Department of Mathematics, Computer Science and Physics, University of Udine

The Safety Fragment of Temporal Logics on Infinite Sequences

Lesson 10

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

May 5th, 2024

Outline

1. Background
   1. Regular and ω-regular languages
   2. The First- and Second-order Theory of One Successor
   3. Automata over finite and infinite words
   4. Linear Temporal Logic
2. The safety fragment of LTL and its theoretical features
   1. Definition of Safety and Cosafety
   2. Characterizations and Normal Forms
   3. Kupferman and Vardi's Classification

3. Recognizing safety
   1. Recognizing safety Büchi automata
   2. Recognizing safety formulas of LTL
   3. Construction of the automaton for the bad prefixes
4. Algorithms and Complexity
   1. Satisfiability
   2. Model Checking
   3. Reactive Synthesis
5. Succinctness and Pastification
   1. Succinctness of Safety Fragments
   2. Pastification Algorithms

MODEL CHECKING OF SAFETY PROPERTIES

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 reachable 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 $s_n \not\models \phi$.

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

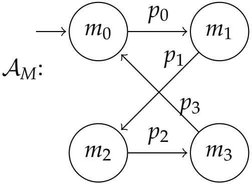
Let M be Kripke structure, s an initial state of M, and ϕ be an LTL formula such that $\mathcal{L}(\phi)$ is *safety*.

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*

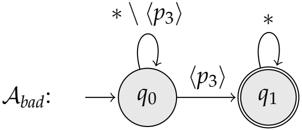
$$\mathsf{G}(\neg \mathit{final})$$

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

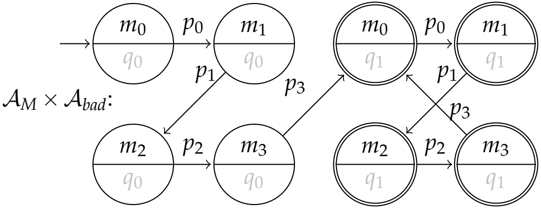
- Kripke Structure M:



- 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.



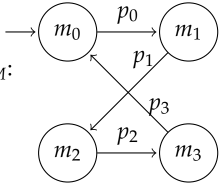
- We reduced the problem $M, s \models \mathsf{A}\,\mathsf{G}(p_0 \vee p_1 \vee p_2)$ to checking whether: (*reachability*)

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

Model Checking of Safety Properties
Example

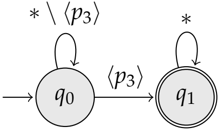
- Kripke Structure M:

$\mathcal{A}_M$:



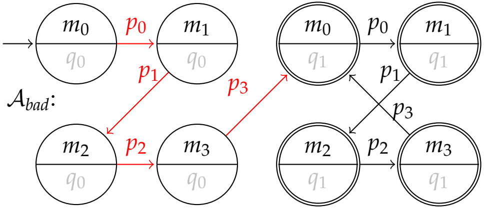
- Automaton for the bad prefixes of $\mathsf{G}(p_0 \vee p_1 \vee p_2)$:

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



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.

$\mathcal{A}_M \times \mathcal{A}_{bad}$:



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

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

- The property does not hold: counterexample trace

K-LIVENESS

- (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

Exploiting the KV's classification

for efficient model checking

- 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.
  - $\Rightarrow$ we want to build an automaton which recognizes only the *informative prefixes*.
  - $\Rightarrow$ only a single exponential blowup: $|\phi| = n \ \rightarrow \ |\mathcal{A}_{bad}| \in 2^{\mathcal{O}(n)}$

Theorem

For every LTL *formula* ϕ *such that* $\mathcal{L}(\phi)$ *is safety, there exists a* NFA $\mathcal{A}$ *that recognizes exactly the informative bad prefixes of* ϕ *and* $|\mathcal{A}| \in 2^{\mathcal{O}(|\phi|)}$.

Proof.

We will prove this result later. □

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 (Tight Automata)

Given a safety language $\mathcal{L}$, a NFA $\mathcal{A}$ is *tight for* $\mathcal{L}$ iff $\mathcal{L}(\mathcal{A}) = \mathtt{bad}(\mathcal{L})$.

Definition (Fine Automata)

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

"In practice, almost all the benefit that one obtain from a tight automaton can also be obtained from a fine automaton."

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 inconsistent or redundant specifications.

