ICS 355: Security and Trust I — Lecture 3: 02/08/2021

Resource security: Dynamic part

Lecturer: Dusko Pavlovic — Place: Online

Contents

1 Histories, properties, safety and liveness

1.1 Histories and properties

In the preceding lecture, we saw how different permission matrices correspond to different states of the world, and how different states of the world arise from subjects' actions. Now we need to reason about the results of these state transitions as the time goes by.

The **basic facts and notations for lists and strings** are given in the Appendix, at the end of this note.
Definition 1. *For a type* Σ *of* events*, the finite sequences*

$$\vec{x} = \left[x_0\, x_1\, \ldots\, x_n \right] \quad \in \quad \Sigma^*$$

are called the Σ-traces *or* histories*. A* Σ-property*, is expressed as a set of histories*

$$P \quad \subseteq \quad \Sigma^*$$

Trivial properties. The simplest properties are the set of all traces Σ^*, and the empty set $\emptyset$. If properties are thought of as requirement constraints, then the former is the least constraining requirement, as every trace satisfies it, whereas the latter is the most constraining requirement, as no trace satisfies it. A singleton $\left\{\vec{t}\right\} \subseteq \Sigma^*$ is a maximally constrained non-trivial property. Larger properties are less constraining.

Standard and refined Σ types. In most models, an event $x \in \Sigma$ is an action of a subject on object, i.e. $x = \langle a, i, u \rangle$, and thus

$$\Sigma \quad = \quad \mathbb{A} \times \mathbb{J} \times \mathbb{S} \tag{1}$$

In many cases, only a subset of actions $\mathbb{A}_{ui} \subseteq \mathbb{A}$ is applicable to an object $i \in \mathbb{J}$ by a subject $u \in \mathbb{S}$; and often only a subset of objects $\mathbb{J}_u \subseteq \mathbb{J}$ is accessible by a subject $u \in \mathbb{S}$, and the set of events that may actually occur is in fact the disjoint union[1]

$$\widetilde{\Sigma} \quad = \quad \coprod_{u \in \mathbb{S}} \coprod_{i \in \mathbb{J}_u} \mathbb{A}_{ui} \tag{2}$$

But since $\widetilde{\Sigma} \subseteq \Sigma$, we stick with the simpler form (1), and restrict to (2) using ad hoc constraints, e.g. in Sec. 2.2.

Access matrices as event spaces. If an event is an action of a subject on an object, then a permission matrix corresponds to a set of permitted events, along the one-to-one correspondence

$$\frac{\mathbb{S} \times \mathbb{J} \xrightarrow{M} \wp\mathbb{A}}{\widehat{M} \subseteq \Sigma}$$

defined by

$$\langle a, i, u \rangle \in \widehat{M} \quad \Longleftrightarrow \quad a \in M_{ui}$$

Since any access matrix $\mathbb{S} \times \mathbb{J} \xrightarrow{B} \wp\mathbb{A}$ similarly corresponds to a set of events $\widehat{B^q} \subseteq \Sigma$, the access control security requirement in (**??**) now becomes $B \subseteq M$. The access matrices from Ch. **??** correspond to subsets of events. At each moment in time, the matrix view, and the event space view are equivalent. The latter is, however, more convenient when we need to take a "historic perspective", and study resource security of processes in time.

To get going, let us have a look at a couple of examples of histories and properties. In general, events are actions of subject on objects. But when subjects don't matter, or when there is just one, then the events are actions on objects.

[1] In dependent type theory, this type would be written as $\sum u : \mathbb{S}\ \sum i : \mathbb{J}(u).\ \mathbb{A}(u, i)$. The notational clash of the symbol Σ used for labels, alphabets, and events in semantics of comutation, and for sum types in type theory, is accidental.

When objects don't matter, or there is just one, then the events are just actions. We begin from this simplest case, and then broaden to richer frameworks.

Example 1: sheep. We begin with Alice as the only subject, and her sheep as the only object, so that an event is just one of Alice's actions on the sheep, i.e.

$$\Sigma \;=\; \mathbb{A} \;=\; \{\text{milk}, \text{wool}, \text{meat}\}$$

The trace

$$\vec{m} \;=\; \big[\,\text{milk milk milk milk wool milk}\,\big]$$

means that Alice milked her sheep 4 times, then sheared the wool, and then milked her again. Consider the following trace properties:

$$\begin{aligned}
\text{MilkWool} &= \{\text{milk}, \text{wool}\}^* && (3)\\
\text{MilkWoolMeat} &= \{\text{milk}, \text{wool}\}^* :: \text{meat} && (4)\\
\text{MilkWoolWool} &= \{\text{milk}, \text{wool}\}^* :: \text{wool} && (5)\\
\text{MilkWoolMeatMeat} &= \{\text{milk}, \text{wool}, \text{meat}\}^* :: \text{meat} && (6)\\
\text{MeatWoolMilkMilk} &= \{\text{meat}, \text{wool}, \text{milk}\}^* :: \text{milk} && (7)\\
\text{MilkWoolAnnual} &= \text{milk}^* \cup \big[\underbrace{\text{milk milk} \ldots \text{milk}}_{\geq 365 \text{ times}}\ \text{wool}\big] :: \text{MilkWoolAnnual} && (8)
\end{aligned}$$

They constrain how Alice uses her sheep resources as follows:

- (3) says that Alice can milk and shear her sheep at will, but cannot use its meat;
- (4) says that Alice can milk and shear her sheep at will, but only until she eats its meat;
- (5) says that Alice can milk and shear her sheep at will, and in the end has to shear it;
- (6) says that Alice can milk, shear, and eat her sheep at will, and in the end has to eat it;
- (7) says that Alice can milk, shear, and eat her sheep at will, and in the end has to milk it;
- (8) says that Alice can use milk and wool, provided that she shears the wool at most once per year

Example 2: elevator. Time passed, and Alice and Bob moved from their houses on Uruq Lane into apartments in a tall building with an elevator, like in fig. 1. They are now sharing not only the public spaces in their neighborhood, but also the elevator in their building. Making sure that the elevator is safe and secure requires modeling how it works. For simplicity, we first assume works the same for everyone, and omit subjects from the model. The relevant types are thus:

- objects $\mathbb{J} \;=\; \{0, 1, 2, \ldots, n\}$ — the floors of the building;
- actions $\mathbb{A} \;=\; \{\langle\,\rangle, (\,)\}$ — "call/go to" and "arrive to", respectively;
- events $\Sigma \;=\; \mathbb{J} \times \mathbb{A} \;=\; \{\langle i \rangle, (i) \;\big|\; i = 0, 1, 2, \ldots, n\}$ — "call to floor i" and "arrive to floor i", respectively[2].

An elevator history is thus a sequence of calls $\langle i \rangle$ and services (i), i.e. an element of the set

$$\Sigma^* \;=\; \big\{\,\big[\langle x_0 \rangle\, \langle x_1 \rangle\, (x_0)\, \langle x_2 \rangle\, \langle x_3 \rangle \ldots\, (x_\ell)\big] \;\big|\; x_0, x_1, \ldots, x_k = 0, 1, 2, \ldots, n\big\}$$

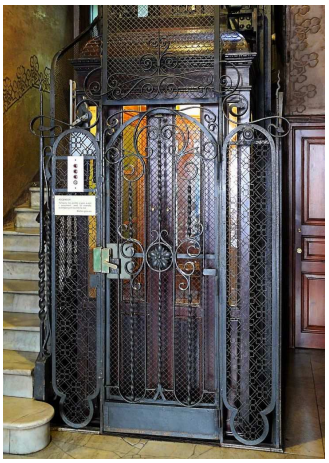


Figure 1: The elevator has a call button at each floor, and the call buttons for all floors in the cabin.

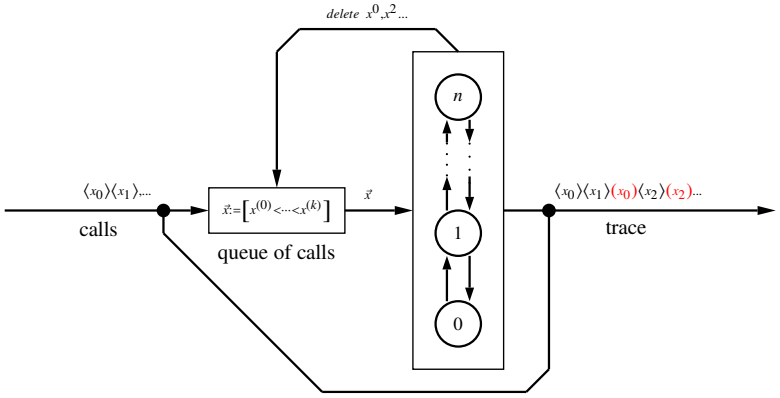


Figure 2: Elevator requests flow in on the left and come shuffled with services on the right

where the variables x_k denote the floor numbers where the elevator is called or sent, and where it provides service. It receives calls as inputs, and provides services as outputs. A process diagram is displayed in fig. 2. A call history is input on the left, and a history of calls and services shuffled in-between them is output on the right. In-between is a scheduler, which maintains a queue of open calls, i.e. those that have not yet been responded to by a service. When a service is provided, the scheduler pops the corresponding call from the queue, and the call is closed. There is thus a feedback from the elevator controller, that provides the services. to the scheduler, that maintains the queue of calls. The process can be viewed as a channel with feedback. The scheduler usually sorts the calls in order of the floors, and changes the priorities depending on the state: it discharges the next floor up on the way up, and the next floor down on the say down. This is, of course, just a very crude picture.

The required properties of processes, such as an elevator, or Bob's sheep bank from Sec. **??**, can be conveniently expressed as trace properties, which we discuss next.

1.2 Safety and liveness

The idea of safety and liveness in theory of processes [**?**] is that a process is

- safe if *bad things do not happen*, and
- live if *nice things do happen*.

The properties expressible in terms of safety and liveness properties are called the *dependability* properties. Thm. **??** says that in fact *all* properties from Def.. **??**, i.e. all properties expressible as sets are dependability properties, as each of them can be expressed as an intersection of a safety and a liveness property. Magic. But how does it work?

Safety captures the intuition that a bad thing, once it happens, remains bad in the future: *an unsafe history remains unsafe forever*. More precisely, if a history $\vec{x}$ does not satisfy a safety requirement $S \subseteq \Sigma^*$, then no future $\vec{x} :: \vec{y}$ can satisfy it either, i.e.

$$\forall \vec{x} \in \Sigma^* \; \forall \vec{y} \in \Sigma^*. \;\; \vec{x} \notin S \quad \Rightarrow \quad \left(\vec{x} :: \vec{y}\right) \notin S \tag{9}$$

The other way around (and equivalently), if $\vec{x} :: \vec{y}$ is safe, then its past $\vec{x}$ must also be safe, i.e.

$$\forall \vec{x} \in \Sigma^* \; \forall \vec{y} \in \Sigma^*. \;\; \left(\vec{x} :: \vec{y}\right) \in S \quad \Rightarrow \quad \vec{x} \in S \tag{10}$$

Liveness implements the idea that there is always a nice thing in the future: *a property is alive if it is eventually satisfied in all histories*.In other words, any history $\vec{x}$ has a future $\vec{x} :: \vec{y}$ that is alive. A property $L \subseteq \Sigma^*$ is thus liveness

$$\forall \vec{x} \in \Sigma^* \; \exists \vec{y} \in \Sigma^*. \quad \left(\vec{x} :: \vec{y}\right) \in L \tag{11}$$

In terms of the prefix ordering $\sqsubseteq$ from Sec. A(49), this means that safety properties are the $\sqsubseteq$-lower closed sets, whereas liveness properties are reached from below.

Definition 2. *The families of safety properties and of liveness properties over the event set* Σ *are respectively*

$$\mathsf{Safe}_\Sigma \;=\; \left\{ S \in \wp(\Sigma^*) \mid \forall \vec{x} \in \Sigma^* \; \forall \vec{z} \in S. \;\; \vec{x} \sqsubseteq \vec{z} \Rightarrow \vec{x} \in S \right\} \tag{12}$$

$$\mathsf{Live}_\Sigma \;=\; \left\{ L \in \wp(\Sigma^*) \mid \forall \vec{x} \in \Sigma^* \; \exists \vec{z} \in L. \;\; \vec{x} \sqsubseteq \vec{z} \right\} \tag{13}$$

When confusion seems unlikely, we omit the subscript Σ *from* Safe_Σ *and* Live_Σ.

Remark. Note that the definition of safety in (12) is equivalent with (10), whereas the definition of liveness in (13) is equivalent with (11).

[2] An action a performed on an object i is usually written as a_i. Here we write $\langle\, i \,\rangle$ instead of $()_i$ and $(\, i \,)$ instead of $\langle\rangle_i$.

