

# A Science of Security

$$\frac{df}{dt} = \lim_{h \rightarrow 0} \frac{f(t+h) - f(t)}{h}$$

# A scientific blueprint for security

A uniform **conceptual** framework that **precisely defines** system security will help **analyze security, support comparative evaluation**, and **develop useful insight into design, implementation, and deployment decisions.**

Quali sono le decisioni che devono essere prese in fase di progetto significa quali sono i meccanismi e le contromisure che mette in opera quando progetta un sistema.

# The ultimate goals

DEVE ESSERE UN MODELLO CHE NON  
DIPENDE DALLA PARTICOLARE  
TECNOLOGIA O CLASSE DI  
APPLICAZIONE

- **transcend specific technologies and attacks**
  - yet still be applicable in real settings,
- **introduce new models and abstractions**,
  - bringing pedagogical value besides predictive power,
- **facilitate discovery of new defenses**
  - describe non-obvious connections between attacks, defenses, and policies,
  - providing a better understanding of the landscape.

The approach:  
Security model

Il modello del sistema è un  
qualcosa che ci descrive i  
comportamenti

# System Model

# Threat Model

# Security Policies

quali sono le capacità in termini di  
potenza computazionale minimi  
che un attaccante può fare. Non ci  
sono difese valide per tutti gli  
attaccanti

Describe i comportamenti del sistema, ovvero quello che deve essere fatto dal sistema quando è in operazione su particolari condizioni. Una definizione dei comportamenti nell'ambiente di esecuzione. Per esempio nel caso del Networking ci sono degli standard per descrivere il comportamento (RFC).

# System Model

- A clear definition of the system of interest:
    - how the system **behaves** when subjected to its **intended operating conditions**, as well as **unintended input or operating conditions**.
  - Several techniques
    - Standards
    - Software Architecture UML per esempio
    - Formal Specification
    - :
- 
- ci sono diversi strumenti per annotare le proprietà di una determinata classe o metodo in termini di post e pre condizioni e che verificano se il codice scritto rispetta tali condizioni

# Threat Model

- Definition of attackers' **computational resources** and **system access**.
  - **Network attackers** might have access to network messages but not to the internal state of hosts communicating on the network
  - **Crypto attackers** have insufficient computational power to break cryptography
  - **OS attackers** might be able to place malicious code in a user process but unable to modify the OS kernel.

# Security Policies

- Define the **properties** we hope to prevent attackers from violating.
- Ability to **determine** whether the desired security properties hold or fail **on all** system behavior

Le politiche devono valere per tutti

## Security Policies: the CIA factor

- 1. *Confidentiality*** (no sensitive information is revealed),
- 2. *Integrity*** (attackers can't destroy the system's meaningfully operable condition),
- 3. *Availability*** (the attacker can't render the system unavailable to intended users).

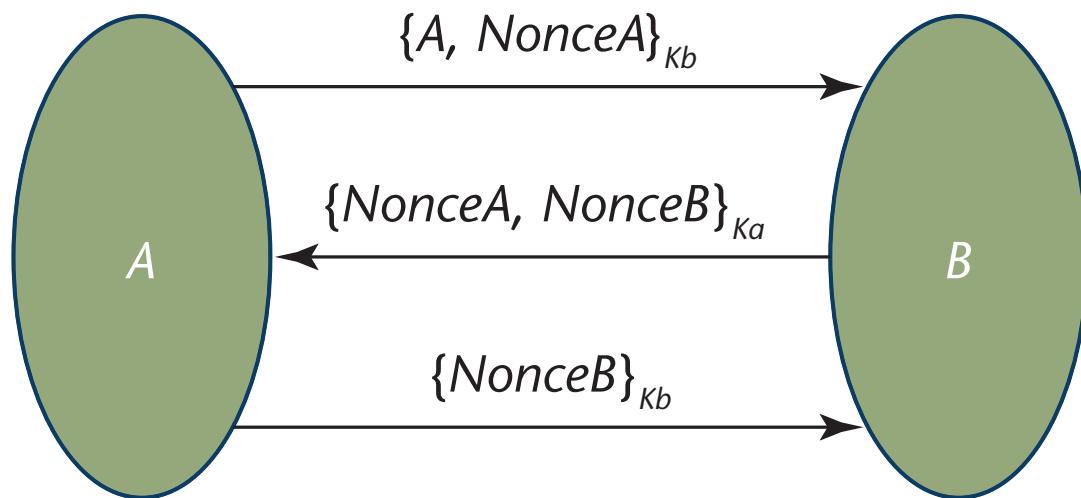
# An example cryptography

- Cryptography issues
  - Understand the design and limitations of algorithms and protocols to compute certain kinds of results in the presence of certain kinds of adversaries who have access to some, but not all, information involved in the computation.
- Cryptography is one of many cybersecurity building blocks.
- A science of cybersecurity would have to encompass richer kinds of building blocks.
- Quoting Peter Neumann "**If you think cryptography is the answer to your problem, then you don't know what your problem is.**"

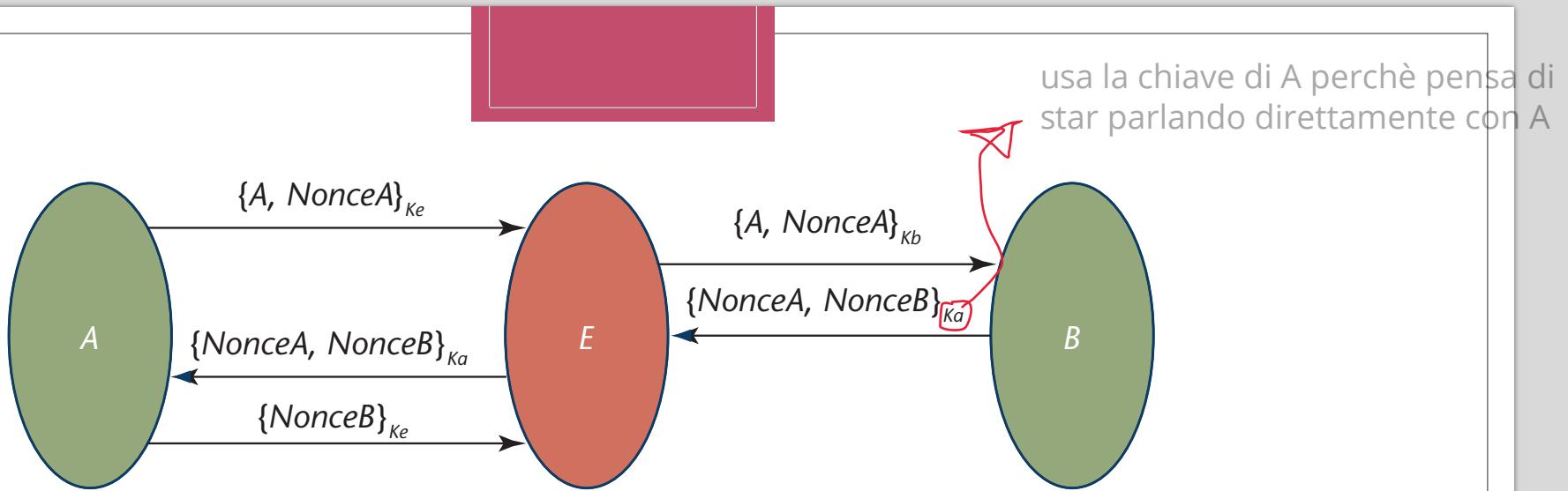
# Security Analysis

- Security models (System, Threat, Policies) provide the basis structure for security analysis
  - The **security analysis** is the process of **evaluating** whether the system design **achieves** the desired **properties** against the chosen **threat model**.
  - SE Point of view: SA is the **methodology** to compare the relative strengths of different system designs.

Abbiamo 2 principali A e B che si scambiano dei dati in modo sicuro. A invia a B il nonce che è un numero pseudo-random necessario per individuare quella particolare comunicazione e questo messaggio è cifrato con la chiave pubblica di B. B cifra con la chiave pubblica di A il nonce di A e un nonce generato al volo da se stesso. A questo punto A è in grado di decifrare il messaggio, vedere la sua nonce inviata prima e avere la certezza che sta parlando con B, vedere la nonce di B e reinviarla criptandola con la chiave pubblica di B.



