Traces Properties

Semantics and applications to verification

Xavier Rival

École Normale Supérieure

February 20th, 2026

Program of this lecture

Goal of verification

Prove that $[\![P]\!] \subseteq \mathcal{S}$
i.e., prove that for all behavior $b \in [\![P]\!]$, $b \in \mathcal{S}$
(i.e., all behaviors of P satisfy specification $\mathcal{S}$)
where $[\![P]\!]$ is the **program semantics** and $\mathcal{S}$ the **desired specification**

Last week, we studied a form of $[\![P]\!]$...

Today's lecture: we look back at program's properties

- **families of properties:**
  what properties can be considered "similar" ? in what sense ?
- **proof techniques:**
  how can those kinds of properties be established ?
- **specification of properties:**
  are there languages to describe properties ?

A high level overview

- In this lecture we look at **trace properties**
- A property is **a set of traces**, defining the **admissible** executions

Safety properties:

- **something (e.g., bad) will never happen**
- proof by invariance

Liveness properties:

- **something (e.g., good) will eventually happen**
- proof by variance

Beyond safety and liveness: hyperproperties (e.g., **security**...)

State properties

As usual, we consider $\mathcal{S} = (\mathbb{S}, \rightarrow, \mathbb{S}_\mathcal{I})$

First approach: properties as sets of states

- A property $\mathcal{P}$ is **a set of states** $\mathcal{P} \subseteq \mathbb{S}$
- $\mathcal{P}$ is satisfied if and only if all reachable states belong to $\mathcal{P}$, i.e., $[\![\mathcal{S}]\!]_\mathcal{R} \subseteq \mathcal{P}$ where $[\![\mathcal{S}]\!]_\mathcal{R} = \{s_n \in \mathbb{S} \mid \exists \langle s_0, \ldots, s_n \rangle \in [\![\mathcal{S}]\!]^*,\ s_0 \in \mathbb{S}_\mathcal{I}\}$

Examples:

- **Absence of runtime errors**:

$$\mathcal{P} = \mathbb{S} \setminus \{\Omega\} \quad \text{where } \Omega \text{ is the error state}$$

- **Non termination** (e.g., for an operating system):

$$\mathcal{P} = \{s \in \mathbb{S} \mid \exists s' \in \mathbb{S}, s \rightarrow s'\}$$

Trace properties

Second approach: properties as sets of traces

- A property $\mathcal{T}$ is **a set of traces** $\mathcal{T} \subseteq \mathbb{S}^{\infty}$
- $\mathcal{T}$ is satisfied if and only if all traces belong to $\mathcal{T}$, i.e., $[\![\mathcal{S}]\!]^{\infty} \subseteq \mathcal{T}$

Examples:

- Obviously, **state properties** are trace properties
- **Functional properties**:
  e.g., "program P takes one integer input x and returns its absolute value"
- **Termination**: $\mathcal{T} = \mathbb{S}^{*}$ (i.e., the system should have no infinite execution)

Monotonicity

Property 1

Let $\mathcal{P}_0, \mathcal{P}_1 \subseteq \mathbb{S}$ be two state properties, such that $\mathcal{P}_0 \subseteq \mathcal{P}_1$.
Then $\mathcal{P}_0$ **is stronger than** $\mathcal{P}_1$, i.e. if program $\mathcal{S}$ satisfies $\mathcal{P}_0$, then it also satisfies $\mathcal{P}_1$.

Property 2

Let $\mathcal{T}_0, \mathcal{T}_1 \subseteq \mathbb{S}$ be two trace properties, such that $\mathcal{T}_0 \subseteq \mathcal{T}_1$.
Then $\mathcal{T}_0$ **is stronger than** $\mathcal{T}_1$, i.e. if program $\mathcal{S}$ satisfies $\mathcal{T}_0$, then it also satisfies $\mathcal{T}_1$.

Property 3

Let $\mathcal{S}_0, \mathcal{S}_1$ two transition systems, such that $\mathcal{S}_1$ has more behaviors than $\mathcal{S}_0$ (i.e., $[\![\mathcal{S}_0]\!] \subseteq [\![\mathcal{S}_1]\!]$), and $\mathcal{P}$ be a (trace or state) property. Then, if $\mathcal{S}_1$ satisfies $\mathcal{P}$, so does $\mathcal{S}_0$.

Proofs: straightforward application of the definitions

Outline

Safety properties

Informal definition: safety properties

A safety property is a property which specifies that some (bad) **behavior defined by a finite, irrecoverable observation will never occur**, at any time

- **Absence of runtime errors** is a safety property ("bad thing": error)
- **State properties** is a safety property ("bad thing": reaching $\mathbb{S} \setminus \mathcal{P}$)
- **Non termination** is a safety property ("bad thing": reaching a blocking state)
- **"Not reaching state *b* after visiting state *a*"** is a safety property (and **not** a state property)
- **Termination** is **not** a safety property

We now intend to provide a **formal definition** of safety.

Towards a formal definition

How to refute a safety property ?

- We assume $\mathcal{S}$ does **not** satisfy safety property $\mathcal{P}$
- Thus, there exists a **counter-example trace** $\sigma = \langle s_0, \ldots, s_n, \ldots \rangle \in [\![\mathcal{S}]\!] \setminus \mathcal{P}$;
  at this point of our study, the trace may be finite or infinite...
- The intuitive definition says this trace **eventually exhibits some bad behavior**, that is **irrecoverable** at **observed at some given time**, thus the observation corresponds to some index i
- Therefore, trace $\sigma' = \langle s_0, \ldots, s_i \rangle$ violates $\mathcal{P}$, i.e. $\sigma' \notin \mathcal{P}$
- Due to the irrecoverability of the observation, the same goes for any trace with the same prefix
- We remark σ' **is finite**

A safety property that does not hold
can always be refuted with a finite, irrecoverable counter-example

A Few Operators on Traces

Prefix: We write $\sigma_{\lceil i}$ for the prefix of length i of trace σ:

$$\begin{array}{rcl} \langle s_0, \ldots, s_n \rangle_{\lceil 0} & = & \epsilon \\ \langle s_0, \ldots, s_n \rangle_{\lceil i+1} & = & \begin{cases} \langle s_0, \ldots, s_i \rangle & \text{if } i < n \\ \langle s_0, \ldots, s_n \rangle & \text{if } i \geq n \end{cases} \\ \langle s_0, \ldots \rangle_{\lceil i+1} & = & \langle s_0, \ldots, s_i \rangle \end{array}$$

Suffix (or tail):

$$\begin{array}{rcll} \sigma_{i\rceil} & = & \epsilon & \text{if } |\sigma| < i \\ (\langle s_0, \ldots, s_{i-1}, s_i, \ldots, s_n \rangle)_{i\rceil} & ::= & \langle s_i, \ldots, s_n \rangle & \text{if } |\sigma| \geq i \\ (\langle s_0, \ldots, s_{i-1}, s_i, \ldots \rangle)_{i\rceil} & ::= & \langle s_i, \ldots \rangle & \text{if } |\sigma| = \infty \end{array}$$

Upper closure operators

Definition: upper closure operator (uco)

We consider a preorder $(\mathcal{S}, \sqsubseteq)$. Function $\phi : \mathcal{S} \to \mathcal{S}$ is an **upper closure operator** iff:

- **monotone**
- **extensive:** $\forall x \in \mathcal{S},\ x \sqsubseteq \phi(x)$
- **idempotent:** $\forall x \in \mathcal{S},\ \phi(\phi(x)) = \phi(x)$

Dual: lower closure operator, monotone, reductive, idempotent

Examples:

- on real/decimal numbers, or on fraction:
  the ceiling operator, that returns the next integer is an upper-closure operator

Prefix closure

Definition: prefix closure

The prefix closure operator is defined by:

$$\begin{array}{rccc} \mathrm{PCl}: & \mathcal{P}(\mathbb{S}^{\infty}) & \longrightarrow & \mathcal{P}(\mathbb{S}^{*}) \\ & X & \longmapsto & \{\sigma_{\lceil i} \mid \sigma \in X,\, i \in \mathbb{N}\} \end{array}$$

Example: assuming $\mathcal{S} = \{\langle a, b, c\rangle, \langle a, c\rangle\}$ then,

$$\mathrm{PCl}(\mathcal{S}) = \{\epsilon, \langle a\rangle, \langle a, b\rangle, \langle a, b, c\rangle, \langle a, c\rangle\}$$

Properties:

- PCl is monotone
- PCl is idempotent, i.e., $\mathrm{PCl} \circ \mathrm{PCl}(X) = \mathrm{PCl}(X)$
- PCl is not extensive on $\mathcal{P}(\mathbb{S}^{\infty})$ (infinite traces do not appear anymore) its restriction to $\mathcal{P}(\mathbb{S}^{*})$

Limit

Definition: limit

The **limit operator** is defined by:

$$\begin{array}{rccl} \mathrm{Lim}: & \mathcal{P}(\mathbb{S}^{\propto}) & \longrightarrow & \mathcal{P}(\mathbb{S}^{\propto}) \\ & X & \longmapsto & X \cup \{\sigma \in \mathbb{S}^{\propto} \mid \forall i \in \mathbb{N},\ \sigma_{\lceil i} \in X\} \end{array}$$

Operator Lim **is an upper-closure operator**

Proof: exercise!

Example: assuming

$$\begin{array}{rcll} \mathcal{S} & = & \{ & \epsilon, \quad \langle a \rangle \\ & & & \langle a, b \rangle \quad \langle a, b, a \rangle \\ & & & \langle a, b, a, b \rangle \quad \langle a, b, a, b, a \rangle \quad \ldots \quad \} \end{array}$$

then,

$$\mathrm{Lim}(\mathcal{S}) = \mathcal{S} \uplus \{\langle a, b, a, b, a, b, \ldots \rangle\}$$

Towards a formal definition for safety

Operator Safe

Operator Safe is defined by $\text{Safe} = \text{Lim} \circ \text{PCl}$.

Operator Safe **saturates** a set of traces S with

- prefixes
- infinite traces all finite prefixes of which can be observed in S

Thus, if $\text{Safe}(S) = S$ and σ is a trace, to establish that σ **is not in** S, it is sufficient to discover a **finite prefix of** σ that cannot be observed in S.

- if σ is finite the result is clear (consider σ)
- otherwise, if all finite prefixes of σ are in S, then σ is in the limit, thus in S.

Safety: definition

A trace property $\mathcal{T}$ is a **safety** property if and only if $\text{Safe}(\mathcal{T}) = \mathcal{T}$

Safety properties: formal definition

An upper closure operator

Operator Safe is an upper closure operator over $\mathcal{P}(\mathbb{S}^\infty)$

Proof:

Safe **is monotone** since Lim and PCl are monotone

Safe **is extensive:**
indeed if $X \subseteq \mathbb{S}^\infty$ and $\sigma \in X$, we can show that $\sigma \in \text{Safe}(X)$:

- if σ is a finite trace, it is one of its prefixes, so $\sigma \in \text{PCl}(X) \subseteq \text{Lim}(\text{PCl}(X))$
- if σ is an infinite trace, all its prefixes belong to $\text{PCl}(X)$, so $\sigma \in \text{Lim}(\text{PCl}(X))$

Safety properties: formal definition

Proof (continued):

Safe **is idempotent:**

- as Safe is extensive and monotone $\text{Safe} \subseteq \text{Safe} \circ \text{Safe}$, so we simply need to show that $\text{Safe} \circ \text{Safe} \subseteq \text{Safe}$
- let $X \subseteq \mathbb{S}^\infty, \sigma \in \text{Safe}(\text{Safe}(X))$; then:

$$
\begin{array}{lll}
 & \sigma \in \text{Safe}(\text{Safe}(X)) & \\
\Rightarrow & \forall i,\ \sigma_{\lceil i} \in \text{PCl} \circ \text{Safe}(X) & \text{by def. of Lim} \\
\Rightarrow & \forall i,\ \exists \sigma', j,\ \sigma_{\lceil i} = \sigma'_{\lceil j} \wedge \sigma' \in \text{Safe}(X) & \text{by def. of PCl} \\
\Rightarrow & \forall i,\ \exists \sigma', j,\ \sigma_{\lceil i} = \sigma'_{\lceil j} \wedge \forall k,\ \sigma'_{\lceil k} \in \text{PCl}(X) & \\
 & \multicolumn{2}{l}{\text{by def. of Lim and case analysis over finiteness of } \sigma'} \\
\Rightarrow & \forall i,\ \exists \sigma', j,\ \sigma_{\lceil i} = \sigma'_{\lceil j} \wedge \sigma'_{\lceil j} \in \text{PCl}(X) & \text{if we take } k = j \\
\Rightarrow & \forall i,\ \sigma_{\lceil i} \in \text{PCl}(X) & \text{by simplification} \\
\Rightarrow & \sigma \in \text{Lim} \circ \text{PCl}(X) & \text{by def. of Lim} \\
\Rightarrow & \sigma \in \text{Safe}(X) &
\end{array}
$$

Safety properties: formal definition

Safety: definition

A trace property $\mathcal{T}$ is a **safety** property if and only if $\text{Safe}(\mathcal{T}) = \mathcal{T}$

Theorem

If $\mathcal{T}$ is a trace property, then $\text{Safe}(\mathcal{T})$ **is a safety property**

Proof:

Straightforward, by idempotence of Safe

Intuition:

- if $\mathcal{T}$ is a trace property (not necessarily a safety property), $\text{Safe}(\mathcal{T})$ **is the strongest safety property, that is weaker than** $\mathcal{T}$
- at this point, this observation is not so useful...
  but it will be soon!

Example

We assume that:

- $\mathbb{S} = \{a, b\}$
- $\mathcal{T}$ states that a **should not be visited after state** b **is visited**; elements of $\mathcal{T}$ are of the general form

$$\langle a, a, a, \ldots, a, b, b, b, b, \ldots \rangle \text{ or } \langle a, a, a, \ldots, a, a, \ldots \rangle$$

Then:

- $\mathrm{PCl}(\mathcal{T})$ elements are all finite traces which are of the above form (i.e., made of n occurrences of a followed by m occurrences of b, where n, m are positive integers)
- $\mathrm{Lim}(\mathrm{PCl}(\mathcal{T}))$ adds to this set the trace made made of infinitely many occurrences of a and the infinite traces made of n occurrences of a followed by infinitely many occurrences of b
- thus, $\mathrm{Safe}(\mathcal{T}) = \mathrm{Lim}(\mathrm{PCl}(\mathcal{T})) = \mathcal{T}$

Therefore $\mathcal{T}$ is indeed formally **a safety property**.

State properties are safety properties

Theorem

Any **state property** is also a **safety property**.

