

# Traces Properties

## Semantics and applications to verification

Xavier Rival

École Normale Supérieure

March 1st, 2024

# Program of this lecture

## Goal of verification

**Prove that  $\llbracket P \rrbracket \subseteq \mathcal{S}$**

(i.e., all behaviors of  $P$  satisfy specification  $\mathcal{S}$ )

where  $\llbracket P \rrbracket$  is the **program semantics** and  $\mathcal{S}$  the **desired specification**

Last week, we studied a form of  $\llbracket P \rrbracket$ ...

**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  $\mathcal{S} = (\mathbb{S}, \rightarrow, \mathbb{S}_I)$

## 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.,  $\llbracket \mathcal{S} \rrbracket_{\mathcal{R}} \subseteq \mathcal{P}$  where  $\llbracket \mathcal{S} \rrbracket_{\mathcal{R}} = \{s_n \in \mathbb{S} \mid \exists \langle s_0, \dots, s_n \rangle \in \llbracket \mathcal{S} \rrbracket^*, s_0 \in \mathbb{S}_I\}$

## 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.,  $\llbracket \mathcal{S} \rrbracket^\infty \subseteq \mathcal{T}$

## 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.,  $\llbracket \mathcal{S}_0 \rrbracket \subseteq \llbracket \mathcal{S}_1 \rrbracket$ ), 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

- 1 Safety properties
  - Informal and formal definitions
  - Proof method
- 2 Liveness properties
- 3 Decomposition of trace properties
- 4 A specification language: temporal logic
- 5 Beyond safety and liveness
- 6 Conclusion

# 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**  
 $\sigma = \langle s_0, \dots, s_n, \dots \rangle \in \llbracket \mathcal{S} \rrbracket \setminus \mathcal{P}$ ;  
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  $\sigma' = \langle s_0, \dots, s_i \rangle$  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  $\sigma'$  **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  $\sigma$ :

$$\begin{aligned}\langle s_0, \dots, s_n \rangle_{\lceil 0} &= \epsilon \\ \langle s_0, \dots, s_n \rangle_{\lceil i+1} &= \begin{cases} \langle s_0, \dots, s_i \rangle & \text{if } i < n \\ \langle s_0, \dots, s_n \rangle & \text{otherwise} \end{cases} \\ \langle s_0, \dots \rangle_{\lceil i+1} &= \langle s_0, \dots, s_i \rangle\end{aligned}$$

**Suffix** (or tail):

$$\begin{aligned}\sigma_{i\rfloor} &= \epsilon & \text{if } |\sigma| < i \\ (\langle s_0, \dots, s_i \rangle \cdot \sigma)_{i+1\rfloor} &::= \sigma & \text{otherwise}\end{aligned}$$

# Upper closure operators

## Definition: upper closure operator (uco)

We consider a preorder  $(\mathcal{S}, \sqsubseteq)$ . Function  $\phi : \mathcal{S} \rightarrow \mathcal{S}$  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:

$$\begin{aligned} \text{PCI} : \mathcal{P}(\mathbb{S}^\infty) &\longrightarrow \mathcal{P}(\mathbb{S}^*) \\ X &\longmapsto \{\sigma_{\upharpoonright i} \mid \sigma \in X, i \in \mathbb{N}\} \end{aligned}$$

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

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

## Properties:

- PCI is monotone
- PCI is idempotent, i.e.,  $\text{PCI} \circ \text{PCI}(X) = \text{PCI}(X)$
- PCI 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:

$$\begin{aligned} \text{Lim} : \mathcal{P}(\mathbb{S}^\infty) &\longrightarrow \mathcal{P}(\mathbb{S}^\infty) \\ X &\longmapsto X \cup \{\sigma \in \mathbb{S}^\infty \mid \forall i \in \mathbb{N}, \sigma_{\upharpoonright i} \in X\} \end{aligned}$$

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

**Proof:** exercise!

**Example:** assuming

$$\mathcal{S} = \left\{ \begin{array}{ll} \epsilon, & \langle a \rangle \\ \langle a, b \rangle & \langle a, b, a \rangle \\ \langle a, b, a, b \rangle & \langle a, b, a, b, a \rangle \quad \dots \end{array} \right\}$$

then,

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

# Towards a formal definition for safety

## Operator Safe

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

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  $\sigma$  is a trace, to establish that  $\sigma$  is not in  $S$ , it is sufficient to discover a **finite prefix of  $\sigma$**  that cannot be observed in  $S$ .

