HighGuard: Cross-Chain Business Logic Monitoring of Smart Contracts

HighGuard: Cross-Chain Business Logic Monitoring of Smart Contracts

Mojtaba Eshghie
KTH Royal Institute of Technology
Stockholm, Sweden
eshghie@kth.se

Cyrille Artho
KTH Royal Institute of Technology
Stockholm, Sweden
artho@kth.se

Hans Stammler
KTH Royal Institute of Technology
Stockholm, Sweden
stammler@kth.se

Wolfgang Ahrendt
Chalmers University of Technology
Gothenburg, Sweden
ahrendt@chalmers.se

Thomas T. Hildebrandt
University of Copenhagen
Copenhagen, Denmark
hilde@di.ku.dk

Gerardo Schneider
University of Gothenburg
Gothenburg, Sweden
gerardo.schneider@gu.se

ABSTRACT

Logical flaws in smart contracts are often exploited, leading to significant financial losses. Our tool, HighGuard, detects transactions that violate business logic specifications of smart contracts. HighGuard employs dynamic condition response (DCR) graph models as formal specifications to verify contract execution against these models. It is capable of operating in a cross-chain environment for detecting business logic flaws across different blockchain platforms. We demonstrate HighGuard's effectiveness in identifying deviations from specified behaviors in smart contracts without requiring code instrumentation or incurring additional gas costs. By using precise specifications in the monitor, HighGuard achieves detection without false positives. Our evaluation, involving 54 exploits, confirms HighGuard's effectiveness in detecting business logic vulnerabilities.

Our open-source implementation of HighGuard and a screencast of its usage are available at:
https://github.com/mojtaba-eshghie/HighGuard
https://www.youtube.com/watch?v=sZYVV-slDaY

CCS CONCEPTS

• **Software and its engineering** → **Dynamic analysis**; **Software verification**; **Model checking**; **Functionality**; **Formal software verification**; **Software testing and debugging**; • **Security and privacy** → *Formal security models.*

KEYWORDS

Smart Contracts, DCR Graphs, Runtime Monitoring, Blockchain Security

ACM Reference Format:
Mojtaba Eshghie, Cyrille Artho, Hans Stammler, Wolfgang Ahrendt, Thomas T. Hildebrandt, and Gerardo Schneider. 2024. HighGuard: Cross-Chain Business Logic Monitoring of Smart Contracts. In *39th IEEE/ACM International Conference on Automated Software Engineering (ASE '24), October 27-November 1, 2024, Sacramento, CA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3691620.3695356

1 INTRODUCTION

Smart contracts are computer programs that execute on blockchain platforms and manage digital assets. Smart contracts operate autonomously according to a predefined set of rules implemented in high-level programming languages such as Solidity [4]. They embody complex business processes, which in lack of business process-oriented development languages may lead to implementations that deviate from the intended business logic of the contract. Such flaws enable attackers to exploit the contracts [12].

Popular programming lanaguages for smart contracts, such as Solidity, lack explicit support for process-oriented concepts such as roles, action dependencies, and time. This makes it difficult to design and analyze business logic directly in the smart contract. To address this problem, we use dynamic condition response (DCR) graphs to model smart contracts and their corresponding business processes [20]. DCR graphs are a well-established declarative business process notation that extended with data and time provide a clear and concise model of the smart contract [26].

We leverage DCR graphs to express the intended design of a smart contract throughout its development cycle (see Fig. 1). The formal contract model helps convey the protocol designers' intentions to developers (stage one in Fig. 1) and supports later development and maintenance stages (stages two and three in Fig. 1).

Business logic flaws in smart contracts account for a significant portion of the total losses in recent smart contract attacks [12]. Previous research has largely overlooked business-logic vulnerabilities, focusing instead on well-known issues like reentrancy and integer overflow [16, 22]. This is because the business logic of a smart contract is application-specific. Identifying business logic flaws requires understanding the contract's intended behavior. These flaws are not easily recognizable as they deviate from the expected program behavior and do not follow traditional patterns, complicating their detection by (esp. static) analysis tools.

