence $\mathsf{U} = \langle \mathsf{U}_0, \ldots, \mathsf{U}_n \rangle$ of choices made by *Environment* so far, determines the current choices $\mathsf{C}_n = g(\mathsf{U})$ of *Controller*.

Definition (Strategy)

Let $\Sigma = \mathcal{C} \cup \mathcal{U}$ be an alphabet partitioned into the set of controllable variables $\mathcal{C}$ and the set of uncontrollable ones $\mathcal{U}$, such that $\mathcal{C} \cap \mathcal{U} = \varnothing$. A *strategy for Controller* is a function

$$g : (2^{\mathcal{U}})^+ \to 2^{\mathcal{C}}$$

that, given the sequence $\mathsf{U} = \langle \mathsf{U}_0, \ldots, \mathsf{U}_n \rangle$ of choices made by *Environment* so far, determines the current choices $\mathsf{C}_n = g(\mathsf{U})$ of *Controller*.

Definition (Realizability and Synthesis)

Let ϕ be a temporal formula over the alphabet $\Sigma = \mathcal{C} \cup \mathcal{U}$. We say that ϕ is *realizable* if and only if

- $\exists g : (2^{\mathcal{U}})^+ \to 2^{\mathcal{C}}$
- $\forall \omega$-sequence $\mathsf{U} = \langle \mathsf{U}_0, \mathsf{U}_1, \ldots \rangle \in (2^{\mathcal{U}})^\omega$
- $\langle \mathsf{U}_0 \cup g(\langle \mathsf{U}_0 \rangle), \mathsf{U}_1 \cup g(\langle \mathsf{U}_0, \mathsf{U}_1 \rangle), \ldots \rangle \models \phi$

In this case, g is called *winning strategy*. If ϕ is realizable, the synthesis problem is the problem of computing such a strategy g.

Definition (Finitely representable strategies)

Let $g : (2^{\mathcal{U}})^{+} \rightarrow 2^{\mathcal{C}}$ be a strategy. We say that g is *finitely representable* iff there exists a Mealy machine M_g *"equivalent"* to g.

Proposition (Small model property of LTL)

Let ϕ be an LTL *formula and $n = |\phi|$. If ϕ is realizable, then there exists a finitely representable winning strategy g such that its corresponding Mealy machine has at most $2^{2^{c \cdot n}}$ states, for some constant c.*

Reference:

Amir Pnueli and Roni Rosner (1989). "On the Synthesis of a Reactive Module". In: *Proceedings of POPL'89*. ACM Press, pp. 179–190. DOI: 10.1145/75277.75293

- Realizability is modeled as a *two-players game* over an *arena/automaton* $\mathcal{A}_\phi$ built from ϕ:
  - Controller player: his objective is to enforce the satisfaction of the specification, no matter the choices of the other player (winning strategy)
  - Environment player: his objective is to enforce the violation of the specification, no matter the choices of the other player
- Environment player moves first.
- The game is played on *deterministic* automata obtained from the initial specification.
  - there are simple algorithms for synthesis over deterministic arenas

$\Rightarrow$ backward fixpoint computations

  - LTL formula $\phi \rightsquigarrow$ DRA $\mathcal{A}_\phi$

We consider first the case of *finite words*.

Standard Approach:

LTLf ϕ

↓

NFA $\mathcal{A}_\phi$

↓ · determinization

DFA $\mathcal{A}'_\phi$

↓

reachability game

- The DFA $\mathcal{A}'_\phi$ is *equivalent* to ϕ:

$$\mathcal{L}(\mathcal{A}'_\phi) = \mathcal{L}(\phi)$$

- Controller can force to the game to reach a *final* state of $\mathcal{A}'_\phi$ iff there is a winning strategy for the formula ϕ:
  - playing over the DFA $\mathcal{A}'_\phi$ is equivalent to solve the reactive synthesis problem for ϕ.

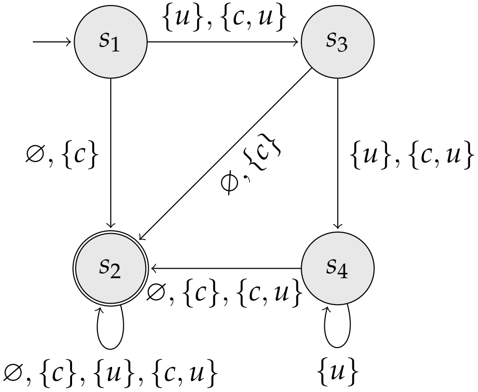
Definition (Strong Predecessor)

Let $\mathcal{A} = \langle Q, 2^{\mathcal{U}} \cup 2^{\mathcal{C}}, q_0, \delta, F\rangle$ be a DFA and let $S \subseteq Q$. We define the *strong precedessors* of S as follows:

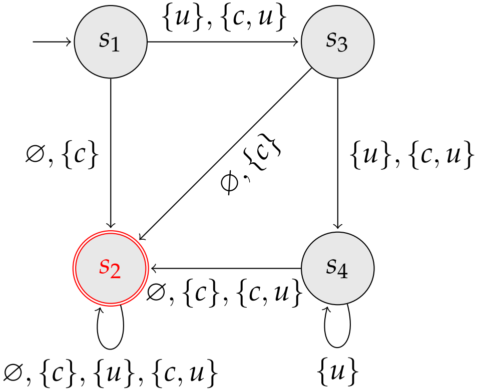
$$\mathtt{pre}(S) := \{s \in Q \mid \forall u \in 2^{\mathcal{U}} \,.\, \exists c \in 2^{\mathcal{C}} \,.\, s \xrightarrow{u,c} s', \text{ for some } s' \in S\}$$

$\mathtt{pre}(S)$ is the set of states of $\mathcal{A}$ from which Controller can force the game into a state of S in one step.

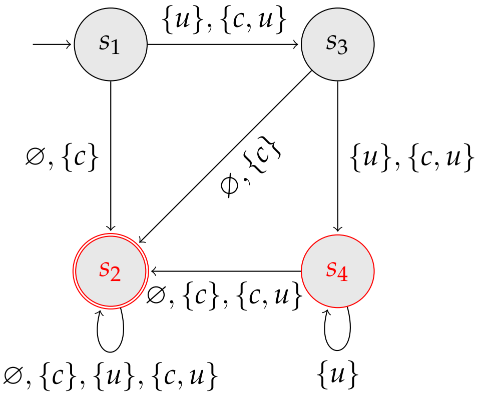
- The *winning region* is the set of states from which Controller can force the game to reach a final state.
  - $\Rightarrow$ reachability games
- Computation of the winning region (greatest fixed point):
  - $W_0 := F$
  - $W_{i+1} := W_i \cup \mathtt{pre}(W_i)$
- We stop when $W_i = W_{i+1}$ (fixed point).
- Controller wins iff $q_0 \in W_i$. The initial specification is *realizable*.
- Otherwise, Environment has a strategy for violating the specification.



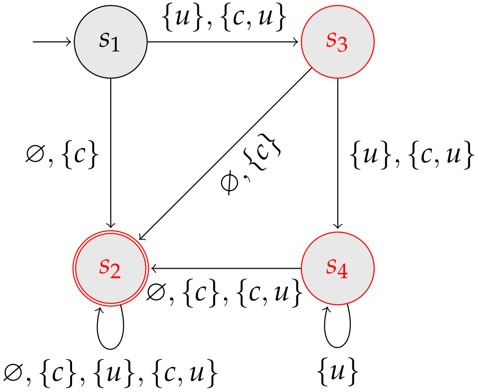
- DFA for the formula $\mathsf{F}(u \rightarrow \mathsf{XX}c)$, with $u \in \mathsf{U}$ and $c \in \mathsf{C}$.