Different situations require different safety and liveness requirements. There are examples where both $S \subseteq \Sigma^*$ and $\neg S = \Sigma^* \setminus W$ are safety properties. There are examples where both $L \subseteq \Sigma^*$ and $\neg L = \Sigma^* \setminus W$ are liveness properties. This does not mean that a system can be both safe and unsafe at the same time, or both alive and dead. It means that desirable concepts of safety and liveness may vary from situation to situation. The intent of Def. 2 is not to provide objective or universal properties, that every safe process should satisfy. Def. 2 describes the properties of properties which can be meaninfully declared to be the desired safety or liveness notions for a particular family of processes.

While there are many properties that are both safe and unsafe, and properties that are both alive and dead, the only property that is both safe and alive is the trivial one, i.e $\mathsf{Safe} \cap \mathsf{Live} = \{\Sigma^*\}$, whereby every history is safe. More examples follow.

Example 1: Safety and liveness of sheep. The properties of Alice's interactions with her sheep using $\Sigma = \{\text{milk}, \text{wool}, \text{meat}\}$ are as follows:

- (3) MilkWool $\in \mathsf{Safe} \setminus \mathsf{Live}$
- (4) MilkWoolMeat $\notin \mathsf{Safe} \cup \mathsf{Live}$;
- (5) MilkWoolWool $\notin \mathsf{Safe} \cup \mathsf{Live}$;
- (6) MilkWoolMeatMeat $\in \mathsf{Live} \setminus \mathsf{Safe}$;
- (7) MeatWoolMilkMilk $\in \mathsf{Live} \setminus \mathsf{Safe}$;
- (8) MilkWoolAnnual $\in \mathsf{Safe} \setminus \mathsf{Live}$.

Example 2: Safety and liveness of elevator. An example of reasonable dependability properties required from an elevator are:

- ***safety:*** the elevator should *only* come to a floor to which it was called, and
- ***liveness:*** the elevator should eventually go to each floor where it was called.

Using the notation from Sec. A, these properties can be formally stated as

$$\begin{array}{lcll} \overline{(i)} & \subseteq & \overline{\exists \langle i \rangle \prec (i)} & \rightsquigarrow \textit{safety} \\ \overline{\langle i \rangle} & \subseteq & \overline{\langle i \rangle \prec \exists (i)} & \rightsquigarrow \textit{liveness} \end{array}$$

Safety says that every arrival (i) must be preceded by a call $\langle i \rangle$. Liveness says that every call $\langle i \rangle$ must eventually be followed by an arrival (i). The properties that we specified are thus

$$\text{SafElev} = \left\{ \vec{t} \in \Sigma^* \mid \forall i \in \mathbb{I}.\, \vec{t} \in \overline{(i)} \implies \vec{t} \in \overline{\exists \langle i \rangle \prec (i)} \right\} = \bigcap_{i=0}^{n} \neg \overline{(i)} \cup \overline{\exists \langle i \rangle \prec (i)} \tag{14}$$

$$\text{LivElev} = \left\{ \vec{t} \in \Sigma^* \mid \forall i \in \mathbb{I}.\, \vec{t} \in \overline{\langle i \rangle} \implies \vec{t} \in \overline{\langle i \rangle \prec \exists (i)} \right\} = \bigcap_{i=0}^{n} \neg \overline{\langle i \rangle} \cup \overline{\langle i \rangle \prec \exists (i)} \tag{15}$$

where $\neg X = \Sigma^* \setminus X$ denotes the complement set.[3]

Exercises. Prove that

a) the properties defined in (3–8) satisfy the claims made in Dependability Example 1;

[3] Note that $\overline{a} \supseteq \overline{b \prec a}$ is true, but that $\overline{a} \supseteq \overline{\exists b \prec a}$ may not be true.

b) the properties defined in (14–15) satisfy $S \in \mathsf{Safe} \setminus \mathsf{Live}$ and $L \in \mathsf{Live} \setminus \mathsf{Safe}$

c) the following are safety properties:

i. $\neg\overline{a}$
ii. $\neg\overline{C}$
iii. $\neg a_0 a_1 \cdots a_n \Sigma^*$
iv. $\neg(\neg a_0 a_1 \cdots a_n \Sigma^*)\, \Sigma^*$

d) the following are liveness properties:

i. $\overline{a}$
ii. $\overline{C \prec \exists C}$
iii. $\overline{a_0 \prec a_1 \prec a_n}$
iv. $\Sigma^* a_0 a_1 \cdots a_n$

where $a, a_0, \ldots a_n \in \Sigma$, $C \subseteq \Sigma$, and $\neg X = \Sigma^* \setminus X$.

2 Authority and availability

2.1 Local events and properties

As indicated in Sec. 1.1, we model security properties as sets of histories $P \subseteq \Sigma^*$, where $\Sigma = \mathbb{A} \times \mathbb{J} \times \mathbb{S}$, or more generally $\Sigma = \mathbb{A} \times \mathbb{J} \times \mathbb{S} \times \mathbb{L}$, when security levels need to be taken into account. An event $x \in \Sigma$ is thus an action of a subject on an object, i.e. $x = \langle a, i, u \rangle$, or $x = \langle a, i, u, \ell \rangle$, where ℓ is a security level. There are other notions of event, suitable for other situations, but for the moment we stick with this one.

Local views of events. The event type Σ thus comes with the projections

$$\boldsymbol{s}\colon \Sigma \longrightarrow \mathbb{S} \qquad \boldsymbol{o}\colon \Sigma \longrightarrow \mathbb{J} \qquad \boldsymbol{a}\colon \Sigma \longrightarrow \mathbb{A} \qquad \boldsymbol{\ell}\colon \Sigma \longrightarrow \mathbb{L} \tag{16}$$

assigning to each event $x \in \Sigma$ its action $\boldsymbol{a}(x)$, object $\boldsymbol{o}(x)$, subject $\boldsymbol{s}(x)$, and the security level $\boldsymbol{\ell}(x)$. All such projections $\boldsymbol{p}\colon \Sigma \longrightarrow \mathbb{V}$ induce the *local* event types $\Sigma_v = \{x \in \Sigma \mid \boldsymbol{p}(x) = v\}$, which partition the event type as disjoint unions $\Sigma = \coprod_{v \in \mathbb{V}} \Sigma_v$, where $\mathbb{V}$ can be $\mathbb{A}, \mathbb{J}, \mathbb{S}$, and sometimes also $\mathbb{L}$. The specific local event types that we will be working with here are:

- $\Sigma_u = \{\langle a, i, u \rangle \mid a \in \mathbb{A}, i \in \mathbb{J}\}$ — the events observable by the subject u;
- $\Sigma_i = \{\langle a, i, u \rangle \mid a \in \mathbb{A}, u \in \mathbb{S}\}$ — the events involving the object i;
- $\Sigma_a = \{\langle a, i, u \rangle \mid i \in \mathbb{J}, A \in \mathbb{S}\}$ — the events where someone performs the action a on something.

Local histories. Process models are usually set up so that Alice only sees her own actions, i.e. that an event $x = \langle a, i, u \rangle \in \Sigma$ is observable for Alice just when $u = A$. When needed, this is refined using a lattice of security levels, as explained in the Prerequisites section. In this way, Alice may also be able to see Bob's actions, but not the other way around. In any case, Alice's view of a history $\vec{z} \in \Sigma^*$ is the *local history* $\vec{z}\restriction_A \in \Sigma_A^*$, where the *purge* operation $\restriction_A\colon \Sigma^* \longrightarrow \Sigma_A^*$ is defined like in (50) in the Prerequisites section, which in this case instantiates to

$$\begin{aligned} () \restriction_A &= () \\ (x{::}\vec{y}) \restriction_A &= \begin{cases} x{::}(\vec{y}) \restriction_A & \text{if } x = \langle a, i, A \rangle \text{ for some } a \in \mathbb{A} \text{ and } i \in \mathbb{J} \\ (\vec{y}) \restriction_A & \text{otherwise} \end{cases} \end{aligned} \tag{17}$$

The purge operations can be defined in a similar way for any of the projections in (16).

Notations. We write $x_A \in \Sigma_A$ for local events, and $\vec{x}_A \in \Sigma_A^*$ for local histories. Note the difference between

- the local purge $\vec{x}\restriction_A \in \Sigma_A^*$ of a given global history $\vec{x} \in \Sigma^*$, and
- a local history $\vec{x}_A \in \Sigma_A^*$, which is given on its own.

Alice's *local view* of the property $P \subseteq \Sigma^*$ is

$$P_A \quad = \quad \left\{ \vec{x}\restriction_A \mid \vec{x} \in P \right\} \tag{18}$$

Localized properties. When Alice observes a local history $\vec{t}_A \in \Sigma_A^*$, she knows that the state of the world is some history $\vec{z} \in \Sigma^*$ with $\vec{z}\restriction_A = \vec{t}_A$. But there are, of course, infinitely many such histories, as $\vec{z}$ may contain any number of events that are unobservable to Alice.

If Alice observes $\vec{z}\restriction_A$, Bob observes $\vec{z}\restriction_B$, and they tell each other what they have seen, they will still not be able to derive $\vec{z}$, even if they are alone in the world. The reason is that neither of them can observe which of Alice's actions occur before which of Bob's actions, and which after.

Example. Let $\Sigma = \Sigma_A \coprod \Sigma_B$ where $\Sigma_A = \{a\}$ and $\Sigma_B = \{b\}$. Suppose that a history $\vec{t}$ satisfies the property $P = \left\{ [\, aaaa \,], [\, aabb \,], [\, baab \,], [\, bbbb \,] \right\}$. If Alice observes $\vec{t}_A = [\, aa \,]$ and Bob observes $\vec{t}_B = [\, bb \,]$, can they be sure that the property P has been satisfied? The possible states of the world consistent with Alice's and Bob's observations are

$$\begin{aligned} \widehat{t} \quad &= \quad \left\{ \vec{z} \in \Sigma^* \mid \vec{z}\restriction_A = [\, aa \,] \wedge \vec{z}\restriction_B = [\, bb \,] \right\} \\ &= \quad \left\{ [\, aabb \,], [\, abab \,], [\, abba \,], [\, baab \,], [\, baba \,], [\, bbaa \,] \right\} \end{aligned}$$

Any of these actions could have taken place. Alice and Bob will only be able to verify locally a property P is satisfied if it is a *localized* property, according to the next definition.

Definition 3. *The* localization *of a property $P \subseteq \Sigma^*$ at the partition $\Sigma = \coprod_{u \in \mathbb{S}} \Sigma_u$ is the the set $\widehat{P}$ of all histories with the projections satisfying P, i.e.*

$$\begin{aligned} \widehat{P} \quad &= \quad \left\{ \vec{z} \in \Sigma^* \mid \forall u \in \mathbb{S}.\, \vec{z}\restriction_u \in P_u \right\} \\ &\qquad \textit{where } P_u = \left\{ \vec{x}\restriction_u \mid \vec{x} \in P \right\} \end{aligned} \tag{19}$$

A property $P \subseteq \Sigma^*$ is localized *when $P = \widehat{P}$, i.e.*

$$\vec{t} \in P \quad \Longleftrightarrow \quad \forall u \in \mathbb{S}.\, \vec{t}\restriction_u \in P_u \tag{20}$$

The family of localized properties is denoted

$$\mathsf{Loc} \quad = \quad \{P \in \wp(\Sigma^*) \mid P = \widehat{P}\} \tag{21}$$

2.2 Resource security as localized safety and liveness

Authority and availability are localized versions of safety and liveness. More precisely, a process is

- authorized if bad things do not happen to any of the subjects: *all bad resource requests are rejected*;
- available of nice things happen for all of the subjects: i.e. *all nice resource requests are eventually accepted*.

The task is again to formalize this intuition. Refining the idea of safety from (9), we say that $F \subseteq \Sigma^*$ is an authorization property if it satisfies the following condition:

$$\forall \vec{x}\vec{z} \in \Sigma^*.\ \vec{x} \notin F\ \wedge \vec{x} \sqsubseteq \vec{z}\ \ \Rightarrow\ \ \exists u \in \mathbb{S}.\ \vec{z}\restriction_u \notin F_u \tag{22}$$

In words, if there is an authority breach in some history, then in every future of that history, some subject will observe that their authority has been breached. Every authority breach is a breach of someone's authority. The logical converse of (22), refining (10), characterizes authority (or synonymously, an authorization property) by

$$\forall \vec{x}\vec{z} \in \Sigma^*.\ \left(\vec{x} \sqsubseteq \vec{z}\ \wedge\ \forall u \in \mathbb{S}.\ \vec{z}\restriction_u \in F_u\right)\ \Longrightarrow\ \vec{x} \in F \tag{23}$$

In other words, if a history $\vec{x}::\vec{y}$ is authorized by everyone locally, then its past $\vec{x}$ must have been authorized globally.