To address the gap in detection of business logic exploits, we introduce HighGuard, a runtime monitoring tool for

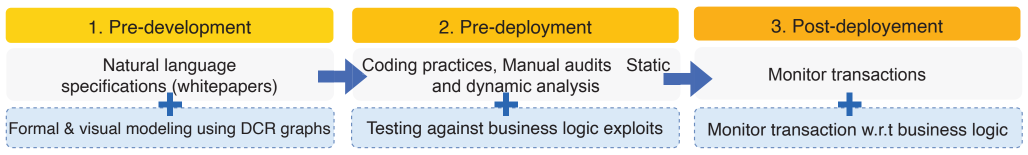

1. Pre-development
2. Pre-deployment
3. Post-deployement
Natural language specifications (whitepapers)
Coding practices, Manual audits Static and dynamic analysis
Monitor transactions
Formal & visual modeling using DCR graphs
Testing against business logic exploits
Monitor transaction w.r.t business logic

Figure 1: Software maintenance ecosystem of smart contracts (bottom blue row: HighGuard's approach)

Table 1: SotA Smart contract monitoring tools

Tool	Monitor Placement	Evaluation Dataset Size	Target Vulnerabilities
ContractLarva [25]	On-chain	1 contract [18]	–
Solythesis [27]	On-chain	23 contr. [28]	–
ContraMaster [3]	Instr. EVM	218 contr. [33]	Reentrancy Exception Disorder Integer Over/underflow
Dynamit [19]	Off-chain	105 tx [21]	Reentrancy
SCMon [17]	Instr. EVM	1 contract [17]	–
Xscope [35]	Off-chain	4 cross-chain bridges [34]	Unrestricted Deposit Inconsistent Event Parsing Unauthorized Unlocking
Annotation [30]	On-chain	50 contr. [30]	Reentrancy, Type cast, Tx order non-determinism Exception disorder
Scribble [6]	On-chain	–	–
Tx Monitors [15]	Instr. EVM	–	–
Forta [1]	Off-chain	–	–
HAL Streams [9]	Off-chain	–	–
OpenZeppelin [10]	Off-chain	–	–
HighGuard	Off-chain	54 exploits	Business logic flaws

smart contracts. It leverages DCR graph specifications as oracles to differentiate between intended behavior and interactions that violate a contract's intended business logic. DCR graphs have been established as a suitable formalism to capture the security properties in smart contracts [20].

To mitigate the performance overhead of runtime monitoring (additional *gas* usage), we execute the monitor *off-chain*. Nevertheless, HighGuard is an *online* monitor, as it observes the transactions as they are appended to the blockchain in near real-time.

Thanks to its architecture, HighGuard can support multiple chain environments and even monitor the business logic of cross-chain transactions, making it (to our knowledge) the first tool that is capable of this.

2 RELATED WORK

Smart contract analysis tools include static and dynamic methods. Table 1 summarizes state-of-the-art monitoring tools.

Static Analysis. Tools like Slither [23], SmartCheck [31], and Securify [32], identify patterns in code but often produce false positives due to syntax-level checks [30, 33].

Dynamic Analysis. Dynamic analysis tools vary in their monitoring approach and target vulnerabilities. ContractLarva adds runtime checks based on automaton-based specifications [18, 25], while Solythesis enforces invariants through a source-to-source compiler [27, 28]. Shyamasundar's framework allows in-code constraints enforced through

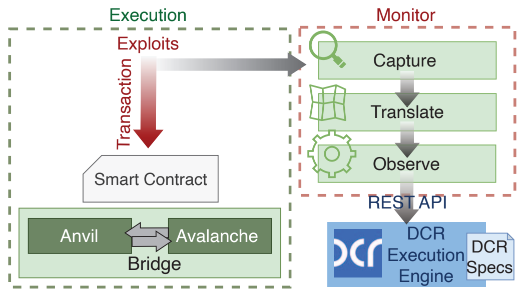


Figure 2: HighGuard system architecture

safeguards [30]. Scribble generates runtime assertions from annotations for pre-deployment testing [2]. Capretto et al. propose transaction monitors validating transaction conditions, requiring new blockchain instructions [15].

Pre-deployment tools like ContraMaster use grey-box fuzzing to test attack transactions [3, 33]. SCMon logs and visualizes function-level execution metrics [17].