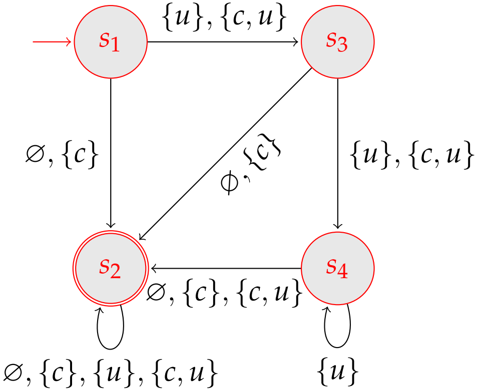
- DFA for the formula $\mathsf{F}(u \to \mathsf{XX}c)$, with $u \in \mathsf{U}$ and $c \in \mathsf{C}$.
- $W_0 := \{s_2\}$



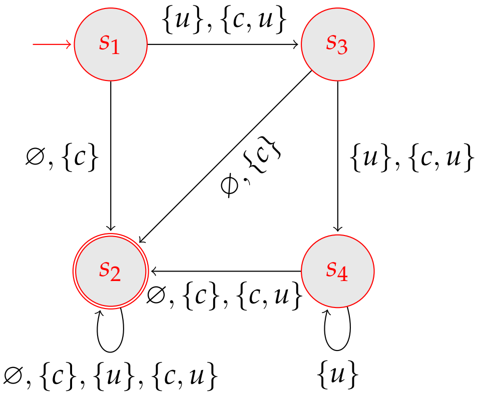
- DFA for the formula $\mathsf{F}(u \rightarrow \mathsf{XX}c)$, with $u \in \mathsf{U}$ and $c \in \mathsf{C}$.
- $W_0 := \{s_2\}$
- $W_1 := \{s_2, s_4\}$



- DFA for the formula $\mathsf{F}(u \rightarrow \mathsf{XX}c)$, with $u \in \mathsf{U}$ and $c \in \mathsf{C}$.
- $W_0 := \{s_2\}$
- $W_1 := \{s_2, s_4\}$
- $W_2 := \{s_2, s_4, s_3\}$



- DFA for the formula $\mathsf{F}(u \to \mathsf{XX}c)$, with $u \in \mathsf{U}$ and $c \in \mathsf{C}$.
- $W_0 := \{s_2\}$
- $W_1 := \{s_2, s_4\}$
- $W_2 := \{s_2, s_4, s_3\}$
- $W_3 := \{s_2, s_4, s_3, s_1\}$



- DFA for the formula $\mathsf{F}(u \rightarrow \mathsf{XX}c)$, with $u \in \mathsf{U}$ and $c \in \mathsf{C}$.
- $W_0 := \{s_2\}$
- $W_1 := \{s_2, s_4\}$
- $W_2 := \{s_2, s_4, s_3\}$
- $W_3 := \{s_2, s_4, s_3, s_1\}$
- $W_3 \cap I \neq \varnothing \Rightarrow$ the formula is realizable.

The case of Infinite Words

Standard approach:

LTL ϕ

↓

NBA $\mathcal{A}(\phi)$

↓ determinization

DRA $\mathcal{A}(\phi)$

↓

game solver

The case for *infinite words* (like in the case for LTL) is much more difficult. Two reasons:

- Büchi games
- NBA cannot be determinized easily. Indeed, *Safra's construction* is:
  - very complicated
  - difficult to implement
  - not amenable to optimizations

The case of Infinite Words

Standard approach:

LTL ϕ

$\downarrow$

NBA $\mathcal{A}(\phi)$

$\downarrow$ · determinization

DRA $\mathcal{A}(\phi)$

$\downarrow$

game solver

Research mainly focused on two lines

1. finding good algorithms for the average case
   - Safraless approaches
     - Bounded synthesis
2. restricting the expressiveness of the specification language
   - GR(1)
   - SafetyLTL

SafetyLTL ϕ

↓ · bad prefixes

DFA $\mathcal{A}_{bad}$

↓

reachability game

Game:

- Now, Controller moves first
- Goal of Controller: always avoid final states of $\mathcal{A}_{bad}$.
- Goal of Environment: reach a final state of $\mathcal{A}_{bad}$.

SafetyLTL ϕ

↓ bad prefixes

DFA $\mathcal{A}_{bad}$

↓

reachability game

Pros:

- infinite words $\rightsquigarrow$ finite word
- Safra's algorithm is *avoided*.
- We use standard subset construction for $\mathcal{A}_{bad}$:
  - easily implementable
  - easily optimizable

Cons:

- the size of $\mathcal{A}_{bad}$ is $2^{2^{\Theta(n)}}$.
- this is prohibitive when ϕ is large.

SafetyLTL ϕ

↓ bad prefixes

DFA $\mathcal{A}_{bad}$

↓

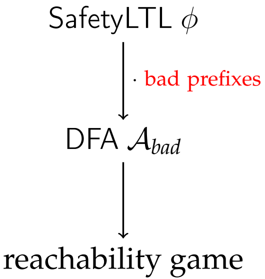
reachability game

Tool: SSyft

Reference:

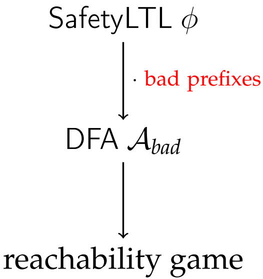
Shufang Zhu et al. (2017). "A Symbolic Approach to Safety LTL Synthesis". In: *Proceedings of the 13th International Haifa Verification Conference*. Ed. by Ofer Strichman and Rachel Tzoref-Brill. Vol. 10629. Lecture Notes in Computer Science. Springer, pp. 147–162. DOI: 10.1007/978-3-319-70389-3_10

Link: https://github.com/Shufang-Zhu/Syft-safety



Tool: SSyft

1. Let ϕ be a SafetyLTL formula.
2. Translate $\neg\phi$ into an equivalent formula ψ of S1S[FO] interpreted over finite words.
   - the models of ψ are exactly the *bad prefixes* of ϕ
3. Call the tool MONA for building the equivalent and *minimal* DFA.
4. Solve a reachability game.



- MONA is a very efficient tool for the construction of automata starting from formulas.
- MONA implements decision procedures for the Weak Second-order Theory of One or Two successors.
- Link : `https://www.brics.dk/mona/`

SafetyLTL ϕ

$\downarrow$ bad prefixes

DFA $\mathcal{A}_{bad}$

$\downarrow$

reachability game

Theorem

SafetyLTL *realizability is* 2EXPTIME-*complete.*

Reference:

Alessandro Artale, Luca Geatti, et al. (2023b). "Complexity of Safety and coSafety Fragments of Linear Temporal Logic". In: *Proc. of the 36th AAAI Conf. on Artificial Intelligence*. AAAI Press

Logics	Problems		
	satisfiability	model checking	realizability
coSafetyLTL	PSPACE-c	???	2EXPTIME-c
F(pLTL)	PSPACE-c	???	EXPTIME-c
LTL[X, F]	NP-c	???	EXPTIME-c

Logics	Problems		
	satisfiability	model checking	realizability
SafetyLTL	PSPACE-c	???	2EXPTIME-c
G(pLTL)	PSPACE-c	???	EXPTIME-c
LTL[$\widetilde{X}$, G]	PSPACE-c	???	EXPTIME-c

Logics	Problems		
	satisfiability	model checking	realizability
coSafetyLTL	PSPACE-c	???	2EXPTIME-c
F(pLTL)	PSPACE-c	???	EXPTIME-c
LTL[X, F]	NP-c	???	EXPTIME-c

Logics	Problems		
	satisfiability	model checking	realizability
SafetyLTL	PSPACE-c	???	2EXPTIME-c
G(pLTL)	PSPACE-c	???	EXPTIME-c
LTL[$\widetilde{\mathsf{X}}$, G]	PSPACE-c	???	EXPTIME-c

G(pLTL) ϕ

↓ · deterministic automaton · of singly exponential size

DFA $\mathcal{A}$

↓

safety game

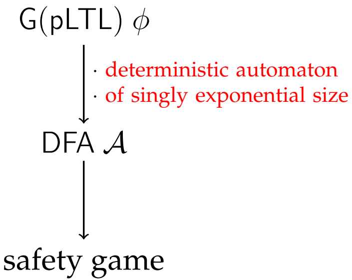
Algorithm:

1. Let $\mathsf{G}(\alpha)$ be a formula of G(pLTL).

Theorem

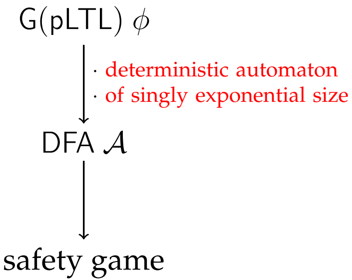
ϕ is realizable (with Environment moving first) iff $\neg\phi$ is unrealizable (with Controller moving first).