Proof:

Let us consider **state property** $\mathcal{P}$.
It is equivalent to **trace property** $\mathcal{T} = \mathcal{P}^{\propto}$:

$$\begin{aligned} \mathsf{Safe}(\mathcal{T}) &= \mathsf{Lim}(\mathsf{PCl}(\mathcal{P}^{\propto})) \\ &= \mathsf{Lim}(\mathcal{P}^{*}) \\ &= \mathcal{P}^{*} \cup \mathcal{P}^{\omega} \\ &= \mathcal{P}^{\propto} \\ &= \mathcal{T} \end{aligned}$$

Therefore $\mathcal{T}$ is indeed a safety property.

Intuition of the formal definition

Operator Safe **saturates** a set of traces S with

- prefixes
- infinite traces all finite prefixes of which can be observed in S

Thus, if $\text{Safe}(S) = S$ and σ is a trace, to establish that σ **is not in** S, it is sufficient to discover a **finite prefix of** σ that cannot be observed in S.

Alternatively, if all finite prefixes of σ belong to S or can observed as a prefix of another trace in S, by definition of the limit operator, σ **belongs to** S (even if it is infinite).

Thus, our definition **indeed captures properties that can be disproved with a finite counter-example**.

Outline

Proof by invariance

- We consider transition system $\mathcal{S} = (\mathbb{S}, \rightarrow, \mathbb{S}_{\mathcal{I}})$, and safety property $\mathcal{T}$. Finite traces semantics is the least fixpoint of F_*.
- We seek a way of **verifying that $\mathcal{S}$ satisfies $\mathcal{T}$**, i.e., that $[\![\mathcal{S}]\!]^{\propto} \subseteq \mathcal{T}$

Principle of invariance proofs

Let $\mathbb{I}$ be a set of finite traces; it is said to be an **invariant** if and only if:

- $\forall s \in \mathbb{S}_{\mathcal{I}},\ \langle s \rangle \in \mathbb{I}$
- $F_*(\mathbb{I}) \subseteq \mathbb{I}$

It is stronger than $\mathcal{T}$ if and only if $\mathbb{I} \subseteq \mathcal{T}$.

The **"by invariance"** proof method is based on finding an invariant that is stronger than $\mathcal{T}$.

Soundness

Theorem: soundness

The invariance proof method is **sound**: if we can find an invariant for $\mathcal{S}$, that is stronger than safety property $\mathcal{T}$, then $\mathcal{S}$ satisfies $\mathcal{T}$.

Proof:

We assume that $\mathbb{I}$ is an invariant of $\mathcal{S}$ and that it is stronger than $\mathcal{T}$, and we show that $\mathcal{S}$ satisfies $\mathcal{T}$:

- by induction over n, we can prove that $F_*^n(\{\langle s \rangle \mid s \in \mathbb{S}_{\mathcal{I}}\}) \subseteq F_*^n(\mathbb{I}) \subseteq \mathbb{I}$
- therefore $[\![\mathcal{S}]\!]^* \subseteq \mathbb{I}$
- thus, $\text{Safe}([\![\mathcal{S}]\!]^*) \subseteq \text{Safe}(\mathbb{I}) \subseteq \text{Safe}(\mathcal{T})$ since Safe is monotone
- we remark that $[\![\mathcal{S}]\!]^\propto = \text{Safe}([\![\mathcal{S}]\!]^*)$
- $\mathcal{T}$ is a safety property so $\text{Safe}(\mathcal{T}) = \mathcal{T}$
- we conclude $[\![\mathcal{S}]\!]^\propto \subseteq \mathcal{T}$, i.e., **$\mathcal{S}$ satisfies property $\mathcal{T}$**

Completeness

Theorem: completeness

The invariance proof method is **complete**: if $\mathcal{S}$ satisfies safety property $\mathcal{T}$, then we can find an invariant $\mathbb{I}$ for $\mathcal{S}$, that is stronger than $\mathcal{T}$.

Proof:

We assume that $[\![\mathcal{S}]\!]^*$ satisfies $\mathcal{T}$, and show that we can exhibit an invariant.

Then, $\mathbb{I} = [\![\mathcal{S}]\!]^*$ is an invariant of $\mathcal{S}$ by definition of $[\![.]\!]^*$, and it is stronger than $\mathcal{T}$.

Caveat:

- $[\![\mathcal{S}]\!]^\propto$ is most likely **not** a very easy to express invariant
- it is just a convenient completeness argument
- so, completeness does not mean the proof is easy !

Example

We consider the proof that the program below **computes the sum of the elements of an array**, i.e., when the exit is reached, $\mathtt{s} = \sum_{k=0}^{n-1} \mathtt{t}[k]$:

$\mathtt{i}, \mathtt{s}$ integer variables
$\mathtt{t}$ integer array of length n

ℓ_0 : $(\!|\textbf{true}|\!)$
$\mathtt{s} = 0;$
ℓ_1 : $(\!|\mathtt{s} = 0|\!)$
$\mathtt{i} = 0;$
ℓ_2 : $(\!|\mathtt{i} = 0 \wedge \mathtt{s} = 0|\!)$
while$(\mathtt{i} < n)\{$
ℓ_3 : $(\!|0 \leq \mathtt{i} < n \wedge \mathtt{s} = \sum_{k=0}^{\mathtt{i}-1} \mathtt{t}[k]|\!)$
$\mathtt{s} = \mathtt{s} + \mathtt{t}[\mathtt{i}];$
ℓ_4 : $(\!|0 \leq \mathtt{i} < n \wedge \mathtt{s} = \sum_{k=0}^{\mathtt{i}} \mathtt{t}[k]|\!)$
$\mathtt{i} = \mathtt{i} + 1;$
ℓ_5 : $(\!|1 \leq \mathtt{i} \leq n \wedge \mathtt{s} = \sum_{k=0}^{\mathtt{i}-1} \mathtt{t}[k]|\!)$
$\}$
ℓ_6 : $(\!|\mathtt{i} = n \wedge \mathtt{s} = \sum_{k=0}^{n-1} \mathtt{t}[k]|\!)$

Principle of the proof:

- for each program point ℓ, we have a **local invariant** $\mathbb{I}_\ell$ (denoted by a logical formula instead of a set of states in the figure)
- the global **invariant** $\mathbb{I}$ is defined by:

$$\mathbb{I} = \{\langle(\ell_0, m_0), \ldots, (\ell_n, m_n)\rangle \mid \forall n,\ m_n \in \mathbb{I}_{\ell_n}\}$$

Outline

Liveness properties

Informal definition: liveness properties

A liveness property is a property which specifies that some (good) behavior **will eventually occur**, and that this behavior **may still occur after any finite observation**.

- **Termination** is a liveness property
  "good behavior": reaching a blocking state (no more transition available)
- **"State *a* will eventually be reached by any execution"** is a liveness property
  "good behavior": reaching state *a*
- The **absence of runtime errors** is *not* a liveness property

As for safety properties, we intend to provide a **formal definition** of liveness.

Intuition towards a formal definition

How to refute a liveness property ?

- We consider liveness property $\mathcal{T}$ (think $\mathcal{T}$ **is termination**)
- We assume $\mathcal{S}$ does **not** satisfy liveness property $\mathcal{T}$
- Thus, there exists a **counter-example trace** $\sigma \in [\![\mathcal{S}]\!] \setminus \mathcal{T}$;
- The informal definition says:

  . . . **may still occur after any finite observation**

  thus, each finite trace σ' can be extended into a good trace
- **Conclusion of this discussion**: σ is necessarily **infinite**

To prove that a liveness property does not hold
we need to look for an infinite counter-example
i.e., no finite trace is a counter-example

Intuition towards a formal definition

To refute a liveness property, we need to look at infinite traces.

Example: if we run a program, and do not see it return...

- should we do `Ctrl+C` and conclude it does not terminate ?
- should we just wait a few more seconds minutes, hours, years ?

