UNIVERSITÀ DI PARMA

Dipartimento di Scienze Matematiche, Fisiche e Informatiche
Corso di Laurea Magistrale in Scienze Informatiche

TESI DI LAUREA MAGISTRALE

Rilevamento Statico di Access Control Incompleteness in Smart Contract EVM

Analisi Taint Relazionale per Bridge Cross-Chain

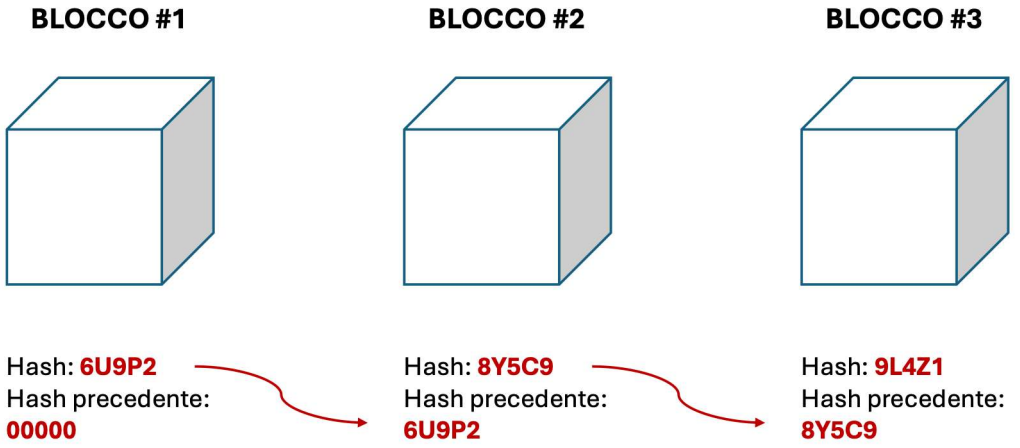
CANDIDATO
Saverio Mattia Merenda
Matricola 376544

RELATORE
Prof. Vincenzo Arceri

ANNO ACCADEMICO
2025 / 2026

Introduzione alla Blockchain

- **Registro pubblico condiviso**
- **Blocchi concatenati**
- **Immutabile**
- **Nessuna autorità centrale**



