

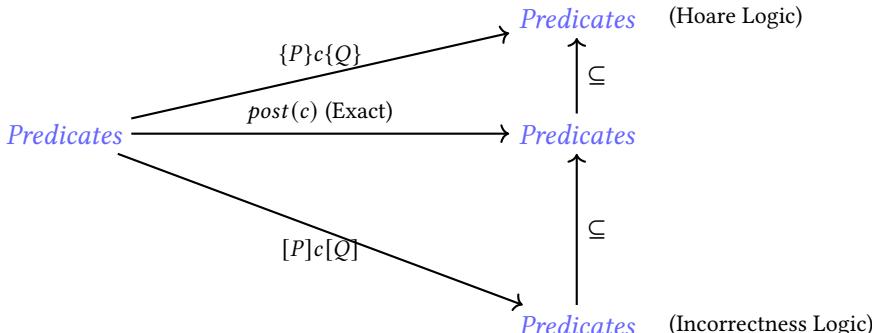
# 1 Skja: Adversarial Logic on Steroids

2  
3 AKA SAI LALITH KUMAR, University of Colorado Boulder, USA

4 We introduce *Skja*, a new incorrectness logic [9] which extends on *Adversarial Logic* [11] with spi-calculus.  
5 Adversarial logic is very interesting in that it lets you apply under-approximation for exploitability analysis.  
6 The current core limitation of adversarial logic is that it only allows for parallel composition over multiple  
7 programs but it has problems with the model. With Skja, we bring the rich theory of spi-calculus to let the  
8 logic be composable for any infinite arbitrary set of programs which may be adversarial or not. This lets us  
9 encode and talk about attacks like Man-in-the-middle (MiTM) attacks and DDoS (Denial of Service) attacks.  
10 For this end, we first prove that parallel composition is a monoid over the adversarial logic grammar. We  
11 change the denotational semantics and program rules to support n-ary parallel composition while also proving  
12 that the changes do not effect the soundness. We also show.

## 13 1 INTRODUCTION

14 When one reasons about programmes, we have access to a rich variety of logics, and one of them  
15 would be incorrectness logic [9] which is equivalent to [5]. The logic targets under-approximating  
16 the post of a condition  $c$ ; this is symmetrical to the work done by Hoare[7] which over-approximates.  
17 The way we can understand this through diagrammatic representation is by visualising the state  
18 space as a hierarchy of set inclusions, taken from O'Hearn's presentation. As shown in Figure 1,  
19 the central arrow represents the *strongest postcondition* ( $\text{post}(c)$ ), which describes exactly the set of  
20 states reachable by the program.  
21



35 Fig. 1. The hierarchy of program logic. Hoare logic (top) over-approximates the reachable states, while  
36 Incorrectness logic (bottom) under-approximates them.

37 In this hierarchy, the top layer corresponds to standard Hoare Logic. It provides an *over-*  
38 *approximation*, meaning the postcondition  $Q$  must contain all reachable states ( $\text{post}(c) \subseteq Q$ ).  
39 This is useful for proving safety (i.e., "nothing bad happens").  
40

41 Conversely, the bottom layer represents under-approximation. Here, the postcondition  $Q$  must be  
42 a *subset* of the reachable states ( $Q \subseteq \text{post}(c)$ ). This logic is useful for proving the existence of bugs  
43 or specific reachable states (i.e., "something bad definitely happens"). Now, you can also observe,  
44 the triangle for top is rather short compared to the triangle for the bottom. This is a deliberate choice  
45 to highlight the fact that over-approximation enjoyed 5 decades of being in the front seat [1–3]  
46 and hence the closeness to the perfect postcondition.  
47

To show case an example with under-approximation, we take a rather simple looking but tricky example derived from [5] below comparing the validity of the triplets shown below:

$\langle T \rangle \quad x := y; z := x \quad \langle z = y \rangle \quad \text{Not Valid}$

This triplet is invalid in under-approximation logic. While it is true that  $z = y$  in the final state, the assertion  $\langle z = y \rangle$  as a set includes states where  $x \neq y$  (e.g., state  $\{x = 0, z = 1, y = 1\}$ ). However, the program guarantees  $x = y$ . Because the assertion includes "extra" states that the program logic cannot reach, it is not a valid subset (under-approximation) of the true postcondition.