Towards a formal definition:

we expect any finite trace be the prefix of a trace in $\mathcal{T}$

... since finite executions cannot be used to disprove $\mathcal{T}$

Formal definition (incomplete)

$$\mathsf{PCl}(\mathcal{T}) = \mathbb{S}^*$$

Definition

Formal definition

Operator Live is defined by $\mathrm{Live}(\mathcal{T}) = \mathcal{T} \cup (\mathbb{S}^\infty \setminus \mathrm{Safe}(\mathcal{T}))$. Given property $\mathcal{T}$, the following three statements are equivalent:

(*i*) $\mathrm{Live}(\mathcal{T}) = \mathcal{T}$

(*ii*) $\mathrm{PCl}(\mathcal{T}) = \mathbb{S}^*$

(*iii*) $\mathrm{Lim} \circ \mathrm{PCl}(\mathcal{T}) = \mathbb{S}^\infty$

When they are satisfied, $\mathcal{T}$ is said to be a **liveness property**

Example: **termination**

- The property is $\mathcal{T} = \mathbb{S}^*$
  (i.e., there should be no infinite execution)
- Clearly, it satisfies (*ii*): $\mathrm{PCl}(\mathcal{T}) = \mathbb{S}^*$
  thus termination indeed satisfies this definition

Proof of equivalence

Proof of equivalence:

(*i*) implies (*ii*):

We assume that $\mathrm{Live}(\mathcal{T}) = \mathcal{T}$, i.e., $\mathcal{T} \cup (\mathbb{S}^\infty \setminus \mathrm{Safe}(\mathcal{T})) = \mathcal{T}$
therefore, $\mathbb{S}^\infty \setminus \mathrm{Safe}(\mathcal{T}) \subseteq \mathcal{T}$.
Let $\sigma \in \mathbb{S}^*$, and let us show that $\sigma \in \mathrm{PCl}(\mathcal{T})$; clearly, $\sigma \in \mathbb{S}^\infty$, thus:

- either $\sigma \in \mathrm{Safe}(\mathcal{T}) = \mathrm{Lim}(\mathrm{PCl}(\mathcal{T}))$, so all its prefixes are in $\mathrm{PCl}(\mathcal{T})$ and $\sigma \in \mathrm{PCl}(\mathcal{T})$
- or $\sigma \in \mathcal{T}$, which implies that $\sigma \in \mathrm{PCl}(\mathcal{T})$

(*ii*) implies (*iii*):

If $\mathrm{PCl}(\mathcal{T}) = \mathbb{S}^*$, then $\mathrm{Lim} \circ \mathrm{PCl}(\mathcal{T}) = \mathbb{S}^\infty$

(*iii*) implies (*i*):

If $\mathrm{Lim} \circ \mathrm{PCl}(\mathcal{T}) = \mathbb{S}^\infty$, then
$\mathrm{Live}(\mathcal{T}) = \mathcal{T} \cup (\mathbb{S}^\infty \setminus (\mathrm{Lim} \circ \mathrm{PCl}(\mathcal{T}))) = \mathcal{T} \cup (\mathbb{S}^\infty \setminus \mathbb{S}^\infty) = \mathcal{T}$

Example

We assume that:

- $\mathbb{S} = \{a, b, c\}$
- $\mathcal{T}$ states that b **should eventually be visited, after** a **has been visited**; elements of $\mathcal{T}$ can be described by

$$\mathcal{T} = \mathrm{PCl}(\mathbb{S}^* \cdot a \cdot \mathbb{S}^* \cdot b \cdot \mathbb{S}^\infty)$$

Then $\mathcal{T}$ **is a liveness property:**

- let $\sigma \in \mathbb{S}^*$; then $\sigma \cdot a \cdot b \in \mathcal{T}$, so $\sigma \in \mathrm{PCl}(\mathcal{T})$
- thus, $\mathrm{PCl}(\mathcal{T}) = \mathbb{S}^*$

A property of Live

Theorem

If $\mathcal{T}$ is a trace property, then $\text{Live}(\mathcal{T})$ **is a liveness property** (i.e., operator Live is **idempotent**).

Proof: we show that $\text{PCl} \circ \text{Live}(\mathcal{T}) = \mathbb{S}^*$, by considering $\sigma \in \mathbb{S}^*$ and proving that $\sigma \in \text{PCl} \circ \text{Live}(\mathcal{T})$; we first note that:

$$\begin{aligned} \text{PCl} \circ \text{Live}(\mathcal{T}) &= \text{PCl}(\mathcal{T}) \cup \text{PCl}(\mathbb{S}^\infty \setminus \text{Safe}(\mathcal{T})) \\ &= \text{PCl}(\mathcal{T}) \cup \text{PCl}(\mathbb{S}^\infty \setminus \text{Lim} \circ \text{PCl}(\mathcal{T})) \end{aligned}$$

- if $\sigma \in \text{PCl}(\mathcal{T})$, this is obvious.
- if $\sigma \notin \text{PCl}(\mathcal{T})$, then:
  - $\sigma \notin \text{Lim} \circ \text{PCl}(\mathcal{T})$ by definition of the limit
  - thus, $\sigma \in \mathbb{S}^\infty \setminus \text{Lim} \circ \text{PCl}(\mathcal{T})$
  - $\sigma \in \text{PCl}(\mathbb{S}^\infty \setminus \text{Lim} \circ \text{PCl}(\mathcal{T}))$ as PCl is extensive when applied to sets of finite traces, which proves the above result

Outline

Termination proof with ranking function

- We consider only **termination**
- We consider transition system $\mathcal{S} = (\mathbb{S}, \rightarrow, \mathbb{S}_{\mathcal{I}})$, and liveness property $\mathcal{T}$
- We seek a way of **verifying that $\mathcal{S}$ satisfies termination**, i.e., that $[\![\mathcal{S}]\!]^{\infty} \subseteq \mathbb{S}^*$

Definition: ranking function

A **ranking function** is a function $\phi : \mathbb{S} \rightarrow E$ where:

- $(E, \sqsubseteq)$ is a **well-founded ordering**
- $\forall s_0, s_1 \in \mathbb{S},\ s_0 \rightarrow s_1 \Longrightarrow \phi(s_1) \sqsubset \phi(s_0)$

Theorem

If $\mathcal{S}$ has a ranking function ϕ, it satisfies termination.

Example

We consider the termination of the array sum program:

$\mathtt{i}, \mathtt{s}$ integer variables
$\mathtt{t}$ integer array of length n

ℓ_0 : $\mathtt{s} = 0;$
ℓ_1 : $\mathtt{i} = 0;$
ℓ_2 : $\mathtt{while}(\mathtt{i} < n)\{$
ℓ_3 : $\quad \mathtt{s} = \mathtt{s} + \mathtt{t}[\mathtt{i}];$
ℓ_4 : $\quad \mathtt{i} = \mathtt{i} + 1;$
ℓ_5 : $\}$
ℓ_6 : $\ldots$

Ranking function:

$$
\begin{array}{lrcl}
\phi: & \mathbb{S} & \longrightarrow & \mathbb{N} \\
& (\ell_0, m) & \longmapsto & 3 \cdot n + 6 \\
& (\ell_1, m) & \longmapsto & 3 \cdot n + 5 \\
& (\ell_2, m) & \longmapsto & 3 \cdot n + 4 \\
& (\ell_3, m) & \longmapsto & 3 \cdot (n - m(\mathtt{i})) + 3 \\
& (\ell_4, m) & \longmapsto & 3 \cdot (n - m(\mathtt{i})) + 2 \\
& (\ell_5, m) & \longmapsto & 3 \cdot (n - m(\mathtt{i})) + 4 \\
& (\ell_6, m) & \longmapsto & 0
\end{array}
$$