Strengthening the idea of liveness from (11), we say that a property is availability if any subject u can make any history available:

$$\forall \vec{x} \in \Sigma^*\ \forall u \in \mathbb{S}\ \ \exists \vec{z} \in D.\ \ \vec{x}\restriction_u \sqsubseteq \vec{z}\restriction_u \tag{24}$$

In other words, an availability property is a liveness property where any subject on their own can assure the liveness.

Definition 4. *For any family of subjects $\mathbb{S}$ for events partitioned into $\Sigma = \coprod_{A\in\mathbb{S}} \Sigma_u$, the authorization and the availability properties are respectively defined*

$$\mathsf{Auth}_\Sigma \ = \ \left\{F \in \wp(\Sigma^*)\ |\ \forall \vec{x} \in \Sigma^*\ \forall \vec{y} \in \Sigma^*.\ \left(\forall u \in \mathbb{S}.\ \left(\vec{x} :: \vec{y}\right)\restriction_u \in F_u\right) \Rightarrow \vec{x} \in F\right\} \tag{25}$$

$$\mathsf{Avail}_\Sigma \ = \ \left\{D \in \wp(\Sigma^*)\ |\ \forall \vec{x} \in \Sigma^*\ \forall u \in \mathbb{S}\ \ \exists \vec{y}_u \in \Sigma_u^*.\ \ \vec{x}\restriction_u :: \vec{y}_u \in D_u\right\} \tag{26}$$

When confusion seems unlikely, we omit the subscript Σ.

Proposition 5. *Let $P \subseteq \Sigma^*$ where $\Sigma = \coprod_{u\in\mathbb{S}} \Sigma_u$. Then the following statements are true.*

a) Authorization is localized and it boils down to safety for all subjects, whereas availability implies local liveness for all subjects:

$$P \in \mathsf{Auth}_\Sigma \ \Longleftrightarrow\ \forall u \in \mathbb{S}.\ P_u \in \mathsf{Safe}_{\Sigma_u}\ \wedge\ P \in \mathsf{Loc}_\Sigma \tag{27}$$

$$P \in \mathsf{Avail}_\Sigma \ \Rightarrow\ \forall u \in \mathbb{S}.\ P_u \in \mathsf{Live}_{\Sigma_u} \tag{28}$$

b) Authorization is localized global safety, whereas availability is implied by global liveness, which is implied by localized availability:

$$\mathsf{Auth}_\Sigma \ = \ \mathsf{Safe}_\Sigma \cap \mathsf{Loc}_\Sigma \tag{29}$$

$$\mathsf{Avail}_\Sigma \ \subseteq \ \mathsf{Live}_\Sigma \tag{30}$$

Proof. ***a)*** Since $\mathsf{Auth}_\Sigma \subseteq \mathsf{Loc}_\Sigma$ follows from (25) for $\vec{z} = \vec{x}$, proving (27) boils down to showing that the following two implications are equivalent

$$\left(\forall u \in \mathbb{S}.\ \left(\vec{x}::\vec{y}\right)\restriction_u \in P_u\right)\ \Longrightarrow\ \left(\forall u \in \mathbb{S}.\ \vec{x}\restriction_u \in P_u\right) \qquad \text{— which means} \qquad P \in \mathsf{Auth}_\Sigma$$

$$\left(\forall u \in \mathbb{S}.\ \left(\vec{x}_u::\vec{y}_u\right) \in P_u\ \Longrightarrow\ \vec{x}_u \in P_u\right) \qquad \text{— which means}\ \ \forall u \in \mathbb{S}.\ P_u \in \mathsf{Safe}_{\Sigma_u}$$

for all $\vec{x}, \vec{y} \in \Sigma^*$ and all $\vec{x}_u, \vec{y}_u \in \Sigma_u^*$. The bottom-up direction is valid for all predicates in first-order logic: the second implication is stronger than the first one. Towards the top-down direction, fix a subject A, take arbitrary histories $\vec{x}_A, \vec{y}_A \in \Sigma_A^*$ such that $\left(\vec{x}_A :: \vec{y}_A\right) \in P_A$, and set $\vec{x} = \vec{x}_A$ and $\vec{y} = \vec{y}_A$. For an arbitrary subject $u \in \mathbb{S}$ we have

$$\left(\vec{x}::\vec{y}\right)\restriction_u \ = \ \left(\vec{x}_A :: \vec{y}_A\right)\restriction_u \ = \ \begin{cases} \left(\vec{x}_A :: \vec{y}_A\right) \in P_A & \text{if } u = A \\ () \in P_u & \text{if } u \neq A \end{cases}$$

Since $() \in P_u$ follows from part *(b)* below, and $\left(\vec{x}_A :: \vec{y}_A\right) \in P_A$ was assumed, we have $\forall u \in \mathbb{S}.\ \vec{x} \restriction_u \in P_u$, as claimed. Towards (28), we need to show that the first of the following statements implies the second:

$$\begin{array}{lll} \forall \vec{x} \in \Sigma^*\ \exists \vec{y}_u \in \Sigma_u^*.\ \left(\vec{x} :: \vec{y}_u\right) \in P & \text{— which means} & P \in \mathsf{Avail}_\Sigma \\ \forall \vec{x}_u \in \Sigma_u^*\ \exists \vec{y}_u \in \Sigma_u^*.\ \left(\vec{x}_u :: \vec{y}_u\right) \in P_u & \text{— which means} & \forall u \in \mathbb{S}.\ P_u \in \mathsf{Live}_{\Sigma_u} \end{array}$$

But the second is just a special case of the first one, obtained by taking $\vec{x} = \vec{x}_u$, and noting that $\vec{x}_u :: \vec{y}_u \in P$ implies $\vec{x}_u :: \vec{y}_u = \left(\vec{x}_u :: \vec{y}_u\right) \restriction_u \in P_u$.

b) Towards (29), consider

$$\begin{array}{rcll} \left(\forall u \in \mathbb{S}.\ \left(\vec{x}::\vec{y}\right) \restriction_u \in P_u\right) & \Longrightarrow & \vec{x} \in P & \text{— which means } P \in \mathsf{Auth}_\Sigma \\ \left(\vec{x}::\vec{y}\right) \in P & \Rightarrow & \vec{x} \in P & \text{— which means } P \in \mathsf{Safe}_\Sigma \end{array}$$

The fact that $\mathsf{Auth}_\Sigma \subseteq \mathsf{Safe}_\Sigma$ means that the second implication follows from the first one. This is true because $\left(\vec{x}::\vec{y}\right) \in P$ implies $\left(\vec{x}::\vec{y}\right) \restriction_u \in P_u$ for all u. The converse clearly also holds as soon as P is local.

And (30) is obvious, since $\vec{y}_u \in \Sigma_u^* \subseteq \Sigma^*$. □

Example 1: Authority and availability of sheep and oil. To model sheep security, we zoom out again, and go back to the situation where Alice and Bob need to share some of their resources. The subject type is thus $\mathbb{S} = \{A, B\}$, the objects are from $\mathbb{J} = \{\text{sheep}, \text{oil}\}$, and the actions are $\mathbb{A} = \{\text{shear}, \text{cook}\}$. The possible events in the simple form of eq:event would thus be all triples from $\mathbb{A} \times \mathbb{J} \times \mathbb{S}$. But since Alice and Bob will not shear the oil or cook the sheep, we take[4]

$$\Sigma \quad = \quad \left\{\text{shear sheep}_A, \text{cook oil}_B\right\}$$

Consider the following properties of Alice's and Bob's interactions:

$$\text{Either} \quad = \quad \text{shear sheep}_A^* \cup \text{cook oil}_B^* \tag{31}$$

$$\text{Alternate} \quad = \quad \left(\text{shear sheep}_A :: \text{cook oil}_B\right)^* \tag{32}$$

$$\text{Finish} \quad = \quad \left\{\text{shear sheep}_A, \text{cook oil}_B\right\}^* :: \text{shear sheep}_A :: \text{cook oil}_B \tag{33}$$

These properties provide counterexamples for the converses of the claims of Prop. 5:

- (31) Either $\in \mathsf{Safe}_\Sigma \setminus (\mathsf{Auth}_\Sigma \cup \mathsf{Loc})$;
- (32) Alternate $\in \left(\mathsf{Live}_{\Sigma_A} \cap \mathsf{Live}_{\Sigma_B}\right) \setminus \mathsf{Live}_\Sigma$;
- (33) Finish $\in \mathsf{Live}_\Sigma \setminus \mathsf{Avail}_\Sigma$.

Example 2: Authorization and availability of elevator. To model security of the elevator, we consider the events from the point of view of the subjects, i.e. in the form

$$\Sigma \quad = \quad \mathbb{J} \times \mathbb{A} \times \mathbb{S} \quad = \quad \left\{ \langle i \rangle_u, (i)_u \;\middle|\; i \in \mathbb{J}, u \in \mathbb{S} \right\}$$

where

- $\mathbb{J} = \left\{0, 1, 2, \ldots, n\right\}$ are the objects again: the floors of the building (denoted by variables $x_0, x_1, \ldots$),

[4] See the remark about *Standard and refined* Σ *types* in Sec. 1.1 The Σ type used here is in a refined form of (2).

- $\mathbb{A} = \{\langle\,\rangle, (\,)\}$ are the actions: "call/go to" and "arrive to", respectively; and moreover
- $\mathbb{S} = \{A, B\}$ are the subjects: Alice and Bob (denoted by variables $Y_0, Y_1, \ldots$)

A history is now in the form

$$\left[\, \langle\, x_0 \,\rangle_{Y_0} \langle\, x_1 \,\rangle_{Y_1} (\, x_2 \,)_{Y_2} \langle\, x_0 \,\rangle_{Y_0} (\, x_0 \,)_{Y_2} \ldots \,\right]$$

meaning that "Y_0 calls to x_0, Y_1 calls to x_1, Y_2 arrives to x_2, Y_0 calls to x_0 again , Y_2 arrives to $x_0 \ldots$". The dependability properties (which required that the elevator should *only* go where called, and that it should *eventually* go where called) are now refined to

- ***authorization:*** the elevator should *only* take *some* subject to a floor if they have a clearance and if they requested it, and
- ***availability:*** the elevator should *eventually* take *every* subject with a clearance for the floor that they requested.

which generalizes Example 2 from Sec. **??** to

$$\begin{array}{lcll} \overline{(\,i\,)_u} & \subseteq & \overline{\exists \langle\, i \,\rangle_u \prec (\,i\,)_u} & \rightsquigarrow \textit{authority} \\ \overline{\langle\, i \,\rangle_u} & \subseteq & \overline{\langle\, i \,\rangle_u \prec \exists (\,i\,)_u} & \rightsquigarrow \textit{availability} \end{array}$$

Note however that here we need to assume that the subject u is somehow authorized to go to the floor i. This is where the static resource security formalism from Chap. **??** need to be imported into the dynamic resource security formalisms of histories and properties. In terms of multi-level security, the clearance assumption would be $c\ell(u) \geq p\ell(i)$. In terms of permission matrices, it would be $\langle -\rangle, (-) \in M_{ui}$. Writing for simplicity either of these assumptions as the predicate $C\ell(u, i)$, the above crude idea of authorization and availability properties of the elevator can be formalized to

$$\text{AuthElev} = \left\{ \vec{t} \in \Sigma^* \mid \forall u \in \mathbb{S}\ \ \forall i \in \mathbb{J}.\ \ \vec{t} \in \overline{(\,i\,)_u} \implies \left(C\ell(u, i) \wedge \vec{t} \in \overline{\exists \langle\, i \,\rangle_u \prec (\,i\,)_u} \right) \right\} \tag{34}$$

$$\text{AvailElev} = \left\{ \vec{t} \in \Sigma^* \mid \forall u \in \mathbb{S}\ \ \forall i \in \mathbb{J}.\ \ \left(\vec{t} \in \overline{\langle\, i \,\rangle_u} \wedge C\ell(u, i) \right) \implies \vec{t} \in \overline{\langle\, i \,\rangle_u \prec \exists (\,i\,)_u} \right\} \tag{35}$$

The reason why $C\ell(u, i)$ occurs on the left of the implication for authorization, and on the right for availability is that the lift should be available to u for i *only if* $C\ell(u, i)$ holds, and u should be authorized to go to i *whenever* $C\ell(u, i)$ holds. Recall from Ch. **??** that permissions and clearances may be declared differently for different states of the system, which means that the clearance predicate would be in the more general form $C\ell\left(\vec{t}, u, i\right)$, which makes it dependent on the history $\vec{t}$. Alice could thus have different authorizations in different historic contexts.