## EXAMPLE: NEEDHAM- SCHROEDER PUBLIC-KEY PROTOCOL



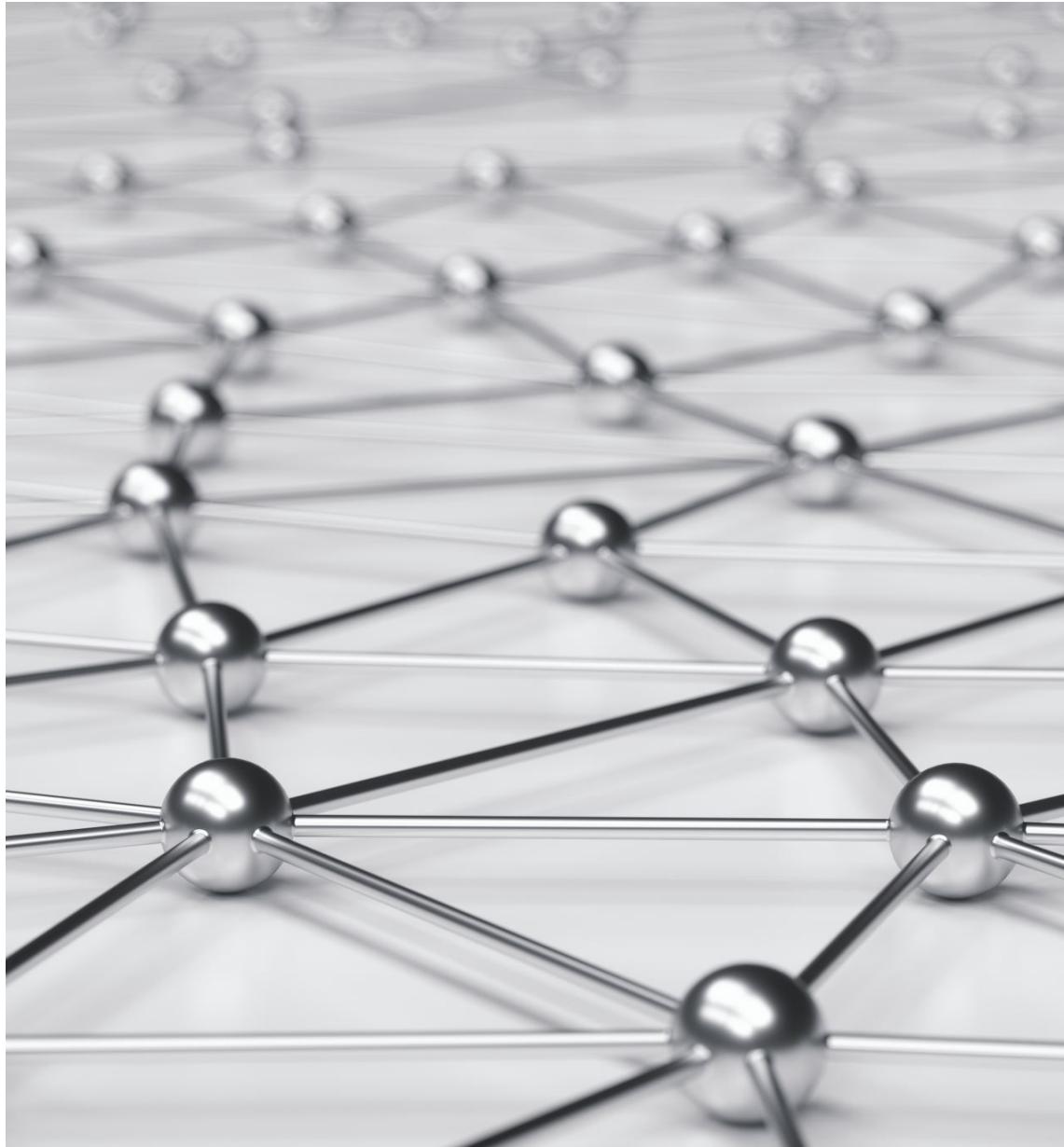
## NEEDHAM-SCHROEDER ATTACK

Evil si spaccia per B, riceve da A la nonce cifrata con la chiave pubblica del man in the middle, legge la nonce A, la cifra con la chiave pubblica di B e si spaccia per B instaurando una connessione intermedia fasulla.

# System Model

il modello del sistema è facile perchè sono le azioni e le regole che caratterizzano lo scambio del messaggio nel protocollo. Si vede quale è il messaggio scambiato, chi sono i principali e quale è la reazione di un principale quando riceve un messaggio da un altro principale.

- The set of action rules “performed” the the protocol steps on the basis of the current state and the validity of information received from the network.
- A chooses a random  $N_A$  and sends it to B, the message is encoded with the public key of B
  - $A \rightarrow B: \{N_A, A\}_{KPB}$  A manda a B un crittogramma che contiene la nonce A e il nome di A cifrato con la chiave pubblica di B
- B chooses a random  $N_B$ , and sends it to A along with  $N_A$  to prove ability to decrypt with  $K_{PB}$ .
  - $B \rightarrow A: \{N_B, N_A\}_{KPA}$  Similmente la risposta di B è quella di mandare un altro messaggio ad A che viene cifrato con la chiave pubblica di A contenente il nonce di A e un nuovo nonce che viene inviato per trovare un identificativo univoco



## System Model

- The network is modeled as a collection of **shared states** between the principals involved in the protocol,
- Each specific network message details a particular network state setting.

# The threat model

2 ricercatori

- The **Dolev Yao threat model**: define attackers' capabilities and operations
  - eavesdropping on any network message and breaking its content (as captured) into parts,
  - recording parts of any eavesdropped packet into storage,
  - removing messages from the network,
  - sending network messages containing new or eavesdropped content to any legitimate party.
  - attackers can't compromise cryptographic protections, such as encryption and signatures, without the appropriate key.

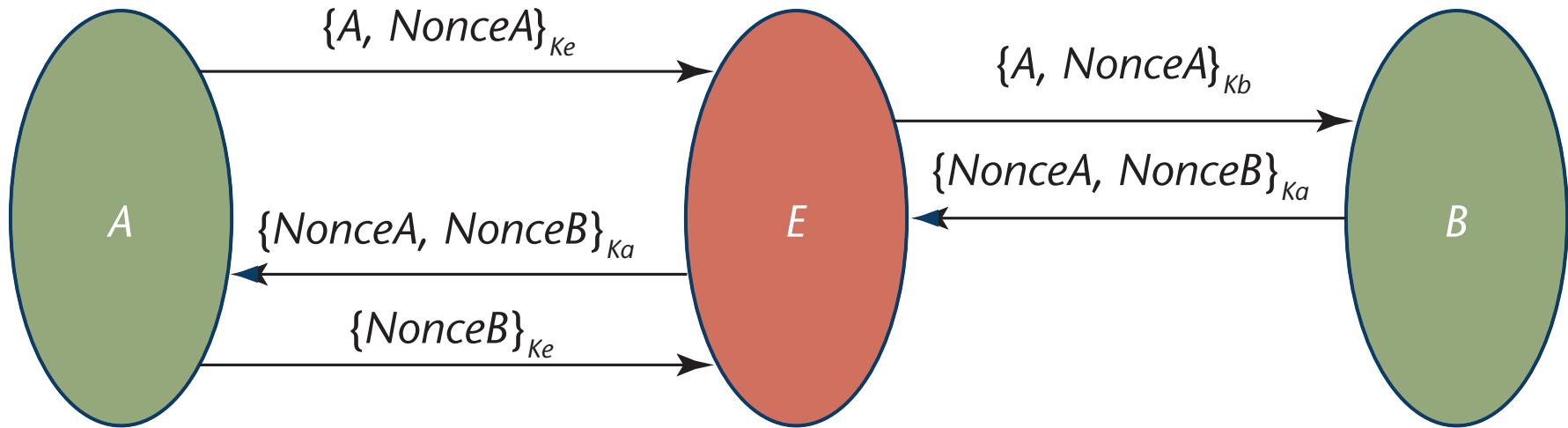
# The threat model: NS protocol

- attackers
  - can only record and replay the ciphertext form of encrypted data captured from the network
  - can't compromise the plaintext content of encrypted data.
  - can create ciphertext using their own private key,
    - decryption can only be performed using the attackers' identifying public key.

le proprietà che vogliamo avere

## Security Property: The NS protocol

- ▣ ° The **integrity** invariants specify that, *for party A, reaching the committed state the secret is NonceB*
- ° vice versa NonceA for party B.
- ° The NS model's **secrecy** invariant: *no attackers can decrypt and learn secrets from any intended parties.*



## THE NS PROTOCOL: VULNERABILITY

# NS Vulnerability

- The weakness was discovered by using a model-checking tools operating on the (state space) of the NS Model
- The model checker finds a protocol-execution trace where the attacker  $E$  learns secrets meant to be kept between parties  $A$  and  $B$  by acting as a man in the middle.
- The discovery of this trace triggers violations of the model's secrecy and integrity invariants.