$\mathsf{G}(\alpha)$ is realizable iff $\mathsf{F}(\neg\alpha)$ is unrealizable (with Controller moving first).



Algorithm:

1. Let $\mathsf{G}(\alpha)$ be a formula of G(pLTL).
   $\mathsf{G}(\alpha)$ is realizable iff $\mathsf{F}(\neg\alpha)$ is unrealizable
2. Build the DFA $\mathcal{A}_{\neg\alpha}$ for $\neg\alpha$
   - this can be done in $2^{\mathcal{O}(n)}$
   - we will see later its construction
3. Solve a reachability game on $\mathcal{A}_{\neg\alpha}$:
   - if Controller (that moves first) wins:
     - $\mathsf{F}(\neg\alpha)$ is realizable
     - $\mathsf{G}(\alpha)$ is unrealizable
   - if Environment wins:
     - $\mathsf{F}(\neg\alpha)$ is unrealizable
     - $\mathsf{G}(\alpha)$ is realizable



- Advantages:
  - The size of $|\mathcal{A}|$ is $2^{\mathcal{O}(n)}$:
    - singly exponential
    - one exponential smaller than the set of bad prefixes of a SafetyLTL formula.
  - The translation from pLTL into DFA can be done in a purely symbolic fashion

Reference:

Alessandro Cimatti et al. (2021). "Extended bounded response LTL: a new safety fragment for efficient reactive synthesis". In: *Formal Methods in System Design*, 1–49 (published online on November 18, 2021, doi: 10.1007/s10703-021-00383–3)

Theorem

For any formula ϕ of pLTL *with $n = |\phi|$, there exists a* DFA *$\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\phi)$ and $|\mathcal{A}| \in 2^{\mathcal{O}(n)}$.*

Reference:

Giuseppe De Giacomo et al. (2021). "Pure-past linear temporal and dynamic logic on finite traces". In: *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pp. 4959–4965

Theorem

For any formula ϕ of pLTL *with $n = |\phi|$, there exists a* DFA *$\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\phi)$ and $|\mathcal{A}| \in 2^{\mathcal{O}(n)}$.*

Intuition:

Since past already happened, there is no need for nondeterminism.

There is this useful asymmetry:

- *The automaton reads from left to right;*
- *The pure past formula predicates from right to left.*

Theorem

For any formula ϕ *of* pLTL *with* $n = |\phi|$*, there exists a* DFA $\mathcal{A}$ *such that* $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\phi)$ *and* $|\mathcal{A}| \in 2^{\mathcal{O}(n)}$.

De Giacomo *et al*. prove the result passing from alternating automata.

Theorem

For any alternating finite automaton $\mathcal{A}$*, there exists a* DFA *for its reverse language of size singly exponential in* $|\mathcal{A}|$.

Reference:

Ashok K. Chandra, Dexter Kozen, and Larry J. Stockmeyer (1981). “Alternation”. In: *J. ACM* 28.1, pp. 114–133. DOI: 10.1145/322234.322243. URL: https://doi.org/10.1145/322234.322243

Theorem

For any formula ϕ *of* pLTL *with* $n = |\phi|$, *there exists a* DFA $\mathcal{A}$ *such that* $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\phi)$ *and* $|\mathcal{A}| \in 2^{\mathcal{O}(n)}$.

Here we give a direct construction.

Definition (Closure of pLTL formulas)

The *closure* of a pLTL formula ϕ over the atomic propositions $\mathcal{AP}$, denoted as $\mathcal{C}(\phi)$, is the smallest set of formulas satisfying the following properties:

- $\mathsf{Y}\phi \in \mathcal{C}(\phi)$
- $\phi \in \mathcal{C}(\phi)$, and, for each subformula ϕ' of ϕ, $\phi' \in \mathcal{C}(\phi)$
- for each $p \in \mathcal{AP}$, $p \in \mathcal{C}(\phi)$ if and only if $\neg p \in \mathcal{C}(\phi)$
- if $\phi_1 \mathsf{S} \phi_2 \in \mathcal{C}(\phi)$, then $\mathsf{Y}(\phi_1 \mathsf{S} \phi_2) \in \mathcal{C}(\phi)$
    - if $\mathsf{O}\phi_1 \in \mathcal{C}(\phi)$, then $\mathsf{Y}(\mathsf{O}\phi_1) \in \mathcal{C}(\phi)$
- if $\phi_1 \mathsf{T} \phi_2 \in \mathcal{C}(\phi)$, then $\widetilde{\mathsf{Y}}(\phi_1 \mathsf{T} \phi_2) \in \mathcal{C}(\phi)$
    - if $\mathsf{H}\phi_1 \in \mathcal{C}(\phi)$, then $\widetilde{\mathsf{Y}}(\mathsf{H}\phi_1) \in \mathcal{C}(\phi)$

- We denote by $\mathcal{C}_{\mathsf{Y}}(\phi)$ the set of formulas of type $\mathsf{Y}\phi_1$ in $\mathcal{C}(\phi)$.
- We denote by $\mathcal{C}_{\widetilde{\mathsf{Y}}}(\phi)$ the set of formulas of type $\widetilde{\mathsf{Y}}\phi_1$ in $\mathcal{C}(\phi)$.

Definition (Stepped Normal Form)

Let ϕ be a pLTL formula over the atomic propositions $\mathcal{AP}$. Its *stepped normal form*, denoted by $\mathrm{snf}(\phi)$, is defined as follows:

$$
\begin{aligned}
\mathrm{snf}(\ell) &= \ell && \text{where } \ell \in \{p, \neg p\}, \text{ for } p \in \mathcal{AP} \\
\mathrm{snf}(\otimes\, \phi_1) &= \otimes\, \phi_1 && \text{where } \otimes \in \{\mathsf{Y}, \widetilde{\mathsf{Y}}\} \\
\mathrm{snf}(\phi_1 \otimes \phi_2) &= \mathrm{snf}(\phi_1) \otimes \mathrm{snf}(\phi_2) && \text{where } \otimes \in \{\wedge, \vee\} \\
\mathrm{snf}(\phi_1 \,\mathsf{S}\, \phi_2) &= \mathrm{snf}(\phi_2) \vee (\mathrm{snf}(\phi_1) \wedge \mathsf{Y}(\phi_1 \,\mathsf{S}\, \phi_2)) \\
\mathrm{snf}(\phi_1 \,\mathsf{T}\, \phi_2) &= \mathrm{snf}(\phi_2) \wedge (\mathrm{snf}(\phi_1) \vee \widetilde{\mathsf{Y}}(\phi_1 \,\mathsf{T}\, \phi_2))
\end{aligned}
$$

Example: $\mathrm{snf}(\mathsf{O}q) = q \vee \mathsf{YO}q$.