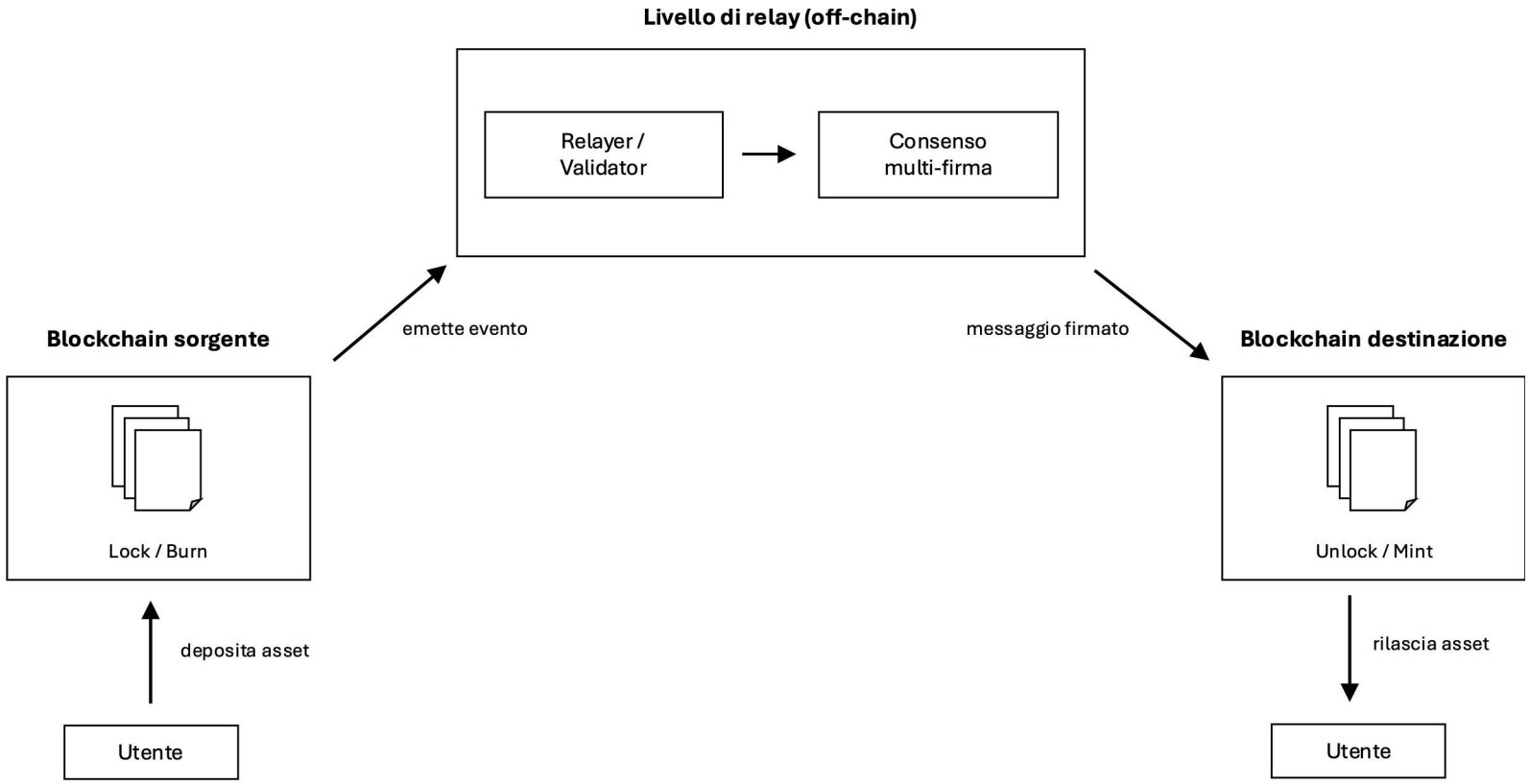
Cosa sono gli Smart Contract

- **Programmi** sulla Blockchain
- Regole eseguite **in automatico**, senza intermediari
- Gestiscono **denaro reale**
- Codice **immutabile → un bug non si corregge!**

Il problema: i Bridge Cross-Chain

- **Collegano** blockchain diverse
- **Custodiscono** ingenti somme
- Furti per **controllo di accesso difettoso:**
  - Ronin bridge $625M (2022)
  - Nomad bridge $190M (2022)

Il problema: i Bridge Cross-Chain



Vulnerabilità: Access Control Incompleteness

- Funzioni **pubbliche**, chiamabili da chiunque
- **Azioni sensibili**: cambiare proprietario, scrivere dati, muovere fondi
- **Vulnerabilità**: input attaccante → azione sensibile, **senza controllo**
- Vale **su ogni cammino**

Vulnerabilità: Access Control Incompleteness

```
contract Vault {
    address public owner;

    // funzione pubblica: chiunque può chiamarla
    function setOwner(address newOwner) public {
        owner = newOwner;   // azione sensibile, nessun controllo
    }
}
```