Proof by variance

- We consider transition system $\mathcal{S} = (\mathbb{S}, \rightarrow, \mathbb{S}_{\mathcal{I}})$, and liveness property $\mathcal{T}$; infinite traces semantics is the greatest fixpoint of F_ω.
- We seek a way of **verifying that $\mathcal{S}$ satisfies $\mathcal{T}$**, i.e., that $[\![\mathcal{S}]\!]^\infty \subseteq \mathcal{T}$

Principle of variance proofs

Let $(\mathbb{I}_n)_{n\in\mathbb{N}}$, $\mathbb{I}_\omega$ be elements of $\mathbb{S}^\infty$; these are said to form a variance proof of $\mathcal{T}$ if and only if:

- $\mathbb{S}^\omega \subseteq \mathbb{I}_0$
- for all $k \in \{1, 2, \ldots, \omega\}$, $\forall s \in \mathbb{S}$, $\langle s \rangle \in \mathbb{I}_k$
- for all $k \in \{1, 2, \ldots, \omega\}$, there exists $l < k$ such that $F_\omega(\mathbb{I}_l) \subseteq \mathbb{I}_k$
- $\mathbb{I}_\omega \subseteq \mathcal{T}$

Proofs of soundness and completeness: exercise, similar to the previous proof but using the definition of $[\![\mathcal{S}]\!]^\infty$ instead

Outline

The decomposition theorem

Theorem

Let $\mathcal{T} \subseteq \mathbb{S}^{\infty}$; it can be decomposed into the **conjunction** of **safety property** $\text{Safe}(\mathcal{T})$ and **liveness property** $\text{Live}(\mathcal{T})$:

$$\mathcal{T} = \text{Safe}(\mathcal{T}) \cap \text{Live}(\mathcal{T})$$

- **Reading**: **Recognizing Safety and Liveness**.
  Bowen Alpern and **Fred B. Schneider**.
  In Distributed Computing, Springer, 1987.
- **Consequence of this result:**
  the proof of any trace property can be decomposed into
  - a proof of safety
  - a proof of liveness

Proof

- **Safety part:**
  Safe is idempotent, so $\mathsf{Safe}(\mathcal{T})$ is a safety property.
- **Liveness part:**
  Live is idempotent, so $\mathsf{Live}(\mathcal{T})$ is a liveness property.
- **Decomposition:**

$$\begin{aligned}
\mathsf{Safe}(\mathcal{T}) \cap \mathsf{Live}(\mathcal{T}) &= \mathsf{Safe}(\mathcal{T}) \cap (\mathcal{T} \cup \mathbb{S}^{\propto} \setminus \mathsf{Safe}(\mathcal{T})) \\
&= \mathsf{Safe}(\mathcal{T}) \cap \mathcal{T} \\
&\qquad \cup \mathsf{Safe}(\mathcal{T}) \cap (\mathbb{S}^{\propto} \setminus \mathsf{Safe}(\mathcal{T})) \\
&= \mathcal{T} \cup \emptyset \\
&= \mathcal{T}
\end{aligned}$$

Example: verification of total correctness

$\mathtt{i}, \mathtt{s}$ integer variables
$\mathtt{t}$ integer array of length n

ℓ_0 : $\mathtt{s} = 0;$
ℓ_1 : $\mathtt{i} = 0;$
ℓ_2 : $\mathtt{while}(\mathtt{i} < n)\{$
ℓ_3 : $\quad \mathtt{s} = \mathtt{s} + \mathtt{t}[\mathtt{i}];$
ℓ_4 : $\quad \mathtt{i} = \mathtt{i} + 1;$
ℓ_5 : $\}$
ℓ_6 : $\ldots$

Property to prove: total correctness

1. the program **terminates**
2. and **does not crash**
3. and **computes the sum of the elements in the array**

Application of the decomposition principle

Conjunction of two proofs:

1. Proved with a **ranking function**
2. Proved with **local invariants**
3. Also proved with **local invariants**

Safety and Liveness Decomposition Example

We consider a very simple **greatest common divider** code function:

```
ℓ0 : int f(int a, int b){
ℓ1 :     while(a > 0){
ℓ2 :         int d = b/a;
ℓ3 :         int r = b − a ∗ d;
ℓ4 :         b = a;
ℓ5 :         a = r;
ℓ6 :     }
ℓ7 :     return b;
ℓ8 : }
```

Specification

When applied to positive integers, function `f` should always return their GCD.

Safety and Liveness Decomposition Example

We consider a very simple **greatest common divider** code function:

```
ℓ0 : int f(int a, int b){
ℓ1 :     while(a > 0){
ℓ2 :         int d = b/a;
ℓ3 :         int r = b − a ∗ d;
ℓ4 :         b = a;
ℓ5 :         a = r;
ℓ6 :     }
ℓ7 :     return b;
ℓ8 : }
```

Specification

When applied to positive integers, function `f` should always return their GCD.

Safety part

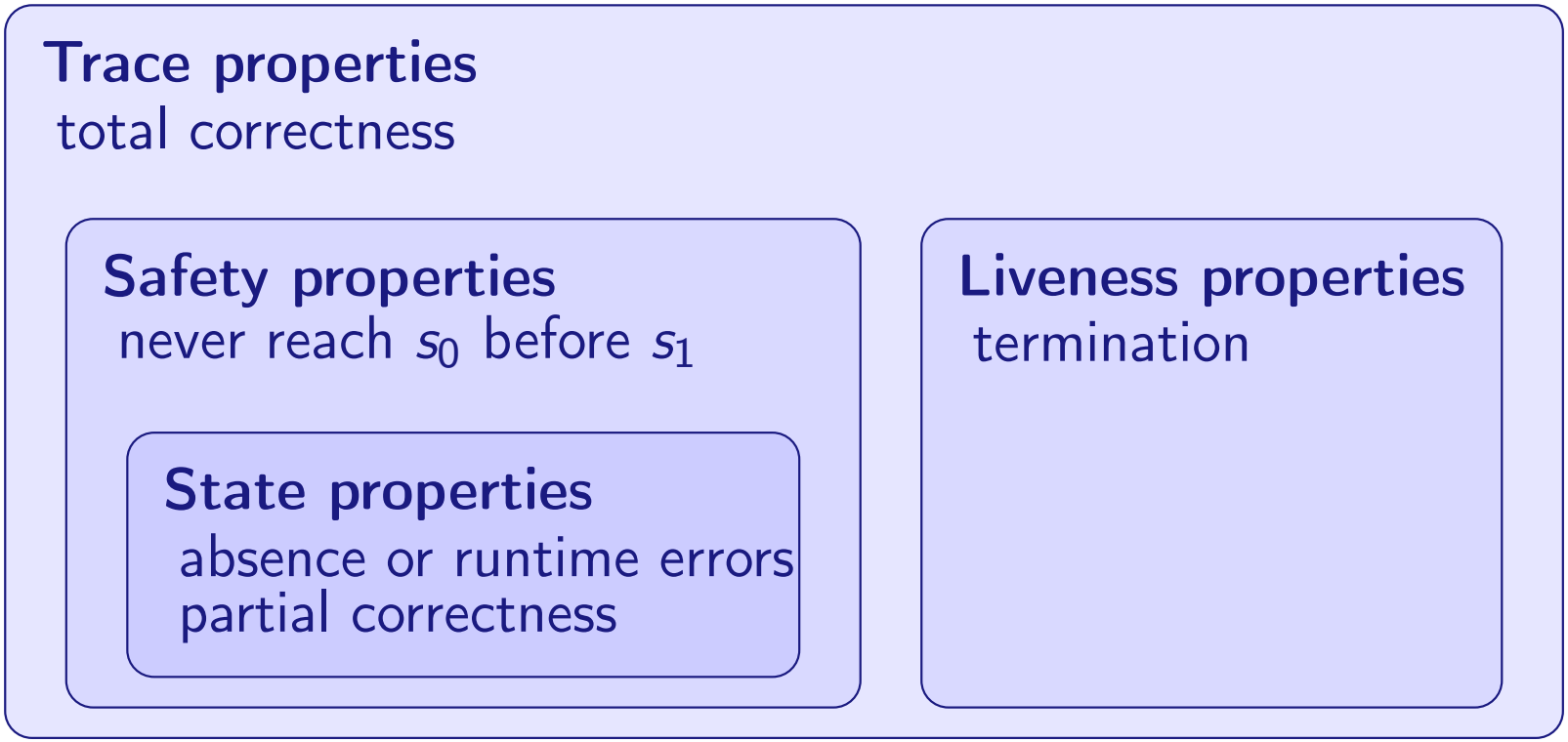
For all trace starting with positive inputs, a **conjunction of two properties:**