Given a set $S \subseteq \mathcal{C}_{\mathsf{Y}}(\phi) \cup \mathcal{C}_{\widetilde{\mathsf{Y}}}(\phi)$ and a $\sigma \in 2^{\mathcal{AP}}$, we write $S, \sigma \models \phi$ iff ϕ is true when:

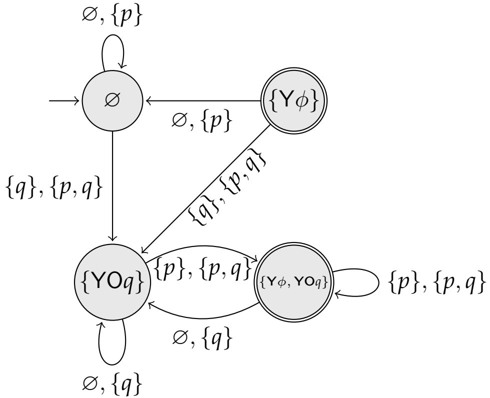
- S is used for interpreting the subformulas of type $\mathsf{Y}\alpha$ and $\widetilde{\mathsf{Y}}\alpha$
- σ is used for interpreting proposition letters in $\mathcal{AP}$

Example:

- $S = \{\mathsf{YO}q\}$
- $\sigma = \varnothing$
- $S, \sigma \models q \vee \mathsf{YO}q$

Given $\phi \in \mathsf{LTL}$ we define the DFA $\mathcal{A}_\phi = \langle Q, \Sigma, q_0, \delta, F \rangle$ as follows:

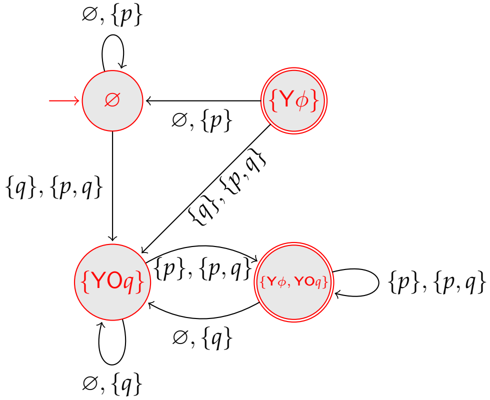
Example: $\phi := p \wedge \mathsf{YO}q$



Given $\phi \in \mathsf{LTL}$ we define the DFA $\mathcal{A}_\phi = \langle Q, \Sigma, q_0, \delta, F \rangle$ as follows:

- $Q = 2^{\mathcal{C}_{\mathsf{Y}}(\phi) \cup \mathcal{C}_{\widetilde{\mathsf{Y}}}(\phi)}$
  - $Q = \{\varnothing, \{\mathsf{Y}\phi\}, \{\mathsf{YO}q\}, \{\mathsf{Y}\phi, \mathsf{YO}q\}\}$

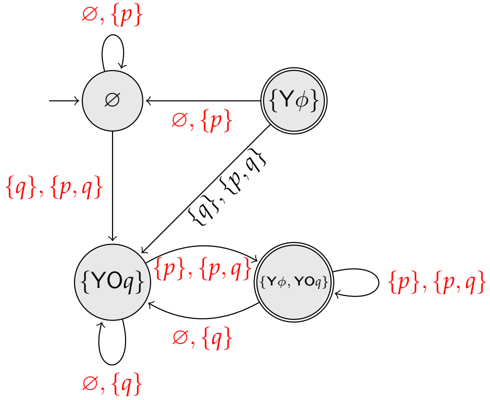
Example: $\phi := p \wedge \mathsf{YO}q$



Given $\phi \in \mathsf{LTL}$ we define the DFA $\mathcal{A}_\phi = \langle Q, \Sigma, q_0, \delta, F \rangle$ as follows:

- $Q = 2^{\mathcal{C}_{\mathsf{Y}}(\phi) \cup \mathcal{C}_{\tilde{\mathsf{Y}}}(\phi)}$
  - $Q = \{\varnothing, \{\mathsf{Y}\phi\}, \{\mathsf{YO}q\}, \{\mathsf{Y}\phi, \mathsf{YO}q\}\}$
- $\Sigma = 2^{\mathcal{AP}}$
  - $\Sigma = \{\varnothing, \{p\}, \{q\}, \{p, q\}\}$

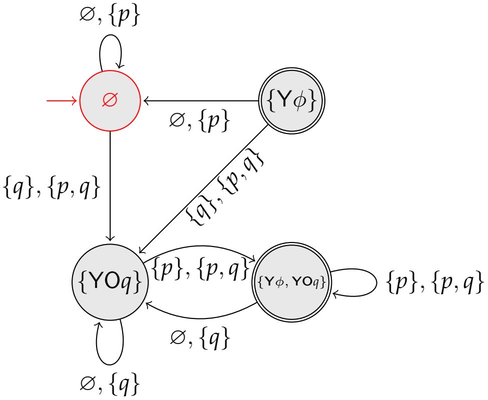
Example: $\phi := p \land \mathsf{YO}q$



Given $\phi \in$ LTL we define the DFA $\mathcal{A}_\phi = \langle Q, \Sigma, q_0, \delta, F \rangle$ as follows:

- $Q = 2^{\mathcal{C}_{\mathsf{Y}}(\phi) \cup \mathcal{C}_{\widetilde{\mathsf{Y}}}(\phi)}$
  - $Q = \{\varnothing, \{\mathsf{Y}\phi\}, \{\mathsf{YO}q\}, \{\mathsf{Y}\phi, \mathsf{YO}q\}\}$
- $\Sigma = 2^{\mathcal{AP}}$
  - $\Sigma = \{\varnothing, \{p\}, \{q\}, \{p, q\}\}$
- $q_0 = \mathcal{C}_{\widetilde{\mathsf{Y}}}(\phi)$
  - $q_0 = \varnothing$

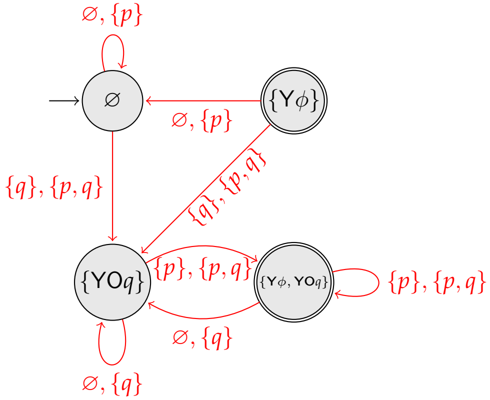
Example: $\phi := p \wedge \mathsf{YO}q$



Given $\phi \in$ LTL we define the DFA $\mathcal{A}_\phi = \langle Q, \Sigma, q_0, \delta, F \rangle$ as follows:

- $Q = 2^{\mathcal{C}_{\mathsf{Y}}(\phi) \cup \mathcal{C}_{\widetilde{\mathsf{Y}}}(\phi)}$
  - $Q = \{\varnothing, \{\mathsf{Y}\phi\}, \{\mathsf{YO}q\}, \{\mathsf{Y}\phi, \mathsf{YO}q\}\}$
- $\Sigma = 2^{\mathcal{AP}}$
  - $\Sigma = \{\varnothing, \{p\}, \{q\}, \{p, q\}\}$
- $q_0 = \mathcal{C}_{\widetilde{\mathsf{Y}}}(\phi)$
  - $q_0 = \varnothing$
- $\delta(q, \sigma) = \{\mathsf{Y}\psi, \widetilde{\mathsf{Y}}\psi \in \mathcal{C}_{\mathsf{Y}}(\phi) \cup \mathcal{C}_{\widetilde{\mathsf{Y}}}(\phi) \mid q, \sigma \models \mathrm{snf}(\psi)\}$
  - see figure

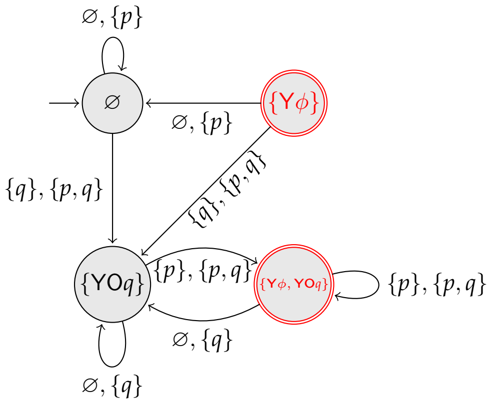
Example: $\phi := p \wedge \mathsf{YO}q$



Given $\phi \in \mathsf{LTL}$ we define the DFA $\mathcal{A}_\phi = \langle Q, \Sigma, q_0, \delta, F \rangle$ as follows:

- $Q = 2^{\mathcal{C}_{\mathsf{Y}}(\phi) \cup \mathcal{C}_{\widetilde{\mathsf{Y}}}(\phi)}$
  - $Q = \{\varnothing, \{\mathsf{Y}\phi\}, \{\mathsf{YO}q\}, \{\mathsf{Y}\phi, \mathsf{YO}q\}\}$
- $\Sigma = 2^{\mathcal{AP}}$
  - $\Sigma = \{\varnothing, \{p\}, \{q\}, \{p, q\}\}$
- $q_0 = \mathcal{C}_{\widetilde{\mathsf{Y}}}(\phi)$
  - $q_0 = \varnothing$
- $\delta(q, \sigma) = \{\mathsf{Y}\psi, \widetilde{\mathsf{Y}}\psi \in \mathcal{C}_{\mathsf{Y}}(\phi) \cup \mathcal{C}_{\widetilde{\mathsf{Y}}}(\phi) \mid q, \sigma \models \mathrm{snf}(\psi)\}$
  - see figure
- $F = \{S \subseteq \mathcal{C}_{\mathsf{Y}}(\phi) \cup \mathcal{C}_{\widetilde{\mathsf{Y}}}(\phi) \mid \mathsf{Y}\phi \in S\}$
  - $F = \{\{\mathsf{Y}\phi\}, \{\mathsf{Y}\phi, \mathsf{YO}q\}\}$

Example: $\phi := p \wedge \mathsf{YO}q$



Theorem

$\mathsf{G}(\mathsf{pLTL})$ *realizability is* EXPTIME*-complete.*

Theorem

$\mathsf{F}(\mathsf{pLTL})$ *realizability is* EXPTIME*-complete.*

Reference:

Alessandro Artale, Luca Geatti, et al. (2023b). "Complexity of Safety and coSafety Fragments of Linear Temporal Logic". In: *Proc. of the 36th AAAI Conf. on Artificial Intelligence*. AAAI Press

Logics	Problems		
	satisfiability	model checking	realizability
coSafetyLTL	PSPACE-c	???	2EXPTIME-c
F(pLTL)	PSPACE-c	???	EXPTIME-c
LTL[X, F]	NP-c	???	EXPTIME-c

Logics	Problems		
	satisfiability	model checking	realizability
SafetyLTL	PSPACE-c	???	2EXPTIME-c
G(pLTL)	PSPACE-c	???	EXPTIME-c
LTL[$\widetilde{X}$, G]	PSPACE-c	???	EXPTIME-c

Theorem

G(pLTL) *realizability is* EXPTIME*-complete.*

- Pure past LTL plays a crucial role for safety fragments
- SafetyLTL realizability is 2EXPTIME-complete
- ...but G(pLTL) and SafetyLTL are expressively equivalent

Theorem

G(pLTL) *realizability is* EXPTIME*-complete.*

- Pure past LTL plays a crucial role for safety fragments
- SafetyLTL realizability is 2EXPTIME-complete
- ...but G(pLTL) and SafetyLTL are expressively equivalent

Two questions:

1. Succinctness

   Can SafetyLTL *be exponentially more succinct than* G(pLTL)*?*

2. Pastification algorithms

   Can a logic be efficiently translated into a pure-past one, by preserving equivalence?

SUCCINCTNESS AND PASTIFICATION

Known results and open questions

Informal definition.

Given two linear-time temporal logics $\mathbb{L}$ and $\mathbb{L}'$, we say that $\mathbb{L}$ can be exponentially more succinct than $\mathbb{L}'$ iff there exists a property such that

- *it can be succinctly expressed in $\mathbb{L}$,*
- *but all formulas of $\mathbb{L}'$ for it are at least exponentially larger.*

Formal definition.

Definition

Given two linear-time temporal logics $\mathbb{L}$ and $\mathbb{L}'$, we say that $\mathbb{L}$ *can be exponentially more succinct than $\mathbb{L}'$ over infinite trace* (resp., *over finite traces*) iff there exists an alphabet Σ and a family of languages $\{\mathcal{L}_n\}_{n>0} \subseteq (2^\Sigma)^\omega$ (resp., $\{\mathcal{L}_n\}_{n>0} \subseteq (2^\Sigma)^*$) such that, for any $n > 0$,

- there exists a formula $\phi \in \mathbb{L}$ over Σ such that its language over infinite traces (resp., over finite traces) is $\mathcal{L}_n$ and $|\phi| \in \mathcal{O}(n)$, and
- for all formulas $\phi' \in \mathbb{L}'$ over Σ, if the language of ϕ' over infinite traces (resp., finite traces) is $\mathcal{L}_n$, then $|\phi'| \in 2^{\Omega(n)}$.

Succinctness

Succinctness is important for various reasons.

In particular,

1. it helps choosing the right formalism when solving problems like reactive synthesis, model checking, and so on;
2. it is an important theoretical tool, that connects the study of computational complexity to that of expressive power.

A well-known result about LTL+P and LTL.

Theorem

LTL+P *can be exponentially more succinct than* LTL.

Reference:

Nicolas Markey (2003). “Temporal logic with past is exponentially more succinct”. In: *Bull. EATCS* 79, pp. 122–128

Succinctness of (co)safety fragments of LTL

Theorem

F(pLTL) *can be exponentially more succinct than* coSafetyLTL.

It follows from the result by Markey.

Here we give a simplified version.

Proof.

Steps (proof by contradiction):

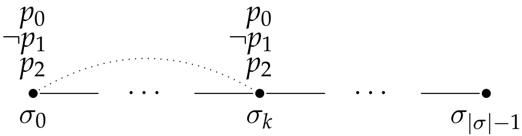
1. For all $n > 0$, find a language A_n such that $\mathcal{L}(\phi_n) = A_n$ and $|\phi_n| \in \mathcal{O}(n)$, for some $\phi_n \in$ F(pLTL).
2. Suppose by contradiction that, for all $n > 0$, there exists a formula ϕ'_n of coSafetyLTL such that $\mathcal{L}(\phi'_n) = \mathcal{L}(\phi_n)$ and $|\phi'_n|$ is polynomial in n.
3. Use ϕ'_n to build a formula ψ_n of LTL+P such that $|\psi_n|$ is polynomial in n. Let $B_n = \mathcal{L}(\psi_n)$.
4. Prove that all NBA for B_n are of size $2^{2^{\Omega(n)}}$.
5. Exploit the fact that there exists a singly exponential translation from LTL+P to equivalent NBA to prove that:
   - all LTL+P formulas of B_n are of size $2^{\Omega(n)}$.
6. Conclude that all formulas of coSafetyLTL that express A_n are of size $2^{\Omega(n)}$.

Succinctness of (co)safety fragments of LTL

1. For all $n > 0$, find a language A_n such that $\mathcal{L}(\phi_n) = A_n$ and $|\phi_n| \in \mathcal{O}(n)$, for some $\phi_n \in \mathsf{F}(\mathsf{pLTL})$.

Let $\Sigma = \{p_0, p_1, \ldots, p_n\}$.

$$A_n \coloneqq \{\sigma \in (2^\Sigma)^+ \mid \exists k > 0 \,.\, (\bigwedge_{i=0}^{n} (p_i \in \sigma_k \leftrightarrow p_i \in \sigma_0))\}$$



1. For all $n > 0$, find a language A_n such that $\mathcal{L}(\phi_n) = A_n$ and $|\phi_n| \in \mathcal{O}(n)$, for some $\phi_n \in \mathsf{F}(\mathsf{pLTL})$.

Let $\Sigma = \{p_0, p_1, \ldots, p_n\}$.

$$A_n \coloneqq \{\sigma \in (2^{\Sigma})^{+} \mid \exists k > 0 \,.\, (\bigwedge_{i=0}^{n} (p_i \in \sigma_k \leftrightarrow p_i \in \sigma_0))\}$$

Lemma

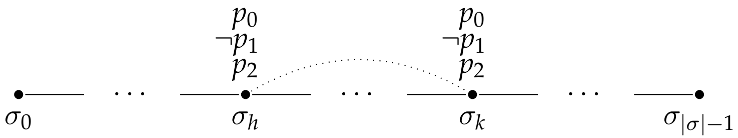
For any $n > 0$, there exists a formula $\phi \in \mathsf{F}(\mathsf{pLTL})$ such that $\mathcal{L}(\phi) = A_n$ and $|\phi| \in \mathcal{O}(n)$.

Proof.

$$\mathsf{F}\Big(\bigwedge_{i=0}^{n} (p_i \leftrightarrow \mathsf{Y}\mathsf{O}(\widetilde{\mathsf{Y}}\bot \wedge p_i))\Big)$$

2. Suppose by contradiction that, for all $n > 0$, there exists a formula ϕ'_n of coSafetyLTL such that $\mathcal{L}(\phi'_n) = \mathcal{L}(\phi_n)$ and $|\phi'_n|$ is polynomial in n.
3. Use ϕ'_n to build a formula ψ_n of LTL+P such that $|\psi_n|$ is polynomial in n. Let $B_n = \mathcal{L}(\psi_n)$.