Vulnerabilità: Access Control Incompleteness

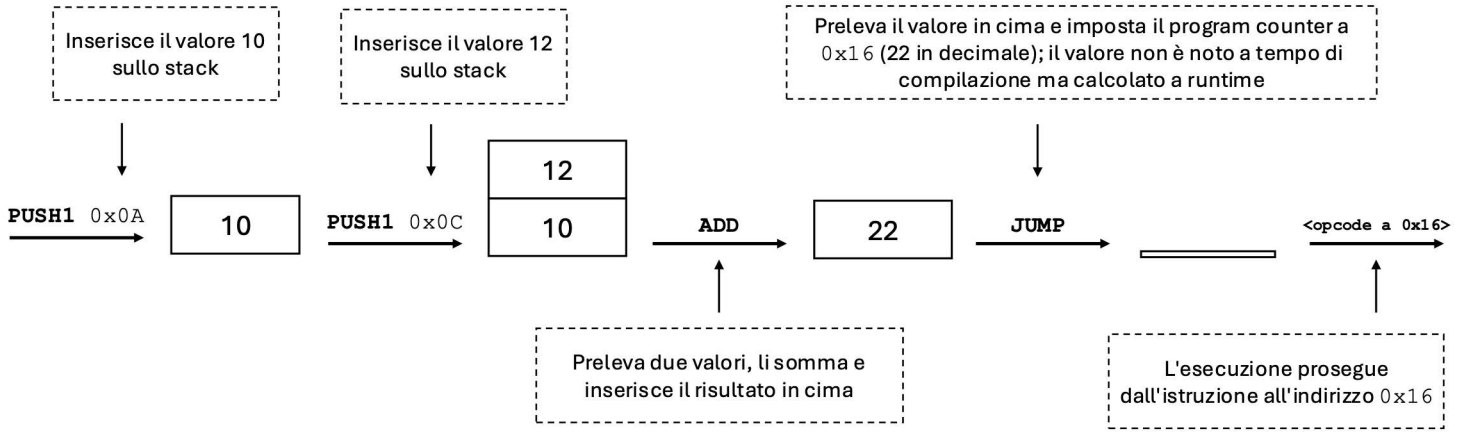
```
contract Vault {
    address public owner;

    function setOwner(address newOwner) public {
        require(msg.sender == owner); // controllo di autorizzazione
        owner = newOwner;             // azione sensibile protetta
    }
}
```

Obiettivo e contributi

- **Obiettivo:** rilevare la vulnerabilità **dal solo EVM bytecode**
- **Contributi:**
  1. Control-Flow Graph **completo *(sound)***
  2. Taint Analysis **Relazionale**
  3. Checker a **due fasi** → protezione su **tutti i cammini**
  4. **Interfaccia modifier**
  5. **Valutazione** su 16 bridge reali

La sfida tecnica: gli *orphan jump*

- Punto di partenza: il **Control-Flow Graph**
- I **salti** (JUMP / JUMPI) non dicono *dove* vanno
- Destinazione **calcolata a runtime**, presa dallo stack
- Stack ignoto → salto irrisolto → **CFG incompleto**



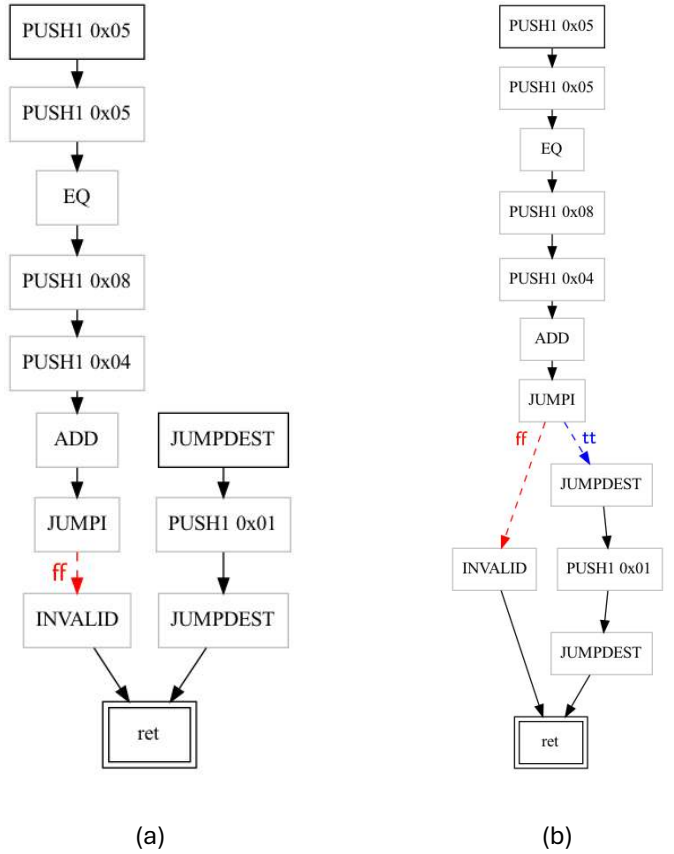
EVMLiSA:[1,2] approssimare lo stack

- **Interpretazione Astratta**: approssimare invece di eseguire
- Si approssima lo **stack** in ogni punto
- Dominio a **3 livelli**:
  - *k*-integers
  - stack di altezza *h*
  - insiemi di ≤ *l* stack

