On the Just-In-Time Discovery of Profit-Generating Transactions in DeFi Protocols

Liyi Zhou, Kaihua Qin, Antoine Cully, Benjamin Livshits and Arthur Gervais
Imperial College London, United Kingdom

2021 IEEE Symposium on Security and Privacy (SP) | 978-1-7281-8934-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/SP40001.2021.00113

***Abstract*—Decentralized Finance (DeFi) is a blockchain-asset-enabled finance ecosystem with millions of daily USD transaction volume, billions of locked up USD, as well as a plethora of newly emerging protocols (for lending, staking, and exchanges). Because all transactions, user balances, and total value locked in DeFi are publicly readable, a natural question that arises is: how can we automatically craft profitable transactions across the intertwined DeFi platforms?**

In this paper, we investigate two methods that allow us to automatically create profitable DeFi trades, one well-suited to arbitrage and the other applicable to more complicated settings. We first adopt the Bellman-Ford-Moore algorithm with DeFiPoser-ARB and then create logical DeFi protocol models for a theorem prover in DeFiPoser-SMT. While DeFiPoser-ARB focuses on DeFi transactions that form a cycle and performs very well for arbitrage, DeFiPoser-SMT can detect more complicated profitable transactions. We estimate that DeFiPoser-ARB and DeFiPoser-SMT can generate an average weekly revenue of 191.48 ETH (76,592 USD) and 72.44 ETH (28,976 USD) respectively, with the highest transaction revenue being 81.31 ETH (32,524 USD) and 22.40 ETH (8,960 USD) respectively. We further show that DeFiPoser-SMT finds the known economic bZx attack from February 2020, which yields 0.48M USD. Our forensic investigations show that this opportunity existed for 69 days and could have yielded more revenue if exploited one day earlier. Our evaluation spans 150 days, given 96 DeFi protocol actions, and 25 assets.

Looking beyond the financial gains mentioned above, *forks* deteriorate the blockchain consensus security, as they increase the risks of double-spending and selfish mining. We explore the implications of DeFiPoser-ARB and DeFiPoser-SMT on blockchain consensus. Specifically, we show that the trades identified by our tools exceed the Ethereum block reward by up to 874×. Given optimal adversarial strategies provided by a Markov Decision Process (MDP), we quantify the value threshold at which a profitable transaction qualifies as Miner Extractable Value (MEV) and would incentivize MEV-aware miners to fork the blockchain. For instance, we find that on Ethereum, a miner with a hash rate of 10% would fork the blockchain if an MEV opportunity exceeds 4× the block reward.

I. Introduction

Blockchain-based decentralized finance protocols (commonly referred to as DeFi) have attracted a recent surge in popularity and value stored exceeding 13 billion USD. The currently most popular DeFi platforms are based on the Ethereum blockchain and its system of smart contracts, which regularly gives nascence to new applications, mirrored and inspired by the traditional centralized finance system. Examples are asset exchanges [24], [58], margin trading [3], [24], lending/borrowing platforms [27], [30], and derivatives [27]. DeFi, moreover, can surprise with novel use-cases such as constant product market maker exchanges [26], [58] and flash loans — instant loans where the lender bears no risk that the borrower does not repay the loan [4], [24], [53].

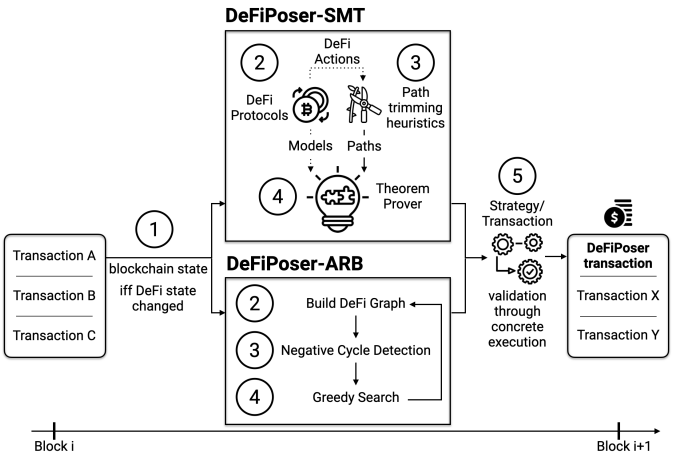


Fig. 1: DeFiPoser-ARB and DeFiPoser-SMT system overview. In DeFiPoser-SMT, we ② Create logical models, ③ paths are created and trimmed with heuristics and ④ used within a theorem prover to generate a transaction. In DeFiPoser-ARB we ② build a graph of the blockchain state, ③ identify negative cycles, ④ perform a local search and repeat. The transaction with the highest revenue is ⑤ concretely evaluated before being mined in the next block.