Example 3: Questions and answers. Suppose that Alice and Bob are having a conversation: one asks a question, the other one answers, or asks another question. We denote the action of asking a question by "?", and the action of answering a question by "!". To distinguish between different questions, and to identify the question that can be repeated, and to bind questions and the corresponding answers, we assume that there is a fixed set of questions $\mathbb{J}$, which we take to be a set of numbers. Thus we have the types

- $\mathbb{J} = \{0, 1, 2, \ldots, n\}$ are the questions that may be asked or answered (viewed as objects, and denoted by variables $x_0, x_1, \ldots$ again),
- $\mathbb{A} = \{?, !\}$ are the actions: "ask question" and "answer question", respectively; and moreover
- $\mathbb{S} = \{A, B\}$ are the subjects: Alice and Bob (denoted by variables $Y_0, Y_1, \ldots$)

which induce the event space in the form

$$\Sigma = \mathbb{S} \times \mathbb{J} \times \mathbb{A} = \left\{ u{:}i?,\ u{:}i! \;\middle|\; u \in \mathbb{S},\ i \in \mathbb{J} \right\}$$

with histories as sequences of events such as

$$\left[\, Y_0 : x_0 ?\quad Y_1 : x_1 ?\quad Y_2 : x_2 !\quad Y_0 : x_0 ?\quad Y_2 : x_2 ! \ldots \,\right]$$

which says that "Y_0 asks the question x_0, then Y_1 asks the question x_1, then Y_2 answers the question x_2, then Y_0 asks x_0 again, then Y_2 answers x_0", etc. Formally, such conversations between Alice and Bob are of course similar to the operations of the elevator that they share in their building.

The difference is, of course, that when Alice calls the elevator, she usually wants to use the service herself; whereas when she asks a question, then she usually expects Bob to provide an answer. This leads to the following properties that may be required in a formal conversation, or perhaps interrogation:

- ***all answers questioned:*** an answer should *only* be given for a question previously asked by someone, and
- ***all questions answered:*** every question that was asked will *eventually* be answered by someone.

Questions and answers in a conversation are matched similarly like calls and services of an elevator, but the matching is slightly more general:

$$\begin{array}{lcll} \overline{u : i !} & \subseteq & \overline{\exists w : i ? \prec u : i !} & \rightsquigarrow \textit{ answers questioned} \\ \overline{u : i ?} & \subseteq & \overline{u : i ? \prec \exists w : i !} & \rightsquigarrow \textit{ questions answered} \end{array}$$

A clearance predicate could moreover be used to constrain who is permitted to ask which questions, and who is permitted to answer them. A predicate in the form $C\ell(u, i, a)$, meaning that the subject u has a permission to perform on the object i the action a, would thus correspond to the declaration $a \in M_{ui}$ in a permission matrix M. The requirements that all answers should be preceded by corresponding questions, and that all questions should eventually be answered, can be refined to

$$\begin{aligned} \text{AskQ} \quad = \quad & \Big\{ \vec{t} \in \Sigma^* \mid \forall u \in \mathbb{S} \;\; \forall i \in \mathbb{J}. \\ & \qquad \Big(\vec{t} \in \overline{u : i !} \;\wedge\; C\ell(u, i, !) \Big) \implies \exists w \in \mathbb{S}. \; \Big(C\ell(w, i, ?) \;\wedge\; \vec{t} \in \overline{\exists w : i ? \prec u : i !} \Big) \Big\} \end{aligned} \tag{36}$$

$$\begin{aligned} \text{AnsQ} \quad = \quad & \Big\{ \vec{t} \in \Sigma^* \mid \forall u \in \mathbb{S} \;\; \forall i \in \mathbb{J}. \\ & \qquad \Big(\vec{t} \in \overline{u : i ?} \;\wedge\; C\ell(u, i, ?) \Big) \implies \exists w \in \mathbb{S}. \; \Big(C\ell(w, i, !) \;\wedge\; \vec{t} \in \overline{u : i ? \prec \exists w : i !} \Big) \Big\} \end{aligned} \tag{37}$$

The fact that AskQ does not guarantee authority and that AnsQ does not guarantee availability points to the limitations of the presented formalizations of authority and availability as tools of resource security. A way to mitigate some of these limitations is explored in exercises below. We shall return to this topic through authentication, in the next chapter.

Remark. The specifications presented here have been chosen to be as simple as possible. They are therefore in some cases quite crude. E.g., the precedence relation in $\exists \langle\, i \,\rangle \prec (\, i \,)$ does not discharge the calls that the elevator has responded to, and thus allows that after a single call to the floor i, the elevator may return there any number of times, while remaining formally safe. Such quirks are discharged in more realistic specifications, which are however significantly more complex.

Exercises. Prove that

a) the properties defined in (32–(33)) are as claimed;

b) the properties defined in (34–35) satisfy AuthElev $\in$ Auth $\setminus$ Avail and AvailElev $\in$ Avail $\setminus$ Auth.

c) the following properties are authorizations:

i. $\neg\overline{\langle\, n \,\rangle_A}$

ii. $\neg\overline{a \prec b}$

iii. $\neg\vec{a}_A\Sigma_A^*$

iv. $\vec{a}_A\Sigma_A^*$

d) the following properties are availabilities:

i. $\overline{\bigcup_{C\ell(u,i)} \langle\, i \,\rangle_u}$

ii. $\overline{(\, n \,)_A}$

iii. $\neg\overline{\langle\, n \,\rangle_A}$

iv. $\Sigma^* a_0 a_1 \cdots a_n$

v. $\neg\Sigma^* a \Sigma^*$

for all $\vec{a} \in \Sigma$.

e) Consider the properties defined in (36) and (37).

i. Prove that AskQ is a safety property, and that AnsQ is a liveness property.

ii. Determine a sufficient condition on the predicate $C\ell$ that will guarantee that AnsQ is an availability property.

iii. Determine a sufficient condition on the predicate $C\ell$ that will guarantee that AskQ is an authorization.

iv. What happens if we prohibit rhetorical questions, i.e. requirement $w \neq u$ in (36) and (37), so that the subjects are not allowed to answer their own questions? What happens if we require that everyone answers their own questions?

Interpret and discuss each of the above answers.

Is it possible that both a property and its negation (i.e. complement) are authorization properties? Can they both be availability properties? Can a property be both an authorization property and an availability property?

Relativity in security. Answering the questions in Sec. **??**, we saw that for every trace $\vec{a} \in \Sigma^*$, both the property $\vec{a}\Sigma^*$ and its complement $\neg\left(\vec{a}\Sigma^*\right)$ are safety properties. Answering the questions above, we see that they are both authorizations as well. We explained that this does not mean that a process can be both safe and unsafe, or both authorized and unauthorized at the same time, but that it just captures the fact that in some cases two opposite and thus complementary properties may be desirable from two opposite points of view, and that each of them can be reasonably declared as a safety requirement, or as an authority requirement. For the latter, this is not just reasonable, but also necessary for capturing real situations. E.g., if Alice is Bob's commanding officer, she needs to be able to authorize him to use a weapon and fight in one situation; and she also needs to be able ti authorize him to not use the weapon, and to surrender in another situation.

Neither safety nor authority are objective or absolute properties of a history. They are declared as desirable properties by some subjects in some situations. In different situations, the same subjects may declare different properties as desirable. In the same situation, different subject may declare different properties as desirable. Both Alice and Bob may claim authority over the same resource. Security science does not prescribe solutions for such conflicts. It only describes the processes in which they arise.

Conflicting availability claims are even more interesting: the availability of a resource to Alice may preempt its availabilty to Bob. This gives rise to the simplest instance of an *attack*: Bob may request a service merely to make it unavailable to Alice. We deal with such attacks in the next section.

3 Denial-of-Service

Although liveness is not a security property, and is generally not a matter of conflict, Alice and Bob may pursue different *nice things* that they want to happen. If each of them pursues different *nice things*, and Alice's *nice things* preclude Bob's *nice things*, then a system that is alive for Alice will be dead for Bob.

If moreover Alice's liveness property happens to be availability, so that every subject can secure it on their own, then Alice can make her *nice things happen*, which then means that Bob's opposite nice things will not happen. In such situations, we say that Alice has launched a *Denial-of-Service (DoS)* attack against Bob.

In general, a DoS phenomenon occurs when both a property P and its complement $\neg P$ are liveness properties, and a history that is alive in one sense must be dead in another sense. When the property $\neg P$ is moreover *available* for some subjects, then those subjects can cause a *Denial-of-Service P*.

Example 3 again: More questions and answers. One thing we didn't mention so far is Alice's and Bob's ages. They are in fact 2 years old. They learned to speak very recently, and they are trying to learn the rules of conversation. While Alice believes that all questions should be answered, Bob's standpoint is that all answers should be questioned.

For Alice, a conversation is alive only when every question can be answered by someone. That is when Alice's requirement is a liveness property. It is moreover an availability property when everyone knows answers to all questions. Only then can anyone satisfy Alice's requirement.

Bob's view is, on the other hand, that a conversation is only alive when there are some unanswered questions. Bob's requirement is *also* a liveness property whenever for any question there is someone who is permitted to ask it. If moreover everyone is permitted to ask any question, then Bob's requirement is even an availability property.

Formally, Alice's requirement is thus $\text{Service}_A = \text{AnsQ}$, whereas Bob's requirement is opposite: $\text{Service}_B = \neg\text{AnsQ}$.

On the other hand, using (37) and the definitions in the sections preceding it, if is can be proved that

$$\begin{aligned}
\forall i \in \mathbb{J}\ \exists w \in \mathbb{S}.\ C\ell(w, i, !) &\quad\Rightarrow\quad \text{AnsQ} \in \mathsf{Live} \\
\forall i \in \mathbb{J}\ \forall w \in \mathbb{S}.\ C\ell(w, i, !) &\quad\Rightarrow\quad \text{AnsQ} \in \mathsf{Avail} \\
\forall i \in \mathbb{J}\ \exists w \in \mathbb{S}.\ C\ell(w, i, ?) &\quad\Rightarrow\quad \neg\text{AnsQ} \in \mathsf{Live} \\
\forall i \in \mathbb{J}\ \forall w \in \mathbb{S}.\ C\ell(w, i, ?) &\quad\Rightarrow\quad \neg\text{AnsQ} \in \mathsf{Avail}
\end{aligned}$$

If there is a particular question $q \in \mathbb{J}$ that Bob is permitted to ask, in the sense that $C\ell(B, q, ?)$ holds true, then the property $\text{Service}_B = \neg\text{AnsQ}$ is available to him, as he can extend any history of questions $\vec{t} \in \Sigma^*$ to $\vec{t} :: (B\!:\!q?) \in \neg\text{AnsQ} = \text{Service}_B$. Since $\text{Service}_A = \text{AnsQ}$, Alice has herewith been denied the service of a final answer, that she normally requires from conversations.

If Alice is, on the other hand, permitted to answer the question q, in the sense that $C\ell(A, q, !)$ is true, then she can extend the current history to $\vec{t} :: (B\!:\!q?) :: (A\!:\!q!) \in \text{AnsQ} = \text{Service}_A$. Now Bob has been denied service.

Alice and Bob can continue to deny service to each other like this, until one of them finds a way to break the symmetry. E.g., Alice may be able to convince their parents Carol and Dave that the questions that have been answered in the past should not be asked again. Bob's Denial-of-Service attack will then depend on how many other questions he is permitted to ask, i.e. for how many different $i \in \mathbb{J}$ does he have a clearance $C\ell(B, i, ?)$. Whether he will win or not also depends on Alice's capability $C\ell(A, i, !)$ to anwer questions.

Example 4: TCP-flooding. The internet transport layer connections are established using the Transmission Control Protocol (TCP). Its rudimentary structure, displayed on the left in section 3, can be construed as an extension of the basic question-answer protocol, where asking and answering a question if followed by a final action, where the answer is accepted. In the TCP terminology, the action of asking a question is called SYN, the action of answering it is called SYN-ACK, and the acceptance of the answer is called ACK. Alice really likes this part, as it makes the TCP connections available, by closing the question-answer sessions with the answer acceptance. After an answer has been accepted, a TCP server establishes the TCP network socket, and releases the protocol state, i.e. forgets the question.