- if  $\sigma$  is finite the result is clear (consider  $\sigma$ )
- otherwise, if all finite prefixes of  $\sigma$  are in  $S$ , then  $\sigma$  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 **PCI** 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  $\sigma$  is a finite trace, it is one of its prefixes, so  $\sigma \in \text{PCI}(X) \subseteq \text{Lim}(\text{PCI}(X))$
- if  $\sigma$  is an infinite trace, all its prefixes belong to  $\text{PCI}(X)$ , so  $\sigma \in \text{Lim}(\text{PCI}(X))$

# 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  $X \subseteq \mathbb{S}^\omega, \sigma \in \text{Safe}(\text{Safe}(X))$ ; then:

$$\begin{aligned}
 & \sigma \in \text{Safe}(\text{Safe}(X)) \\
 \Rightarrow & \forall i, \sigma \upharpoonright_i \in \text{PCI} \circ \text{Safe}(X) && \text{by def. of Lim} \\
 \Rightarrow & \forall i, \exists \sigma', j, \sigma \upharpoonright_i = \sigma' \upharpoonright_j \wedge \sigma' \in \text{Safe}(X) && \text{by def. of PCI} \\
 \Rightarrow & \forall i, \exists \sigma', j, \sigma \upharpoonright_i = \sigma' \upharpoonright_j \wedge \forall k, \sigma' \upharpoonright_k \in \text{PCI}(X) \\
 & \quad \text{by def. of Lim and case analysis over finiteness of } \sigma' \\
 \Rightarrow & \forall i, \exists \sigma', j, \sigma \upharpoonright_i = \sigma' \upharpoonright_j \wedge \sigma' \upharpoonright_j \in \text{PCI}(X) && \text{if we take } k = j \\
 \Rightarrow & \forall i, \sigma \upharpoonright_i \in \text{PCI}(X) && \text{by simplification} \\
 \Rightarrow & \sigma \in \text{Lim} \circ \text{PCI}(X) && \text{by def. of Lim} \\
 \Rightarrow & \sigma \in \text{Safe}(X)
 \end{aligned}$$



# 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

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

Then:

- $\text{PCI}(\mathcal{T})$  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)
- $\text{Lim}(\text{PCI}(\mathcal{T}))$  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,  $\text{Safe}(\mathcal{T}) = \text{Lim}(\text{PCI}(\mathcal{T})) = \mathcal{T}$

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**  $\mathcal{T} = \mathcal{P}^\infty$ :

$$\begin{aligned}\text{Safe}(\mathcal{T}) &= \text{Lim}(\text{PCI}(\mathcal{P}^\infty)) \\ &= \text{Lim}(\mathcal{P}^*) \\ &= \mathcal{P}^* \cup \mathcal{P}^\omega \\ &= \mathcal{P}^\infty \\ &= \mathcal{T}\end{aligned}$$

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  $\sigma$  is a trace, to establish that  $\sigma$  **is not in**  $S$ , it is sufficient to discover a **finite prefix of**  $\sigma$  that cannot be observed in  $S$ .

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

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

# Outline

- 1 Safety properties
  - Informal and formal definitions
  - Proof method
- 2 Liveness properties
- 3 Decomposition of trace properties
- 4 A specification language: temporal logic
- 5 Beyond safety and liveness
- 6 Conclusion

# 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  $\llbracket \mathcal{S} \rrbracket^\infty \subseteq \mathcal{T}$

## 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  $\llbracket \mathcal{S} \rrbracket^* \subseteq \mathbb{I}$
- thus,  $\text{Safe}(\llbracket \mathcal{S} \rrbracket^*) \subseteq \text{Safe}(\mathbb{I}) \subseteq \text{Safe}(\mathcal{T})$  since Safe is monotone
- we remark that  $\llbracket \mathcal{S} \rrbracket^\infty = \text{Safe}(\llbracket \mathcal{S} \rrbracket^*)$
- $\mathcal{T}$  is a safety property so  $\text{Safe}(\mathcal{T}) = \mathcal{T}$
- we conclude  $\llbracket \mathcal{S} \rrbracket^\infty \subseteq \mathcal{T}$ , 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  $\llbracket \mathcal{S} \rrbracket^*$  satisfies  $\mathcal{T}$ , and show that we can exhibit an invariant.