A peculiarity of DeFi platforms is their ability to interoperate; e.g., one may borrow a cryptocurrency asset on one platform, exchange the asset on another, and for instance, lend the resulting asset on a third system. DeFi's *composability* has led to the emergence of chained trading and arbitrage opportunities throughout the tightly intertwined DeFi space. Reasoning about what this easy composition entails is not particularly simple; on one side, atomic composition allows to perform *risk-free arbitrage* — that is to equate asset prices on different DeFi markets. Arbitrage is a benign and important endeavor to keep markets synchronized.

On the other side, we have seen multi-million-revenue trades that cleverly use the technique of flash loans to exploit *economic states* in DeFi protocols (e.g., the economic attack on bZx [3], [53] Harvest Finance [28], Value Defi [23] and others [5], [55]). While exploiting economic states, however, is not a security attack in the traditional sense, the practitioners' community often frames these high-revenue trades as "hacks."

DOI 10.1109/SP40001.2021.00113

Yet, the executing trader follows the rules set forth by the deployed smart contracts. Irrespective of the framing, liquidity providers engaging with DeFi experience millions of USD in unexpected losses. This highlights the need for automated tools that help protocol designers and liquidity providers to understand arbitrage and financial implications in general when engaging with DeFi protocols.

DEFIPOSER-ARB and DEFIPOSER-SMT: This paper presents two tools (cf. Figure 1) that automatically create transactions to compose existing DeFi protocols to generate revenue that can be extracted from the Ethereum ecosystem. They are designed to run in real-time: at every block, they can find (and execute) a new profit-generating transaction; we show how our running time of an unoptimized implementation requires an average of 6.43 seconds and 5.39 seconds on a recent Ethereum block (for DEFIPOSER-ARB and DEFIPOSER-SMT respectively), which is below Ethereum's average block time of 13.5 seconds [9]. We would like to point out that DEFIPOSER-ARB and DEFIPOSER-SMT, are best-effort tools: because the state of the blockchain and DeFi platforms may change at each block, it is important to operate in real-time, otherwise found trading opportunities might be outdated. Therefore, we made the choice of prioritizing execution speed over completeness, and we do not claim to find optimal strategies.

To the best of our knowledge, we are the first to provide automated transaction search mechanisms for composable DeFi protocols. The main risks for a trader using the tools that we consider within this work are currency exposure (i.e., price volatility risks) and the blockchain transaction fees. We discover that significant revenue can be generated with less than 1 ETH of initial capital when using flash loans.

Our contributions are as follows:

- **DEFIPOSER-ARB:** We build a directed DeFi market graph and identify negative cycles with the Bellman-Ford-Moore algorithm. A local search then allows us to discover parameters for profitable arbitrage transactions in near-real-time (average of 6.43 seconds per block).
- **DEFIPOSER-SMT and Space Reduction:** To discover more demanding trades than arbitrage, we model the DeFi systems using a state transition model, which we translate to a logical representation in the Z3 theorem prover. We introduce heuristics to significantly prune the search space to achieve a near real-time transaction discovery (average of 5.39 seconds per block).
- **Miner Extractable Value (MEV) and Security:** We show how DEFIPOSER-SMT discovers the economic attack on bZx, which yields over 0.48M USD, and that this opportunity window was open for over 69 days. Given optimal adversarial mining strategies provided by a Markov Decision Process, we show quantitatively that MEV opportunities can deteriorate the blockchain security. For example, a rational MEV-aware miner with a hash rate of 10% will fork the blockchain if an MEV opportunity exceeds 4 times the block reward and the miner failed to claim the source of MEV.
- **Trading Strategy Validation:** We validate the trading strategies discovered by DEFIPOSER-ARB and DEFIPOSER-SMT on a locally-deployed blockchain that mirrors the real network. We estimate that the found strategies yield 4,103.22 ETH (1,641,288 USD) and 1,552.32 ETH (620,928 USD) of profit between the Ethereum block $9,100,000$ to $10,050,000$ (150 days from December 2019 to May 2020). We demonstrate that our tools' capital requirements are minimal: the majority of the strategies require less than 150.00 ETH (60,000 USD), and only 0.40 ETH (160 USD) when using flash loans.