Theorem

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

- *if* ϕ *is intentionally safe, then* $\mathcal{A}$ *is tight for* ϕ*;*
- *if* ϕ *is accidentally safe, then* $\mathcal{A}$ *is fine for* ϕ*.*

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*, there exists a* NFA $\mathcal{A}$ *such that* $|\mathcal{A}| \in 2^{\mathcal{O}(|\phi|)}$ *and:*

- *if* ϕ *is intentionally safe, then* $\mathcal{A}$ *is tight for* ϕ*;*
- *if* ϕ *is accidentally safe, then* $\mathcal{A}$ *is fine for* ϕ*.*

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*

We know prove this result.

Theorem

For every LTL *formula* ϕ *such that* $\mathcal{L}(\phi)$ *is safety, there exists a* NFA $\mathcal{A}$ *that recognizes exactly the informative bad prefixes of* ϕ *and* $|\mathcal{A}| \in 2^{\mathcal{O}(|\phi|)}$.

ALTERNATING AUTOMATA

Alternating automata

Definition

An *alternating automaton* $\mathcal{A}$ is a tuple $\mathcal{A} = \langle \Sigma, Q, I, \delta, F \rangle$ such that:

- Σ is the alphabet
- Q is the set of states
- $I \subseteq Q$ is the set of initial states
- $\delta : Q \times \Sigma \to \mathbb{B}^+(Q)$
- $F \subseteq Q$ is the set of final states

where $\mathbb{B}^+(Q)$ is the set of *positive* Boolean formulas over the variables in Q.

Example

$\mathcal{A} := \langle \{a, b, c\}, \{q_0, q_A, \overline{q_A}, q_B, \overline{q_B}\}, \{q_0\}, \delta, \{q_A, q_B\} \rangle$ where $\delta : Q \times \Sigma \to \mathbb{B}^+(Q)$ is defined as follows:

- $\delta(q_0, *) = q_A \wedge q_B$
- $\delta(q_A, a) = q_A$
- $\delta(q_A, b) = \overline{q_A}$
- $\delta(q_A, c) = \overline{q_A}$
- $\delta(\overline{q_A}, a) = q_A$
- $\delta(\overline{q_A}, b) = \overline{q_A}$
- $\delta(\overline{q_A}, c) = \overline{q_A}$
- $\delta(q_B, a) = \overline{q_B}$
- $\delta(q_B, b) = q_B$
- $\delta(q_B, c) = \overline{q_B}$
- $\delta(\overline{q_B}, a) = \overline{q_B}$
- $\delta(\overline{q_B}, b) = q_B$
- $\delta(\overline{q_B}, c) = \overline{q_B}$

Definition

An *alternating automaton* $\mathcal{A}$ is a tuple $\mathcal{A} = \langle \Sigma, Q, I, \delta, F \rangle$ such that:

- Σ is the alphabet
- Q is the set of states
- $I \subseteq Q$ is the set of initial states
- $\delta : Q \times \Sigma \rightarrow \mathbb{B}^+(Q)$
- $F \subseteq Q$ is the set of final states

where $\mathbb{B}^+(Q)$ is the set of *positive* Boolean formulas over the variables in Q.

Definition (Run tree)

A run of an alternating automaton $\mathcal{A} = \langle \Sigma, Q, I, \delta, F \rangle$ over a word $\sigma := \langle \sigma_0, \sigma_1, \ldots \rangle$ is a *Q-labeled tree* such that:

- the root is labeled with a initial state in I
- given a node q such that $\delta(q, \sigma) = \Phi$, the set of all its children $\{q_1, \ldots, q_k\}$ must satisfy Φ.

Alternating automata

If $\Phi := \top$, then q does not need to have children: possible *finite* branches even when reading *infinite* words.

Definition (Run tree)

A run of an alternating automaton $\mathcal{A} = \langle \Sigma, Q, I, \delta, F \rangle$ over a word $\sigma := \langle \sigma_0, \sigma_1, \ldots \rangle$ is a *Q-labeled tree* such that:

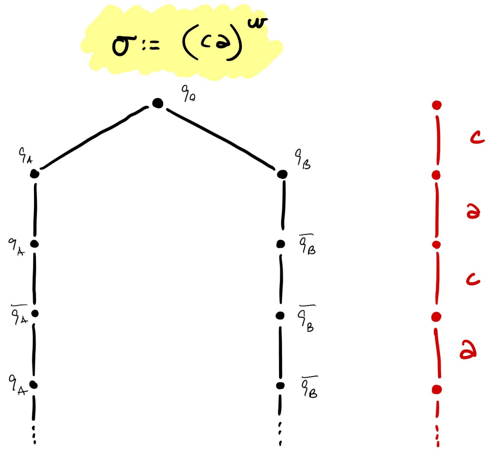
- the root is labeled with a initial state in I
- given a node q such that $\delta(q, \sigma) = \Phi$, the set of all its children $\{q_1, \ldots, q_k\}$ must satisfy Φ.