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

## Caveat:

- $\llbracket \mathcal{S} \rrbracket^\infty$  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,  $s = \sum_{k=0}^{n-1} t[k]$ :

```

    i, s integer variables
    t integer array of length n
ℓ0 : (true)
      s = 0;
ℓ1 : (s = 0)
      i = 0;
ℓ2 : (i = 0 ∧ s = 0)
      while(i < n){
ℓ3 : (0 ≤ i < n ∧ s =  $\sum_{k=0}^{i-1} t[k]$ )
          s = s + t[i];
ℓ4 : (0 ≤ i < n ∧ s =  $\sum_{k=0}^i t[k]$ )
          i = i + 1;
ℓ5 : (1 ≤ i ≤ n ∧ s =  $\sum_{k=0}^{i-1} t[k]$ )
      }
ℓ6 : (i = n ∧ s =  $\sum_{k=0}^{n-1} t[k]$ )
  
```

## Principle of the proof:

- for each program point  $\ell$ , 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:

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

# Outline

- 1 Safety properties
- 2 Liveness properties
  - Informal and formal definitions
  - Proof method
- 3 Decomposition of trace properties
- 4 A specification language: temporal logic
- 5 Beyond safety and liveness
- 6 Conclusion

# 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**  $\sigma \in \llbracket \mathcal{S} \rrbracket \setminus \mathcal{T}$ ;
- The informal definition says:  
... **may still occur after any finite observation**  
thus, each finite trace  $\sigma'$  can be extended into a good trace
- **Conclusion of this discussion:**  $\sigma$  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)

$$\text{PCI}(\mathcal{T}) = \mathbb{S}^*$$

# Definition

## Formal definition

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

- (i)  $\text{Live}(\mathcal{T}) = \mathcal{T}$
- (ii)  $\text{PCI}(\mathcal{T}) = \mathbb{S}^*$
- (iii)  $\text{Lim} \circ \text{PCI}(\mathcal{T}) = \mathbb{S}^\infty$

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):  $\text{PCI}(\mathcal{T}) = \mathbb{S}^*$   
thus termination indeed satisfies this definition

# Proof of equivalence

## Proof of equivalence:

### (i) implies (ii):

We assume that  $\text{Live}(\mathcal{T}) = \mathcal{T}$ , i.e.,  $\mathcal{T} \cup (\mathbb{S}^\infty \setminus \text{Safe}(\mathcal{T})) = \mathcal{T}$   
therefore,  $\mathbb{S}^\infty \setminus \text{Safe}(\mathcal{T}) \subseteq \mathcal{T}$ .

Let  $\sigma \in \mathbb{S}^*$ , and let us show that  $\sigma \in \text{PCI}(\mathcal{T})$ ; clearly,  $\sigma \in \mathbb{S}^\infty$ , thus:

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

### (ii) implies (iii):

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

### (iii) implies (i):

If  $\text{Lim} \circ \text{PCI}(\mathcal{T}) = \mathbb{S}^\infty$ , then

$$\text{Live}(\mathcal{T}) = \mathcal{T} \cup (\mathbb{S}^\infty \setminus (\text{Lim} \circ \text{PCI}(\mathcal{T}))) = \mathcal{T} \cup (\mathbb{S}^\infty \setminus \mathbb{S}^\infty) = \mathcal{T}$$

# 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

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

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

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



# A property of Live

## Theorem

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

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

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

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

# Outline

- 1 Safety properties
- 2 Liveness properties
  - Informal and formal definitions
  - Proof method
- 3 Decomposition of trace properties
- 4 A specification language: temporal logic
- 5 Beyond safety and liveness
- 6 Conclusion

# 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  $\llbracket \mathcal{S} \rrbracket^{\infty} \subseteq \mathbb{S}^*$

## Definition: ranking function

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

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

## Theorem

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

# Example

We consider the termination of the array sum program:

$i, s$  integer variables  
 $t$  integer array of length  $n$

```

 $\ell_0$  :  $s = 0$ ;
 $\ell_1$  :  $i = 0$ ;
 $\ell_2$  : while( $i < n$ ) {
 $\ell_3$  :      $s = s + t[i]$ ;
 $\ell_4$  :      $i = i + 1$ ;
 $\ell_5$  : }
 $\ell_6$  : ...