Paper organization: The remainder of the paper is organized as follows. Section II elaborates on the DeFi background, discusses stable coins and flash loans. Section III describes how we encode DeFi protocols into state transition models. Section IV applies negative cycle detection to find DeFi arbitrage opportunities. Section V presents our heuristics and techniques to enable the autonomous discovery of adversarial strategies. Section VI presents our empirical evaluation and quantitative analysis of the found strategies on previous Ethereum blockchain blocks. Section VII discusses DEFIPOSER's blockchain security implication. We discuss related works in Section VIII and conclude the paper in Section IX.

II. BACKGROUND

In this section, we outline the required background for DeFi. For extensive background on blockchains and smart contracts, we refer the interested reader to [6], [11].

A. Decentralized Finance (DeFi)

Decentralized Finance (DeFi) refers to a financial ecosystem that is built on top of (permissionless) blockchains [59]. DeFi supports a multitude of different financial applications [3], [4], [24], [24], [24], [27], [27], [30], [53], [58]. The current DeFi landscape is mostly built upon smart contract enabled blockchains (e.g., Ethereum). We briefly summarize relevant DeFi platforms.

Automated Market Maker (AMM): In traditional finance, asset exchanges are usually operated in the form of order matching. Asks and bids are matched in a centralized limit order book, typically following the FIFO principle [17]. In DeFi, such an order matching mechanism would be inefficient because the number of transactions per second supported by the underlying blockchain is usually limited. Therefore, AMM minimizes the number of transactions required to balance an on-chain asset exchange. AMM allows liquidity providers, the traders who are willing to provide liquidity to the market, to deposit assets into a liquidity pool. Liquidity takers then directly trade against the AMM liquidity pool according to a predefined pricing mechanism. The constant product AMM is currently the most common model (adopted by over 66% of the AMM DEX), where the core idea is to keep the product of the asset amounts in the liquidity pool constant. Consider a constant product AMM that trades the asset pair X/Y. x and

y are the amount of X and Y respectively in the liquidity pool. A liquidity taker attempts to sell Δx of X and get Δy of Y in exchange. The constant product rule stipulates that $x \times y = (x + \Delta x) \times (y - \Delta y)$. Uniswap [58] is the most dominating constant product AMM with a market capitalization of 1.4B USD [52]. Variant AMMs utilize different pricing formulas, e.g., Bancor [40], while other platforms (e.g., Kyber [46]) aggregate AMMs. When receiving an order from a user, these platforms redirect the order to the AMM, which provides the best asset price.

Stablecoin: Stablecoins are a class of cryptocurrencies designed to alleviate the blockchain price volatility [49]. The most salient solution for stabilization is to peg the price of stablecoins to a less-volatile currency (e.g., USD) [50]. There exist over 200 stablecoin projects announced since 2014 [2]. Among them, SAI and DAI developed by MakerDAO [30] have received extensive attention. Both SAI and DAI are collateral-backed stablecoins. SAI is collateralized solely by ETH, whereas DAI is an SAI upgrade to support multiple assets as collateral. At the time of writing, the collateral locked in MakerDAO amounts to 2.73B USD [52].

Flash Loans: The Ethereum blockchain operates similarly to a replicated state machine. Transactions trigger state transitions and provide the input data necessary for the Ethereum Virtual Machine (EVM) state to change according to rules set by smart contracts. Interestingly, the EVM state is only affected by a transaction if the transaction executes without failure. In the case of a failed transaction, the EVM state is reverted to the previous state, but the transaction fees are still paid to miners (as in to avoid Denial of Service attacks). A transaction can fail due to the following three reasons: Either the transaction sender did not specify a sufficient amount of transaction fees, or the transaction does not meet a condition set forth by the interacting smart contract, or the transaction is conflicting (e.g., double-spending) with another transaction.

This concept of a state reversion enables the introduction of flash loans, short-lived loans that execute atomically within only one blockchain transaction. Within a single transaction, (i) the loan is taken from a liquidity pool, (ii) the loan is put to use, and (iii) the loan (plus interest payment) is paid back to the flash loan pool. If the third condition is not met, i.e., the loan plus interests are not paid back, then the entire flash loan transaction fails. This is equivalent to the case that the loan was never issued because the EVM state is not modified out of the result of a failed transaction.

Flash loans, therefore, entail two interesting properties. First, the lender is guaranteed that the borrower will repay the loan. If the repayment is not performed, the loan would not be given. Second, the borrower can technically request any amount of capital, up to the amount of funds available in a flash loan pool, given a constant payment which corresponds to the blockchain transaction fees (about 10 USD for the most common flash loan providers). The borrower hence can have access to millions of USD with just a few initial USD and hence is not exposed to the currency risk of the lent asset.