1. Arceri, **Merenda**, Dolcetti, Negrini, Olivieri, Zaffanella. *"Towards a Sound Construction of EVM Bytecode Control-Flow Graphs",* doi: 10.1145/3678721.3686227
2. Arceri, **Merenda**, Negrini, Olivieri, Zaffanella. *"EVMLiSA: Sound Static Control-Flow Graph Construction for EVM Bytecode",* doi: 10.1016/j.bcra.2025.100384

EVMLiSA: costruzione del CFG *sound*

- Grafo **incompleto**: salti scollegati
- Stack astratto **scopre** nuove destinazioni → archi
- Nuovi archi → **si ripete** fino a stabilità (***fixpoint***)
- **Terminazione** e **soundness**

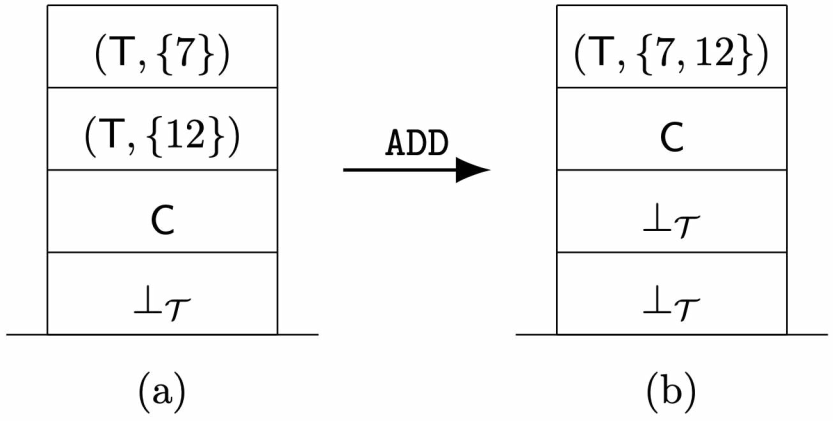


Dal CFG alla Taint Analysis Relazionale

- **Taint Analysis**: seguo i dati *contaminati* fino alle *operazioni sensibili*
  - *Source:* input controllati dal chiamante
  - *Sink:* operazioni che modificano lo stato o trasferiscono fondi
- **Relazionale:** ogni dato ricorda **la provenienza**
- Distinguo flussi diversi anche se si incrociano

Dominio Taint Relazionale

- Taint classico: *tainted / clean*
- Ogni valore *tainted* porta la **provenienza**
- Elemento relazionale: (τ, P)
  - $\tau \in \{\mathsf{T}, \mathsf{C}\}$: tainted / clean
  - $P \subseteq \mathbb{N}$: program point sorgente
- Combinando valori tainted: **unione** delle provenienze
- Vantaggio: **distinguo i flussi**



La regola: protezione su tutti i cammini

- **Sanitizer:** JUMPI sul *program point* corretto
- **Proprietà:** ogni cammino *source* → *sink* attraversa un *sanitizer* pertinente
- **Checker a 2 fasi:**
  - **trova** i sanitizer
  - **ogni cammino** → sanitizer
- Un solo cammino scoperto → **warning**

- *Source:* CALLDATALOAD, CALLDATACOPY, CALLDATASIZE, CALLER, ORIGIN, CALLVALUE
- *Sink:* SSTORE, CALL, CALLCODE, DELEGATECALL, STATICCALL