```

Ranking function:

$$\phi : \mathbb{S} \longrightarrow \mathbb{N}$$

$(\ell_0, m)$	$\longmapsto$	$3 \cdot n + 6$
$(\ell_1, m)$	$\longmapsto$	$3 \cdot n + 5$
$(\ell_2, m)$	$\longmapsto$	$3 \cdot n + 4$
$(\ell_3, m)$	$\longmapsto$	$3 \cdot (n - m(i)) + 3$
$(\ell_4, m)$	$\longmapsto$	$3 \cdot (n - m(i)) + 2$
$(\ell_5, m)$	$\longmapsto$	$3 \cdot (n - m(i)) + 4$
$(\ell_6, m)$	$\longmapsto$	$0$

## Proof by variance

- We consider transition system  $\mathcal{S} = (\mathbb{S}, \rightarrow, \mathbb{S}_I)$ , and liveness property  $\mathcal{T}$ ; infinite traces semantics is the greatest fixpoint of  $F_\omega$ .
- We seek a way of **verifying that  $\mathcal{S}$  satisfies  $\mathcal{T}$** , i.e., that  $\llbracket \mathcal{S} \rrbracket^\omega \subseteq \mathcal{T}$

### Principle of variance proofs

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

- $\mathbb{S}^\omega \subseteq \mathbb{I}_0$
- for all  $k \in \{1, 2, \dots, \omega\}$ ,  $\forall s \in \mathbb{S}$ ,  $\langle s \rangle \in \mathbb{I}_k$
- for all  $k \in \{1, 2, \dots, \omega\}$ , 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  $\llbracket \mathcal{S} \rrbracket^\omega$  instead

# Outline

- 1 Safety properties
- 2 Liveness properties
- 3 Decomposition of trace properties**
- 4 A specification language: temporal logic
- 5 Beyond safety and liveness
- 6 Conclusion

# 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  $\text{Safe}(\mathcal{T})$  is a safety property.

- **Liveness part:**

Live is idempotent, so  $\text{Live}(\mathcal{T})$  is a liveness property.

- **Decomposition:**

$$\begin{aligned}
 \text{Safe}(\mathcal{T}) \cap \text{Live}(\mathcal{T}) &= \text{Safe}(\mathcal{T}) \cap (\mathcal{T} \cup \mathbb{S}^\infty \setminus \text{Safe}(\mathcal{T})) \\
 &= \text{Safe}(\mathcal{T}) \cap \mathcal{T} \\
 &\quad \cup \text{Safe}(\mathcal{T}) \cap (\mathbb{S}^\infty \setminus \text{Safe}(\mathcal{T})) \\
 &= \mathcal{T} \cup \emptyset \\
 &= \mathcal{T}
 \end{aligned}$$



# Example: verification of total correctness

$i, s$  integer variables  
 $t$  integer array of length  $n$

```

 $\ell_0$  :    $s = 0$ ;
 $\ell_1$  :    $i = 0$ ;
 $\ell_2$  :   while( $i < n$ ){
 $\ell_3$  :        $s = s + t[i]$ ;
 $\ell_4$  :        $i = i + 1$ ;
 $\ell_5$  :   }
 $\ell_6$  :   ...
  