- $\psi_n := \mathsf{F}(\phi'_n)$
- $B_n := \{\sigma \in (2^\Sigma)^+ \mid \exists h \geq 0 \,.\, \exists k > h \,.\, (\bigwedge_{i=0}^{n}(p_i \in \sigma_k \leftrightarrow p_i \in \sigma_h))\}$



Succinctness of (co)safety fragments of LTL

2. Suppose by contradiction that, for all $n > 0$, there exists a formula ϕ'_n of coSafetyLTL such that $\mathcal{L}(\phi'_n) = \mathcal{L}(\phi_n)$ and $|\phi'_n|$ is polynomial in n.
3. Use ϕ'_n to build a formula ψ_n of LTL+P such that $|\psi_n|$ is polynomial in n. Let $B_n = \mathcal{L}(\psi_n)$.

Lemma

If there exists a formula of coSafetyLTL for A_n of size less than exponential in n, then there exists a formula of LTL+P for B_n of size less than exponential in n.

4. Prove that all NBA for B_n are of size $2^{2^{\Omega(n)}}$.

Lemma

For any $n > 0$ and any NBA $\mathcal{A}$ over the alphabet 2^{Σ}, if $\mathcal{L}(\mathcal{A}) = B_n$ then $|\mathcal{A}| \in 2^{2^{\Omega(n)}}$.

Reference:

Kousha Etessami, Moshe Y. Vardi, and Thomas Wilke (2002). "First-Order Logic with Two Variables and Unary Temporal Logic". In: *Inf. Comput.* 179.2, pp. 279–295. DOI: 10.1006/inco.2001.2953. URL: https://doi.org/10.1006/inco.2001.2953

4. Exploit the fact that there exists a singly exponential translation from LTL+P to equivalent NBA to prove that:
   - all LTL+P formulas of B_n are of size $2^{\Omega(n)}$.

Proposition

For any LTL *formula* ϕ, *with* $|\phi| = n$, *over the set of atomic propositions* $\mathcal{AP}$, *there exists an* NBA $\mathcal{A}_\phi$ *over the alphabet* $2^{\mathcal{AP}}$ *such that:*

- $\mathcal{L}(\phi) = \mathcal{L}(\mathcal{A}_\phi)$
- $|\,\mathcal{A}_\phi\,| \in 2^{\mathcal{O}(n)}$

Lemma

For any formula $\phi \in$ LTL+P, *if* $\mathcal{L}(\phi) = B_n$, *then* $|\phi| \in 2^{\Omega(n)}$.

4. Conclude that all formulas of coSafetyLTL that express A_n are of size $2^{\Omega(n)}$.

Theorem

For any $n > 0$ and any formula $\phi \in$ coSafetyLTL, if $\mathcal{L}(\phi) = A_n$, then $|\phi| \in 2^{\Omega(n)}$.

Corollary

F(pLTL) *can be exponentially more succinct than* coSafetyLTL.

By a simple duality argument:

Corollary

G(pLTL) *can be exponentially more succinct than* SafetyLTL.

All these results have been collected in:

Reference:

Alessandro Artale, Luca Geatti, et al. (2023c). "LTL over finite words can be exponentially more succinct than pure-past LTL, and vice versa". In: *Proceedings of the 30th International Symposium on Temporal Representation and Reasoning, TIME 2023, September 25-26, 2023, NCSR Demokritos, Athens, Greece*. Ed. by Florian Bruse Alexander Artikis and Luke Hunsberger. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik

Open problem:

Can coSafetyLTL be exponentially more succinct than F(pLTL)?

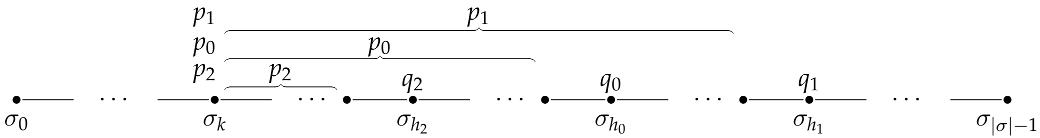
Conjecture:

coSafetyLTL can be $n!$ more succinct than F(pLTL).

Conjecture:

coSafetyLTL can be $n!$ more succinct than F(pLTL).

- $C_n := \{\sigma \in (2^\Sigma)^\omega \mid \exists k \geq 0 \,.\, \bigwedge_{i=1}^{n} (\exists h > k \,.\, (q_i \in \sigma_h \wedge \forall k \leq l < h \,.\, p_i \in \sigma_l))\}$
- $\mathsf{F}(\bigwedge_{i=1}^{n} p_i \,\mathsf{U}\, q_i)$



- In F(pLTL), one needs to specify all permutations of the set $\{q_1, \ldots, q_n\}$.

Recall that $[\![\mathsf{LTL}]\!] \cap \mathsf{SAFETY} = [\![\mathsf{LTL}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega}$

Consider now $\mathsf{LTL_f}$, that is, $[\![\mathsf{LTL}]\!]^{<\omega}$. The following incomparability result holds.

Theorem

- $\mathsf{LTL_f}$ *can be exponentially more succinct than* pLTL.
- pLTL *can be exponentially more succinct than* $\mathsf{LTL_f}$.

Reference:

Alessandro Artale, Luca Geatti, et al. (2023c). “LTL over finite words can be exponentially more succinct than pure-past LTL, and vice versa”. In: *Proceedings of the 30th International Symposium on Temporal Representation and Reasoning, TIME 2023, September 25-26, 2023, NCSR Demokritos, Athens, Greece*. Ed. by Florian Bruse Alexander Artikis and Luke Hunsberger. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik

Succinctness and Pastification

- Let us consider again the case of coSafetyLTL and F(pLTL).
- Succinctness properties can be considered as lower bounds for the transformation of coSafetyLTL into F(pLTL).
- The transformation of a pure future fragment into a pure past one is called

PASTIFICATION

- Originally introduced in the context of synthesis of timed temporal logics:

Reference:

Oded Maler, Dejan Nickovic, and Amir Pnueli (2007). "On synthesizing controllers from bounded-response properties". In: *Proceedings of the International Conference on Computer Aided Verification*. Springer, pp. 95–107. DOI: 10.1023/A:1008734703554

- We now look at some pastification algorithms (upper bounds)

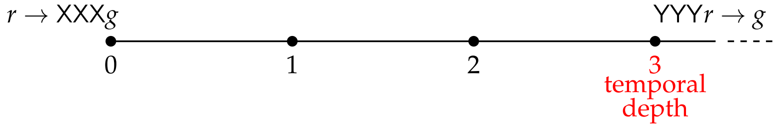
Pastification Algorithms

Let us briefly consider pastification algorithms for the following fragments:

- LTL[X]
  - *polynomial-size pastification*
- LTL[X, F]
  - *exponential-size pastification*
- coSafetyLTL
  - *triply exponential-size pastification*
- $\mathsf{LTL_f}$
  - *triply exponential-size pastification*

Transforming LTL[X] into F(pLTL)

- Let $\phi \in \mathsf{LTL}[\mathsf{X}]$.
- There exists a time point $d \in \mathbb{N}$, that is, the temporal depth of ϕ, such that the subsequent states cannot be constrained by ϕ.
  - temporal depth of ϕ = maximum number of nested X operators
- Thus, we can write a formula (the pastification of ϕ) that uses only past operators and is equivalent to ϕ when interpreted at d.
- Example: $\phi \coloneqq r \rightarrow \mathsf{XXX}g$



It holds that: $r \rightarrow \mathsf{XXX}g \equiv \mathsf{F}(\mathsf{at}_3 \wedge (\mathsf{YYY}r \rightarrow g))$.

- where $\mathsf{at}_3 \coloneqq \widetilde{\mathsf{Y}}\widetilde{\mathsf{Y}}\widetilde{\mathsf{Y}}\bot \wedge \mathsf{YY}\top$.