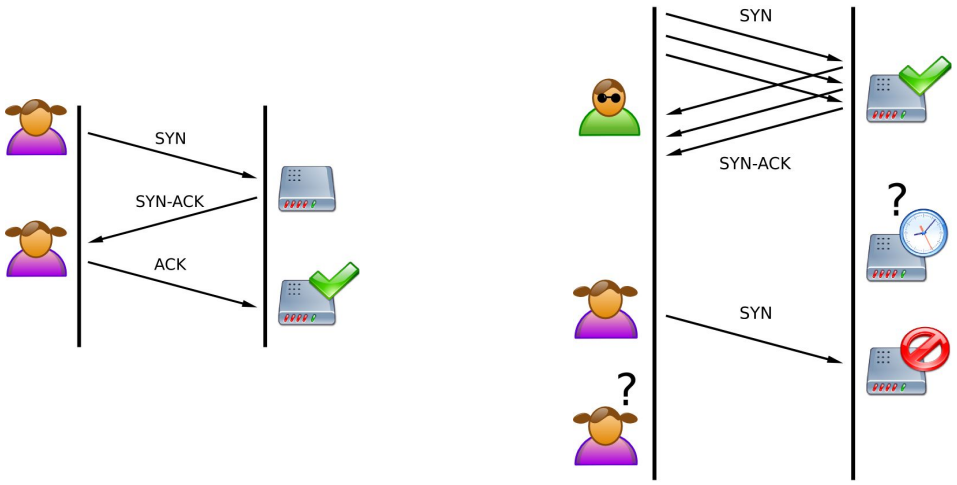


Figure 3: TCP: 3-way handshake and the SYN-flooding

If Bob, however, never accepts any answer, then the the TCP server has in principle to remember lots of questions, i.e. keeps lots of TCP protocol sessions open, and at one point runs out of memory. That is the SYN-flooding attack, displayed in section 3 on the right.

This basic idea of a DoS attack on the internet is very old, almost as old as the internet, and there are in the meantime many methods to prevent it; but there are even more methods to circumvent these preventions. DoS attacks are a big business, both on the internet, and in everyday conversations between the 2-year olds.

Exercises.

a) Just before stepping out of the elevator, 12 year old Johnny pushes all buttons, and the empty elevator proceeds all the way up, stopping at each floor, before becoming available to respond to the call from the ground floor, where Mrs Knaus is waiting for service to her penthouse. Model this situation as a DoS attack.

b) Bob and Charlie went one day to Alice's restaurant without a reservation, and didn't get a table because the restaurant was full. Bob took offense, and the next day he called Alice's restaurant repeatedly, and booked up many tables. Alice rejected the reservation requests from others, because there were no more tables available. But Bob did not show up to take the tables he booked. Dave tried to book a table, and was told that none were available, while many tables later remained empty.

   Suppose that Alice's restaurant has 10 tables that seat up to 4 guests, and 5 tables that seat up to 6 guests. Tasks:

   i. Assuming that the restaurant is continuously open, model a table reservation process where
      - each booking request consists a required number of guests, not greater than 6, and a desired service time;
      - each booking service assigns to a particular booking request a table with enough seats, that is available at the desired time.

   ii. Formulate the dependability requirements for the table reservation service:
      - *safety*: no service should be provided without a prior reservation;
      - *liveness*: every guest who reserves a table should eventually be assigned a table.

   iii. Show that the restaurant process can satisfy the safety and the liveness requirements, if it is continuously open.

   iv. Suppose that the restaurant opening hours are 4PM–1AM, that each booking request is for 1 hour, and that the start times are at full hour. Express these process constraints as a trace property.

v. Show that this restaurant with limited opening hours does not satisfy the liveness requirement.

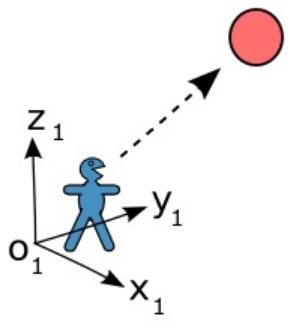
vi. Suppose that the restaurant is a part of corporate headquarters, providing services to authorized employees, with suitable access control credentials. Discuss the difference between the liveness failure under (e) and an availability failure caused by a DoS-attack from an authorized employee Bob.

c) Refine the question-answer model to provide a model of the venerable SYN-flooding attack, or of a contemporaneous version of DoS on the internet, as suitable for your interests and familiarity with network protocols.

4 Geometry of cyber space and computation

4.1 Geometry?

Geometry is the science of space. Space is the way we observe things. Our physical space has 3 dimensions, and we usually subdivide it into cubes, because we see and hear things up or down, left or right, forward or backward. Ants observe the world by way of smells and tastes, and their physical space is structured differently. Computers observe



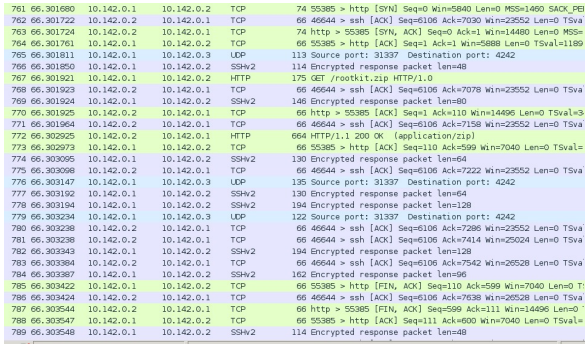


Figure 4: An observation in physical space vs an observation in cyber space

the world as traces of computations, as streams of data, as network logs, as listings of one sort or another. We have been calling such listings *histories*, because they have a known past and an unknown future, separated by the present.

4.2 Geometry of histories and properties

Observations. If we record a string of past events $\vec{a} = [\, a_0\, a_1\, a_2\, \cdots\, a_n\,] \in \Sigma^*$, then we have observed that our history has the property

$$\begin{array}{ccc} \vec{a} :: \Sigma^* & \in & \wp\mathcal{O}(\Sigma^*) \\ \uparrow & & \uparrow \\ past & & futures \end{array}$$

This is an *observation*. Observations are collected in the family of *basic* sets

$$\mathcal{B} \quad = \quad \left\{ \vec{a} :: \Sigma^* \mid \vec{a} \in \Sigma^* \right\} \tag{38}$$

Open observables represent possible observations of bad things that may happen. If such an observation occurs, it will persist, i.e. remain open under all futures. A family O of open observables is thus required to satisfy the following properties:

a) $\mathcal{B} \subseteq \mathcal{O}$ — any observation is open;

b) $\mathcal{X} \subseteq \mathcal{O} \;\Rightarrow\; \bigcup \mathcal{X} \in \mathcal{O}$ — any set of open observables corresponds to a possible observation, i.e. to an open observable;

c) $U, V \in \mathcal{O} \;\Rightarrow\; U \cap V \in \mathcal{O}$ — any finite set of open observables can be observed together, as a single open observable.

The set of observables $\mathcal{O}$ thus has to contain $\mathcal{B}$, and all unions of the elements of $\mathcal{B}$. It follows that

$$\mathcal{O} \quad = \left\{ U \in \wp\Sigma^* \mid \vec{x} \in U \wedge \vec{x} \sqsubseteq \vec{y} \Rightarrow \vec{y} \in U \right\} \tag{39}$$

because $U \in \wp\Sigma^*$ satisfies $vec x \in U \wedge \vec{x} \sqsubseteq \vec{y} \Rightarrow \vec{y} \in U$ if and only if it also satisfies $U = \bigcup\{\vec{a} :: \Sigma^* \subseteq U \mid \vec{a} \in \Sigma^*\}$. It is easy to see that $\mathcal{O}$ from (39) also satisfies (c), since already $\mathcal{B}$ is closed under $\cap$. According to (39), the observables $U \in \mathcal{O}$ are thus the *upper-closed* sets under the prefix order $\sqsubseteq$. This captures that they are *open* into all futures.

Closed observables represent observations that bad things did not happen so far; i.e., they are the complements $F = \neg U$ of open observables $U \in \mathcal{O}$. Since for $\vec{x} \sqsubseteq \vec{y}$ the implication $\vec{x} \in U \Rightarrow \vec{y} \in U$ is equivalent with $\vec{y} \notin U \Rightarrow \vec{x} \notin U$, it follows that the family of closed observables must be in the form

$$\mathcal{F} \quad = \quad \left\{ F \in \wp\Sigma^* \mid \vec{x} \sqsubseteq \vec{y} \wedge \vec{y} \in F \Rightarrow \vec{x} \in F \right\} \tag{40}$$

The closed observables are thus the *lower-closed* sets under the prefix ordering. This captures that they are *closed* under the past: if bad things did not happen now, than that statement was also true at any moment in the past.

Dense observables are those that always remain possible: an observable is dense if it must be observed eventually, after any future-open observation. Each of the three previously defined families of properties can be used to characterize dense properties, which leads to three equivalent characterizations:

$$\mathcal{D} \quad = \quad \left\{ D \in \wp\left(\Sigma^*\right) \mid \forall \vec{a} :: \Sigma^*.\, D \cap \left(\vec{a} :: \Sigma^*\right) \neq \emptyset \right\} \tag{41}$$

$$= \quad \left\{ D \in \wp\left(\Sigma^*\right) \mid \forall U \in \mathcal{O}.\, D \cap U = \emptyset \Rightarrow U = \emptyset \right\} \tag{42}$$

$$= \quad \left\{ D \in \wp\left(\Sigma^*\right) \mid \forall F \in \mathcal{F}.\, D \subseteq F \Rightarrow F = \Sigma^* \right\} \tag{43}$$

Dense observations are used to characterize the nice things that are required to eventually happen. The third characterization says that if nice things are not bad, then nothing is bad; i.e., if a dense property is past-closed, then it contains all histories. The definition is then relativized to any $Y \subseteq \Sigma^*$ as

$$\mathcal{D}_Y \quad = \quad \left\{ D \in \wp(Y) \mid \forall F \in \mathcal{F}.\, D \subseteq F \Rightarrow Y \subseteq F \right\} \tag{44}$$

4.3 Geometry of safety and liveness

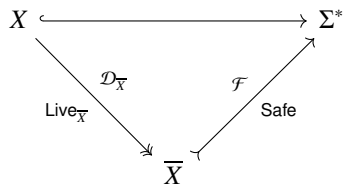
Proposition 6. *A history is*

a) safe if and only if it is closed: $\mathsf{Safe} = \mathcal{F}$

b) alive if and only if it is dense: $\mathsf{Live} = \mathcal{D}$

Proof. By inspection of (12) $\overset{\text{(a)}}{\Longleftrightarrow}$ (40) and (13) $\overset{\text{(b)}}{\Longleftrightarrow}$ (41). □

Corollary 7. *The closure operator* $\overline{=} : \wp(\Sigma^*) \longrightarrow \wp(\Sigma^*)$ *factors any property* $X \subseteq \Sigma^*$ *through* $\overline{X} = \left\{ \vec{y} \sqsubseteq \vec{x} \in X \right\}$



where $\overline{X} \subseteq \Sigma^*$ is closed and $X \subseteq \overline{X}$ is dense.

Requirements are future-open properties. Positive properties, e.g. in the form $\overline{b \prec c}$, can be expressed as unions of basic properties, e.g.

$$\overline{b \prec c} \quad = \quad \bigcup_{\vec{x},\vec{y} \in \Sigma^*} \vec{x} :: b :: \vec{y} :: c :: \Sigma^* \ \in \ O$$

and thus remain open. Further properties can be expressed as fintite intersections of those, e.g.

$$\overline{b_0 \prec b_1 \prec \cdots b_n} \quad = \quad \overline{b_0 \prec b_1} \cap \overline{b_1 \prec b_2} \cap \cdots \overline{b_{n-1} \prec b_n} \ \in \ O$$

Constraints are past-closed, because they correspond to negative requirements, e.g.,

$$\bigcap_{i \in \mathbb{J}} \neg \overline{i! \prec i?} \in \mathcal{F} \quad \text{because} \quad \bigcup_{i \in \mathbb{J}} \overline{i! \prec i?} \in O$$

This explains why safety properties in Sec. 1.2 often occur as negative requirements.

4.4 Geometry of security

Security is all about what the different points of view. We discussed in Sec. 2.1 how security problems arise from local observability, and hiding. I can deceive you if there is something that I can see and you cannot. Physical security

Figure 5: Global observer: "I think, therefore I exist". Local observers: relativistic frames of reference.

prevents burglaries when no one is home. Cyber security requires reconciling Alice's and Bob's different views of histories. National security also requires reconciling different views of histories, although in that context, the histories that are being viewed are much more specific, and harder to model. Behind every security problem, there are different views of some space of histories.

The difference between cyber security and physical security is the difference between the underlying geometries. Physical security is based on physical distances and velocities: smaller animals prevent attacks from bigger animals by keeping a distance, which will allow them to run away. Cyber security cannot be based on outrunning the attacks, because the concept of distance in cyber space is unreliable, as electronic networks have unreliable speeds, and unpredictable optimizations. If the implementation details of network services are abstracted away, then every two nodes in cyber space look like neighbors, because the underlying networks in principle do their best to route traffic as fast as possibles. Many cyber security problems are direct consequences of such geometric peculiarities of cyber space.