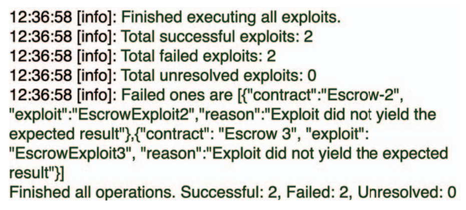
Post-deployment tools monitor deployed contracts for malicious traces. Dynamit uses machine learning to classify transactions, focusing on reentrancy attacks [21]. Forta employs decentralized detection bots [1, 11]. HAL Streams filters blockchain data for specific events [9]. OpenZeppelin Monitors provide alerts for specific events [7, 10].

Chain Interoperability. Centralized bridges enable asset transfer between blockchains [13, 14, 29]. Xscope identifies vulnerabilities by pre-executing transaction sequences as a relayer in cross-chain bridges [35]. Ganguly et al. propose distributed runtime verification across blockchains [24]. Unlike these tools, HighGuard supports both pre- and post-deployment testing and monitoring through its multi-chain execution ecosystem (stages two and three in Fig. 1). Rather than using predefined anomalous transaction sequences, HighGuard relies on the reference DCR model of the contract as the monitoring oracle (Table 1, last row), enabling developers to apply various contract-specific high-level properties to the monitor.

3 HIGHGUARD ARCHITECTURE

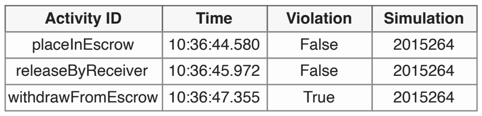
HighGuard requires two inputs from the user: (1) a DCR model of the contract, and (2) the mapping of contract functions, events, and transaction(s) to DCR model activities.

Fig. 2 shows HighGuard's architecture. The system monitors incoming transactions to smart contracts and translates them to DCR graph activities based on transaction information. The monitor then sends requests based on the translated events to the DCR execution engine. The trace of



```
12:36:58 [info]: Finished executing all exploits.
12:36:58 [info]: Total successful exploits: 2
12:36:58 [info]: Total failed exploits: 2
12:36:58 [info]: Total unresolved exploits: 0
12:36:58 [info]: Failed ones are [{"contract":"Escrow-2",
"exploit":"EscrowExploit2","reason":"Exploit did not yield the
expected result"},{"contract": "Escrow 3", "exploit":
"EscrowExploit3", "reason":"Exploit did not yield the expected
result"}]
Finished all operations. Successful: 2, Failed: 2, Unresolved: 0
```

Figure 3: CLI Termination output for Escrow contract.



Activity ID	Time	Violation	Simulation
placeInEscrow	10:36:44.580	False	2015264
releaseByReceiver	10:36:45.972	False	2015264
withdrawFromEscrow	10:36:47.355	True	2015264

Figure 4: Report generated for Escrow contract

executed activities is generated as a report while the monitor is running. If the event is part of a violating trace in the DCR model, it will return an error code that is logged in the monitor for further investigation.

HighGuard offers a command line interface to deploy, run, and report. Fig. 3 shows an example report produced by executing four exploits against four vulnerable Escrow contracts. Fig. 4 shows details of a particular exploit.

HighGuard is intended for two types of usage: (1) In a pre-deployment testing setup (stage two in Fig. 1) through the execution ecosystem of HighGuard (left side of Fig. 2); (2) Plugged into blockchain environments such as Ethereum to monitor contracts post-deployment (stage three in Fig. 1).

HighGuard supports multiple blockchain simulators as *environments* that abstract away details of each simulation platform and expose an API to the monitor for deploying contracts, executing exploits, and monitoring for violations from exploits. Two such blockchain simulators, *Anvil* and *Avalanche*, are implemented as environments [5, 8].

HighGuard is capable of monitoring cross-chain transactions as the business logic specified in the DCR model of the contract is platform-independent [20]. To execute cross-chain transactions, we implement a bridge that supports cross-chain transactions between the two aforementioned environments (left side of Fig. 2).[1]

4 DCR GRAPH MODELING

DCR graphs contain activities (boxes in Fig. 5) and relations between activities (arrows in Fig. 5). Smart contracts implement functionality as functions that affect state variables.