Theorem

There is a polynomial-size pastification of LTL[X] *into* F(pLTL).

Reference:

Oded Maler, Dejan Nickovic, and Amir Pnueli (2007). "On synthesizing controllers from bounded-response properties". In: *Proceedings of the International Conference on Computer Aided Verification*. Springer, pp. 95–107. DOI: 10.1023/A:1008734703554

Transforming LTL[X, F] into F(pLTL)

Theorem

There is a 1 exponential-size pastification of LTL[X, F] *into* F(pLTL).

- Data structure: *dependency trees*
- Candidate lower bound: $\mathsf{F}(\bigwedge_{i=1}^{n}(p_i \vee \mathsf{F}q_i))$

Reference:

Alessandro Artale, Luca Geatti, et al. (2023a). "A Singly Exponential Transformation of LTL[X,F] into Pure Past LTL". In: *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning, KR 2023, Rhodes, Greece. September 2-8, 2023*

Theorem

There is a 3 exponential-size pastification of coSafetyLTL *into* F(pLTL).

Let ϕ be a coSafetyLTL formula.

1. Build the DFA $\mathcal{A}'_\phi$ for the set of *good prefixes* of ϕ:
   - doubly exponential blow-up
2. Use the Krohn-Rhodes Primary Decomposition Theorem to build a cascade product equivalent to $\mathcal{A}'_\phi$.
   - exponential blow-up
3. Translate the cascade product into a formula ψ of pLTL. Return F(ψ).
   - linear

Total: triply exponential pastification algorithm.

Transforming coSafetyLTL into F(pLTL)

Theorem

There is a 3 exponential-size pastification of coSafetyLTL *into* F(pLTL).

Reference:

Oded Maler and Amir Pnueli (1990). "Tight bounds on the complexity of cascaded decomposition of automata". In: *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*. IEEE, pp. 672–682

There are two missing exponentials between the best-known upper and lower bounds:

- best known upper bound: triply exponential
- best known lower bound: singly exponential

As for $\mathsf{LTL_f}$, the best known algorithm is the same as the one for coSafetyLTL.

Let ϕ be a $\mathsf{LTL_f}$ formula.

1. Build a NFA $\mathcal{A}_\phi$ for ϕ.
   - exponential blow-up
2. Determinize $\mathcal{A}_\phi$ into a DFA $\mathcal{A}'_\phi$.
   - exponential blow-up
3. Use the Krohn-Rhodes Primary Decomposition Theorem to build a cascade product equivalent to $\mathcal{A}'_\phi$.
   - exponential blow-up
4. Translate the cascade product into pLTL.
   - linear

Total: triply exponential pastification algorithm.

	Upper bound	Lower bound
LTL[X]	linear	linear
LTL[X, F]	1-exp	?
coSafetyLTL	3-exp	?
$\mathsf{LTL_f}$	3-exp	1-exp

A *polynomial-size* pastification algorithm is a very uncommon feature for a logic.

CONCLUSIONS

Conclusions: results

- Characterizations of safety and cosafety fragments of LTL:
  - reduction from infinite to finite words reasoning
- Role of past temporal operators in the definition of canonical forms
- Kupferman & Vardi's classification of safety properties:
  - intentionally, accidentally, and pathologically safe.
- Algorithms to recognize safety automata and LTL safety formulas
- Algorithms to build the set of bad prefixes
  - *doubly exponential* DFA
- Algorithms for
  - satisfiability checking
  - model checking
    - the worst-case complexity does not change
    - efficient algorithms in practice
  - reactive synthesis
    - avoid Safra's determinization
    - by using past operators, the worst-case complexity can be decreased by one exponential
- Succinctness issues
  - G(pLTL) can be exponentially more succinct than SafetyLTL
- Pastification algorithms

Conclusions: open problems

- Some interesting open problems:
  - Worst-case complexity of safety model checking
  - Succinctness lower bounds
    - $\mathsf{LTL}[\mathsf{X}, \mathsf{F}]$
    - coSafetyLTL
  - Efficient pastification algorithms

REFERENCES

Alessandro Abate et al. (2021). “Rational verification: game-theoretic verification of multi-agent systems”. In: *Applied Intelligence* 51.9, pp. 6569–6584.

Bowen Alpern and Fred B. Schneider (1987). “Recognizing Safety and Liveness”. In: *Distributed Comput.* 2.3, pp. 117–126. DOI: 10.1007/BF01782772. URL: https://doi.org/10.1007/BF01782772.

Alessandro Artale, Luca Geatti, et al. (2023a). “A Singly Exponential Transformation of LTL[X,F] into Pure Past LTL”. In: *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning, KR 2023, Rhodes, Greece. September 2-8, 2023.*

— (2023b). “Complexity of Safety and coSafety Fragments of Linear Temporal Logic”. In: *Proc. of the 36th AAAI Conf. on Artificial Intelligence*. AAAI Press.

Bibliography II

Alessandro Artale, Luca Geatti, et al. (2023c). "LTL over finite words can be exponentially more succinct than pure-past LTL, and vice versa". In: *Proceedings of the 30th International Symposium on Temporal Representation and Reasoning, TIME 2023, September 25-26, 2023, NCSR Demokritos, Athens, Greece.* Ed. by Florian Bruse Alexander Artikis and Luke Hunsberger. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Alessandro Artale, Roman Kontchakov, et al. (2014). "A Cookbook for Temporal Conceptual Data Modelling with Description Logics". In: *ACM Trans. Comput. Log.* 15.3, 25:1–25:50. DOI: 10.1145/2629565.

Fahiem Bacchus and Froduald Kabanza (1998). "Planning for Temporally Extended Goals". In: *Annals of Mathematics in Artificial Intelligence* 22.1-2, pp. 5–27.

Armin Biere et al. (2003). "Bounded model checking". In: *Adv. Comput.* 58, pp. 117–148. DOI: 10.1016/S0065-2458(03)58003-2. URL: https://doi.org/10.1016/S0065-2458(03)58003-2.

Aaron R Bradley (2011). "SAT-based model checking without unrolling". In: *International Workshop on Verification, Model Checking, and Abstract Interpretation*. Springer, pp. 70–87.

Ronen I. Brafman and Giuseppe De Giacomo (2019). "Planning for LTLf /LDLf Goals in Non-Markovian Fully Observable Nondeterministic Domains". In: *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. Ed. by Sarit Kraus. ijcai.org, pp. 1602–1608. DOI: 10.24963/ijcai.2019/222.

J. R. Buechi (1960). "On a decision method in restricted second-order arithmetics". In: *Proc. Internat. Congr. on Logic, Methodology and Philosophy of Science, 1960.*

Roberto Cavada et al. (2014). "The nuXmv symbolic model checker". In: *International Conference on Computer Aided Verification (CAV)*. Springer, pp. 334–342. DOI: 10.1007/s10009-006-0001-2.

Ashok K. Chandra, Dexter Kozen, and Larry J. Stockmeyer (1981). "Alternation". In: *J. ACM* 28.1, pp. 114–133. DOI: 10.1145/322234.322243. URL: https://doi.org/10.1145/322234.322243.

Edward Y. Chang, Zohar Manna, and Amir Pnueli (1992). "Characterization of Temporal Property Classes". In: *Proceedings of the 19th International Colloquium on Automata, Languages and Programming*. Ed. by Werner Kuich. Vol. 623. Lecture Notes in Computer Science. Springer, pp. 474–486. DOI: 10.1007/3-540-55719-9_97.

Alessandro Cimatti et al. (2021). "Extended bounded response LTL: a new safety fragment for efficient reactive synthesis". In: *Formal Methods in System Design*, 1–49 (published online on November 18, 2021, doi: 10.1007/s10703-021-00383–3).

— (2022). "A first-order logic characterisation of safety and co-safety languages". In: *Foundations of Software Science and Computation Structures - 25th International Conference, FOSSACS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings*. Ed. by Patricia Bouyer and Lutz Schröder. Vol. 13242. Lecture Notes in Computer Science. Springer, pp. 244–263. DOI: 10.1007/978-3-030-99253-8_13. URL: https://doi.org/10.1007/978-3-030-99253-8%5C_13.