Alternating automata

Example

$\mathcal{A} := \langle \{a, b, c\}, \{q_0, q_A, \overline{q_A}, q_B, \overline{q_B}\}, \{q_0\}, \delta, \{q_A, q_B\} \rangle$ where $\delta : Q \times \Sigma \to \mathbb{B}^+(Q)$ is defined as follows:

- $\delta(q_0, *) = q_A \wedge q_B$
- $\delta(q_A, a) = q_A$
- $\delta(q_A, b) = \overline{q_A}$
- $\delta(q_A, c) = \overline{q_A}$
- $\delta(\overline{q_A}, a) = q_A$
- $\delta(\overline{q_A}, b) = \overline{q_A}$
- $\delta(\overline{q_A}, c) = \overline{q_A}$
- $\delta(q_B, a) = \overline{q_B}$
- $\delta(q_B, b) = q_B$
- $\delta(q_B, c) = \overline{q_B}$
- $\delta(\overline{q_B}, a) = \overline{q_B}$
- $\delta(\overline{q_B}, b) = q_B$
- $\delta(\overline{q_B}, c) = \overline{q_B}$



Alternating automata

In general, an alternating automaton can have *multiple* run trees over a given word.

Definition (Accepting run tree)

A AFA (*Alternating Finite Automata*) accepts a word σ iff there exists a run tree such that all its branches end in a final state.
A ABA (*Alternating Büchi Automata*) accepts a word σ iff there exists a run tree such that all infinite branches reaches a final state infinitely often.

- Note that in ABA we don't require nothing for branches of finite length.

Alternating automata

In general, an alternating automaton can have *multiple* run trees over a given word.

Definition (Accepting run tree)

A AFA (*Alternating Finite Automata*) accepts a word σ iff there exists a run tree such that all its branches end in a final state.

A ABA (*Alternating Büchi Automata*) accepts a word σ iff there exists a run tree such that all infinite branches reaches a final state infinitely often.

- Note that in ABA we don't require nothing for branches of finite length.
- An NFA is a AFA such that, for each $q \in Q$ and for each $a \in \Sigma$, the Boolean formula $\delta(q, a)$ contains only *disjunctions*.
- An NBA is a ABA such that, for each $q \in Q$ and for each $a \in \Sigma$, the Boolean formula $\delta(q, a)$ contains only *disjunctions*.

In general, an alternating automaton can have *multiple* run trees over a given word.

Definition (Accepting run tree)

A AFA (*Alternating Finite Automata*) accepts a word σ iff there exists a run tree such that all its branches end in a final state.

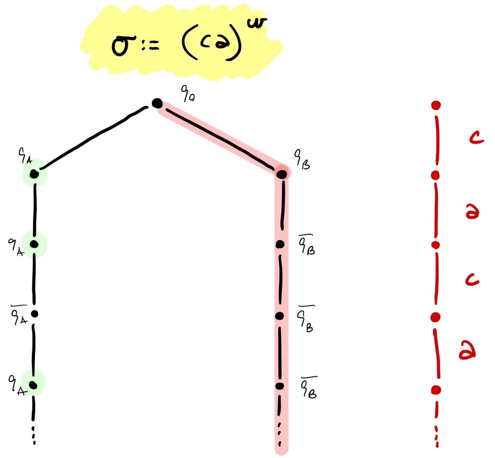
A ABA (*Alternating Büchi Automata*) accepts a word σ iff there exists a run tree such that all infinite branches reaches a final state infinitely often.

- Note that in ABA we don't require nothing for branches of finite length.
- An NFA is a AFA such that, for each $q \in Q$ and for each $a \in \Sigma$, the Boolean formula $\delta(q, a)$ contains only *disjunctions*.
- An NBA is a ABA such that, for each $q \in Q$ and for each $a \in \Sigma$, the Boolean formula $\delta(q, a)$ contains only *disjunctions*.
- If $\delta(q, a)$ contains only *conjunctions*, for each $q \in Q$ and for each $a \in \Sigma$, the automaton is said to be *universal*.