La soluzione illustrata dopo è stata trovata usando un model checker, ovvero uno strumento che analizza i comportamenti del sistema e andando a verificare sui comportamenti del sistema se vale la proprietà di integrity che è la proprietà che vogliamo valga sia per A che per B



## Fixing the attack (G. Lowe, 1995)

The fix involves the modification of message six to include the responder's identity. This means replacing the message

**B -> A :{N<sub>A</sub>, N<sub>B</sub>}<sub>KPA</sub>**

With the message

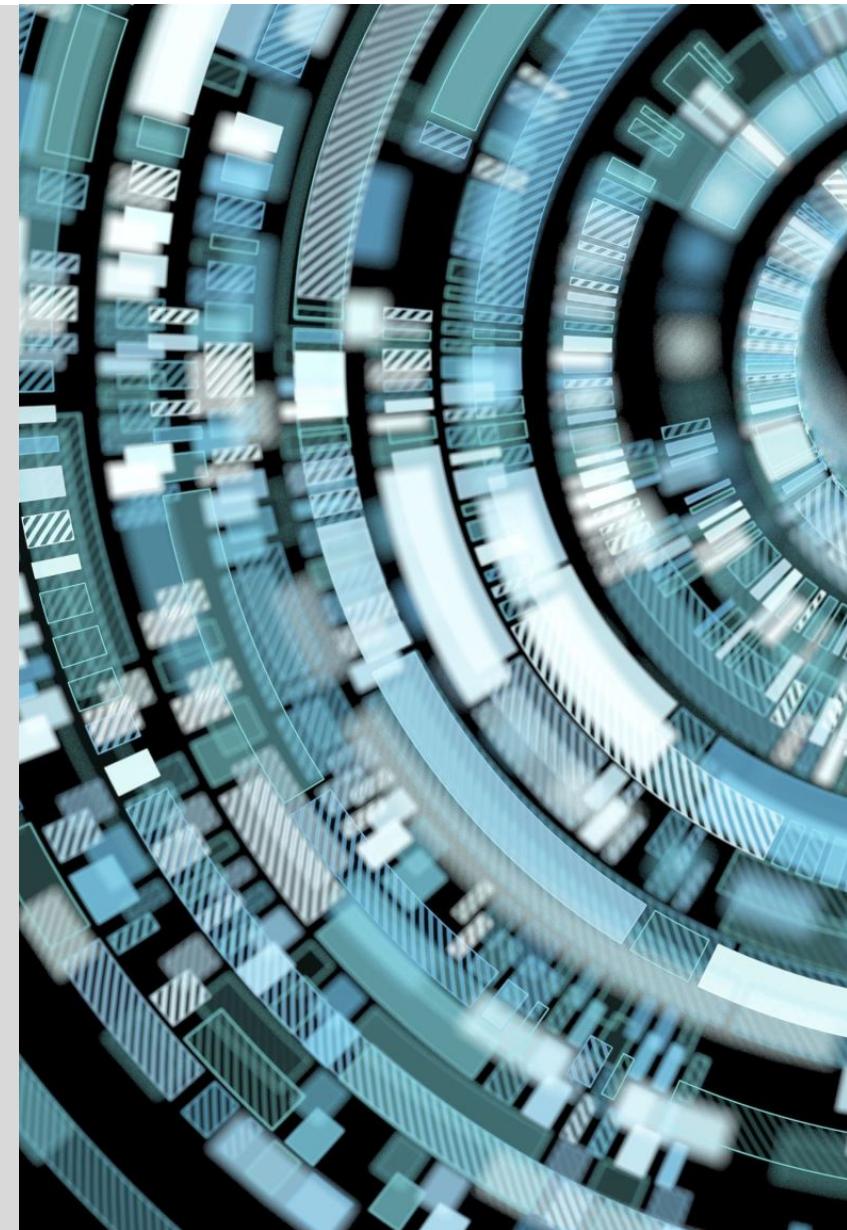
**B -> A: {N<sub>A</sub>, N<sub>B</sub>, B}<sub>KPA</sub>**

The model checker verifies the fixed protocol (NSL) bisogna specificare che B è effettivamente B e che conosce il nonce di A e conosce il suo nonce (questo l'attaccante non può modificarlo perché altrimenti non si avrebbe integrità)



# Science of security

## Multidisciplinary Approach



il system model



specification

ovvero le proprietà che vogliamo avere



e lo esegue ovvero verifica se questo modello  
le rispetta le proprietà

yes  
(O→O→O→O  
witness path)

no  
(O→O→O→O  
counterexample)

DEVE ESSERE IN GRADO DI FORNIRE UN CAMMINO, UN  
COMPORTAMENTO CHE E' IL CONTROESEMPIO OVVERO  
CI FA VEDERE QUALE E' ESATTAMENTE IL CAMMINO CHE  
NON RISPETTA LA PROPRIETA' CHE CI ASPETTIAMO

# MODEL CHECKING?

E' un insieme di stati, un alfabeto, un insieme di stati  $Q_0$  che è l'insieme di stati iniziali e un insieme di stati finali  $F$ . La funzione di transizione prende uno stato e un simbolo dell'alfabeto e ci da un insieme di possibili stati. E' un qualcosa che riconosce una sequenza di caratteri che sono gli elementi di un alfabeto. Mi trovo in uno stato, leggo un carattere, e posso muovermi in un certo insieme di alternative e ne prendo una tra queste e questo è il motivo per cui la funzione di transizione da come risultato  $2^Q$  e non un risultato unico. Se restituisse un risultato unico sarebbe un AUTOMA A STATI FINITI DETERMINISTICO, potrebbe essere vuoto ovviamente.

NFA  $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$

- $Q$  finite set of states
- $\Sigma$  alphabet
- $\delta : Q \times \Sigma \rightarrow 2^Q$  transition relation
- $Q_0 \subseteq Q$  set of initial states
- $F \subseteq Q$  set of **final states**, also called **accept states**

NON  
DETERMINISTIC  
FINITE STATE  
AUTOMATA

NFA  $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$

- $Q$  finite set of states
- $\Sigma$  alphabet
- $\delta : Q \times \Sigma \rightarrow 2^Q$  transition relation
- $Q_0 \subseteq Q$  set of initial states
- $F \subseteq Q$  set of final states, also called accept states

run for a word  $A_0 A_1 \dots A_{n-1} \in \Sigma^*$ :

state sequence  $\pi = q_0 q_1 \dots q_n$  where  $q_0 \in Q_0$   
and  $q_{i+1} \in \delta(q_i, A_i)$  for  $0 \leq i < n$

NFA: RUN

Una RUN è una sequenza di stati a partire da uno stato iniziale  $q_0$ , lo stato successivo è uno stato che appartiene alla funzione di transizione, ovvero è uno stato che appartiene all'insieme di stati che se io applico la funzione delta allo stato precedente  $q_i$  leggendo il simbolo  $A_i$  mi andrà in uno stato  $q_{i+1}$ . Dunque a seconda della scelta che faccio per risolvere un non determinismo vado in uno stato e poi proseguo, quindi una run è un qualcosa che mi fa muovere un automa a stati finiti.