$\langle T \rangle \quad x := y; z := x \quad \langle z = y \wedge x = y \rangle \quad \text{Valid}$

This triplet is valid. By conjoining  $x = y$ , we restrict the postcondition to exclude the unreachable states. The resulting set matches exactly the behavior of the code, satisfying the requirement  $Q \subseteq post(c)$ . Application of under-approximation to security reasoning is a very novel yet fundamental contribution [11]. And this contribution brings proving software or protocols are secure to the realm of finding bugs or vulnerabilities aka exploitability analysis [6]. A simple analysis tool running on a rather not complicated software usually returns many bugs [10] and many of them are usually false positives. In the era of move fast, break fast [4] this puts a lot of burden on security bug triagers who have to classify each bug for its true validity and slows the development cycle a lot given if a team moves with an unsure bug. This breaks rythm and destroys planning of the required

## 2 OVERVIEW

### 3 NEW SEMANTICS

In this section we build all the semantics and rules which transform adversarial logic to Skja. First, we assume the reader is familiar with the semantics of adversarial logic and refer them to the original paper for the full set [11], our work states what is new to the established semantics.

First we change the state space. In AL, states were rigid couples  $(\sigma_p, \sigma_a)$ . To support  $n$ -ary composition, we map Process IDs to their local stores.

**DEFINITION 1 (GENERAL STATE SPACE).**  $\Sigma ::= [PIDs \rightarrow Variables \rightarrow Values]$

Let  $\Pi \in \Sigma$  be a global state. We denote  $\Pi(i)$  as the local store of process  $i$ . We also maintain the global channel state  $\Gamma \in [Channels \rightarrow List(Values)]$ . A configuration is a tuple  $\langle \Pi, \Gamma \rangle$ .

A key notation change from adversarial logic is that instead of using  $[\epsilon : P]$  where  $\epsilon \in \{ad, ok\}$ , we define  $\Delta$  as a mapping from PIDs to assertions.

**DEFINITION 2 (GENERALIZED TRIPLE).**  $[\Delta_{pre}]C[\Delta_{post}]$

For processes Alice ( $A$ ), Bob ( $B$ ), and Eve ( $E$ ),  $\Delta$  maps identities to assertions:  $\{A : P_A, B : P_B, E : P_{Eve}\}$ . The triple  $[\Delta]C[\Delta']$  is valid if, for all processes  $i \in \text{dom}(\Delta)$ , the local execution satisfies the under-approximate relation between  $\Delta(i)$  and  $\Delta'(i)$ .

#### 3.1 Syntax of Assertions

To reason about what an adversary knows, we define the syntax for expressions and assertions, adapting the model from [11].

**DEFINITION 3 (SYNTAX).**

$Variables \ V ::= x \mid n \mid \alpha$

$Expressions \ E ::= V \mid \text{rand}() \mid \text{enc}(E, K) \mid \langle E, E \rangle$

$Assertions \ P, Q ::= E = E \mid \text{Knows}(E) \mid P \wedge Q \mid \exists x.P$

99 Here,  $\text{Knows}(E)$  is a predicate indicating that the value  $E$  is derivable from the agent's current  
 100 knowledge set using Dolev-Yao deduction rules.

### 101 102 3.2 Semantics

103 DEFINITION 4 (SMALL-STEP SEMANTICS FOR COMPOSITION). Let  $C_1 \parallel \dots \parallel C_n$  be a composition of  
 104  $n$  processes.