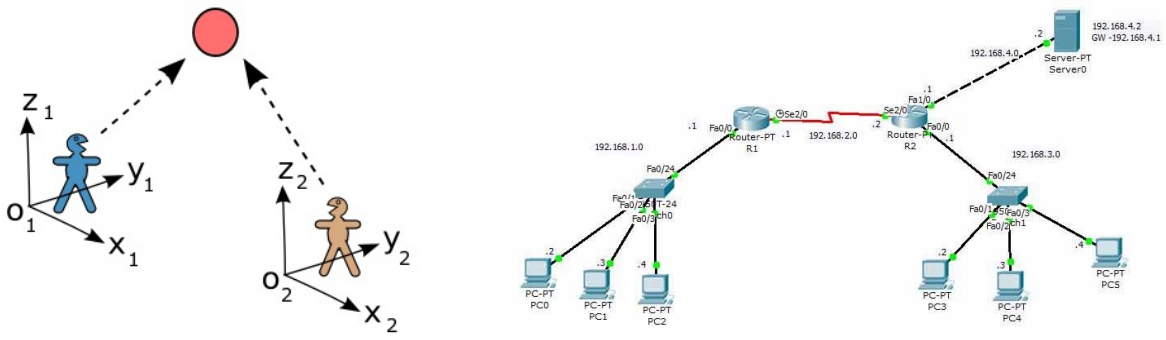


Figure 6: Linking and separating frames of reference in physical space and in cyber space

4.5 Locality and cylinders

In Sec. 2.2, we analyzed how authority and availability be construed as *localized* versions of safety and liveness. Here we spell out the geometric meaning of that observation. It is based on interpreting the purge operations $\restriction_u : \Sigma^* \longrightarrow \Sigma_u^*$ (50) as spacial projections, that reduce global histories to local views. Instantiating *cylinder localizations* from Sec. **??** to the family of views $V = \left\{ \restriction_u : \Sigma^* \longrightarrow \Sigma_u^* \mid u \in \mathbb{S} \right\}$, and then we explore how well can arbitrary properties be approximated by the corresponding cylinders, which are subjects' views.

Lemma 8. *A property is localized in the sense of Def. 3 if and only if it is external cylindric in the sense of Def. 15 in the Prerequisites:*

$$\mathsf{Loc} \;=\; \mathsf{ExCyl}$$

Proof. We prove that the localization $\widehat{Y}$ from Def. 3 is the special case of cylindrification $[\![Y]\!]$ from Def. 15:

$$\widehat{Y} \;=\; \left\{ \vec{y} \mid \forall u \in \mathbb{S}.\, \vec{y}\restriction_u \in Y_u \right\} \;=\; \bigcap_{u \in \mathbb{S}} \left\{ \vec{y} \mid \vec{y}\restriction_u \in Y_u \right\} \;=\; \bigcap_{u \in \mathbb{S}} u^* u_!(Y) \;=\; [\![Y]\!]$$

Hence $\mathsf{Loc} = \{Y \in \wp\mathcal{O}(\Sigma^*) \mid Y = \widehat{Y}\} = \{Y \in \wp\mathcal{O}(\Sigma^*) \mid Y = [\![Y]\!]\} = \mathsf{ExCyl}$. □

4.6 Geometry of authority and availability

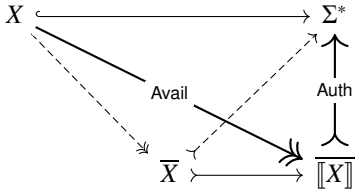
Proposition 9. *A history is*

a) authorized if and only if it is closed and cylindric: $\mathsf{Auth} \;=\; \left\{ P \in \wp\mathcal{O}(\Sigma^*) \mid P \in \mathcal{F} \wedge P = [\![P]\!] \right\} \;=\; [\![\mathcal{F}]\!]$

b) available if and only if its cylindrification is dense: $\mathsf{Avail} \;=\; \left\{ P \in \wp\mathcal{O}(\Sigma^*) \mid [\![P]\!] \in \mathcal{D} \right\} \;=\; [\![\mathcal{D}]\!]^{-1}$

Proof. (a) follows from (29) and Lemma 8. (b) follows from (24). □

Corollary 10. *The closure operator* $\overline{[\![-]\!]} : \wp\mathcal{O}(\Sigma^*) \longrightarrow \wp\mathcal{O}(\Sigma^*)$ *factors any property* $X \subseteq \Sigma^*$ *through its cylindric closure* $\overline{[\![X]\!]} = \left\{ \vec{y} \sqsubseteq \vec{x} \mid \forall u \in \mathbb{S}.\, \vec{x}\restriction_u \in X_u \right\}$



where $\llbracket\overline{X}\rrbracket \subseteq \Sigma^*$ *is closed cylindric whereas* $X \subseteq \llbracket\overline{X}\rrbracket$ *is dense cylindric.*

Remark. The closure operator $\overline{\llbracket - \rrbracket} : \wp(\Sigma^*) \longrightarrow \wp(\Sigma^*)$ is the composite of $\overline{(-)} : \wp(\Sigma^*) \longrightarrow \wp(\Sigma^*)$, defined in (55) and $\llbracket - \rrbracket : \wp(\Sigma^*) \longrightarrow \wp(\Sigma^*)$, defined in (53), both among the prerequisites. It is easy to check that $\llbracket\overline{X}\rrbracket = \overline{\llbracket X \rrbracket}$.

4.7 Normal decompositions of properties

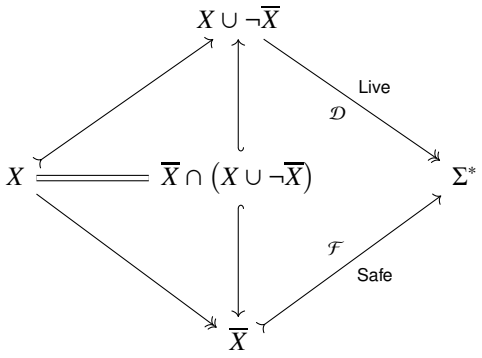
Corollaries 7 and 10 establish how any specified constraint $X \subseteq \Sigma^*$ can be approximated by a safety constraint $\overline{X}$ or by an authority constraint $\llbracket\overline{X}\rrbracket$. This is useful because safety constraints and authority constraints are generally easier to implement and validate than arbitrary constraints.

Safety-driven decompositions. The closure $\overline{X}$ in general declares more histories to be safe than intended by X. This deviation of $\overline{X}$ from X can, however, always be corrected by intersecting $\overline{X}$ with a liveness property in such a way that the intersection contains precisely the original property X. The histories that are in $\overline{X}$ but do not in X will thus be declared safe, but not alive. This is a special case of the geometric decomposition from Sec A.3 of arbitrary sets into closed and dense sets.

Proposition 11. *Any property can be expressed as an intersection of a safety property and a liveness property:*

$$X \quad = \quad \overline{X} \cap \left(X \cup \neg\overline{X}\right)$$

Proof. Recalling that in the space Σ^* of traces the closure is $\overline{X} = \{\vec{y} \sqsubseteq \vec{x} \in X\}$, we have



□

Example 1: sheep life. If Alice wants to shear her sheep at most once every year, and to only use meat in the end, the requirement might be

$$\begin{array}{rcll} \text{SheepLife} & = & \text{MilkWoolMeat} \cap \text{MilkMeatWoolAnnual} & \text{— where} \\ \text{MilkWoolMeat} & = & \{\text{milk}, \text{wool}\}^* :: \text{meat} & \\ \text{MilkMeatWoolAnnual} & = & \{\text{milk}, \text{meat}\}^* \cup \big[\underbrace{\text{milk}\,\text{milk}\,\ldots\,\text{milk}}_{\geq 365 \text{ times}}\,\text{wool}\big] :: \text{MilkWoolAnnual} & \end{array}$$

The normal decomposition is then

$$\text{SheepLife} \quad = \quad \overline{\text{SheepLife}} \cap \Big(\text{SheepLife} \cup \underline{\text{NoSheepLife}}\Big)$$

where

- $\overline{\text{SheepLife}}$ consists of all histories where sheep's wool is only ever used after 365 consecutive milkings, and where sheep's meat is only ever used at the end, after which there are no further events; and
- $\underline{\text{NoSheepLife}} = \neg\overline{\text{SheepLife}}$ consists of all histories where the rules of $\overline{\text{SheepLife}}$ are broken.

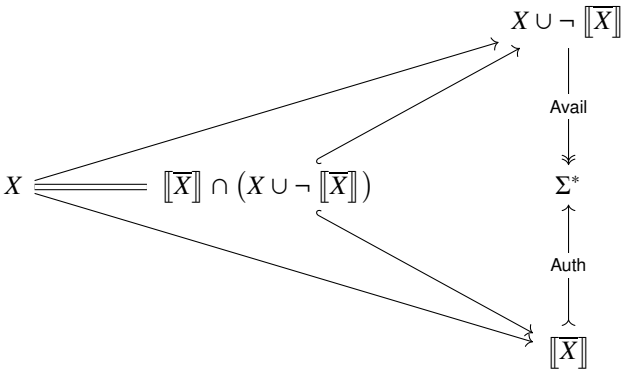
Remark. Note that the liveness part of the SheepLife decomposition is thus unsafe, whereas the safety part of the decomposition is not alive. It is, of course, reasonable to expect this. Yet when the liveness part and of the safety part are tested independently, this sometimes causes problems.

Authority-driven property decompositions. The cylinder closure $[\![\overline{X}]\!]$ in Corollary 10 is the smallest authorization property containing all histories from a given $X \subseteq \Sigma^*$. If a security designer wants to make sure that all histories contained in X are authorized, he will thus use $[\![\overline{X}]\!]$. Some of the histories authorized in $[\![\overline{X}]\!]$ may not be in X; but they can be eliminated as unavailable, by intersecting $[\![\overline{X}]\!]$ with an availability property, as spelled out in Prop. 12. The desired property X now remains as the set of precisely those histories that are both authorized and available, just like it remained as the set of histories that are both safe and alive in the safety-driven decomposition above.

Proposition 12. *Any property can be expressed as an intersection of authorization and availability:*

$$X \quad = \quad [\![\overline{X}]\!] \cap \left(X \cup \neg\, [\![\overline{X}]\!]\right)$$

Proof. Recalling that the cylinder closure is $[\![\overline{X}]\!] = \{\vec{y} \in \Sigma^* \mid \exists \vec{x} \in X\ \forall u \in \mathbb{S}.\ \vec{y}\restriction_u \sqsubseteq \vec{x}\restriction_u\}$, and expanding the diagram from Corollary 10, we have



□

Availability-driven decomposition. The decomposition in Prop. 12 is authority-driven because it is obtained by embedding the desired property X into the smallest authorization property $[\![\overline{X}]\!]$ that contains it. One of the equivalent logical forms of authority derived in Sec. 2.2 was

$$\forall \vec{x}\vec{z} \in \Sigma^*.\ \ \vec{x} \notin F\ \wedge \vec{x} \sqsubseteq \vec{z}\ \ \Rightarrow\ \ \exists u \in \mathbb{S}.\ \vec{z}\restriction_u \notin F_u \tag{45}$$

It says that any unauthorized event will be observed by someone, independently on others. ***No cooperation is needed.*** On the other hand, the corresponding availability property

$$\forall \vec{x} \in \Sigma^*\ \forall u \in \mathbb{S}\ \ \exists \vec{z} \in \Sigma^*.\ \left(\vec{x}\restriction_u \sqsubseteq \vec{z}\restriction_u\ \wedge\ \vec{z} \in D\right) \tag{46}$$

says that in any situation there is a future for all *together*. More precisely, it says that Alice can on her own can find $\vec{y}_A$ such that $\vec{x}\restriction_B :: \vec{y}_A$ is in D_A; and Bob can find $\vec{y}_B$ on his own, such that $\vec{x}\restriction_B :: \vec{y}_B$ is in D_B; but they must come together to find $\vec{z} \in D$ such that $\vec{z}\restriction_A = \vec{x}\restriction_B :: \vec{y}_A$ and $\vec{z}\restriction_B = \vec{x}\restriction_B :: \vec{y}_B$. Finding such $\vec{z}$ requires scheduling local histories like in Lemma 14 in Prerequisites. ***This requires cooperation.***

A *strong availability* requirement, strengthening (46) so that the required actions can be realized by individual subjects with no need for cooperation is

$$\forall \vec{x} \in \Sigma^* \; \forall \vec{z} \in \Sigma^* \; \exists u \in \mathbb{S}. \Big(\vec{x}\restriction_u \sqsubseteq \vec{z}\restriction_u \;\Rightarrow\; \vec{z} \in D \Big) \tag{47}$$

saying there is someone for whom a future is available *independently on others*.