To model the semantics of contracts in DCR graphs, we represent publicly callable functions of a contract as activities in the DCR graph. The condition checks (*require* statements in Solidity) that preserve invariants in the contract are mapped to the relations in DCR graphs. A function's requirement in the form of *require(predicate)* in Solidity is translated to a guarded inclusion (→+) or exclusion (→%) relation [26] in the DCR graph with *predicate* written as a guard on top of the relation in the model.

[1]https://github.com/mojtaba-eshghie/HighGuard/tree/main/CI/envs/bridge-decentralized

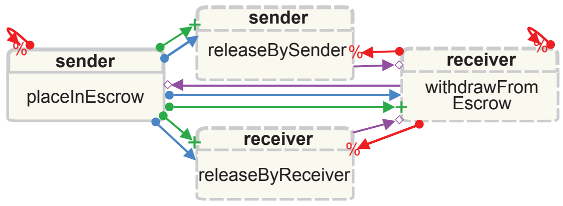


Figure 5: DCR model of the Escrow contract.

A DCR graph activity is enabled if it is included (drawn with solid border, e. g., *placeInEscrow* in Fig. 5). Time constraints are represented as DCR response (●⟶) relations with deadlines; inter-action dependencies can be modeled as milestones (→◇) that govern when actions are enabled [26].

Finally, Solidity roles are directly mapped to roles in DCR models (*sender* and *receiver* in Fig. 5), ensuring that each event is executed only by the permitted actors.

5 EVALUATING HIGHGUARD

The DCR models can be designed and simulated in the dcrgraphs.net online tool and then executed via a REST API.[2] During runtime, a sequence of one or more transactions in the smart contract are translated to one activity execution in the DCR model depending on the mapping given to HighGuard for each contract.

Single-Chain Evaluation. We modeled five contracts using DCR graphs, including the example in Fig. 5. Variants of each contract were deployed with business logic vulnerabilities injected into their Solidity source code, such as altered equations in functions and removed *require* statements. One author reviewed these vulnerabilities to exclude traditional types like reentrancy, focusing instead on deviations from the contract's business specifications.[3] We evaluated HighGuard's ability to detect malicious transactions by running exploits [4] targeting these vulnerabilities using HighGuard's ecosystem (Fig. 2). In total, 52 pairs of vulnerable contract variants and their exploits were tested. Table 2 shows the detection results: HighGuard flagged all exploits with no false positives or false negatives. This experiment was conducted in the *Anvil* environment (Fig. 2).

Cross-Chain Evaluation. We modeled a cross-chain decentralized exchange (DEX) with four contracts: a vault, a token, a price oracle, and a router contract on each of Ethereum and Avalanche blockchains. As mentioned earlier, the execution ecosystem uses our own centralized bridge for pre-deployment testing purposes (Fig. 2). Our cross-chain DEX contract can exchange tokens between two blockchains using the exchange rate updated by the price oracle contract. We instrumented the implementations of the *vault* and *router* contracts to inject two vulnerabilities related to cross-chain transaction expiration times and double-payouts. We ran manually-written exploits[5] targeting the mentioned vulnerabilities which resulted in loss of tokens in victim

[2]The online tool and API can be used freely for academic use.
[3]https://github.com/mojtaba-eshghie/HighGuard/tree/main/contracts/src/synthesized
[4]https://github.com/mojtaba-eshghie/HighGuard/tree/main/CI/exploits/synthesized
[5]https://github.com/mojtaba-eshghie/HighGuard/tree/main/CI/tests

Table 2: Single-chain evaluation results

Contract	Exploits	FP	FN
Governance	15	0	0
Escrow	2	0	0
MultiStageAuction	13	0	0
PrizeDistribution	7	0	0
ProductOrder	15	0	0
Total	52	0	0

contracts. These two exploits were successfully detected by HighGuard with no false positives or false negatives.

Resource Usage. Since the off-chain monitor placement does not affect contracts under observation, there is no on-chain performance overhead. The monitor application, written in NodeJS, runs on a server with memory usage under 1 GB. In a pre-deployment testing setup, HighGuard also provides the smart contract execution platform, with resource usage depending on the testing environment. In the *Anvil* environment, it uses less than 10 MB of RAM and 1% CPU on a MacBook with an Intel i7 2.30 GHz processor.