NFA  $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$

- $Q$  finite set of states
- $\Sigma$  alphabet
- $\delta : Q \times \Sigma \rightarrow 2^Q$  transition relation
- $Q_0 \subseteq Q$  set of initial states
- $F \subseteq Q$  set of final states, also called accept states

run for a word  $A_0 A_1 \dots A_{n-1} \in \Sigma^*$ :

state sequence  $\pi = q_0 q_1 \dots q_n$  where  $q_0 \in Q_0$

and  $q_{i+1} \in \delta(q_i, A_i)$  for  $0 \leq i < n$

run  $\pi$  is called accepting if  $q_n \in F$

una run si dice ACCETTANTE quando se vado a vedere lo stato finale  $q_n$  di questa sequenza di esecuzione su un automa, questo stato appartiene agli stati finali

NFA:  
ACCEPTING



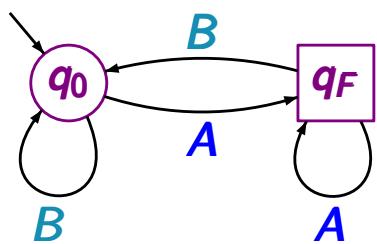
# NFA: LANGUAGES

NFA  $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$

- $Q$  finite set of states
- $\Sigma$  alphabet
- $\delta : Q \times \Sigma \rightarrow 2^Q$  transition relation
- $Q_0 \subseteq Q$  set of initial states
- $F \subseteq Q$  set of final states, also called accept states

accepted language  $\mathcal{L}(\mathcal{A}) \subseteq \Sigma^*$  is given by:

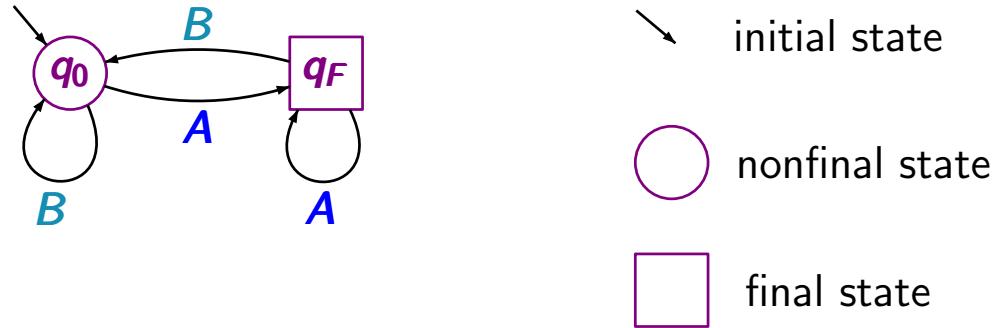
$\mathcal{L}(\mathcal{A}) =$  set of finite words over  $\Sigma$  that have  
an accepting run in  $\mathcal{A}$



- initial state
- nonfinal state
- final state

NFA:  
NOTATION

Se vado a vedere quale è il linguaggio conosciuto ovvero tutte quelle parole che sono accettate e mi portano in uno stato finale ad esempio le parole che iniziano con B mi portano a rimanere sempre in uno stato iniziale mentre tutte le parole che terminano con A sono accettate dal linguaggio perchè non appena ricevo A vado nello stato finale.



accepted language  $\mathcal{L}(\mathcal{A})$ :

set of all finite words over  $\{A, B\}$   
ending with letter  $A$

## NFA: EXAMPLE

# Regular Property

Dato un automa a stati finiti non deterministico A possiamo dire come è fatto il linguaggio, è l'insieme di tutte le tracce e run accettate.

- Given a NFA A the language recognized by A ( **$L(A)$** ) is the set of all accepting runs
- A property is a subset of runs
  - $E \subseteq L(A)$
- A property E is called **regular** iff BadPrefixes recognized by a NFA B
  - $BadPrefixes \subseteq L(B)$

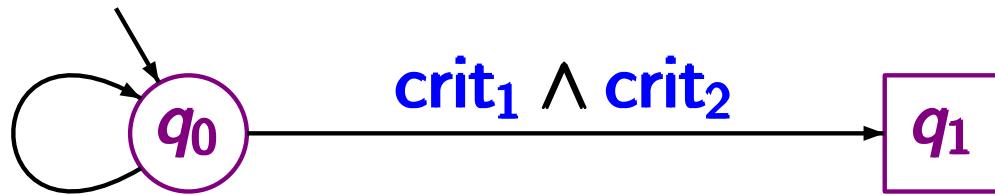
se e solo se  
↑

Questa proprietà E è detta regolare se e solo se i suoi BadPrefixes ovvero le sue sottotracce non accettate, ovvero sequenze di comportamento che sono sotto tracce prefisso delle run, ovvero del sottoinsieme di tracce E sono riconosciute da un altro automa. Abbiamo un sottoinsieme di comportamenti accettabili che mi portano in uno stato finale, ovvero tutte le run che mi portano in uno stato finale. Quelle regolari sono quelle che se riesco a determinare un sottoinsieme dei prefissi della proprietà che sto analizzando e riesco a fare in modo che questo sottoinsieme dei prefissi che portano in stati non accettati che quindi non verificano la proprietà, allora se riesco a determinarlo in modo tale che sia (questo sottoinsieme di prefissi scorretti) il linguaggio riconosciuto da un altro automa allora dico che quella proprietà è una proprietà regolare in quanto riesco a individuare un automa che la riconosce.

## Example

- MUTEX: The two processes are never simultaneously in their critical sections
- BadPrefixes MUTEX

Questo schema fa vedere esattamente l'insieme dei bad prefixes che solo uno dei 2 sta in mutua esclusione. Fa vedere i bad prefixes in quanto lo stato finale, quello accettante è l'unico stato in cui arrivo quando ho esattamente i 2 thread che entrambi sono nello stesso momento nella sezione critica. Riesco a realizzare la mutua esclusione tramite un automa che mi esprime il negato, ovvero esprime la proprietà in termini di accettazione della formula esattamente nella violazione della proprietà.



$\neg\text{crit}_1 \vee \neg\text{crit}_2$

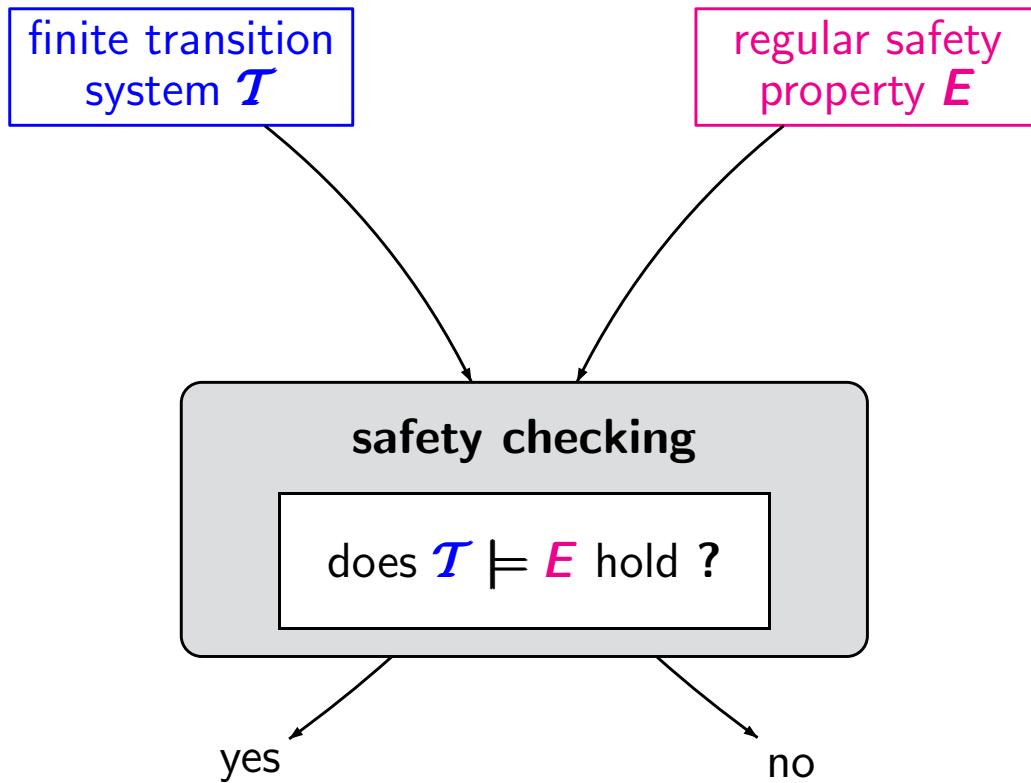
come faccio a descrivere in termini di un automa non la proprietà che voglio ma la negazione della proprietà che voglio? (in questo caso: i 2 thread sono nello stesso momento nella sezione condivisa) questo automa mi dice che nello stato iniziale "o non ho il primo thread o non ho il secondo thread nella sezione condivisa" ovvero uno dei 2 sta nella sezione condivisa e questo continua ad essere uno stato accettabile quando invece ho la proprietà voluta? quando entrambi sono in sezione critica. A questo punto una traccia è "o ch'ho uno o l'altro" e a questo punto ho entrambi dunque questa è una traccia che è un prefisso dei comportamenti che porta in uno stato finale che falsifica la proprietà. Se dunque prima avevo una proprietà, i bad prefixes sono la negazione della proprietà ovvero i comportamenti che non voglio avere. Se riesco a caratterizzare i bad prefixes come un automa a stati finiti allora ho un modo di verificarli usando le tecniche che vedremo (isolamento, obfuscation etc etc...)