- 105 • **Independent Step:** If process  $i$  takes a local step  $\langle \sigma_i, \Gamma \rangle \rightarrow \langle \sigma'_i, \Gamma \rangle$ , then:

$$106 \langle \Pi[i \mapsto \sigma_i], \Gamma \rangle \rightarrow \langle \Pi[i \mapsto \sigma'_i], \Gamma \rangle$$

- 108 • **Communication Step:** If process  $i$  writes to channel  $c$  and process  $j$  reads from channel  $c$ ,  
 109 they synchronize:

$$111 \langle \Pi, \Gamma \rangle \xrightarrow{i!c(v), j?c(v)} \langle \Pi', \Gamma' \rangle$$

112 where  $\Pi'$  updates the local variables of  $i$  and  $j$  appropriately.

- 113 • **Termination:**  $\text{skip} \parallel \dots \parallel \text{skip}$  is the terminal configuration.

114 We restate the properties of channels and local variables for processes from adversarial logic.

115 HYPOTHESIS 1 (OWNERSHIP PRINCIPLE).

- 116 • No two parallel processes share the same local variables and are distinctly separated (Disjointness).  $\forall i \neq j, \text{dom}(\Pi(i)) \cap \text{dom}(\Pi(j)) = \emptyset$ .
- 117 • They can only communicate through channels, which we state are global and represented by  $\Gamma$ .

### 118 119 3.3 Inference Rules

120 To support the  $n$ -ary composition defined above, we extend the rules of adversarial logic by  
 121 modifying the

122 1. **The n-Par Rule.** This rule allows us to combine independent proofs of  $n$  processes. It relies  
 123 on the disjointness hypothesis.

$$128 \frac{\text{n-PAR}}{\forall i \in \{1..n\}, [\Delta_{\text{pre}}(i)]C_i[\Delta_{\text{post}}(i)]} [\Delta_{\text{pre}}]C_1 \parallel \dots \parallel C_n[\Delta_{\text{post}}]$$

129 2. **The n-Com Rule.** This is the workhorse of protocol verification when it comes to Skja. It  
 130 synchronizes a sender  $i$  and a receiver  $j$  over channel  $c$ . Other processes  $k$  remain idle (frame  
 131 property).

$$135 \frac{\text{n-COM}}{[\Delta_{\text{pre}}(i)]\text{out}(c, v)[\Delta_{\text{post}}(i)] \quad [\Delta_{\text{pre}}(j)]\text{in}(c, x)[\Delta_{\text{post}}(j)] \quad \forall k \notin \{i, j\}, \Delta_{\text{pre}}(k) = \Delta_{\text{post}}(k)} [\Delta_{\text{pre}}] (\dots \parallel \text{out}_i \parallel \dots \parallel \text{in}_j \parallel \dots) [\Delta_{\text{post}}]$$

136 Semantically, this implies a logical flow of the value  $v$  from  $i$  to  $j$ :

$$137 \exists v, (\Delta_{\text{pre}}(i) \implies x = v) \wedge (\Delta_{\text{post}}(j) \implies x = v)$$

### 138 139 3.4 The Dolev-Yao Adversary Model

140 To reason about protocols, we adopt the standard Dolev-Yao model, assuming the network is under  
 141 total control of the adversary. The adversary can read, block, and modify messages but cannot  
 142 break cryptography without the correct keys.

We formalize the attacker's capabilities using the deduction relation  $\vdash$ . Let  $\mathcal{K}$  be the set of messages currently known to the attacker. The relation  $\mathcal{K} \vdash m$  (read: "from knowledge  $\mathcal{K}$ , message  $m$  can be derived") is defined inductively:

$$\frac{m \in \mathcal{K}}{\mathcal{K} \vdash m} \text{(Axiom)} \quad \frac{\mathcal{K} \vdash m_1 \quad \mathcal{K} \vdash m_2}{\mathcal{K} \vdash \langle m_1, m_2 \rangle} \text{(Pairing)} \quad \frac{\mathcal{K} \vdash \langle m_1, m_2 \rangle}{\mathcal{K} \vdash m_i} \text{(Proj)}$$

$$\frac{\mathcal{K} \vdash m \quad \mathcal{K} \vdash k}{\mathcal{K} \vdash \{m\}_k} \text{(Encrypt)} \quad \frac{\mathcal{K} \vdash \{m\}_k \quad \mathcal{K} \vdash k^{-1}}{\mathcal{K} \vdash m} \text{(Decrypt)}$$