6 CONCLUSION

We present HighGuard, a tool for detecting business logic flaws in smart contracts. HighGuard takes the DCR model of a contract's business logic as the reference to check the transactions against it, pre- or post-deployment of the contract. It operates off-chain and does not require any code instrumentation. We successfully demonstrated HighGuard's capability of detecting malicious transactions in single-chain and cross-chain setups by evaluating it against 54 smart contract exploits.

REFERENCES

[1] 2022. Forta Litepaper. https://docs.forta.network/en/latest/2022-7-11%20Forta%20Litepaper.pdf

[2] 2023. Introduction | Scribble. https://docs.scribble.codes

[3] 2023. ntu-SRSLab/vultron. https://github.com/ntu-SRSLab/vultron original-date: 2018-11-30T05:36:03Z.

[4] 2023. Solidity documentation. https://docs.soliditylang.org/en/latest/

[5] 2024. Avalanche-CLI | Avalanche Dev Docs. https://docs.avax.network/tooling/avalanche-cli

[6] 2024. Consensys/scribble. https://github.com/Consensys/scribble original-date: 2020-12-04T17:43:07Z.

[7] 2024. Defender - OpenZeppelin Docs. https://docs.openzeppelin.com/defender/v2/

[8] 2024. Foundry Book. https://book.getfoundry.sh/reference/anvil/

[9] 2024. HAL Streams Overview. https://docs.hal.xyz/docs/overview

[10] 2024. Monitor - OpenZeppelin Docs. https://docs.openzeppelin.com/defender/v2/module/monitor

[11] 2024. Network Overview - Forta Docs. https://docs.forta.network/en/latest/network-overview/

[12] 2024. SunWeb3Sec/DeFiHackLabs: Reproduce DeFi Hacked Incidents Using Foundry. https://github.com/SunWeb3Sec/DeFiHackLabs

[13] André Augusto, Rafael Belchior, Miguel Correia, André Vasconcelos, Luyao Zhang, and Thomas Hardjono. 2024. SoK: Security and Privacy of Blockchain Interoperability [Extended Version]. https://doi.org/10.36227/techrxiv.24595764.v2

[14] Vitalik Buterin. 2016. Chain interoperability. *R3 research paper* 9 (2016), 1–25. https://allquantor.at/blockchainbib/pdf/buterin2016chain.pdf

[15] Margarita Capretto, Martin Ceresa, and César Sánchez. 2022. Transaction Monitoring of Smart Contracts. In *Runtime Verification (Lecture Notes in Computer Science)*, Thao Dang and Volker Stolz (Eds.). Springer International Publishing, Cham, 162–180. https://doi.org/10.1007/978-3-031-17196-3_9

[16] Stefanos Chaliasos, Marcos Antonios Charalambous, Liyi Zhou, Rafaila Galanopoulou, Arthur Gervais, Dimitris Mitropoulos, and Ben Livshits. 2023. Smart Contract and DeFi Security: Insights from Tool Evaluations and Practitioner Surveys. https://doi.org/10.48550/arXiv.2304.02981 arXiv:2304.02981 [cs]

[17] Yi Ding, Chenshuo Wang, Qionghui Zhong, Haisheng Li, Jinjing Tan, and Jie Li. 2020. Function-Level Dynamic Monitoring and Analysis System for Smart Contract. *IEEE Access* 8 (2020), 229161–229172. https://doi.org/10.1109/ACCESS.2020.3046005 Conference Name: IEEE Access.

[18] Joshua Ellul and Gordon J. Pace. 2018. Runtime Verification of Ethereum Smart Contracts. In *2018 14th European Dependable Computing Conference (EDCC)*. 158–163. https://doi.org/10.1109/EDCC.2018.00036

[19] Mojtaba Eshghie. 2024. Mojtaba-Eshghie/Dynamit. https://github.com/mojtaba-eshghie/Dynamit

[20] Mojtaba Eshghie, Wolfgang Ahrendt, Cyrille Artho, Thomas Troels Hildebrandt, and Gerardo Schneider. 2023. Capturing Smart Contract Design with DCR Graphs. https://doi.org/10.48550/arXiv.2305.04581 arXiv:2305.04581 [cs].