Proposition 13. *Any property can be expressed as a union of a strong availability and an authority breach:*

$$X \;=\; (\!|\underline{X}|\!) \;\cup\; \Big(X \cap \neg (\!|\underline{X}|\!) \Big) \tag{48}$$

Proof. The largest strong availability property contained in X is

$$(\!|\underline{X}|\!) \;=\; \big\{ \vec{y} \mid \exists u \in \mathbb{S}.\, \vec{y}\restriction_u \in X_u \Rightarrow \vec{y} \in \underline{X} \big\} \;=\; \bigcup_{u \in \mathbb{S}} \big\{ \vec{y} \mid \vec{y}\restriction_u \in X_u \Rightarrow \vec{y} \in \underline{X} \big\} \;=\; \bigcup_{u \in \mathbb{S}} u^* u_* (\underline{X}) \;=\; \neg \, [\![\overline{\neg X}]\!]$$

Since Prop. 12 gives $\neg X \;=\; [\![\overline{\neg X}]\!] \;\cap\; \big(\neg X \cup \neg \, [\![\overline{\neg X}]\!] \big)$, the claim follows by taking complements on both sides. □

4.8 What did we learn?

The geometric view allows us to express any set as an intersection of a closed and a dense set. Since safety and liveness properties of computations turn out to be past-closed and dense sets, any property of computations X can thus be approximated by a safety property, and precisely recovered by intersecting that safety property with a liveness property.

In cyber space, the authority properties turn out to be past-closed *cylindric* sets, whereas the availability properties are the sets whose cylindrifications are dense. Therefore, any resource security property can be approximated by an authority property, and precisely recovered by intersecting it with an availability property.

4.9 Geometry of security?!

The challenge of security is that that it defies our intuitions: *we are used to thinking locally, whereas our security requirements concern non-local interactions*. Strategic thinking about security requires expanding our views beyond our local horizons, and taking into account the non-local interactions. Formal models are not just means to increase precision, assurance, or reliability of reasoning about security. They are our indispensable tools for non-local reasoning, just like rockets and airplanes are our indispensable tool for flying. Without them, we cannot overcome our limitations.

Using the tools that overcome our limitations is, of course, a challenge in itself. It requires science. It requires theories and experiments; and it requires intuitive interfaces. The geometric view provides an intuitive interface for security science.

In this lecture, we saw how to reduce arbitrary, often very complex dependability and resource security requirements to a normal form, consisting of a single authorization, and a single availability requirement. A general test whether an arbitrary security requirement is satisfied is simplified down to two very special, standard tests. In the next lecture, we shall see how non-local properties, that cannot be tested locally at all, can be established and assured through non-local interactions.

A Appendix: Lists and strings

List and string constructors and notations

- **lists:** $X^* \;=\; \left\{ \left[x_1\ x_2 \cdots x_n \right] \in X^n \mid n = 0, 1, 2, \ldots \right\}$
- **strings:** $X^+ = \left\{ \left[x_1\ x_2 \cdots x_n \right] \in X^n \mid n = 1, 2, 3 \ldots \right\}$

Note that the types of strings and of lists differ just by a single element: the empty list ( ). Strings are the nonempty lists, and a list can be either a string of letters, or empty:

$$X^* \quad = \quad X^+ \cup \left\{ [\ ] \right\}$$

Lists are inductively generated by the empty list and concatenation:

$$\begin{array}{rl} 1 \xrightarrow{[\ ]} X^* & (\textit{empty list}) \\ X^* \times X \xrightarrow{::} X^* & (\textit{concatenation}) \\ \left\langle \left[y_n \cdots y_2\ y_1 \right], x_0 \right\rangle \longmapsto \left[y_n \cdots y_2\ y_1\ x_0 \right] & \end{array}$$

whereas strings are generated by the letters and concatenation:

$$\begin{array}{rl} X \xrightarrow{[-]} X^+ & (\textit{singletons}) \\ X^+ \times X \xrightarrow{::} X^+ & (\textit{concatenation}) \\ \left\langle \left[y_n \cdots y_2\ y_1 \right], x_0 \right\rangle \longmapsto \left[y_n \cdots y_2\ y_1\ x_0 \right] & \end{array}$$

We write lists and strings as vectors[5]

$$\vec{x} \quad = \quad \left[x_0\ x_1\ x_2\ \cdots x_n \right]$$

The indices can also be in the descending order, e.g. $\vec{x} = \left[x_n\ x_{n-1}\ \cdots x_1\ x_0 \right]$ when convenient.

The inductive constructors of these datatypes allow inductive definitions of operations on them. The most obvious one is the *concatenation*, which is on lists

$$\begin{array}{ccccc} X^* \times X^* & \xrightarrow{@} & X^* & \xleftarrow{[\ ]} & 1 \\ \left\langle \vec{z}, [\ ] \right\rangle & \longmapsto & \vec{z} & & \\ \left\langle \vec{z}, \vec{y}{::}x \right\rangle & \longmapsto & \left(\vec{z}@\vec{y}\right){::}x & & \end{array}$$

This makes X^* into a monoid, with the unit [ ]. More precisely, X^* is the free monoid over X, whereas X^+ is the free semigroup. The monoid operations easily extend from lists to sets of lists

$$\frac{X^* \times X^* \xrightarrow{@} X^* \xleftarrow{[\ ]} 1}{\wp\mathcal{O}(X^*) \times \wp\mathcal{O}\,(X^*) \xrightarrow{@} \wp\mathcal{O}\,(X^*) \xleftarrow{\{[\ ]\}} 1}$$

by defining the concatenation of $C, D \subseteq X^*$ to be the set of concatenations of their elements

$$CD \;=\; C@D \quad = \quad \{c@d \in X^* \mid c \in C, d \in D\}$$

[5] Functional programmers write $xs \;=\; \left[x1\ x2 \cdots xn \right]$

When no confusion is likely, we simplify notation, and abbreviate $C@D$ to CD as above, and also $\vec{x}@\vec{y}$ to $\vec{x} :: \vec{y}$, or $\vec{x}\vec{y}$. Lastly, the *prefix* relation $\sqsubseteq$ defined by

$$\vec{x} \sqsubseteq \vec{y} \quad \Longleftrightarrow \quad \exists \vec{z}.\ \vec{x} :: \vec{z} = \vec{y} \tag{49}$$

is a partial ordering of both lists and strings, where $\vec{x}$ is a prefix of $\vec{y}$ in the sense that $\vec{x} \sqsubseteq \vec{y}$ means that there is $\vec{z}$ such that

$$\begin{array}{rcl} & & \left[\, x_1\ x_2 \cdots x_k\ z_1\ \cdots\ z_{n-k} \,\right] \\ & = & \left[\, y_1\ y_2 \cdots y_k\ y_{k+1} \cdots y_n \,\right] \end{array}$$

so that $x_i = y_i$ for $i \leq k$ and $z_i = y_{i+k}$ for $i \leq n - k$. This partial ordering makes any set of lists X^* into a meet semilattice, with the greatest lower bound $\vec{x} \sqcap \vec{y}$ the common prefix of $\vec{x}$ and $\vec{y}$. The set of strings X^+ is not a semilattice only because there is no empty string, which would be the greatest lower bound of the strings that begin with different symbols.

Specifying properties. For any pair of events $a, b \in \Sigma$, and for arbitrary sets of events $C, D \subseteq \Sigma$, some of the properties that can be defined are:

$$\begin{array}{rcl} \overline{a} & = & \{\vec{x} :: a :: \vec{y} \mid \vec{x}, \vec{y} \in \Sigma^*\} \\ \overline{a \prec b} & = & \left\{\vec{x} :: a :: \vec{y} :: b :: \vec{z} \ \mid\ \vec{x}, \vec{y}, \vec{z} \in \Sigma^*\right\} \\ \overline{C} & = & \{\vec{x} :: c :: \vec{y} \mid \vec{x}, \vec{y} \in \Sigma^*, c \in C\} \\ \overline{C \prec D} & = & \left\{\vec{x} :: c :: \vec{y} :: d :: \vec{z} \ \mid\ \vec{x}, \vec{y}, \vec{z} \in \Sigma^*,\ c \in C,\ d \in D\right\} \\ \overline{C \prec \exists D} & = & \left\{\vec{t} \in \Sigma^* \mid \forall c \in C.\ \vec{t} = \vec{x} :: c :: \vec{y} \ \Rightarrow\ \exists d \in D.\ \vec{y} = \vec{y}' :: d :: \vec{y}''\right\} \\ \overline{\exists C \prec D} & = & \left\{\vec{t} \in \Sigma^* \mid \forall d \in D.\ \vec{t} = \vec{x} :: d :: \vec{y} \ \Rightarrow\ \exists c \in C.\ \vec{x} = \vec{x}' :: c :: \vec{x}''\right\} \end{array}$$

A.1 Local views and localized properties

An arbitrary function $\boldsymbol{w} : \Sigma \longrightarrow \mathbb{W}$ can be considered as a *projection* of an event on its shadow, or as a *local view*, mapping an event σ to its view $\boldsymbol{w}(\sigma)$ at a *locality* $\mathbb{W}$. Hence the partition

$$\Sigma \quad = \quad \coprod_{w \in \mathbb{W}} \Sigma_w$$

where $\Sigma_w = \{\sigma \in \Sigma \mid \boldsymbol{w}(\sigma) = w\}$ are just the events corresponding to a locality $w \in \mathbb{W}$. When $\mathbb{W}$ is $\mathbb{S}$, then Σ_w are the actions performed by the subect w; if $\mathbb{W}$ is $\mathbb{J}$, Σ_w are the actions performed on the obect w. The localities $\mathbb{W}$ can also be security levels $\mathbb{L}$, actions $\mathbb{A}$, and various other options that we didn't even mention.

In any case, for any locality $w \in \mathbb{W}$, the *purge* operation $\restriction_w \colon \Sigma^* \longrightarrow \Sigma_w^*$ is defined inductively by

$$\begin{array}{rcl} [\,] \restriction_w & = & [\,] \\ \left(x :: \vec{y}\right) \restriction_w & = & \begin{cases} x :: \left(\vec{y} \restriction_w\right) & \text{if } \boldsymbol{w}(x) \leq w \\ \vec{y} \restriction_w & \text{otherwise} \end{cases} \end{array} \tag{50}$$

The *local view* of a property $P \subseteq \Sigma^*$ is then defined

$$P_w \quad = \quad \left\{\vec{x} \restriction_w \ \mid \vec{x} \in P\right\}$$

A set of traces $P \subseteq \Sigma^*$ is called a *localized property* when

$$\vec{t} \in P \quad \Longleftrightarrow \quad \forall i \in \mathbb{I}.\ \vec{t} \restriction_i \in P_i \tag{51}$$

Example. Let $\Sigma = \Sigma_0 \coprod \Sigma_1$ where $\Sigma_0 = \{a\}$ and $\Sigma_1 = \{b\}$. Then $C = \{aabb\}$ is not local. Its local views are $C_0 = \{aa\}$ and $C_1 = \{bb\}$, so that $\vec{t} = (abab)$ satisfies $\vec{t} \restriction_0 \in C_0$ and $\vec{t} \restriction_1 \in C_1$, but $\vec{t} \notin C$. The right-hand side of (51) is thus true, but the left-hand side is not. The smallest local property containing C is $\widehat{C} = \{aabb, abab, baab, baba, bbaa\}$.

Lemma 14. *Any history $\vec{t}$ of events $\Sigma = \coprod_{w\in\mathbb{W}} \Sigma_w$ is completely determined by*

- *the restrictions $\vec{t}\restriction_w$ for $w \in \mathbb{W}$, and*
- *their schedule , which is function $s : |\vec{t}| \longrightarrow \mathbb{W}$ such that*

$$s(k) = w \quad \Longleftrightarrow \quad t_k \in \Sigma_w \tag{52}$$

where $\vec{t} = \left[\, t_0\ t_1 \cdots t_n \,\right]$ and $|\vec{t}| = n + 1$.