- **no runtime errors**
- **the value of `b` is the GCD**

Liveness part

Termination, on all traces starting with positive inputs

The Zoo of semantic properties: current status



- **Safety:** if wrong, can be refuted with a **finite trace**
  proof done by **invariance**
- **Liveness:** if wrong, has to be refuted with an **infinite trace**
  proof done by **variance**

Outline

Notion of specification language

- Ultimately, we would like to **verify or compute** properties
- So far, we simply describe properties with **sets of executions** or worse, with English / French / ... statements
- Ideally, we would prefer to use a **mathematical language** for that
  - to **gain in concision**, **avoid ambiguity**
  - to **define sets of properties to consider**, fix **the form of inputs for verification tools...**

Definition: specification language

A **specification language** is a set of terms $\mathbb{L}$ with an **interpretation function** (or **semantics**)

$$[\![.]\!] : \mathbb{L} \longrightarrow \mathcal{P}(\mathbb{S}^\infty) \qquad (\text{resp., } \mathcal{P}(\mathbb{S}))$$

- We are now going to consider specification languages **for states**, **for traces**...

A State specification language

A first **example** of a (simple) specification language:

A state specification language

- **Syntax:** we let terms of $\mathbb{L}_{\mathbb{S}}$ be defined by:

$$p \in \mathbb{L}_{\mathbb{S}} ::= @\ell \mid \mathtt{x} < \mathtt{x}' \mid \mathtt{x} < n \mid \neg p' \mid p' \wedge p'' \mid \Omega$$

- **Semantics:** $[\![p]\!]_{\mathtt{s}} \subseteq \mathbb{S}_\Omega$ is defined by

$$\begin{array}{rcl} [\![@\ell]\!]_{\mathtt{s}} & = & \{\ell\} \times \mathbb{M} \\ [\![\mathtt{x} \leq \mathtt{x}']\!]_{\mathtt{s}} & = & \{(\ell, m) \in \mathbb{S} \mid m(\mathtt{x}) \leq m(\mathtt{x}')\} \\ [\![\mathtt{x} \leq n]\!]_{\mathtt{s}} & = & \{(\ell, m) \in \mathbb{S} \mid m(\mathtt{x}) \leq n\} \\ [\![\neg p]\!]_{\mathtt{s}} & = & \mathbb{S}_\Omega \setminus [\![p]\!]_{\mathtt{s}} \\ [\![p \wedge p']\!]_{\mathtt{s}} & = & [\![p]\!]_{\mathtt{s}} \cap [\![p']\!]_{\mathtt{s}} \\ [\![\Omega]\!]_{\mathtt{s}} & = & \{\Omega\} \end{array}$$

Exercise: add $=$, $\vee$, $\Longrightarrow$...

State properties: examples

Unreachability of control state ℓ_0:

- specification: $\vee\neg@\ell_0$
- property: $[\![\neg@\ell_0]\!]_{\mathbf{s}} = \mathbb{S}_\Omega \setminus \{(\ell_0, m) \mid m \in \mathbb{M}\}$

Absence of runtime errors:

- specification: $\neg\Omega$
- property: $[\![\neg\Omega]\!]_{\mathbf{s}} = \mathbb{S}_\Omega \setminus \{\Omega\} = \mathbb{S}$

Intermittent invariant:

- principle: attach a local invariant to each control state
- example:

$$
\begin{array}{lll}
\ell_0: & \mathsf{if}(\mathtt{x} \geq 0)\{ & \\
\ell_1: & \quad \mathtt{y} = \mathtt{x}; & @\ell_1 \Longrightarrow \mathtt{x} \geq 0 \\
\ell_2: & \}\mathsf{else}\{ & \wedge \quad @\ell_2 \Longrightarrow \mathtt{x} \geq 0 \wedge \mathtt{y} \geq 0 \\
\ell_3: & \quad \mathtt{y} = -\mathtt{x}; & \wedge \quad @\ell_3 \Longrightarrow \mathtt{x} < 0 \\
\ell_4: & \} & \wedge \quad @\ell_4 \Longrightarrow \mathtt{x} < 0 \wedge \mathtt{y} > 0 \\
\ell_5: & \ldots & \wedge \quad @\ell_5 \Longrightarrow \mathtt{y} \geq 0
\end{array}
$$

Propositional temporal logic: syntax

We now consider the **specification of trace properties**

- **Temporal logic:** specification of properties in terms of events that occur at distinct times in the execution (hence, the name "temporal")
- There are **many** instances of temporal logic
- We study a simple one: **Pnueli's Propositional Temporal Logic**

Definition: syntax of PTL (Propositional Temporal Logic)

Properties over traces are defined as terms of the form

$$
\begin{array}{lcll}
t(\in \mathbb{L}_{\mathsf{PTL}}) & ::= & p & \text{state property, i.e., } p \in \mathbb{L}_{\mathbb{S}} \\
& | & t' \vee t'' & \text{disjunction} \\
& | & \neg t' & \text{negation} \\
& | & \bigcirc\, t' & \text{"next"} \\
& | & t' \,\mathfrak{U}\, t'' & \text{"until", i.e., } t' \text{ until } t''
\end{array}
$$

Propositional temporal logic: semantics

The semantics of a temporal property is a set of traces, and it is defined by induction over the syntax:

Semantics of Propositional Temporal Logic formulae

$$
\begin{aligned}
[\![p]\!]_{\mathtt{t}} &= \{s \cdot \sigma \mid s \in [\![p]\!]_{\mathtt{s}} \wedge \sigma \in \mathbb{S}^{\infty}\} \\
[\![t_0 \vee t_1]\!]_{\mathtt{t}} &= [\![t_0]\!]_{\mathtt{t}} \cup [\![t_1]\!]_{\mathtt{t}} \\
[\![\neg t_0]\!]_{\mathtt{t}} &= \mathbb{S}^{\infty} \setminus [\![t_0]\!]_{\mathtt{t}} \\
[\![\bigcirc\, t_0]\!]_{\mathtt{t}} &= \{s \cdot \sigma \mid s \in \mathbb{S} \wedge \sigma \in [\![t_0]\!]_{\mathtt{t}}\} \\
[\![t_0\, \mathfrak{U}\, t_1]\!]_{\mathtt{t}} &= \{\sigma \in \mathbb{S}^{\infty} \mid \exists n \in \mathbb{N},\ \forall i < n,\ \sigma_{i]} \in [\![t_0]\!]_{\mathtt{t}} \wedge \sigma_{n]} \in [\![t_1]\!]_{\mathtt{t}}\}
\end{aligned}
$$

Temporal logic operators as syntactic sugar

Many useful operators can be added:

- **Boolean constants:**

$$\textbf{true} ::= (\mathtt{x} < 0) \vee \neg(\mathtt{x} < 0)$$
$$\textbf{false} ::= \neg\textbf{true}$$

- **Sometime:**

$$\Diamond\, t ::= \textbf{true}\, \mathfrak{U}\, t$$

  intuition: there exists a rank n at which t holds

- **Always:**

$$\Box\, t ::= \neg(\Diamond(\neg t))$$

  intuition: there is no rank at which the negation of t holds

Exercise: what do $\Diamond\,\Box\, t$ and $\Box\,\Diamond\, t$ mean ?