Example

$\mathcal{A} := \langle \{a, b, c\}, \{q_0, q_A, \overline{q_A}, q_B, \overline{q_B}\}, \{q_0\}, \delta, \{q_A, q_B\} \rangle$ where $\delta : Q \times \Sigma \to \mathbb{B}^+(Q)$ is defined as follows:

- $\delta(q_0, *) = q_A \wedge q_B$
- $\delta(q_A, a) = q_A$
- $\delta(q_A, b) = \overline{q_A}$
- $\delta(q_A, c) = \overline{q_A}$
- $\delta(\overline{q_A}, a) = q_A$
- $\delta(\overline{q_A}, b) = \overline{q_A}$
- $\delta(\overline{q_A}, c) = \overline{q_A}$
- $\delta(q_B, a) = \overline{q_B}$
- $\delta(q_B, b) = q_B$
- $\delta(q_B, c) = \overline{q_B}$
- $\delta(\overline{q_B}, a) = \overline{q_B}$
- $\delta(\overline{q_B}, b) = q_B$
- $\delta(\overline{q_B}, c) = \overline{q_B}$



Which is the ω-language of this alternating automaton?

Example

$\mathcal{A} := \langle \{a, b, c\}, \{q_0, q_A, \overline{q_A}, q_B, \overline{q_B}\}, \{q_0\}, \delta, \{q_A, q_B\} \rangle$ where $\delta : Q \times \Sigma \to \mathbb{B}^+(Q)$ is defined as follows:

- $\delta(q_0, *) = q_A \wedge q_B$
- $\delta(q_A, a) = q_A$
- $\delta(q_A, b) = \overline{q_A}$
- $\delta(q_A, c) = \overline{q_A}$
- $\delta(\overline{q_A}, a) = q_A$
- $\delta(\overline{q_A}, b) = \overline{q_A}$
- $\delta(\overline{q_A}, c) = \overline{q_A}$
- $\delta(q_B, a) = \overline{q_B}$
- $\delta(q_B, b) = q_B$
- $\delta(q_B, c) = \overline{q_B}$
- $\delta(\overline{q_B}, a) = \overline{q_B}$
- $\delta(\overline{q_B}, b) = q_B$
- $\delta(\overline{q_B}, c) = \overline{q_B}$

Alternating automata

Example

$\mathcal{A} := \langle \{a, b, c\}, \{q_0, q_A, \overline{q_A}, q_B, \overline{q_B}\}, \{q_0\}, \delta, \{q_A, q_B\} \rangle$ where $\delta : Q \times \Sigma \to \mathbb{B}^+(Q)$ is defined as follows:

- $\delta(q_0, *) = q_A \wedge q_B$
- $\delta(q_A, a) = q_A$
- $\delta(q_A, b) = \overline{q_A}$
- $\delta(q_A, c) = \overline{q_A}$
- $\delta(\overline{q_A}, a) = q_A$
- $\delta(\overline{q_A}, b) = \overline{q_A}$
- $\delta(\overline{q_A}, c) = \overline{q_A}$
- $\delta(q_B, a) = \overline{q_B}$
- $\delta(q_B, b) = q_B$
- $\delta(q_B, c) = \overline{q_B}$
- $\delta(\overline{q_B}, a) = \overline{q_B}$
- $\delta(\overline{q_B}, b) = q_B$
- $\delta(\overline{q_B}, c) = \overline{q_B}$

Which is the ω-language of this alternating automaton?

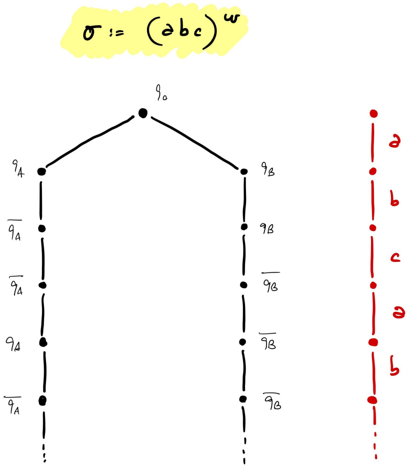
$$\mathcal{L}(\mathcal{A}) = \{\sigma \in \Sigma^\omega \mid \exists^\omega i \,.\, \sigma_i = a \wedge \exists^\omega i \,.\, \sigma_i = b\}$$