```

**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

## Trace properties

total correctness

### Safety properties

never reach  $s_0$  before  $s_1$

### State properties

absence or runtime errors  
partial correctness

### Liveness properties

termination

- **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

- 1 Safety properties
- 2 Liveness properties
- 3 Decomposition of trace properties
- 4 A specification language: temporal logic**
- 5 Beyond safety and liveness
- 6 Conclusion

# 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**)

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

- 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:

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

- **Semantics:**  $\llbracket p \rrbracket_{\mathbb{S}} \subseteq \mathbb{S}_{\Omega}$  is defined by

$$\begin{aligned} \llbracket @\ell \rrbracket_{\mathbb{S}} &= \{\ell\} \times \mathbb{M} \\ \llbracket x \leq x' \rrbracket_{\mathbb{S}} &= \{(\ell, m) \in \mathbb{S} \mid m(x) \leq m(x')\} \\ \llbracket x \leq n \rrbracket_{\mathbb{S}} &= \{(\ell, m) \in \mathbb{S} \mid m(x) \leq n\} \\ \llbracket \neg p \rrbracket_{\mathbb{S}} &= \mathbb{S}_{\Omega} \setminus \llbracket p \rrbracket_{\mathbb{S}} \\ \llbracket p \wedge p' \rrbracket_{\mathbb{S}} &= \llbracket p \rrbracket_{\mathbb{S}} \cap \llbracket p' \rrbracket_{\mathbb{S}} \\ \llbracket \Omega \rrbracket_{\mathbb{S}} &= \{\Omega\} \end{aligned}$$

**Exercise:** add  $=$ ,  $\vee$ ,  $\implies \dots$

# State properties: examples

## Unreachability of control state $\ell_0$ :

- **specification:**  $\forall \neg @\ell_0$
- **property:**  $\llbracket \neg @\ell_0 \rrbracket_s = S_\Omega \setminus \{(\ell_0, m) \mid m \in \mathbb{M}\}$

## Absence of runtime errors:

- **specification:**  $\neg \Omega$
- **property:**  $\llbracket \neg \Omega \rrbracket_s = S_\Omega \setminus \{\Omega\} = S$

## Intermittent invariant:

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

$$\begin{array}{ll}
 \ell_0 : & \text{if}(x \geq 0)\{ \\
 \ell_1 : & \quad y = x; \qquad \qquad @\ell_1 \implies x \geq 0 \\
 \ell_2 : & \quad \text{ } \} \text{else}\{ \qquad \qquad \wedge \quad @\ell_2 \implies x \geq 0 \wedge y \geq 0 \\
 \ell_3 : & \quad y = -x; \qquad \qquad \wedge \quad @\ell_3 \implies x < 0 \\
 \ell_4 : & \quad \} \qquad \qquad \wedge \quad @\ell_4 \implies x < 0 \wedge y > 0 \\
 \ell_5 : & \quad \dots \qquad \qquad \wedge \quad @\ell_5 \implies y \geq 0
 \end{array}$$



# 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

$t(\in \mathbb{L}_{\text{PTL}})$	$::=$	$p$	state property, i.e., $p \in \mathbb{L}_{\mathbb{S}}$
		$t' \vee t''$	disjunction
		$\neg t'$	negation
		$\bigcirc t'$	"next"
		$t' \mathbin{\text{U}} t''$	"until", i.e., $t'$ until $t''$

# 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

$$\begin{aligned}
 \llbracket p \rrbracket_t &= \{s \cdot \sigma \mid s \in \llbracket p \rrbracket_s \wedge \sigma \in \mathbb{S}^\infty\} \\
 \llbracket t_0 \vee t_1 \rrbracket_t &= \llbracket t_0 \rrbracket_t \cup \llbracket t_1 \rrbracket_t \\
 \llbracket \neg t_0 \rrbracket_t &= \mathbb{S}^\infty \setminus \llbracket t_0 \rrbracket_t \\
 \llbracket \bigcirc t_0 \rrbracket_t &= \{s \cdot \sigma \mid s \in \mathbb{S} \wedge \sigma \in \llbracket t_0 \rrbracket_t\} \\
 \llbracket t_0 \mathbin{\text{U}} t_1 \rrbracket_t &= \{\sigma \in \mathbb{S}^\infty \mid \exists n \in \mathbb{N}, \forall i < n, \sigma_{i\downarrow} \in \llbracket t_0 \rrbracket_t \wedge \sigma_n \in \llbracket t_1 \rrbracket_t\}
 \end{aligned}$$

# Temporal logic operators as syntactic sugar

Many useful operators can be added:

- **Boolean constants:**

$$\mathbf{true} ::= (x < 0) \vee \neg(x < 0)$$

$$\mathbf{false} ::= \neg \mathbf{true}$$

- **Sometime:**

$$\Diamond t ::= \mathbf{true} \mathbin{\text{U}} t$$

**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:

```

 $\ell_0$  :  x = input();
 $\ell_1$  :  if(x < 8){
 $\ell_2$  :      x = 0;
 $\ell_3$  :  } else {
 $\ell_4$  :      x = 1;
 $\ell_5$  :  }
 $\ell_6$  :  ...