Koen Claessen and Niklas Sörensson (2012). "A liveness checking algorithm that counts". In: *2012 Formal Methods in Computer-Aided Design (FMCAD)*. IEEE, pp. 52–59.

Edmund M Clarke et al. (2018). *Model checking*. MIT press.

Giuseppe De Giacomo, Marco Favorito, et al. (2020). "Imitation Learning over Heterogeneous Agents with Restraining Bolts". In: *Proceedings of the 13th International Conference on Automated Planning and Scheduling*. AAAI Press, pp. 517–521.

Giuseppe De Giacomo and Moshe Y. Vardi (2013). "Linear Temporal Logic and Linear Dynamic Logic on Finite Traces". In: *Proceedings of the 23rd International Joint Conference on Artificial Intelligence*. Ed. by Francesca Rossi. IJCAI/AAAI, pp. 854–860.

Giuseppe De Giacomo et al. (2021). “Pure-past linear temporal and dynamic logic on finite traces”. In: *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pp. 4959–4965.

D. Della Monica et al. (2017). “Bounded Timed Propositional Temporal Logic with Past Captures Timeline-based Planning with Bounded Constraints”. In: *Proc. of the 26th International Joint Conference on Artificial Intelligence*, pp. 1008–1014. DOI: 10.24963/ijcai.2017/140.

Calvin C Elgot (1961). “Decision problems of finite automata design and related arithmetics”. In: *Transactions of the American Mathematical Society* 98.1, pp. 21–51. DOI: 10.1090/S0002-9947-1961-0139530-9.

Kousha Etessami, Moshe Y. Vardi, and Thomas Wilke (2002). “First-Order Logic with Two Variables and Unary Temporal Logic”. In: *Inf. Comput.* 179.2, pp. 279–295. DOI: 10.1006/inco.2001.2953. URL: https://doi.org/10.1006/inco.2001.2953.

Bibliography VIII

Maria Fox and Derek Long (2003). "PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains". In: *J. Artif. Intell. Res.* 20, pp. 61–124. DOI: 10.1613/jair.1129.

Dov M. Gabbay et al. (1980). "On the Temporal Analysis of Fairness". In: *Conference Record of the Seventh Annual ACM Symposium on Principles of Programming Languages, Las Vegas, Nevada, USA, January 1980*. Ed. by Paul W. Abrahams, Richard J. Lipton, and Stephen R. Bourne. ACM Press, pp. 163–173. URL: https://doi.org/10.1145/567446.567462.

Lewis Hammond et al. (2021). "Multi-Agent Reinforcement Learning with Temporal Logic Specifications". In: *Proceedings of the 20th International Conference on Autonomous Agents and Multiagent Systems*. ACM, pp. 583–592. DOI: 10.5555/3463952.3464024.

Swen Jacobs et al. (2017). “The first reactive synthesis competition (SYNTCOMP 2014)”. In: *Int. J. Softw. Tools Technol. Transf.* 19.3, pp. 367–390. DOI: 10.1007/s10009-016-0416-3.

Johan Anthony Wilem Kamp (1968). “Tense logic and the theory of linear order”. In.

Orna Kupferman and Moshe Y Vardi (2001). “Model checking of safety properties”. In: *Formal Methods in System Design* 19.3, pp. 291–314. DOI: 10.1023/A:1011254632723.

Richard E Ladner (1977). “Application of model theoretic games to discrete linear orders and finite automata”. In: *Information and Control* 33.4, pp. 281–303. DOI: 10.1016/S0019-9958(77)90443-0.

Orna Lichtenstein, Amir Pnueli, and Lenore Zuck (1985). “The glory of the past”. In: *Workshop on Logic of Programs*. Springer, pp. 196–218. DOI: 10.1007/3-540-15648-8_16.

Oded Maler, Dejan Nickovic, and Amir Pnueli (2007). "On synthesizing controllers from bounded-response properties". In: *Proceedings of the International Conference on Computer Aided Verification*. Springer, pp. 95–107. DOI: 10.1023/A:1008734703554.

Oded Maler and Amir Pnueli (1990). "Tight bounds on the complexity of cascaded decomposition of automata". In: *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*. IEEE, pp. 672–682.

Zohar Manna and Amir Pnueli (1990). "A hierarchy of temporal properties (invited paper, 1989)". In: *Proceedings of the 9th annual ACM symposium on Principles of distributed computing*, pp. 377–410. DOI: 10.1145/93385.93442.

— (1995). *Temporal verification of reactive systems - safety*. Springer. ISBN: 978-0-387-94459-3.

Nicolas Markey (2003). "Temporal logic with past is exponentially more succinct". In: *Bull. EATCS* 79, pp. 122–128.

Robert McNaughton (1966). "Testing and generating infinite sequences by a finite automaton". In: *Information and control* 9.5, pp. 521–530. DOI: 10.1016/S0019-9958(66)80013-X.

Robert McNaughton and Seymour A Papert (1971). *Counter-Free Automata (MIT research monograph no. 65)*. The MIT Press.

Amir Pnueli (1977). "The temporal logic of programs". In: *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*. IEEE, pp. 46–57. DOI: 10.1109/SFCS.1977.32.

Amir Pnueli and Roni Rosner (1989). "On the Synthesis of a Reactive Module". In: *Proceedings of POPL'89*. ACM Press, pp. 179–190. DOI: 10.1145/75277.75293.

Arthur N Prior (2003). *Time and modality*. John Locke Lecture.

Sven Schewe (2009). "Büchi Complementation Made Tight". In: *26th International Symposium on Theoretical Aspects of Computer Science, STACS 2009, February 26-28, 2009, Freiburg, Germany, Proceedings*. Ed. by Susanne Albers and Jean-Yves Marion. Vol. 3. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany, pp. 661–672. DOI: 10.4230/LIPIcs.STACS.2009.1854. URL: https://doi.org/10.4230/LIPIcs.STACS.2009.1854.

Ina Schiering and Wolfgang Thomas (1996). "Counter-free automata, first-order logic, and star-free expressions extended by prefix oracles". In: *Developments in Language Theory, II (Magdeburg, 1995), Worl Sci. Publishing, River Edge, NJ*, pp. 166–175.

A Prasad Sistla (1994). "Safety, liveness and fairness in temporal logic". In: *Formal Aspects of Computing* 6.5, pp. 495–511. DOI: 10.1007/BF01211865.

Bibliography XIII

A Prasad Sistla and Edmund M Clarke (1985). "The complexity of propositional linear temporal logics". In: *Journal of the ACM (JACM)* 32.3, pp. 733–749. DOI: 10.1145/3828.3837.

Wolfgang Thomas (1979). "Star-free regular sets of ω-sequences". In: *Information and Control* 42.2, pp. 148–156. DOI: 10.1016/S0019-9958(79)90629-6.

— (1981). "A combinatorial approach to the theory of ω-automata". In: *Information and Control* 48.3, pp. 261–283. DOI: 10.1016/S0019-9958(81)90663-X.

Johan van Benthem et al. (2009). "Merging Frameworks for Interaction". In: *J. Philos. Log.* 38.5, pp. 491–526. DOI: 10.1007/s10992-008-9099-x.

Moshe Y Vardi (1996). "An automata-theoretic approach to linear temporal logic". In: *Logics for concurrency*. Springer, pp. 238–266.

Moshe Y Vardi and Pierre Wolper (1986). "An automata-theoretic approach to automatic program verification". In: *Proceedings of the First Symposium on Logic in Computer Science*. IEEE Computer Society, pp. 322–331.

Pierre Wolper (1983). "Temporal logic can be more expressive". In: *Information and control* 56.1-2, pp. 72–99. DOI: 10.1016/S0019-9958(83)80051-5.

Shufang Zhu et al. (2017). "A Symbolic Approach to Safety LTL Synthesis". In: *Proceedings of the 13th International Haifa Verification Conference*. Ed. by Ofer Strichman and Rachel Tzoref-Brill. Vol. 10629. Lecture Notes in Computer Science. Springer, pp. 147–162. DOI: 10.1007/978-3-319-70389-3_10.

Lenore Zuck (1986). "Past temporal logic". In: *Weizmann Institute of Science* 67.