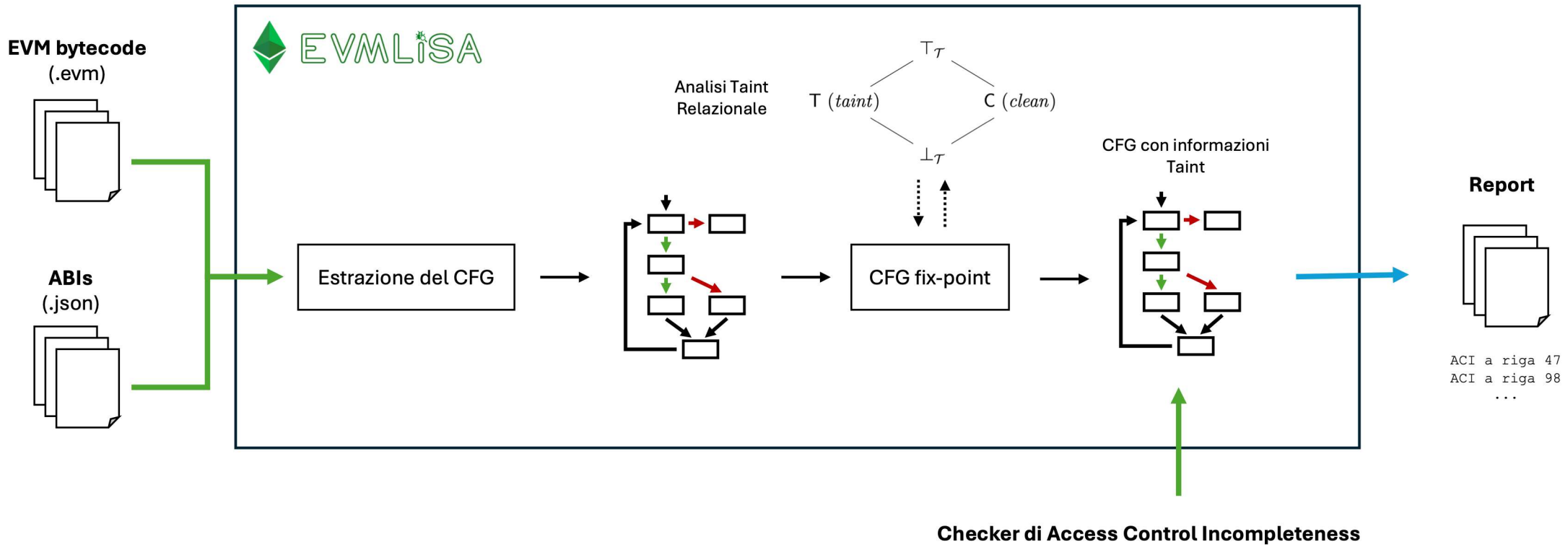
L'ostacolo dei *modifier*

- Controlli scritti come ***modifier*** (es. onlyOwner)
- Compilatore: **espansione inline** → guardia = ramo qualunque
- Checker non la riconosce → **falso positivo**
- **Soluzione**: annotare i *modifier*, soppressione **conservativa**

L'ostacolo dei *modifier* (esempio)

```
function setContent(bytes32 node, bytes32 hash) onlyOwner(node) public {
  records[node].content = hash;
}

modifier onlyOwner(bytes32 node) {
  if (ens.owner(node) != msg.sender)
    throw;
  _;
}
```

Struttura di EVMLiSA



Valutazione: il benchmark

- Dataset **SmartAxe:**[1]
  - 16 bridge
  - 1.003 contratti
  - 109.952 righe
- Etichette originali inaffidabili: loro tool sul loro benchmark → **0 warning**
- Ground truth ricostruita **a mano** → **301** vulnerabilità

1. Liao, Nan, Liang, Hao, Zhai, Wu, Zheng. *"SmartAxe: Detecting Cross-Chain Vulnerabilities in Bridge Smart Contracts via Fine-Grained Static Analysis",* doi: 10.1145/3643738

Risultati

Configurazione	Warning	True Positive	False Positive	Precision	Recall
SmartAxe (etichette pubbl.)	19	11	8	57,89%	3,65%
SmartAxe (riesecuzione)	0	0	0	n/d	0%
solo *bytecode* + *ABI*	411	301	110	73,24%	**100%**
con interfaccia *modifier*	319	301	18	**94,36%**	**100%**

- **Recall 100%** in entrambe le configurazioni di EVMLiSA
- Interfaccia *modifier:* falsi positivi **110** → **18**
- Prestazioni: **mediana 5,79s**

Conclusioni e lavori futuri

- **CFG sound** + **Dominio Taint Relazionale** + **verifica *all-paths***
- Bridge reali: **100% recall** e **94,36% precision**
- Lavori futuri:
  - Analisi inter-contratto
  - Gestione **atomicità**