Alternating automata

Example

$\mathcal{A} := \langle \{a, b, c\}, \{q_0, q_A, \overline{q_A}, q_B, \overline{q_B}\}, \{q_0\}, \delta, \{q_A, q_B\} \rangle$ where $\delta : Q \times \Sigma \to \mathbb{B}^+(Q)$ is defined as follows:

- $\delta(q_0, *) = q_A \wedge q_B$
- $\delta(q_A, a) = q_A$
- $\delta(q_A, b) = \overline{q_A}$
- $\delta(q_A, c) = \overline{q_A}$
- $\delta(\overline{q_A}, a) = q_A$
- $\delta(\overline{q_A}, b) = \overline{q_A}$
- $\delta(\overline{q_A}, c) = \overline{q_A}$
- $\delta(q_B, a) = \overline{q_B}$
- $\delta(q_B, b) = q_B$
- $\delta(q_B, c) = \overline{q_B}$
- $\delta(\overline{q_B}, a) = \overline{q_B}$
- $\delta(\overline{q_B}, b) = q_B$
- $\delta(\overline{q_B}, c) = \overline{q_B}$

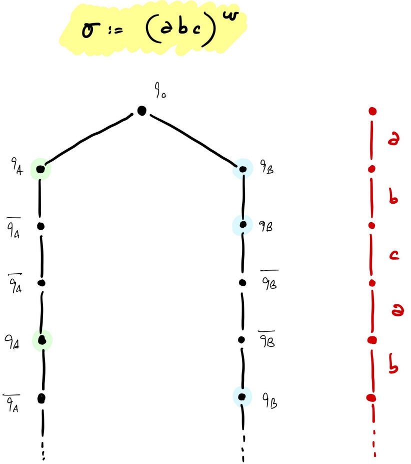


Alternating automata

Example

$\mathcal{A} := \langle \{a,b,c\}, \{q_0, q_A, \overline{q_A}, q_B, \overline{q_B}\}, \{q_0\}, \delta, \{q_A, q_B\}\rangle$ where $\delta : Q \times \Sigma \to \mathbb{B}^+(Q)$ is defined as follows:

- $\delta(q_0, *) = q_A \wedge q_B$
- $\delta(q_A, a) = q_A$
- $\delta(q_A, b) = \overline{q_A}$
- $\delta(q_A, c) = \overline{q_A}$
- $\delta(\overline{q_A}, a) = q_A$
- $\delta(\overline{q_A}, b) = \overline{q_A}$
- $\delta(\overline{q_A}, c) = \overline{q_A}$
- $\delta(q_B, a) = \overline{q_B}$
- $\delta(q_B, b) = q_B$
- $\delta(q_B, c) = \overline{q_B}$
- $\delta(\overline{q_B}, a) = \overline{q_B}$
- $\delta(\overline{q_B}, b) = q_B$
- $\delta(\overline{q_B}, c) = \overline{q_B}$



Definition

Given a LTL formula ϕ over $\mathcal{AP}$, we can effectively construct a ABA $\mathcal{A}_\phi = \langle \Sigma, Q, I, \delta, F \rangle$ with $\Sigma = 2^{\mathcal{AP}}$ such that $\mathcal{L}(\mathcal{A}_\phi) = \mathcal{L}(\phi)$ and $|\mathcal{A}_\phi| \in \mathcal{O}(|\phi|)$.

Proof.

We define the closure of ϕ, denoted with $\mathcal{C}(\phi)$, as the set of subformulas of ϕ (included ϕ itself) and their negations.
We define the set of states Q of $\mathcal{A}_\phi$ as $\mathcal{C}(\phi)$.

- $\Rightarrow$ states of $\mathcal{A}_\phi$ are subformulas of ϕ

Definition

Given a LTL formula ϕ over $\mathcal{AP}$, we can effectively construct a ABA $\mathcal{A}_\phi = \langle \Sigma, Q, I, \delta, F \rangle$ with $\Sigma = 2^{\mathcal{AP}}$ such that $\mathcal{L}(\mathcal{A}_\phi) = \mathcal{L}(\phi)$ and $|\mathcal{A}_\phi| \in \mathcal{O}(|\phi|)$.

Proof.