In Skja, this deduction relation connects the operational semantics to the assertion logic. We define the semantics of the assertion  $\text{Knows}(m)$  as follows:

$$\sigma \models \text{Knows}(m) \iff \sigma(\text{knowledge}) \vdash m$$

This definition is crucial for under-approximation. Unlike safety verification, where one must compute the potentially infinite set of all derivable knowledge to prove a secret is *never* leaked, Skja only requires demonstrating the existence of a single derivation tree. If there exists a sequence of rule applications allowing the adversary to construct the message  $m$  needed for the next step of the attack trace, the assertion holds. This makes the check constructive and efficient for exploit generation.

## 4 PROOFS

### 4.1 Proof of Composition ( $\parallel$ ) forms a monoid

The reason we would like to have composition as a commutative monoid is that it frees us from thinking about ordering and composition of operators, allowing us to treat the system as a "soup" of processes, consistent with the chemical abstract machine model often used for process calculi.

First, we define the structural congruence relation ( $\equiv$ ) for our logic. This relation identifies process terms that are syntactically different but semantically identical.

**DEFINITION 5 (STRUCTURAL CONGRUENCE).** *The parallel composition operator  $\parallel$  satisfies the following laws:*

$$P \parallel Q \equiv Q \parallel P \quad (\text{Commutativity}) \quad (1)$$

$$(P \parallel Q) \parallel R \equiv P \parallel (Q \parallel R) \quad (\text{Associativity}) \quad (2)$$

$$P \parallel \text{skip} \equiv P \quad (\text{Identity}) \quad (3)$$

**THEOREM 1.** *(Process,  $\parallel$ , skip) form a commutative monoid.*

**Proof:** A commutative monoid has three properties for it to satisfy. We show that  $(\parallel)$  satisfies all below:

- **Commutativity:** For some processes  $P$  and  $Q$ ,  $P \parallel Q \equiv Q \parallel P$ . Every derivation of a step from  $P \parallel Q$  is obtained by applying one of the small-step rules. Swapping the syntactic positions of  $P$  and  $Q$  gives a derivation of the same step from  $Q \parallel P$  (process IDs are intrinsic to the state, not the syntactic position). Therefore the sets of possible traces are identical.
- **Associativity:** For some processes  $P$ ,  $Q$  and  $R$ ,  $(P \parallel Q) \parallel R \equiv P \parallel (Q \parallel R)$ . The transition relation is defined on the set of PIDs. Grouping  $(P \parallel Q)$  is merely syntactic sugar for a set of processes  $\{P, Q\}$ . The execution semantics operate on the union of disjoint heaps. Since set union is associative, the composition is associative.

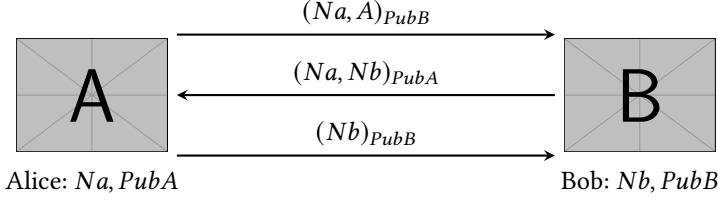


Fig. 2. Needham-Schroder

- **Identity:** For some process P,  $(P \parallel \text{skip}) \equiv P$ . The *skip* process has no transitions and modifies no state. Its interleaving with P adds no new traces and restricts no existing traces of P.

LEMMA 1 (PERMUTATIONS OF  $\parallel$ ). *For any set of processes  $\{C_1, \dots, C_n\}$ , any syntactic permutation of their composition results in an equivalent system under the defined denotational semantics.*

## 4.2 Soundness

We prove the soundness of the inference rules by showing that if the premises hold, the conclusion satisfies the under-approximation reachability property (Lemma 2).