UNIVERSITÀ
DI PARMA

Rilevamento Statico di Access Control Incompleteness in Smart Contract EVM

Analisi Taint Relazionale per Bridge Cross-Chain

PUBBLICAZIONI E CONTRIBUTI SCIENTIFICI

1. Arceri, **Merenda**, Dolcetti, Negrini, Olivieri, Zaffanella
   Towards a Sound Construction of EVM Bytecode Control-Flow Graphs
   FTfJP 2024 (ACM) - doi: 10.1145/3678721.3686227

2. Arceri, **Merenda**, Negrini, Olivieri, Zaffanella
   EVMLiSA: Sound Static Control-Flow Graph Construction for EVM Bytecode
   Blockchain: Research and Applications, 2025 - doi: 10.1016/j.bcra.2025.100384

3. **Merenda**, Olivieri, Arceri
   Access Control Incompleteness Detection in Cross-Chain Bridge Contracts via Static Taint Analysis
   IEEE TOPS - **sottomesso (2026)**

CANDIDATO
Saverio Mattia Merenda
Matricola 376544

RELATORE
Prof. Vincenzo Arceri

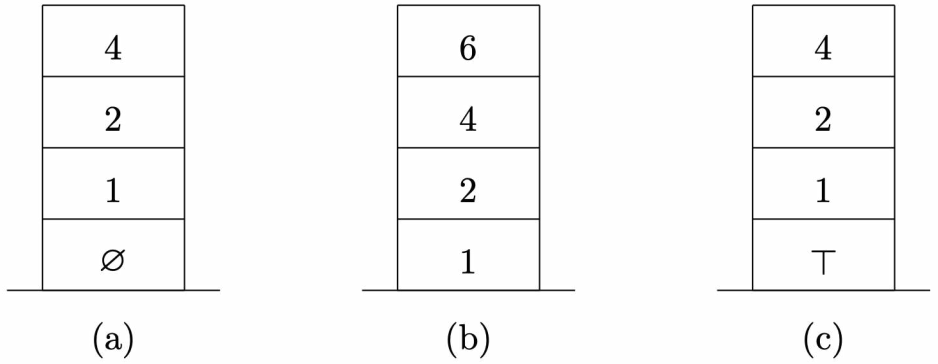
ANNO ACCADEMICO
2025 / 2026

Bonus: dominio dei *k*-interi

- Astrae i **valori singoli** sullo stack
- $\left(\mathbb{Z}^{\sharp} = \mathbb{Z} \cup \emptyset, \top_{\overline{\mathbb{Z}}}, \top_{\mathbb{Z}^{\sharp}}\right)$
  - $\emptyset$: slot vuoto
  - $\top_{\overline{\mathbb{Z}}}$: ignoto, **non utilizzabile** come JUMPDEST
  - $\top_{\mathbb{Z}^{\sharp}}$: completamente ignoto → serve ad **evitare archi falsi**

Bonus: stack astratti di altezza *h*

- Astrae lo **stack operandi**
- $\Sigma^h = [v_0, \ldots, v_{h-1}]$ con $v_i \in \mathbb{Z}^\sharp$
  - Al più le prime *h* posizioni
  - Cima dello stack a **destra**
- Stack più corti: in basso ∅
- Stack più profondi: parte bassa non tracciata



Bonus: insiemi di stack astratti

- Più cammini $\rightarrow$ stesso program point
- Dominio: $\Sigma^{h,l} = \wp_{\leq l}(\Sigma^h)$
- Insiemi di al più l stack astratti
- Join:
  - cardinalità $\leq l$: stack distinti
  - oltre l: collasso a $\top$
- Effetto: **terminazione** del punto fisso

Bonus: un falso positivo onesto: _permit

```
function _permit(address owner, address spender, uint256 value,
                 uint256 deadline, uint8 v, bytes32 r, bytes32 s) internal {

    bytes memory data = abi.encode(
        PERMIT_TYPEHASH, owner, spender, value,
        _permitNonces[owner]++,  // SSTORE su owner non validato (sink)
        deadline);

    // verifica firma DOPO: se fallisce, revert annulla l'SSTORE
    require(
        EIP712.recover(DOMAIN_SEPARATOR, v, r, s, data) == owner,
            "EIP2612: invalid signature");

    _approve(owner, spender, value);
}
```