The ABA $\mathcal{A}_\phi = \langle \Sigma, Q, I, \delta, F \rangle$ is defined as follows:

- $\Sigma = 2^{\mathcal{AP}}$
- $Q := \mathcal{C}(\phi)$
- $I = \{\phi\}$
- $F = \{\psi := \neg(\alpha \mathsf{U} \beta) \mid \psi \in \mathcal{C}(\phi)\}$

Intuition on F:

- an infinite branch of a run tree reaching a state $\mathsf{G}\alpha$ ($\equiv \neg(\top \mathsf{U} \neg\alpha)$) correctly ensures the realization of α at every step.
- an infinite branch of a run tree reaching a state $\alpha \mathsf{U} \beta$ can postpone the realization of β forever.

Definition

Given a LTL formula ϕ over $\mathcal{AP}$, we can effectively construct a ABA $\mathcal{A}_\phi = \langle \Sigma, Q, I, \delta, F \rangle$ with $\Sigma = 2^{\mathcal{AP}}$ such that $\mathcal{L}(\mathcal{A}_\phi) = \mathcal{L}(\phi)$ and $|\mathcal{A}_\phi| \in \mathcal{O}(|\phi|)$.

Proof.

The ABA $\mathcal{A}_\phi = \langle \Sigma, Q, I, \delta, F \rangle$ is defined as follows:

- $\Sigma = 2^{\mathcal{AP}}$
- $Q := \mathcal{C}(\phi)$
- $I = \{\phi\}$
- $F = \{\psi := \neg(\alpha \mathsf{U} \beta) \mid \psi \in \mathcal{C}(\phi)\}$

Intuition on F:

- We should *not allow* infinite branches which visit infinitely often a state $\alpha \mathsf{U} \beta$.
- The branches starting from a state $\alpha \mathsf{U} \beta$ that will realize β in the future will eventually take a $\top$-transition and thus are finite branches.

Definition

Given a LTL formula ϕ over $\mathcal{AP}$, we can effectively construct a ABA $\mathcal{A}_\phi = \langle \Sigma, Q, I, \delta, F \rangle$ with $\Sigma = 2^{\mathcal{AP}}$ such that $\mathcal{L}(\mathcal{A}_\phi) = \mathcal{L}(\phi)$ and $|\mathcal{A}_\phi| \in \mathcal{O}(|\phi|)$.

Proof.

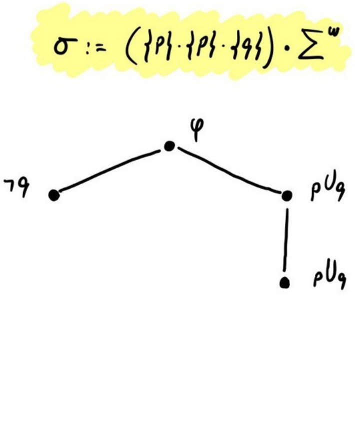
For each $q \in Q$ and for each $a \in \Sigma$, we define $\delta(q, a)$ as follows:

- $\delta(p, a) = \begin{cases} \top & \text{if } p \in a \\ \bot & \text{otherwise} \end{cases}$
- $\delta(\neg\psi, a) = \neg\delta(\psi, a)$
- $\delta(\psi_1 \wedge \psi_2, a) = \delta(\psi_1, a) \wedge \delta(\psi_2, a)$
- $\delta(\mathsf{X}\psi, a) = \psi$
- $\delta(\psi_1 \mathsf{U} \psi_2, a) =$ $\delta(\psi_2, a) \wedge (\delta(\psi_1, a) \vee \psi_1 \mathsf{U} \psi_2)$

Example

Let $\phi := \neg q \land \mathsf{X}\neg q \land p \,\mathsf{U}\, q$. We define the ABA $\mathcal{A}_\phi$ equivalent to ϕ as $\langle 2^{\mathcal{AP}}, Q, \{\phi\}, \delta, F\rangle$ where:

- $Q := \mathcal{C}(\phi) = \{\phi, \neg\phi, \neg q, q, \mathsf{X}\neg q, \neg\mathsf{X}\neg q, \ldots, p \,\mathsf{U}\, q, \neg(p \,\mathsf{U}\, q)\}$
- $F := \{\neg(p \,\mathsf{U}\, q)\}$