LEMMA 2 (CHARACTERIZATION FOR N-ARY SYSTEMS). *The triple  $[\Delta_{\text{pre}}]C[\Delta_{\text{post}}]$  is true iff every state in  $[\Delta_{\text{post}}]$  is reachable from a state in  $[\Delta_{\text{pre}}]$  via the operational semantics of C.*

THEOREM 2 (SOUNDNESS OF N-PAR). *The n-Par rule is valid.*

PROOF. Let  $\Pi' \in [\Delta_{\text{post}}]$  be a target global state. By the definition of the state space,  $\Pi'$  decomposes into local stores  $\Pi'(i)$ . From the premises, for each  $i$ ,  $[\Delta_{\text{pre}}(i)]C_i[\Delta_{\text{post}}(i)]$  is valid, implying  $\exists \sigma_i \in [\Delta_{\text{pre}}(i)]$  such that  $\sigma_i \rightarrow^* \sigma'_i$  via  $C_i$ . By the **Ownership Principle** (Hypothesis 1), domains are disjoint. Consequently, the **Independent Step** semantics apply, allowing local transitions to be interleaved without interference. Thus, starting from  $\Pi = \bigcup_i \sigma_i$ , the global system reaches  $\Pi'$  via  $C_1 \parallel \dots \parallel C_n$ .  $\square$

THEOREM 3 (SOUNDNESS OF N-COM). *The n-Com rule is valid.*

PROOF. Let  $\Pi' \in [\Delta_{\text{post}}]$ . For idle processes  $k \notin \{i, j\}$ ,  $\Delta_{\text{pre}}(k) = \Delta_{\text{post}}(k)$ , satisfying the frame property via trivial reachability (0 steps). For interacting processes, the premises guarantee the existence of local pre-states  $\sigma_i \in [\Delta_{\text{pre}}(i)]$  and  $\sigma_j \in [\Delta_{\text{pre}}(j)]$  that can perform output and input respectively. The **Communication Step** semantics  $\langle \Pi, \Gamma \rangle \xrightarrow{i!c(v), j?c(v)} \langle \Pi', \Gamma' \rangle$  synchronizes these steps. Specifically, the semantic rule updates the receiver's store  $\sigma'_j$  such that  $x = v$ . Since the premises hold and the operational semantics explicitly enforce the data flow  $v \rightarrow x$ , the global post-state where receiver variable  $x$  holds sender value  $v$  is reachable from  $[\Delta_{\text{pre}}]$ .  $\square$

## 5 REASONING WITH SJKA: NEEDHAM-SCHRODER

### 5.1 Needham-Schroder

Needham-Schroder(NS)[8], is an authentication protocol. The way it runs is as follows:

We refer to figure 1, which shows the steps of NS protocol. First we see Alice initiating the protocol with Bob by sending here Nonce Na with her own ID paired together. This is done to make sure Bob knows who he is communicating with. Now, we see Bob responds to Alice with his own Nonce Nb and Na paired together; this is done so that Alice can confirm her nonce and

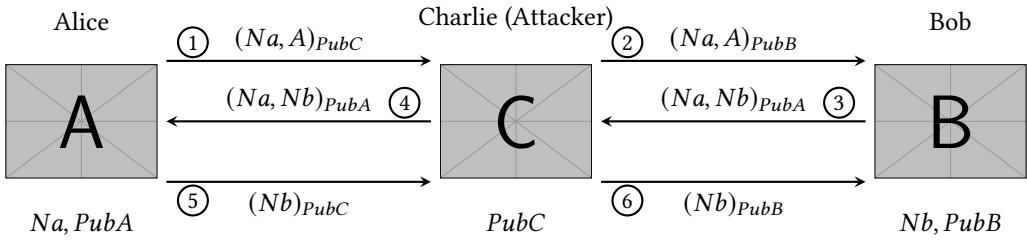


Fig. 3. Lowe's man-in-the-middle attack, where Charlie impersonates Alice.

ensure she is not talking to another third party, as only Bob can deduce her nonce. Now, finally, Alice sends back Bob's nonce in the same spirit as above.