Abbiamo un modello dei comportamenti con tutte le nostre tracce, abbiamo una proprietà che è un sottoinsieme di tutte queste tracce, quando è che il sistema, ovvero l'insieme delle tracce soddisfa la proprietà e in che modo possiamo gestire questa verifica? Se riuscissimo a gestirla con un algoritmo in un tempo ragionevole avremmo lo strumento automatico che permette di verificare in modo automatico delle proprietà su classi di comportamento.

## Verifying regular properties

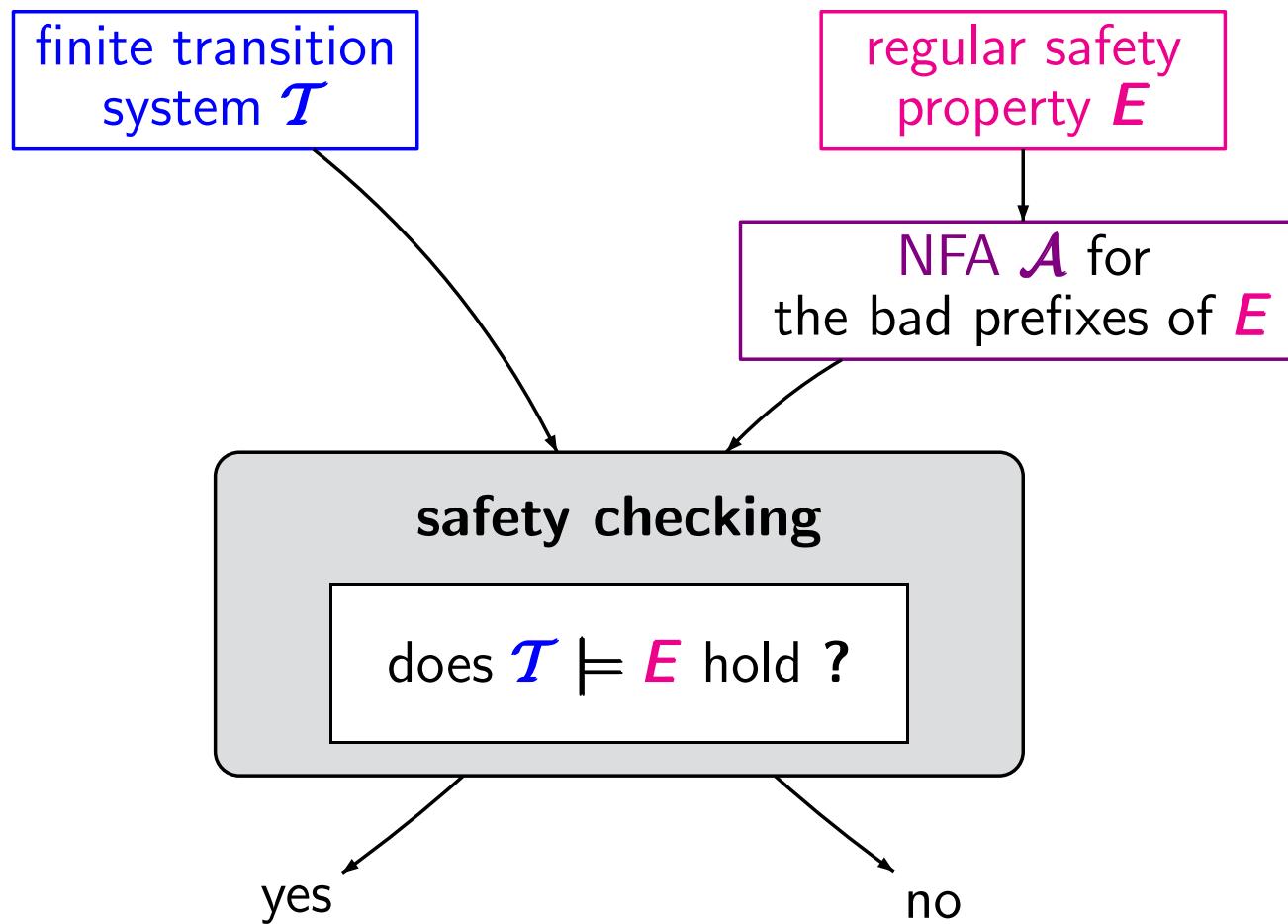
- Given a System Model  $T$  (set of runs) a regular property  $E$  (represented by an NFA for its bad prefixes), the main question is
  - does  $T$  satisfy  $E$ ?
  - $T \models E$  hold ?

Abbiamo il nostro sistema che sono le transizioni  $T$ , una proprietà regolare  $E$  e vogliamo fare il model checking risolvendo il problema se  $T$  verifica la proprietà  $E$ .

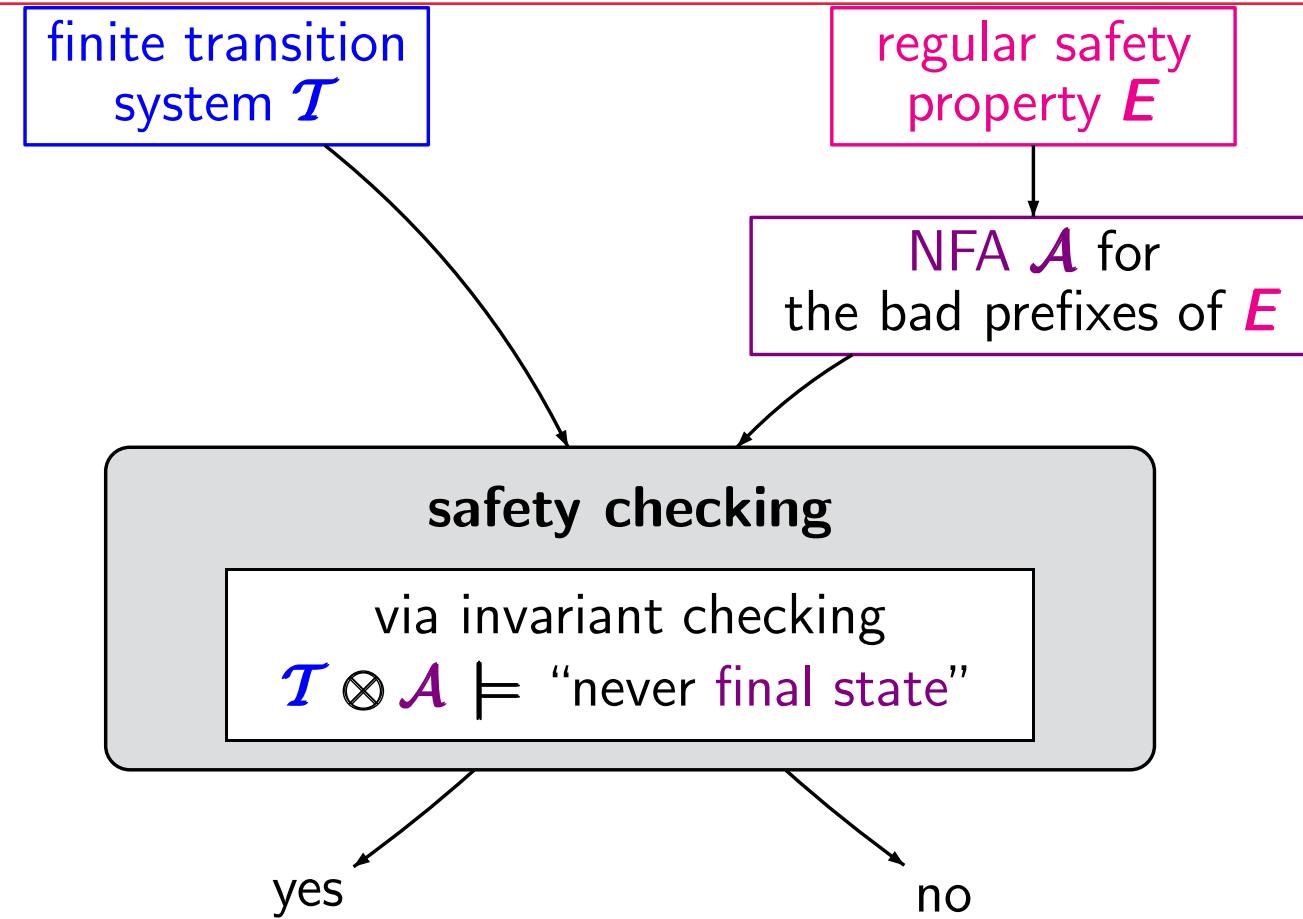


## VERIFYING REGULAR PROPERTIES

In realtà non ci interessa la proprietà  $E$  ma ci interessa il suo negato quindi costruiamo l'automa a stati finiti del negato della proprietà. L'automa a stati finiti del negato della proprietà == insieme dei bad prefixes. Avendo in pasto il negato la domanda rimane uguale ovvero: verifica o no la proprietà?