[21] Mojtaba Eshghie, Cyrille Artho, and Dilian Gurov. 2021. Dynamic Vulnerability Detection on Smart Contracts Using Machine Learning. In *EASE 2021*. ACM, 305–312.

[22] Josselin Feist, Gustavo Grieco, and Alex Groce. 2019. Slither: a static analysis framework for smart contracts. In *2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*. IEEE, 8–15.

[23] Josselin Feist, Gustavo Grieco, and Alex Groce. 2023. Slither Analyzer. https://github.com/crytic/slither original-date: 2018-09-05T21:56:35Z.

[24] Ritam Ganguly, Yingjie Xue, Aaron Jonckheere, Parker Ljung, Benjamin Schornstein, Borzoo Bonakdarpour, and Maurice Herlihy. 2022. Distributed Runtime Verification of Metric Temporal Properties for Cross-Chain Protocols. In *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*. 23–33. https://doi.org/10.1109/ICDCS54860.2022.00012 ISSN: 2575-8411.

[25] gordonpace. 2024. gordonpace/contractLarva. https://github.com/gordonpace/contractLarva original-date: 2017-12-14T19:27:41Z.

[26] Thomas T. Hildebrandt, Håkon Normann, Morten Marquard, Søren Debois, and Tijs Slaats. 2022. Decision Modelling in Timed Dynamic Condition Response Graphs with Data. In *Business Process Management Workshops*. Springer, Cham, 362–374.

[27] Ao Li. 2024. aoli-al/Solythesis. https://github.com/aoli-al/Solythesis original-date: 2019-04-05T01:29:12Z.

[28] Ao Li, Jemin Andrew Choi, and Fan Long. 2020. Securing smart contract with runtime validation. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2020)*. Association for Computing Machinery, New York, NY, USA, 438–453. https://doi.org/10.1145/3385412.3385982

[29] Wei Ou, Shiying Huang, Jingjing Zheng, Qionglu Zhang, Guang Zeng, and Wenbao Han. 2022. An overview on cross-chain: Mechanism, platforms, challenges and advances. *Computer Networks* 218 (2022), 109378. https://doi.org/10.1016/j.comnet.2022.109378

[30] R. K. Shyamasundar. 2022. A Framework of Runtime Monitoring for Correct Execution of Smart Contracts. In *Blockchain – ICBC 2022 (Lecture Notes in Computer Science)*, Shiping Chen, Rudrapatna K. Shyamasundar, and Liang-Jie Zhang (Eds.). Springer Nature Switzerland, Cham, 92–116. https://doi.org/10.1007/978-3-031-23495-8_7

[31] Sergei Tikhomirov, Ekaterina Voskresenskaya, Ivan Ivanitskiy, Ramil Takhaviev, Evgeny Marchenko, and Yaroslav Alexandrov. 2018. SmartCheck: Static Analysis of Ethereum Smart Contracts. https://doi.org/10.1145/3194113.3194115

[32] Petar Tsankov, Andrei Dan, Dana Drachsler-Cohen, Arthur Gervais, Florian Bünzli, and Martin Vechev. 2018. Securify: Practical Security Analysis of Smart Contracts. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18)*. Association for Computing Machinery, New York, NY, USA, 67–82. https://doi.org/10.1145/3243734.3243780

[33] Haijun Wang, Ye Liu, Yi Li, Shang-Wei Lin, Cyrille Artho, Lei Ma, and Yang Liu. 2022. Oracle-Supported Dynamic Exploit Generation for Smart Contracts. *IEEE Transactions on Dependable and Secure Computing* 19, 3 (2022), 1795–1809. https://doi.org/10.1109/TDSC.2020.3037332 Conference Name: IEEE Transactions on Dependable and Secure Computing.

[34] Xscope-Tool. 2024. Xscope-Tool/Results. https://github.com/Xscope-Tool/Results original-date: 2022-05-23T02:19:43Z.

[35] Jiashuo Zhang, Jianbo Gao, Yue Li, Ziming Chen, Zhi Guan, and Zhong Chen. 2023. Xscope: Hunting for Cross-Chain Bridge Attacks. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22)*. Association for Computing Machinery, New York, NY, USA, 1–4. https://doi.org/10.1145/3551349.3559520