Propositional temporal logic: examples

We consider the program below:

$$
\begin{array}{ll}
\ell_0: & \mathtt{x} = \mathsf{input}(); \\
\ell_1: & \mathsf{if}(\mathtt{x} < 8)\{ \\
\ell_2: & \qquad \mathtt{x} = 0; \\
\ell_3: & \} \, \mathsf{else} \, \{ \\
\ell_4: & \qquad \mathtt{x} = 1; \\
\ell_5: & \} \\
\ell_6: & \ldots
\end{array}
$$

Examples of properties:

- "when ℓ_4 is reached, $\mathtt{x}$ is positive"

$$\Box(@\ell_4 \Longrightarrow \mathtt{x} \geq 0)$$

- "if the value read at point ℓ_0 is negative, and when ℓ_6 is reached, $\mathtt{x}$ is equal to 0"

$$\Box\left((@\ell_1 \wedge \mathtt{x} < 0) \Longrightarrow \Box(@\ell_6 \Longrightarrow \mathtt{x} = 0)\right)$$

Outline

Security properties

We now consider other interesting properties of programs, and show that they do not all reduce to trace properties

i.e., not all interesting properties write as $\forall \sigma \in P^*,\ \sigma \in \mathcal{S}$...

Security

- Collects many kinds of properties
- So we consider just one:

  an unauthorized observer should not be able to guess anything about private information by looking at public information

- **Example:** another user should not be able to guess the content of an email sent to you
- We need to **formalize this property**

A few definitions

Assumptions:

- We let $\mathcal{S} = (\mathbb{S}, \rightarrow, \mathbb{S}_{\mathcal{I}})$ be a transition system
- States are of the form $(\ell, m) \in \mathbb{L} \times \mathbb{M}$
- Memory states are of the form $\mathbb{X} \rightarrow \mathbb{V}$
- We let $\ell, \ell' \in \mathbb{L}$ (program entry and exit) and $\mathtt{x}, \mathtt{x}' \in \mathbb{X}$ (private and public variables)

Security property we are looking at

Observing the value of $\mathtt{x}'$ at ℓ' gives no information on the value of $\mathtt{x}$ at ℓ

A few examples

A secure program (**no information flow**, no way to guess $\mathtt{x}$):

$$\begin{array}{ll} \ell: & \mathtt{x}' = 84; \\ \ell': & \ldots \end{array}$$

An insecure program (**explicit information flow**, $\mathtt{x}'$ gives a lot of information about $\mathtt{x}$, so that we can simply recompute it):

$$\begin{array}{ll} \ell: & \mathtt{x}' = \mathtt{x} - 2; \\ \ell': & \ldots \end{array}$$

An insecure program (**implicit information flow**, through a test):

$$\begin{array}{ll} \ell: & \mathtt{if}(\mathtt{x} < 0)\{\mathtt{x}' = 0;\} \\ \ell': & \ldots \end{array}$$

How to characterize information flow in the semantic level ?

Non-interference

We consider the **transformer** Φ defined by:

$$\begin{array}{rccl} \Phi: & \mathbb{M} & \longrightarrow & \mathcal{P}(\mathbb{M}) \\ & m & \longmapsto & \{m' \in \mathbb{M} \mid \exists \sigma = \langle (\ell, m), \ldots, (\ell', m') \rangle \in [\![\mathcal{S}]\!]\} \end{array}$$

Definition: non-interference

There is **no interference** between (ℓ, x) and (ℓ', x') and we write $(l', x') \not\rightsquigarrow (l, x)$ if and only if the following property holds:

$$\begin{array}{l} \forall m \in \mathbb{M}, \forall v_0, v_1 \in \mathbb{V}, \\ \quad \{m'(\mathrm{x}') \mid m' \in \Phi(m[\mathrm{x} \leftarrow v_0])\} = \{m'(\mathrm{x}') \mid m' \in \Phi(m[\mathrm{x} \leftarrow v_1])\} \end{array}$$

Intuition:

- if two observations at point ℓ differ only in the value of x, there is no difference in observation of x' at ℓ'
- in other words, observing x' at ℓ' (even on many executions) gives no information about the value of x at point ℓ...

Non-interference is not a trace property

- We assume $\mathbb{V} = \{0, 1\}$ and $\mathbb{X} = \{\mathrm{x}, \mathrm{x}'\}$ (store m is defined by the pair $(m(\mathrm{x}), m(\mathrm{x}'))$, and denoted by it)
- We assume $\mathbb{L} = \{\ell, \ell'\}$ and consider two systems such that all transitions are of the form $(\ell, m) \to (\ell', m')$
  (i.e., system $\mathcal{S}$ is isomorphic to its transformer $\Phi[\mathcal{S}]$)

$$\Phi[\mathcal{S}_0]: \begin{array}{lcl} (0,0) & \longmapsto & \mathbb{M} \\ (0,1) & \longmapsto & \mathbb{M} \\ (1,0) & \longmapsto & \mathbb{M} \\ (1,1) & \longmapsto & \mathbb{M} \end{array} \qquad \Phi[\mathcal{S}_1]: \begin{array}{lcl} (0,0) & \longmapsto & \mathbb{M} \\ (0,1) & \longmapsto & \mathbb{M} \\ (1,0) & \longmapsto & \{(1,1)\} \\ (1,1) & \longmapsto & \{(1,1)\} \end{array}$$

- $\mathcal{S}_1$ has fewer behaviors than $\mathcal{S}_0$: $[\![\mathcal{S}_1]\!]^* \subset [\![\mathcal{S}_0]\!]^*$
- $\mathcal{S}_0$ **has the non-interference property**, but $\mathcal{S}_1$ **does not**
- If non interference was a trace property, $\mathcal{S}_1$ should have it (monotony)

Thus, the non interference property is not a trace property

Dependence properties

Dependence property

- Many notions of dependences
- So we consider just one:

  what inputs may influence on the observation of a given output

- **Applications:**
  - **reverse engineering:** understand how an input gets computed
  - **slicing:** extract the fragment of a program that is relevant to a result
- This corresponds to the **negation** of non-interference

Interference

Definition: interference

There is **interference** between $(\ell, \mathtt{x})$ and $(\ell', \mathtt{x}')$ and we write $(l', x') \rightsquigarrow (l, x)$ if and only if the following property holds:

$$\exists m \in \mathbb{M}, \exists v_0, v_1 \in \mathbb{V},$$
$$\{m'(\mathtt{x}') \mid m' \in \Phi(m[\mathtt{x} \leftarrow v_0])\} \neq \{m'(\mathtt{x}') \mid m' \in \Phi(m[\mathtt{x} \leftarrow v_1])\}$$