Facciamo il prodotto tra il modello del sistema e l'automa che nega la proprietà. A questo punto il problema di verificare E viene ridotto al problema di dire: il sistema dei comportamenti iniziali prodotto il sistema che ottengo negando la proprietà ( fare il prodotto vuol dire vedere ed eseguire passo passo per tutti i possibili comportamenti di T il negato della proprietà.) e la verificherà quando l'automa che ottengo facendo questo prodotto non raggiunge mai uno stato finale, se non raggiunge mai uno stato finale vuol dire che il sistema dei BadPrefixes non è soddisfatto e quindi vuol dire che E è soddisfatta. Se l'intersezione tra i comportamenti di T e i comportamenti dei prefissi sbagliati non è vuota, se non è vuota significa che ho un comportamento che mi va nello stato finale e dunque il sistema non verifica la proprietà.



finite transition system  $\mathcal{T}$

regular safety property  $E$

NFA  $\mathcal{A}$  for the bad prefixes of  $E$

### safety checking

via invariant checking

$\mathcal{T} \otimes \mathcal{A} \models$  “never final state”

yes

intersezione vuota

no + error indication

intersezione non vuota dunque posso prendere una traccia che è una che mi porta nello stato finale erroneamente dunque questo è esattamente il CONTROESEMPIO di cui parlavamo prima.

C'è una teoria che dice che queste 3 proprietà sono equivalent:

1) il sistema  $\mathcal{T}$  verifica la proprietà  $E$  se le tracce di  $\mathcal{T}$  intersecate con il linguaggio dei bad prefixes di  $E$  è vuoto vuol dire che il comportamento intersecato con le tracce che io ho cattive non ha niente in comune dunque il sistema verifica la proprietà. In termini logici si esprime dicendo che: il prodotto del mio comportamento con l'automa associato ai bad prefixes mi da la proprietà invariante "always not  $F$ " ovvero non raggiungo mai lo stato finale

The following statements are equivalent:

- (1)  $\mathcal{T} \models E$
- (2)  $Traces_{fin}(\mathcal{T}) \cap \mathcal{L}(\mathcal{A}) = \emptyset$
- (3)  $\mathcal{T} \otimes \mathcal{A} \models$  invariant "always  $\neg F$ "

where " $\neg F$ " denotes  $\bigwedge_{q \in F} \neg q$

THEORY  
HELPS

a questo punto si ha un modo per costruire il model checker che può essere visto come l'algoritmo di costruzione del model checker. Ho un modello del comportamento  $T$ , l'automa a stati finiti della negazione della proprietà e l'output deve essere "si verifica la proprietà oppure un controesempio" allora avendo  $T$  ed  $A$  posso costruire esattamente il prodotto di 2 automi, a questo punto vado a vedere se il prodotto dei 2 automi non mi porta mai nello stato finale "always not  $F$ " e questo punto dico sì o no a seconda del risultato, ma come faccio a fare questa operazione?

*input:* finite TS  $\mathcal{T}$ ,  
NFA  $\mathcal{A}$  for the bad prefixes of  $E$   
*output:* “yes” if  $\mathcal{T} \models E$   
otherwise “no” + error indication

construct product transition system  $\mathcal{T} \otimes \mathcal{A}$

check whether  $\mathcal{T} \otimes \mathcal{A} \models \text{"always } \neg F\text{"}$

if so, then return “yes”

if not, then return “no”  $\leftarrow$  and an error indication

where  $F = \text{set of final states in } \mathcal{A}$

# MODEL CHECKING

vado a vedere se ho un cammino e lo faccio per tutti i possibili cammini dell'intersezione, a questo punto quello che viene fuori è che se trovo un cammino che mi sta nell'intersezione dico "no ed avrò un conto esempio" se invece trovo un intersezione vuota dico "si". Notare che questa operazione dal punto di vista della complessità è della dimensione del modello degli stati per la dimensione per la formule, quindi vuol dire che chiaramente è qualcosa di automatizzabile dunque ci sono degli algoritmi che lo fanno, è un procedimento effettivo però ho il problema della esclusione combinatoria degli stati perché se ho un modello con stati particolarmente grandi sono della size del numero di stati, il problema del model checking è quello di trovare dei modelli in grado di ridurre il numero degli stati. Siamo partiti dall'affermazione che nel 95 ha fissato Lowe il protocollo perchè lo spazio degli stati era ridotto e dunque è riuscito a fare questa operazione che abbiamo descritto molto velocemente in questo contesto.

construct product transition system  $\mathcal{T} \otimes \mathcal{A}$

IF  $\mathcal{T} \otimes \mathcal{A} \models \text{"always } \neg F\text{"}$

THEN return "yes"

ELSE compute a counterexample for  $\mathcal{T} \otimes \mathcal{A}$  and  
the invariant "always  $\neg F$ ",

i.e., an initial path fragment in the product

$\langle s_0, p_0 \rangle \langle s_1, p_1 \rangle \dots \langle s_n, p_n \rangle$  where  $p_n \in F$

return "no" and  $s_0 s_1 \dots s_n$

FI

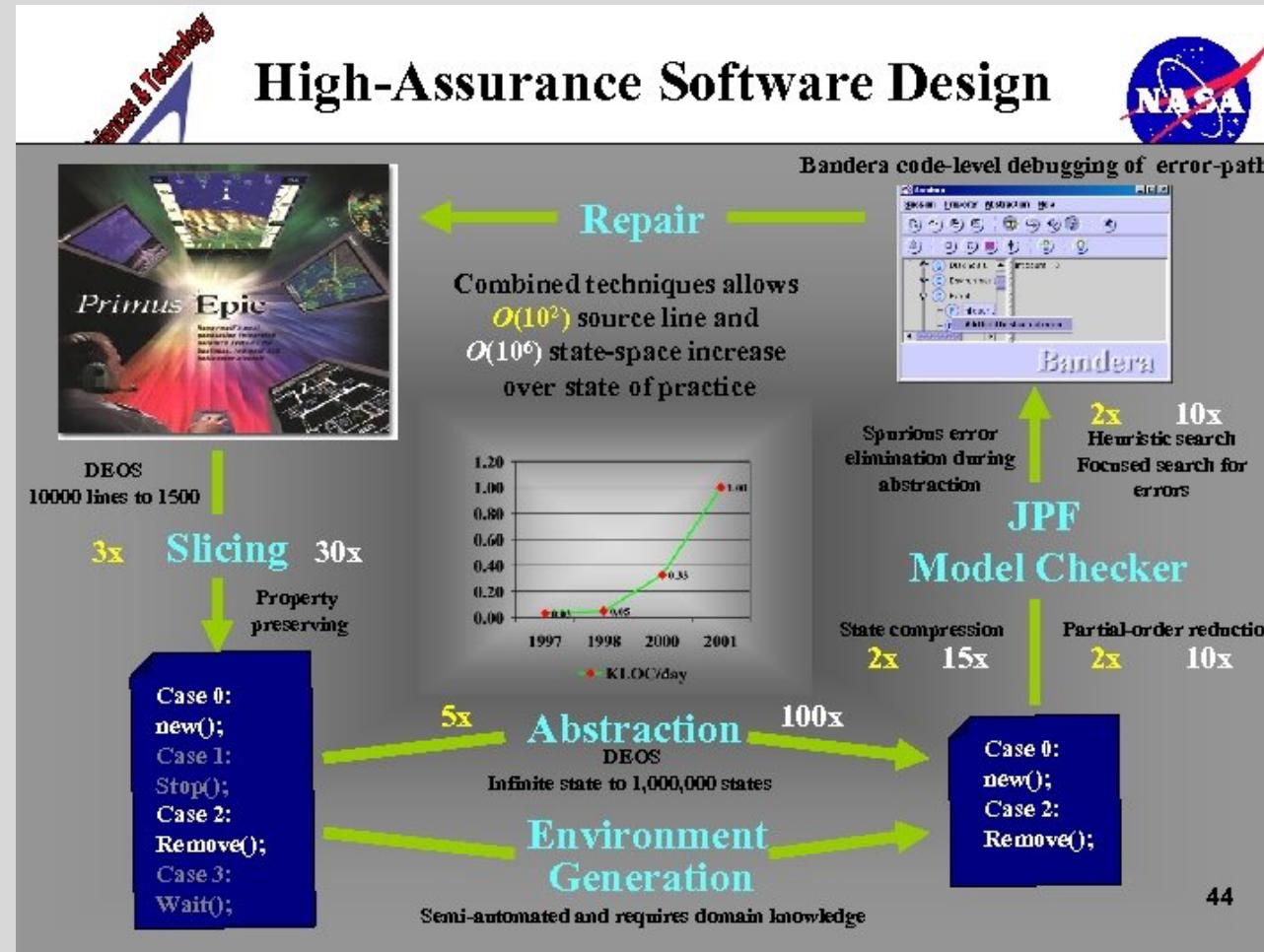
time complexity:  $\mathcal{O}(\text{size}(\mathcal{T}) \cdot \text{size}(\mathcal{A}))$