This protocol is very sensible but hides a cunning attack in plain sight. We refer the reader to figure 2 for Lowe's attack. Here you see, Charlie is an impersonator. Who can be called a Dolev-Yao adversary as the semantics and powers of the same are defined in the section 2. To recap, he can replay messages, stop messages and forward them and strip messages if they're encrypted with his private key. Now in Lowe's attack, the security guarantees of Alice are maintained, meaning that they won't share the nonce with some other person who they don't know they are talking to, but for Bob it's broken. This can be observed at the end of the protocol run where Bob thinks he is talking to Alice, but it's actually Charlie who now has Bob's and Alice's nonce, so Charlie can act as Alice and keep the conversation going. This is a failure of authentication.

## 5.2 Formalizing the Attack in Skja

We demonstrate the validity of Lowe's attack by tracing the evolution of the generalized assertion map  $\Delta$  through the application of the **n-Com** rule.

Let the initial state be  $\Delta_0$ . The participants have the following initial knowledge:

$$\Delta_0 = \{A : \text{Knows}(Na, PrivA), B : \text{Knows}(Nb, PrivB), C : \text{Knows}(PrivC)\}$$

### Phase 1: Initiation and Impersonation (Steps 1 & 2)

Alice intends to talk to Charlie. She executes  $\text{out}(c, \{Na, A\}_{PubC})$ . Charlie reads from  $c$ . Applying **n-Com**:

$$[\Delta_0(A)]\text{out} \cdots \parallel \text{in}[\Delta_1(C)]$$

The synchronization ensures the value flows to Charlie. Since Charlie possesses  $PrivC$ , he decrypts the message.

$$\Delta_1(C) \implies \text{Knows}(Na)$$

Using this new knowledge, Charlie impersonates Alice. He constructs  $\{Na, A\}_{PubB}$  and sends it to Bob. Applying **n-Com** between Charlie and Bob:

$$\Delta_2(B) \implies \text{Received}(\{Na, A\}_{PubB})$$

Bob decrypts this. Crucially, Bob's internal state now asserts he is communicating with Alice.

### Phase 2: The Oracle Step (Steps 3 & 4)

Bob responds to the perceived initiator. He sends  $\{Na, Nb\}_{PubA}$ . Charlie intercepts this via **n-Com**.

$$\Delta_3(C) \implies \text{Knows}(\{Na, Nb\}_{PubA})$$

At this stage, Charlie *cannot* decrypt the message to learn  $Nb$  because he lacks  $PrivA$ . This is where the logic highlights the vulnerability. Charlie forwards the blob opaque to Alice (Step 4). Applying

295   **n-Com** between Charlie and Alice:

$$\Delta_4(A) \implies \text{Received}(\{Na, Nb\}_{PubA})$$

296   Alice, believing this is a valid response from Charlie (her intended partner), decrypts it. She verifies  
 297   298   *Na* and learns *Nb*.

### 300   **Phase 3: The Leak (Step 5)**

301   This is the fatal step. Following the protocol, Alice confirms the session by sending *Nb* encrypted  
 302   with her partner's key (*PubC*).

$$[\Delta_4(A)]\text{out}(c, \{Nb\}_{PubC}) \parallel \text{in}(c, x)[\Delta_5(C)]$$

303   Applying **n-Com**, Charlie receives  $x = \{Nb\}_{PubC}$ . Since  $\Delta_5(C)$  includes *PrivC*:

$$\Delta_5(C) \implies \text{Knows}(Nb)$$

307   The adversary has successfully extracted the secret nonce *Nb*.

### 308   **Phase 4: Completion (Step 6)**

309   Charlie re-encrypts *Nb* with *PubB* and sends it to Bob via **n-Com**. Bob verifies *Nb* and completes  
 310   the session.