- This expresses that there is at least one case, where the value of $\mathtt{x}$ at ℓ has an impact on that of $\mathtt{x}'$ at ℓ'
- It may not hold even if the computation of $\mathtt{x}'$ reads $\mathtt{x}$:

$$\begin{aligned} \ell : &\quad \mathtt{x}' = 0 \star \mathtt{x}; \\ \ell' : &\quad \ldots \end{aligned}$$

Interference is not a trace property

- We assume $\mathbb{V} = \{0, 1\}$ and $\mathbb{X} = \{\mathrm{x}, \mathrm{x}'\}$ (store m is defined by the pair $(m(\mathrm{x}), m(\mathrm{x}'))$, and denoted by it)
- We assume $\mathbb{L} = \{\ell, \ell'\}$ and consider two systems such that all transitions are of the form $(\ell, m) \to (\ell', m')$
  (i.e., system $\mathcal{S}$ is isomorphic to its transformer $\Phi[\mathcal{S}]$)

$$\Phi[\mathcal{S}_0]: \begin{array}{lcl} (0,0) & \longmapsto & \mathbb{M} \\ (0,1) & \longmapsto & \mathbb{M} \\ (1,0) & \longmapsto & \{(1,1)\} \\ (1,1) & \longmapsto & \{(1,1)\} \end{array} \qquad \Phi[\mathcal{S}_1]: \begin{array}{lcl} (0,0) & \longmapsto & \{(1,1)\} \\ (0,1) & \longmapsto & \{(1,1)\} \\ (1,0) & \longmapsto & \{(1,1)\} \\ (1,1) & \longmapsto & \{(1,1)\} \end{array}$$

- $\mathcal{S}_1$ has fewer behavior than $\mathcal{S}_0$: $[\![\mathcal{S}_1]\!]^* \subset [\![\mathcal{S}_0]\!]^*$
- $\mathcal{S}_0$ **has the interference property**, but $\mathcal{S}_1$ **does not**
- If interference was a trace property, $\mathcal{S}_1$ should have it (monotony)

Thus, the interference property is not a trace property

Hyperproperties

Conclusion:

- The absence of interference between (ℓ, x) and (ℓ', x') **is not a trace property**:
  we cannot describe as the set of programs the semantics of which is included into a given set of traces.
- This is **not surprising**: the definition of non-interference refers to **two execution traces**
- It can however **be described by a set of sets of traces**:
  we simply collect the set of program semantics that satisfy the property.

This is what we call a **hyperproperty**:

Hyperproperties

- **Trace hyperproperties** are described by sets of sets of executions
- **Trace properties** are described by sets of executions

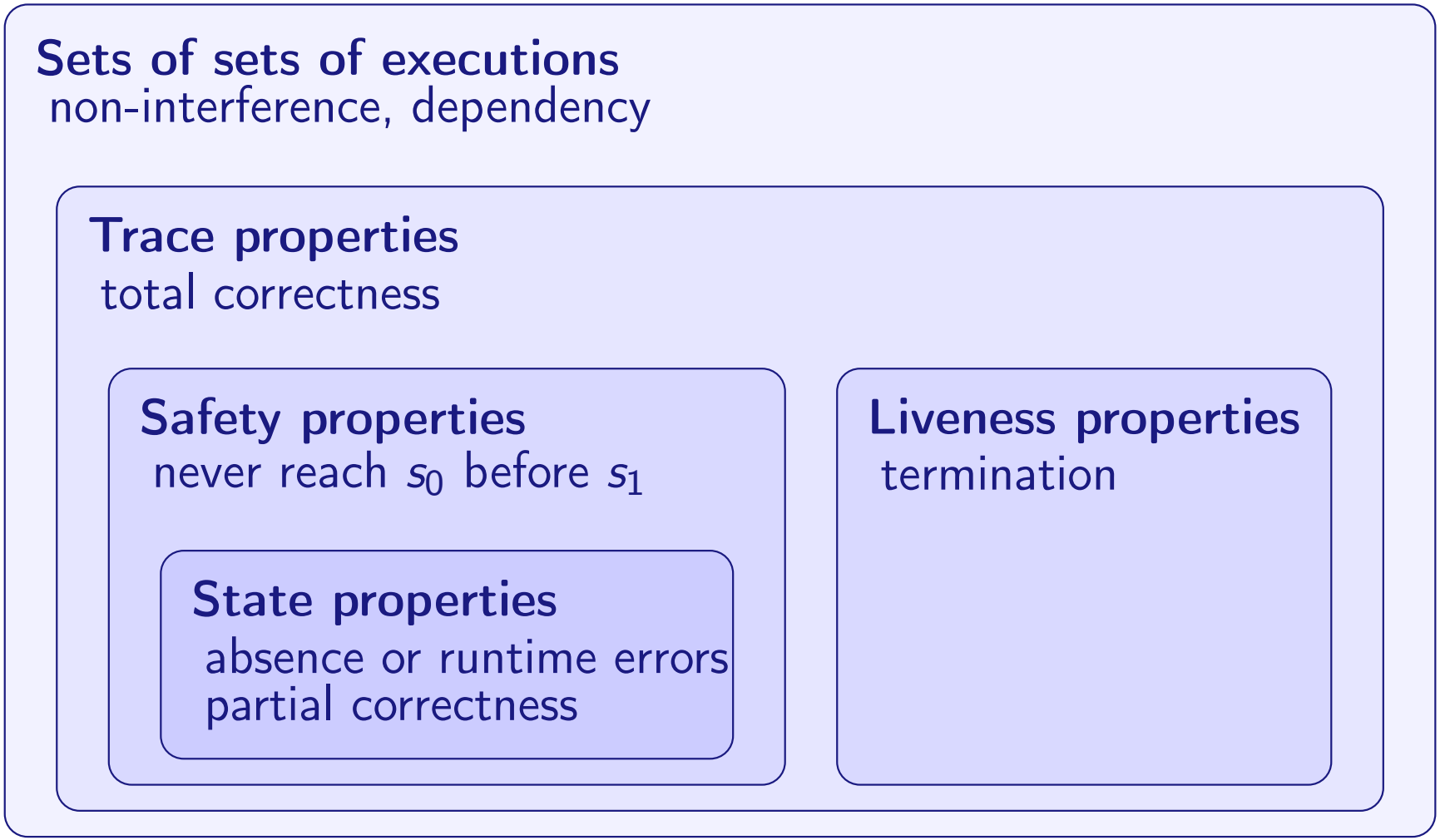
Some classes of hyperproperties

A **very elegant framework, with many classes**:

- **2-safety**:
  Writes down as $\forall \sigma_0, \sigma_1 \in [\![P]\!]^*$, $\texttt{Pred}(\sigma_0, \sigma_1)$.
  For instance, to disprove the absence of interference (i.e., to show there exists an interference), we simply need to exhibit **two finite traces**.
  On proof methods: self-composition for k-safety...
- **k-safety**:
  Same, but with k execution traces instead of 2...
- **Hypersafety**:
  Counterexamples are always finite (finitely many finite traces)
- **Hyperliveness**:
  Counterexamples need to be infinite (infinitely many traces, or some infinite traces...)

Outline

The Zoo of semantic properties



Summary

To sum-up:

- **Trace properties** allow to express a large range of program properties
- **Safety = absence of bad behaviors**
- **Liveness = existence of good behaviors**
- Trace properties can be **decomposed** as conjunctions of safety and liveness properties, with **dedicated proof methods**
- Some interesting properties are **not trace properties**
  security properties are *sets of sets of executions*
- Notion of **specification languages** to describe program properties