## MODEL CHECKING



EDMUND M. CLARKE, E. ALLEN EMERSON, JOSEPH SIFAKIS  
Model Checking: An Automated Quality Assurance Method

la NASA usa il software Model Checking per sviluppare il software di controllo dei loro sistemi e usa delle tecniche del model checking che proprio per evitare il problema delle estrusioni degli stati usano dei meccanismi di astrazione che permettono di ridurre la dimensione dello spazio degli stati da esaminare, in modo particolare in questo sistema che si chiama Bandera usa un model checking che si chiama JAVA PATH FINDER che è un model checker che vede l'assenza di violazioni in sezioni critiche di thread java



# Science of security

01

Transcend specific technologies and attacks, yet still be applicable in real settings,

essere in grado di affrontare problemi che vanno al di là di particolari tecnologie

02

Introduce new models and abstractions, thereby bringing pedagogical value besides predictive power,

essere in grado di introdurre dei nuovi modelli e astrazioni che possono affrontare bene la possibilità di predire nuovi comportamenti o astrazioni

03

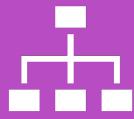
Facilitate discovery of new defenses as well as describe non-obvious connections between attacks, defenses, and policies, thus providing a better understanding of the landscape.

dare delle risposte delle nuove contromisure agli attacchi perché definire le contromisure in termini di modello ci permette di caratterizzare le proprietà del sistema che stiamo realizzando

# Security policies



**Confidentiality** refers to which principals are allowed to learn what information.



**integrity** refers to what changes to the system (stored information and resource usage) and to its environment (outputs) are allowed.



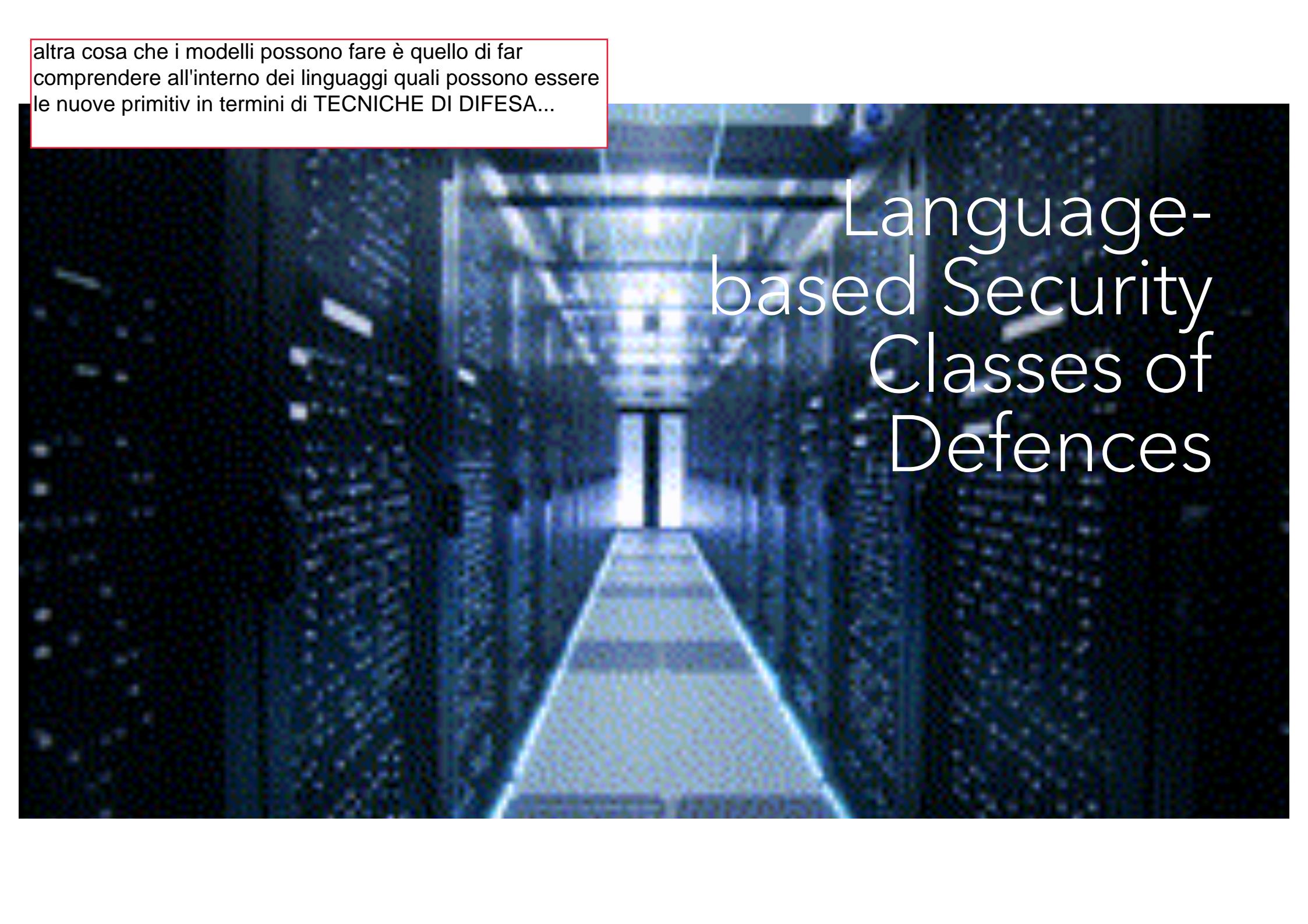
**Availability** refers to when must inputs be read or outputs produced.

# Issues

- Needed: widespread agreement on mathematical definitions for confidentiality, integrity, and availability.
- The CIA requirements are not orthogonal.
  - Any confidentiality property can be satisfied by enforcing a weak enough availability property: a system that does nothing cannot be accessed by attackers to learn secret information.
- Any classification of security policies is likely to be associated with some kind of **system model** with the **interfaces** the model defines (hence the **operations available to adversaries**)
  - A science of cybersecurity will not be built around a single model or around a single classification of policies.
    - **Traces - OS Reference Monitor**
    - **Hypertraces - Secure Compilation**

Un sistema che non fa niente ovvero un sistema che non ha comportamenti sicuramente garantisce l'integrity in quanto se non fa niente anche l'attaccante non è in grado di non far niente però ovviamente uno vorrebbe avere dei sistemi che fanno qualcosa e garantire le proprietà che quei sistemi in realtà qualcosa la fanno. In modo particolare andremo a vedere che tutta la teoria dei sistemi operativi che ha a che vedere con i monitor dei sistemi operativi ha un modello formale di comportamento che è quello delle tracce che abbiamo visto esattamente oggi per parlare di tracce safe dei protocolli di sicurezza. Per comprendere le proprietà dei reference monitor che vengono usati nei sistemi operativi per isolare dei comportamenti ha tutta la sua base formale in termini delle tracce come abbiamo visto oggi in termini dei protocolli. Quando andremo a vedere invece le tecniche di compilazione sicura in cui un astrazione ad alto livello viene mappata in un astrazione a basso livello dal compilatore, invece avremo bisogno di un qualcosa di più forte che sono le HYPERTRACES ovvero insiemi di insiemi di tracce che ci dicono come l'astrazione a livello alto viene compilata in un astrazione a livello più basso preservando le proprietà di sicurezza e quindi ci definisce esattamente il perimetro per cui possiamo fare delle affermazioni di correttezza di quello che stiamo facendo

altra cosa che i modelli possono fare è quello di far comprendere all'interno dei linguaggi quali possono essere le nuove primitiv in termini di TECNICHE DI DIFESA...