Proof. The claim is that the function $s : |\vec{t}| \longrightarrow \mathbb{W}$ satisfying (52) provides enough information to reconstruct $\vec{t}$ from its restrictions $\vec{t}\restriction_w$, given for all $w \in \mathbb{W}$. The restrictions can be scheduled in many ways, but (52) says that k-th component t_k of $\vec{t}$ comes from the restriction $\vec{t}\restriction_{s(k)}$. To simplify notation, we write $\vec{t}\restriction_{s(k)}$ as $\vec{\tau}^{(k)}$, so that σ-th component of $\vec{t}\restriction_{s(k)}$ becomes $\tau^{(k)}_\sigma$. So we know that $t_k = \tau^{(k)}_\sigma$ for some σ. The question is: *what is σ?*

Towards the answer, we use $s : |\vec{t}| \longrightarrow \mathbb{W}$ to define the function $\varsigma : |\vec{t}| \times \mathbb{S} \longrightarrow |\vec{t}| + 1$ which counts the number of w-actions in the k-length prefix of $\vec{t}$. The definition is by recursion:

$$\begin{aligned}
\varsigma(0, w) &= \begin{cases} 1 & \text{if } s(0) = w \\ 0 & \text{otherwise} \end{cases} \\
\varsigma(n + 1, w) &= \begin{cases} \varsigma(n) + 1 & \text{if } s(n + 1) = w \\ \varsigma(n) & \text{otherwise} \end{cases}
\end{aligned}$$

It is easy to show by induction that $\varsigma(k, w)$ is the length of the w-restriction of the k-prefix $\vec{t}_k \sqsubseteq \vec{t}$, i.e. $\varsigma(k, w) = |\vec{t}_k\restriction_w|$. Hence $t_k = \tau^{(k)}_{\varsigma(k,s(k))}$ is thus the $\varsigma(k, s(k))$-th component of $\vec{\tau}^{(k)} = \vec{t}\restriction_{s(k)}$. □

Proposition. *A property P is localized if and only if together with every $\vec{t}$ it contains all $\vec{s}$ such that $\vec{t}\restriction_w = \vec{s}\restriction_w$ for all $w \in \mathbb{W}$.*

Localization. In general, the smallest local property containing $P \subseteq \Sigma^*$ with respect to the partition $\Sigma = \coprod_{i\in\mathbb{I}} \Sigma_i$ is

$$\widehat{P} \quad = \quad \left\{\vec{t} \in \Sigma^* \mid \forall i \in \mathbb{I}.\ \vec{t}\restriction_i \in P_i\right\}$$

The property $\widehat{P}$ is called the *localization* of P.

A.2 Images, cylinders and cylindrifications

Direct and inverse images. Any function $f : A \longrightarrow B$ induces two direct image maps and one inverse image map:

$$\begin{aligned}
f_! : \wp A &\longrightarrow \wp B \\
X &\longmapsto \{y \mid \exists x.\ f(x) = y \wedge x \in X\}
\end{aligned}$$

$$\begin{aligned}
f^* : \wp B &\longrightarrow \wp A \\
V &\longmapsto \{x \mid f(x) \in V\}
\end{aligned}$$

$$\begin{aligned}
f_* : \wp A &\longrightarrow \wp B \\
X &\longmapsto \{y \mid \forall x.\ f(x) = y \Rightarrow x \in X\}
\end{aligned}$$

It is easy to show that they satisfy

$$\begin{aligned}
f_!(X) \subseteq Y \quad &\Longleftrightarrow \quad X \subseteq f^*(Y) \text{ and} \\
f^*(Y) \subseteq X \quad &\Longleftrightarrow \quad Y \subseteq f_*(X)
\end{aligned}$$

and hence for any $X \in \wp\mathcal{O}A$ holds

$$f^* f_*(X) \subseteq X \subseteq f^* f_!(X)$$

Cylinders. We call $f^* f_*(X)$ the *internal f-cylinder* contained in X, whereas $f^* f_!(X)$ is the *external f-cylinder* containing X. An external f-cylinder is thus the smallest inverse image of a set in B along f that contains X, whereas an internal f-cylinder is the largest inverse image of a set in B along f that is contained X.

Cylinder closures and interiors. Suppose that we are given a family of functions $V = \{v : A \longrightarrow B_v \mid v \in \mathbb{V}\}$, where $\mathbb{V}$ is an arbitrary set of indices.

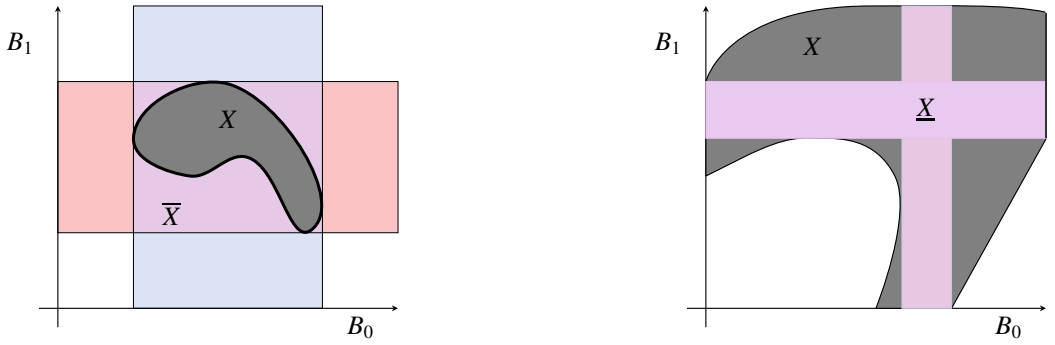


Figure 7: Cylinder closure and interior

A *cylinder closure* and *cylinder interior* operators are defined by

$$\begin{array}{rcl} [\![-]\!] : \wp\mathcal{O}A & \longrightarrow & \wp\mathcal{O}A \\ X & \longmapsto & [\![X]\!] = \bigcap_{v \in \mathbb{V}} v^* v_!(X) \end{array} \tag{53}$$

$$\begin{array}{rcl} (\!|-|\!) : \wp\mathcal{O}A & \longrightarrow & \wp\mathcal{O}A \\ X & \longmapsto & (\!|X|\!) = \bigcup_{v \in \mathbb{V}} v^* v_*(X) \end{array} \tag{54}$$

where the sets

$$\begin{array}{rcll} [\![X]\!]_v & = & v^* v_!(X) & = \{z \in A \mid \exists x. v(z) = v(x) \wedge x \in X\} \\ (\!|X|\!)_v & = & v^* v_*(X) & = \{z \in A \mid \forall x. v(z) = v(x) \Rightarrow x \in X\} \end{array}$$

are respectively the external v-cylinders around X and the internal v-cylinders inside X. It is easy to see that the $[\![-]\!]$ is a closure operator and that $(\!|-|\!)$ is an interior operator, which means that they satisfy

$$\begin{array}{cc} (\!|(\!|X|\!)|\!) = (\!|X|\!) & [\![[\![X]\!]]\!] = [\![X]\!] \\ (\!|X|\!) \subseteq X \subseteq [\![X]\!] & \end{array}$$

They are complementary in the sense

$$\neg(\!|X|\!) = [\![\neg X]\!] \qquad \neg [\![X]\!] = (\!|\neg X|\!)$$

Definition 15. *$X \in \wp\mathcal{O}A$ is* external cylindric *if $X = [\![X]\!]$ and* internal cylindric *if $X = (\!|X|\!)$.*

A.3 Topology concepts

Neighborhoods. Topology is the most general theory of space. Spaces usually model the realms of observations, and their structures therefore correspond to what is observed: e.g., metric spaces capture distances, vector spaces also capture angles, etc. Topological spaces only capture *neighborhoods*, viewed as sets that are inhabited by some objects together. Smaller neighborhoods suggest that the objects are closer together, and the set inclusion of neighborhoods thus tells which objects are closer together, and which ones are further apart. But these distinctions are made without assigning any numeric values to the distances between objects, as it is done in metric spaces. Metric spaces can thus be viewed as a special case of topological spaces.

For any set E, a (topological) space structure is by either of the following:

- *open neighborhoods* are represented by a family $\mathcal{O}_E \subseteq \wp(E)$ closed under finite $\cap$ and arbitrary $\bigcup$; whereas
- *closed neighborhoods* are represented by a family $\mathcal{F}_E \subseteq \wp(E)$ closed under finite $\cup$ and arbitrary $\bigcap$

The two families determine each other, in the sense that a set is open if and only if its complement is closed, i.e.

$$\begin{array}{rcl} \mathcal{O}_E & = & \left\{U \in \wp(E) \mid \neg U \in \mathcal{F}_E\right\} \\ \mathcal{F}_E & = & \left\{F \in \wp(E) \mid \neg F \in \mathcal{O}_E\right\} \end{array}$$

Any family $\mathcal{B}_E \subseteq \wp(E)$ can be declared to be open (or closed) neighborhoods, and used to generate the full space structure. If we want $\mathcal{B}_E$ to be

- open, then set

$$\begin{array}{rcl} \mathcal{B}_E^{\cap} & = & \left\{ \cap_{i=0}^{n} B_i \mid B_0, \ldots, B_n \in \mathcal{B}_E \right\} \\ \mathcal{O}_E & = & \left\{ \bigcup \mathcal{V} \mid \mathcal{V} \subseteq \mathcal{B}_E^{\cap} \right\} \end{array}$$

- closed, then set

$$\begin{array}{rcl} \mathcal{B}_E^{\cup} & = & \left\{ \cup_{i=0}^{n} B_i \mid B_0, \ldots, B_n \in \mathcal{B}_E \right\} \\ \mathcal{F}_E & = & \left\{ \bigcap \mathcal{V} \mid \mathcal{V} \subseteq \mathcal{B}_E^{\cup} \right\} \end{array}$$

Closure operators. For any given topology on E, the family $\mathcal{F}_E$ can be equivalently presented in terms of the *closure operator* that it induces:

$$\overline{(-)} : \wp(E) \longrightarrow \wp(E) \tag{55}$$

$$X \longmapsto \overline{X} = \bigcap \{F \in \mathcal{F}_E \mid X \subseteq F\} \tag{56}$$

It is easy to see that the general requirements from a closure operator are satisfied:

$$X \subseteq \overline{X} = \overline{\overline{X}}$$

and that $\mathcal{F}_E = \{X \in \wp(E) \mid \overline{X} = X\}$.

Interior operators are dual to closure operators. This means that for any given topology on E, the family $\mathcal{O}_E$ of open sets (dual to the closed sets $\mathcal{F}_E$) can be equivalently presented in terms of the *interior operator* that it induces:

$$\begin{array}{rcl} \underline{-} : \wp(E) & \longrightarrow & \wp(E) \\ X & \longmapsto & \underline{X} = \bigcup \{U \in \mathcal{O}_E \mid U \subseteq X\} \end{array}$$

It is easy to see that the general requirements from an interior operator are satisfied:

$$X \supseteq \underline{X} = \underline{\underline{X}}$$

and that $\mathcal{O}_E = \{X \in \wp(E) \mid \underline{X} = X\}$.

It is easy to see that the two operators determine each other by

$$\neg\overline{X} = \underline{\neg X} \qquad\qquad \overline{\neg X} = \neg\underline{X}$$

Density can be defined by saying for $X, Y \in \wp(E)$ that X *is dense on* Y if $\overline{X} \supseteq Y$. This can be equivalently written in the form $Y \cap \underline{\neg X} = \emptyset$. We sometimes describe such situations by saying that $\neg Y$ *is codense on* $\neg X$. In general, U is codense on V when $\int V \setminus U = \emptyset$. The family of sets that are dense on Y, or on which Y is codense, is

$$\mathcal{D}_Y = \left\{D \in \wp(Y) \mid \forall F \in \mathcal{F}_E.\, D \subseteq F \Rightarrow Y \subseteq F\right\}$$
$$\left\{D \in \wp(Y) \mid \forall U \in \mathcal{O}_E.\, Y \cap U \neq \emptyset \;\Rightarrow D \cap U \neq \emptyset\right\}$$

Closed-dense $\cap$-decomposition. Any inclusion $X \subseteq E$ clearly factors as $X \subseteq \overline{X} \subseteq E$ where $\overline{X} \in \mathcal{F}_E$ is closed, and $X \in \mathcal{D}_{\overline{X}}$ is relatively dense.

Therefore any $X \subseteq E$ is an intersection of a closed and a dense set

$$X = \overline{X} \cap (X \cup \neg\overline{X})$$

where $\overline{X} \in \mathcal{F}_E$ is closed, and $X \cup \neg\overline{X} \in \mathcal{D}_E$ is dense everywhere.

Open-codense $\cup$-decomposition is induced by the closed-dense $\cap$-decomposition of the complement $\neg X \subseteq E$ through $\overline{\neg X} = \neg\underline{X} \subseteq E$. More precisely $\neg X = \overline{\neg X} \cap (\neg X \cup \neg\overline{\neg X}) = \neg\underline{X} \cap \neg\,(X \cap \neg\underline{X})$ becomes

$$X = \underline{X} \cup \left(X \setminus \underline{X}\right)$$

where $\underline{X} \in \mathcal{O}_E$ is open, and $X \setminus \underline{X}$ is codense.