```

## Examples of properties:

- “when  $\ell_4$  is reached, x is positive”

$$\Box(@\ell_4 \implies x \geq 0)$$

- “if the value read at point  $\ell_0$  is negative, and when  $\ell_6$  is reached, x is equal to 0”

$$\Box((@\ell_1 \wedge x < 0) \implies \Box(@\ell_6 \implies x = 0))$$

# Outline

- 1 Safety properties
- 2 Liveness properties
- 3 Decomposition of trace properties
- 4 A specification language: temporal logic
- 5 Beyond safety and liveness**
- 6 Conclusion

# Security properties

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

## 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  $\mathcal{S} = (\mathbb{S}, \rightarrow, \mathbb{S}_I)$  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  $x, x' \in \mathbb{X}$  (private and public variables)

Security property we are looking at

Observing the value of  $x'$  at  $\ell'$   
gives no information on the value of  $x$  at  $\ell$

## A few examples

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

$$\begin{aligned}\ell : & \quad x' = 84; \\ \ell' : & \quad \dots\end{aligned}$$

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

$$\begin{aligned}\ell : & \quad x' = x - 2; \\ \ell' : & \quad \dots\end{aligned}$$

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

$$\begin{aligned}\ell : & \quad \text{if}(x < 0)\{x' = 0;\} \\ \ell' : & \quad \dots\end{aligned}$$

**How to characterize information flow in the semantic level ?**



# Non-interference

We consider the **transformer**  $\Phi$  defined by:

$$\begin{aligned}\Phi : \mathbb{M} &\longrightarrow \mathcal{P}(\mathbb{M}) \\ m &\longmapsto \{m' \in \mathbb{M} \mid \exists \sigma = \langle (\ell, m), \dots, (\ell', m') \rangle \in \llbracket \mathcal{S} \rrbracket\}\end{aligned}$$

## Definition: non-interference

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

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

## Intuition:

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

# Non-interference is not a trace property

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

$$\begin{aligned}\Phi[\mathcal{S}_0] : \quad (0, 0) &\mapsto \mathbb{M} \\ (0, 1) &\mapsto \mathbb{M} \\ (1, 0) &\mapsto \mathbb{M} \\ (1, 1) &\mapsto \mathbb{M}\end{aligned}$$

$$\begin{aligned}\Phi[\mathcal{S}_1] : \quad (0, 0) &\mapsto \mathbb{M} \\ (0, 1) &\mapsto \mathbb{M} \\ (1, 0) &\mapsto \{(1, 1)\} \\ (1, 1) &\mapsto \{(1, 1)\}\end{aligned}$$

- $\mathcal{S}_1$  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  $(\ell, x)$  and  $(\ell', x')$  and we write  $(\ell', x') \rightsquigarrow (\ell, x)$  if and only if the following property holds:

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

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

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

# Interference is not a trace property

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

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

- $\mathcal{S}_1$  has fewer behavior than  $\mathcal{S}_0$ :  $\llbracket \mathcal{S}_1 \rrbracket^* \subset \llbracket \mathcal{S}_0 \rrbracket^*$
- $\mathcal{S}_0$  **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  $(\ell, x)$  and  $(\ell', x')$  **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
- 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

**2-safety**: to disprove the absence of interference (i.e., to show there exists an interference), we simply need to exhibit **two finite traces**

# Outline

- 1 Safety properties
- 2 Liveness properties
- 3 Decomposition of trace properties
- 4 A specification language: temporal logic
- 5 Beyond safety and liveness
- 6 Conclusion

# The Zoo of semantic properties

**Sets of sets of executions**  
non-interference, dependency

**Trace properties**  
total correctness

**Safety properties**  
never reach  $s_0$  before  $s_1$

**State properties**  
absence or runtime errors  
partial correctness

**Liveness properties**  
termination



# 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