# Language-based Security Classes of Defences

l'isolamento a livello hw sappiamo esistere, per esempio con macchine virtuali, un processo ma questo avviene a basso livello quando invece potrei avere nel linguaggio delle primitive che mi isolano i comportamenti. Chiaramente poi abbiamo il problema della compilazione, il problema di comprendere che questo isolamento a livello del linguaggio è corretto. Il problema è che si potrebbe avere un enclave sicura a livello del linguaggio di programmazione ma sotto la compila in qualcosa dove tutti possono fare ciò che pare allora a quel punto è inutile che ce l'ho a livello del linguaggio di programmazione se il compilatore nel backend non mi garantisce la proprietà. Però analizzarlo in termini del modello ci fa vedere come possiamo estrarre dal modello delle caratteristiche che poi corrispondono a proprietà linguistiche

# ISOLATION

- Execution of one program is somehow prevented from accessing interfaces that are associated with the execution of others.
- Examples include physically isolated hardware, virtual machines, and processes (which, by definition, have isolated memory segments).

il monitoring è una cosa tipica dei sistemi operativi. Nei sistemi operativi si ha in genere il reference monitor che analizza il comportamento del sistema ispezionando il comportamento e può bloccarlo, può guidare il comportamento verso altre operazioni nel caso in cui l'attaccante o il programma in esecuzione cerca di fare cose che non sono prescritte. Posso fare la stessa cosa a livello del linguaggio di programmazione: come programmo il codice? come lo compilo? come faccio in modo che la proprietà di sicurezza del monitor valga sia a livello dell'astrazione all'interno del linguaggio che all'interno del codice compilato. Cosa mi garantisce che quello che ho scritto a livello del linguaggio poi sotto ha le stesse proprietà? In che modo riesco a farlo?

## MONITORING

- A *reference monitor* is guaranteed to receive control whenever any operation in some specified set is invoked;
- it further has the capacity to block subsequent execution, which it does to prevent an operation from proceeding when that execution would not comply with whatever policy is being enforced.
- Examples include memory mapping hardware, processors having modes that disable certain instructions, operating system kernels, and firewalls.

Nei protocolli di crittografia rendo il contenuto non visibile ma quando metto a disposizione il mio codice in rete ad esempio, l'attaccante potrebbe ad esempio andare a vedere il mio codice e comprendere le proprietà del mio codice. Vorrei poter offuscare delle parti di codice, quelle critiche e pur mandando il codice in rete vorrei offuscarlo in modo tale che uno non comprenda quali sono gli aspetti di comportamento critici e ovviamente quello che uno vorrebbe avere sono delle tecniche di compilazione che fanno trasformazione di programmi preservando offuscamento del codice perchè se la trasformazione del programma me lo mette in evidenza allora a quel punto il codice prodotto l'attaccante lo va a vedere con un opera di reverse engineering .

## OBFUSCATION

- Code or data is transmitted or stored in a form that can be understood only with knowledge of a secret. That secret is kept from the attacker, who then is unable to abuse, understand, or alter in a meaningful way the content being protected.
- Examples include data encryption, digital signatures, and program transformations that increase the work factor needed to craft attacks.

gli errori non avvengono solo in termini della sicurezza ma anche quando uno sviluppa programmi che non devono essere sicuri infatti i metodi di analisi formale valgono anche in caso di programmi in generale ma nel caso della sicurezza abbiamo anche delle ragioni stringenti che ci portano ad essere sicuri che il programma che abbiamo sviluppato non ha dei flow interni perchè i flow interni potrebbero essere usati da attaccanti.

## Why Formal Methods?

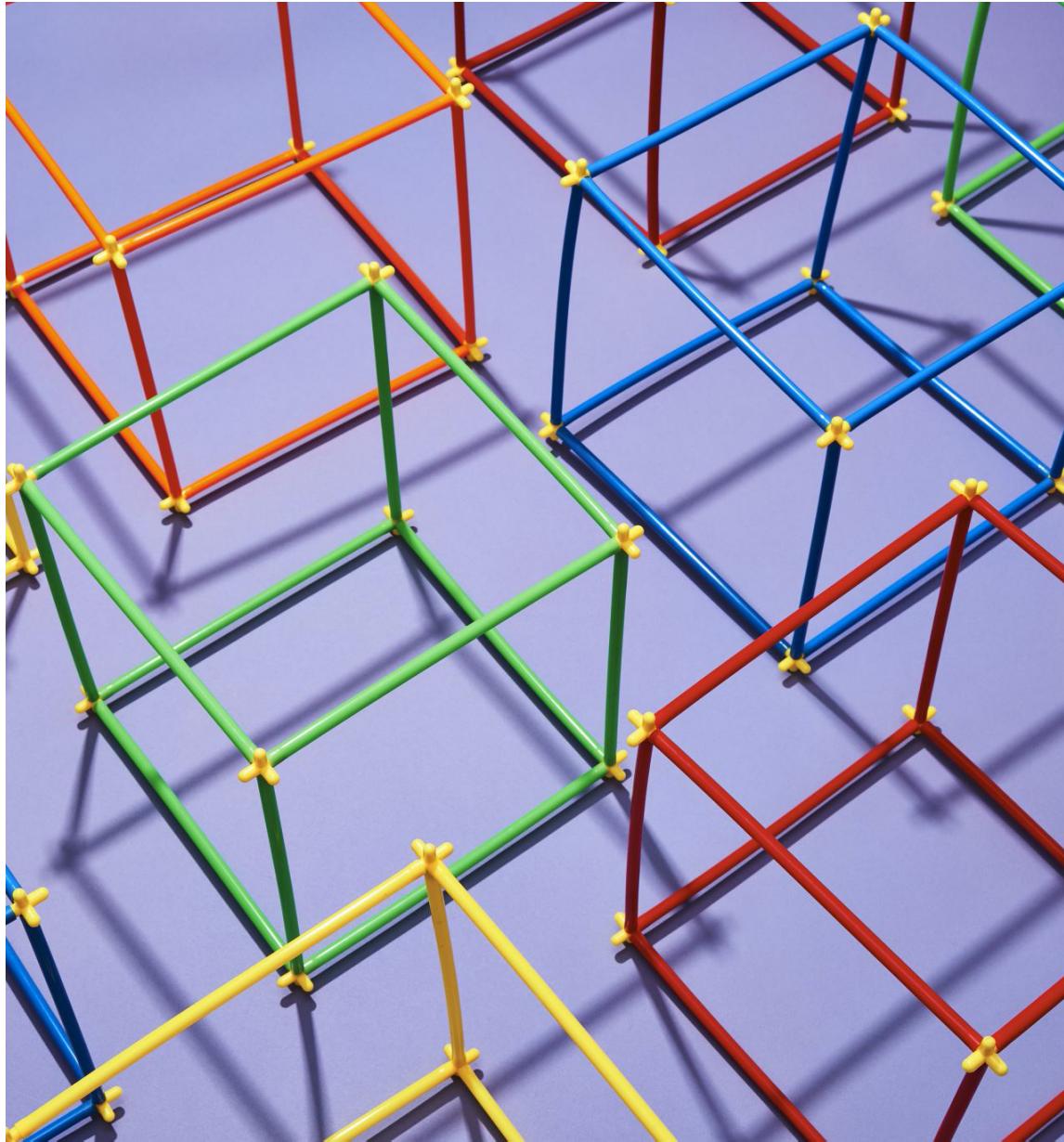
- Attacks are possible only because a system we deploy has flaws in its implementation, design, specification, or requirements.
- Eliminate the flaws and we eliminate the need to deploy defenses.
- But even when the systems on which we rely aren't being attacked, we should want confidence that they will function correctly. The presence of flaws undermines that confidence.
- So cybersecurity is not the only compelling reason to eliminate flaws.

# Formal Methods

- They are used by companies like Microsoft to validate device drivers and Intel to validate chip designs.
- Formal methods are the engine behind strong type-checking in modern programming languages (for example, Java and C#) and various code-analysis tools used in security audits (JSFLOW).

# Experimental computer science

- Computer systems are very complex
  - experimentation is useful as in the natural sciences,
  - we expect to find experimentation an integral part of a science of security.



## Concluding Remarks

- The science of cybersecurity should lead to new ideas about how to build systems and defenses, the validation of those proposals could require building prototypes.
- Prototypes are built in support of a science of cybersecurity expressly to allow validation of assumptions and observation of emergent behaviors.
- The science of cybersecurity will involve some amount of formal methods as well as some amount of experimentation.
- ... this is the core idea of Language-based Technology for Security.

# Discussion