### 311   **Conclusion**

312   The final state  $\Delta_{final}$  contains a contradiction to the authentication specification:

- 314   (1)  $\Delta_{final}(B) \implies \text{Partner} = Alice$
- 315   (2)  $\Delta_{final}(C) \implies \text{Knows}(Na, Nb)$

316   In Skja, the reachability of  $\Delta_{final}$  where the adversary knows *Nb* constitutes a proof of the exploit.  
 317   The proof script could be more integrated to look like the reasoning of adversarial logic, but we  
 318   choose to keep it focused on the rules we have defined.

## 319   6 RELATED WORK

321   Related work in program analysis and formal verification of protocols and softwares come with a  
 322   rather rich history. We stand on the shoulders of the giants, and a small exposition in this section  
 323   will never give with an *n*-ary composition model based on the spi-calculus, allowing for more  
 324   complex network topologies.

## 325   7 CONCLUSION

327   In this paper, we adapted under-approximation logic for security and boosted its versatility by  
 328   integrating techniques from the spi-calculus. We demonstrated that this extension preserves the  
 329   soundness of the underlying logic through formal proofs. Finally, we applied Skja to the Needham-  
 330   Schroeder protocol, formally proving the validity of the classic Man-in-the-Middle vulnerability.

## 331   ACKNOWLEDGMENTS

333   I thank my advisor, Dr. Gowtham Kaki. I acknowledge Dr. Bohr-Yuh Evan Chang and Kirby Linvill  
 334   for their instruction on PL. Thanks to Dakota Brayan for feedback, and Dr. Fabio Somenzi for  
 335   teaching me about logic.

## 336   REFERENCES

- 338   [1] Nahid A Ali. 2017. A survey of verification tools based on hoare logic. *International Journal of Software Engineering & Applications* 8 (2017), 87–100.
- 339   [2] Krzysztof R Apt. 1981. Ten years of Hoare's logic: A survey—part I. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 3, 4 (1981), 431–483.
- 340   [3] Krzysztof R Apt. 1983. Ten years of Hoare's Logic: a survey—Part II: nondeterminism. *Theoretical Computer Science* 28, 1-2 (1983), 83–109.

- 344 [4] David Cohen, Mikael Lindvall, and Patricia Costa. 2004. An introduction to agile methods. *Advances in computers* 62,  
345 03 (2004), 1–66.
- 346 [5] Edsko De Vries and Vasileios Koutavas. 2011. Reverse hoare logic. In *International Conference on Software Engineering  
and Formal Methods*. Springer, 155–171.
- 347 [6] Sarah Elder, Md Rayhanur Rahman, Gage Fringer, Kunal Kapoor, and Laurie Williams. 2024. A survey on software  
348 vulnerability exploitability assessment. *Comput. Surveys* 56, 8 (2024), 1–41.
- 349 [7] Charles Antony Richard Hoare. 1969. An axiomatic basis for computer programming. *Commun. ACM* 12, 10 (1969),  
350 576–580.
- 351 [8] Roger M Needham and Michael D Schroeder. 1978. Using encryption for authentication in large networks of computers.  
*Commun. ACM* 21, 12 (1978), 993–999.
- 352 [9] Peter W O’Hearn. 2019. Incorrectness logic. *Proceedings of the ACM on Programming Languages* 4, POPL (2019), 1–32.
- 353 [10] Kosta Serebryany. 2016. Continuous fuzzing with libfuzzer and addresssanitizer. In *2016 IEEE Cybersecurity Development  
(SecDev)*. IEEE, 157–157.
- 355 [11] Julien Vanegue. 2022. Adversarial logic. In *International Static Analysis Symposium*. Springer, 422–448.

356

357

358

359

360

361

362

363

364

365

366

367

368

369

370

371

372

373

374

375

376

377

378

379

380

381

382

383

384

385

386

387

388

389

390

391

392