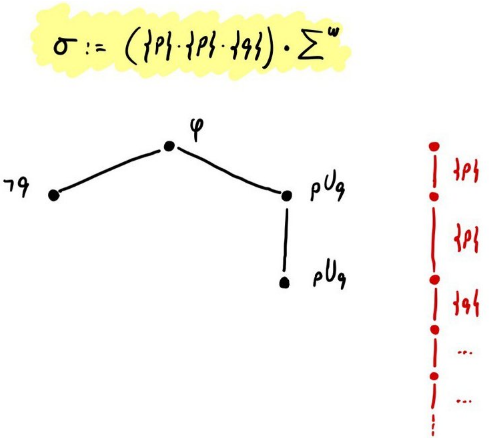


From LTL to Alternating Büchi Automata

Example

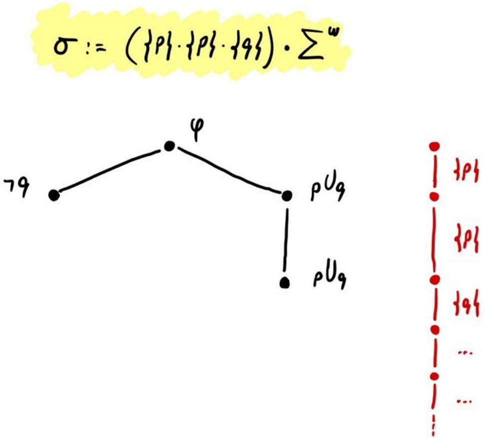
Let $\phi := \neg q \land \mathsf{X}\neg q \land p \,\mathsf{U}\, q$. We define the ABA $\mathcal{A}_\phi$ equivalent to ϕ as $\langle 2^{\mathcal{AP}}, Q, \{\phi\}, \delta, F\rangle$ where:

- $\delta(\neg q, a) = \begin{cases} \top & \text{if } q \notin a \\ \bot & \text{otherwise} \end{cases}$
- $\delta(p, a) = \begin{cases} \top & \text{if } p \in a \\ \bot & \text{otherwise} \end{cases}$
- $\delta(\phi, a) =$
  $\delta(\neg q, a) \land \delta(\mathsf{X}\neg q, a) \land \delta(p \,\mathsf{U}\, q, a)$
- $\delta(p \,\mathsf{U}\, q, a) = \delta(q, a) \lor (\delta(p, a) \land p \,\mathsf{U}\, q)$



There are no infinite branches in the run tree

- in ABA, we don't require nothing for finite branches
- $\Rightarrow$ the word is accepted

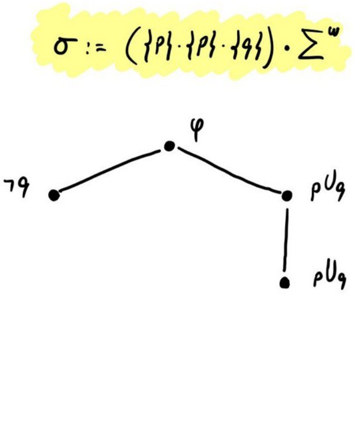


There are no infinite branches in the run tree

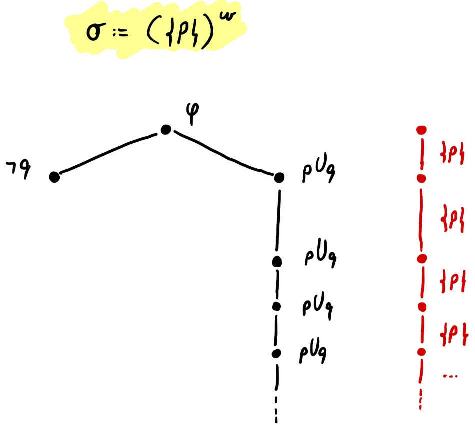
- in ABA, we don't require nothing for finite branches
- $\Rightarrow$ the word is accepted

Note the similarities between taking a transition of type $\delta(q, a) = \top$ and informative prefixes:

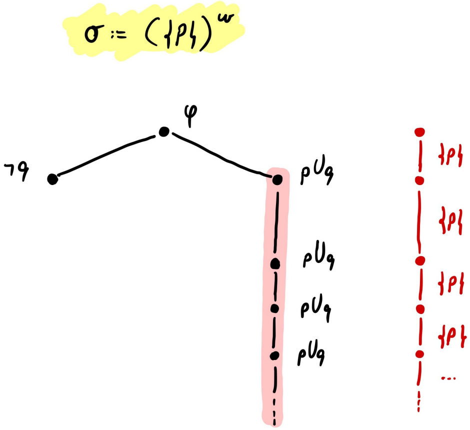
- $\Rightarrow$ a finite word σ induces a run tree of $\mathcal{A}_\phi$ that contains only branches reaching a transition of type $\delta(q, a) = \top$ **iff** σ is informative for ϕ



