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

The Safety Fragment of Temporal Logics on Infinite Sequences

Lesson 1

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

April 4th, 2024

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

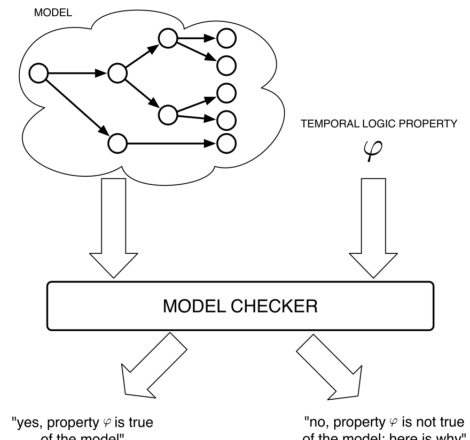
Edmund M Clarke et al. (2018). ***Model checking.*** **MIT press**

Important techniques in formal verification:

- consistency checking
- model checking
- reactive synthesis
- …

The course

Relationship between Linear Temporal Logic and Formal Verification

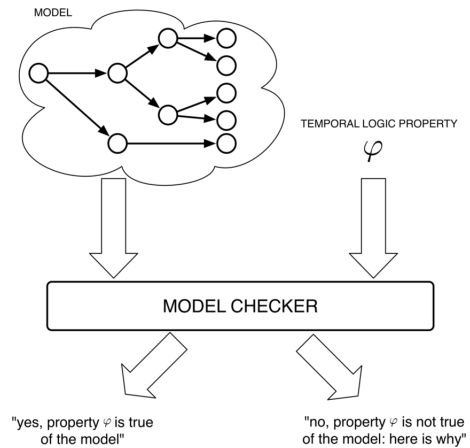


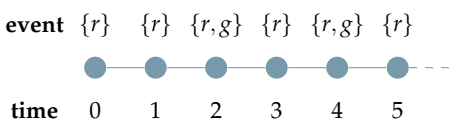
We are interested in *infinite linear time*: specification of *Reactive Systems*.

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**

The course

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.

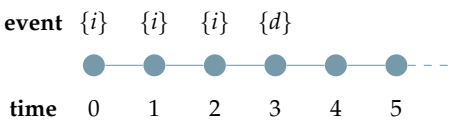
The Cosafety Fragment

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.

The course

Safety

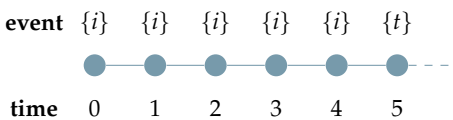
Property: *"The program never enters a deadlock"*



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

Cosafety

Property: *"The program terminates"*



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

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

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

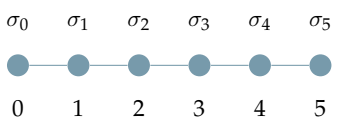
We fix a finite alphabet Σ.

Finite Words

- Modal interpretation:

$\sigma \in \Sigma^*$

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

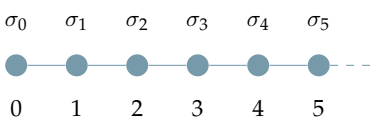


Infinite Words

- Modal interpretation:

$\sigma \in \Sigma^\omega$

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



Finite and Infinite Words

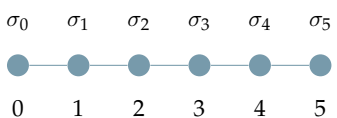
We fix a finite alphabet Σ.

Finite Words

- Modal interpretation:

$\sigma \in \Sigma^*$

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



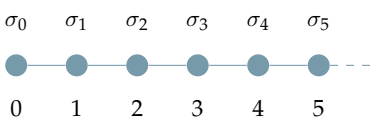
- *length* of σ: $|\sigma| = n + 1$
- word = synonym of finite word

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

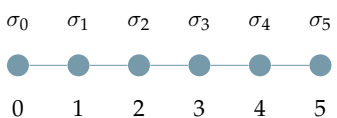
We fix a finite alphabet Σ.

Finite Words

- Modal interpretation:

$\sigma \in \Sigma^*$

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



Example

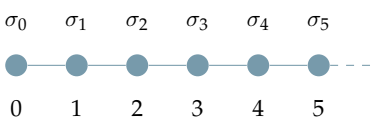
$\Sigma := \{a, b\}$ and $\sigma = ababab$

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*

Regular Languages

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 an ω-language that can be built using an ω-regular expression.
- We denote by 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 $\Sigma^* \coloneqq \overline{\varnothing}$.

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.

REFERENCES

Bibliography I

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

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