- any run tree for the ω-word $(\{p\})^\omega$ has an infinite branch going through state $p \mathsf{U} q$ infinitely many times
- $p \mathsf{U} q$ is not a final state
- $(\{p\})^\omega$ is rejected



- any run tree for the ω-word $(\{p\})^\omega$ has an infinite branch going through state $p \, \mathsf{U} \, q$ infinitely many times
- $p \, \mathsf{U} \, q$ is not a final state
- $(\{p\})^\omega$ is rejected



An AFA for the informative prefixes

Theorem

For every LTL *formula ϕ such that $\mathcal{L}(\phi)$ is safety, there exists a* AFA *$\mathcal{A}$ that recognizes exactly the informative bad prefixes of ϕ and $|\mathcal{A}| \in \mathcal{O}(|\phi|)$.*

Proof.

Let $\mathcal{A}_{\neg\phi} = \langle \Sigma, Q, I, \delta, F \rangle$ be the ABA for $\neg\phi$ (its size in linear in $|\phi|$).
We define $\mathcal{A}'_{\neg\phi}$ as $\langle \Sigma, Q, I, \delta, \varnothing \rangle$.

- the only way for $\mathcal{A}'_{\neg\phi}$ to accept a word having a run tree in which all branches take a transition of type $\delta(q, a) = \top$.
- this means that the word must be *informative* for $\neg\phi$.

The AFA for the informative bad prefixes of ϕ is obtained from $\mathcal{A}'_{\neg\phi}$ by setting the accepting condition to the case of finite words. □

Counterexample to minimality

Consider the formula:

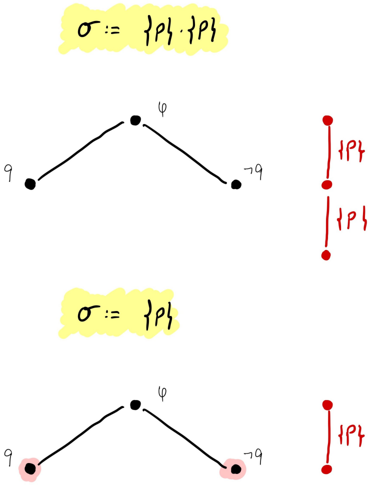
$$\phi \coloneqq \mathsf{G}(p \rightarrow (\mathsf{X}q \wedge \mathsf{X}\neg q))$$

which is equivalent to $G(p)$.
We have:

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

The AFA for the informative bad prefixes of ϕ is such that:

- it accepts the word $\{p\} \cdot \{p\}$, which is informative
- but it does not accept the *minimal* bad prefix $\{p\}$, which is not informative



Informative prefixes

Theorem

For every SafetyLTL *formula* ϕ*, there exists an* AFA $\mathcal{A}$ *such that* $|\mathcal{A}| \in \mathcal{O}(|\phi|)$ *and*

- *if* ϕ *is intentionally safe, then* $\mathcal{A}$ *is tight for* ϕ*;*
- *if* ϕ *is accidentally safe, then* $\mathcal{A}$ *is fine for* ϕ*;*

Proof.

Trivially follows from these three points.

- Let $\mathcal{A}$ be the automaton for the informative bad prefixes of ϕ.
- Every bad prefix of an intentionally safe formula is informative.
  - $\Rightarrow \mathcal{A}$ is tight for ϕ
- Every violation of an accidentally safe formula contains an informative prefix.
  - $\Rightarrow \mathcal{A}$ is fine for ϕ

$\square$

Theorem

For each AFA $\mathcal{A}$ *there exists an* NFA $\mathcal{A}'$ *such that* $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$ *and* $|\mathcal{A}'| \in 2^{\mathcal{O}(|\mathcal{A}|)}$.

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 every LTL *formula* ϕ *such that* $\mathcal{L}(\phi)$ *is safety, there exists a* NFA $\mathcal{A}$ *that recognizes exactly the informative bad prefixes of* ϕ *and* $|\mathcal{A}| \in 2^{\mathcal{O}(|\phi|)}$.

REFERENCES

Bibliography I

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

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.

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.

Bibliography II

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.