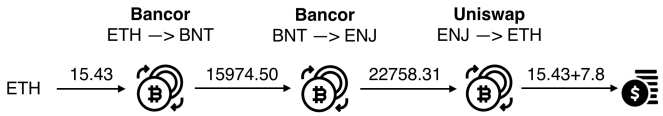


Fig. 2: Example strategy across three DeFi markets, identified at Ethereum block $10,001,087$, which would yield a revenue of 7.81 ETH (3,124 USD).

III. DeFi Modeling

We proceed to introduce our system, trader, and state transition model for the interaction between DeFi platforms. On a high level, our model state consists of the DeFi market states, as well as the cryptocurrency asset balances of a trader $\mathbb{T}$. The transitions represent DeFi actions performed by the trader $\mathbb{T}$ on the respective DeFi platforms. The goal of the trader is to maximize the amount of cryptocurrency assets held.

A. System Model

Our system consists of a blockchain with financial cryptocurrency assets (i.e., coins or tokens). Cryptocurrency assets can be used within DeFi platforms (i.e., markets), such as exchanges, lending, and borrowing platforms. Each DeFi platform offers a set of *actions*, which can be triggered by a transaction. Actions take an asset as input and yield, for instance, another asset as output. Multiple actions can be encapsulated in one transaction and executed atomically in sequence. A *path*, is a sequence of actions across DeFi platforms. We denote as *strategy* (cf. Figure 2), or transaction, a path with parameters for each action (such as coin amounts, etc.). We consider a state of a DeFi market to change whenever an action manipulates the amount of assets within this DeFi market. Note that we only consider the blockchain state at block-height i after the execution of all transactions within a block i (i.e., we do not consider intermittent block states).

B. Trader Model

We consider a computationally bounded trader (denoted by $\mathbb{T}$) which is capable of executing transactions (i.e., perform actions) across a set of DeFi platforms. $\mathbb{T}$'s cryptocurrency assets are limited by the supply of liquidity available in public flash loan pools [53]. The trader is capable of reading the blockchain contents but is not expected to observe unconfirmed blockchain transactions on the network layer. We assume that the trader is capable of placing a transaction ahead of other DeFi transactions within a future blockchain block. Practically, this requires the trader to pay a higher transaction fee, as most miners appear to order transactions based on gas price. We assume that the trader is not colluding with a miner, while this may present an interesting avenue for future work.

We assume that the trader is operating on the blockchain head, i.e., the most recently mined, valid block, of the respective blockchain. In the case of a Proof-of-Work (PoW) blockchain, the most recent block shall also be the one with the most PoW (i.e., the greatest difficulty). For simplicity, we ignore complications resulting from blockchain forks.

C. Notation

To ease the understanding of the following paragraphs, we proceed by introducing the utilized notation.

Assets: The set C denotes the collection of cryptocurrency assets, which the trader uses to generate trading strategies.

Actions: The set A denotes the collection of actions the trader selects from the DeFi protocols.

Parameters: The trader $\mathbb{T}$ must supply parameters to execute actions $a \in A$, e.g., the amounts of cryptocurrency assets $\mathbb{T}$ sends to the corresponding DeFi platforms.

Path: A path $p \in P$ is a sequence of n non-repeated actions drawn from A. We denote the power set of all actions with $\wp(A)$, which consists of all subsets of the action set A, including the empty set. Given a subset $K \in \wp(A)$, we denote the permutations set of K with $\mathfrak{S}(K)$. The collection of all paths P can then be defined using Equation 1. Note P consists of paths of different lengths.

$$P = \cup_{K}^{\wp(A)} \mathfrak{S}(K), \quad \text{s.t.} \quad \begin{aligned} &\forall p = (a_1, a_2, ..., a_n) \in P \\ &a_i \in A, \forall i \in [1, n] \\ &a_i \neq a_j, \forall a_i, a_j \in A, i \neq j \end{aligned} \tag{1}$$

Strategy: A strategy consists a path $p \in P$ with n actions, a list of parameters $[x_1, \ldots, x_n]$ for each action in p, and an initial state (cf. Equation 4) of the model.

Balance function: Given a strategy with n actions, the balance function $\mathcal{B}_i^{\mathbb{T}}(c)$ denotes $\mathbb{T}$'s balance for cryptocurrency asset c after performing the i^{th} action, where $0 \leq i \leq n$ and $c \in C$.

Storage function: $\mathcal{K}(a)$ denotes the set of smart contract storage variable addresses an action a reads from and writes to. These addresses are identified from the underlying blockchain runtime environment. We use $\mathcal{K}^{\mathbb{T}}(a)$ to denote a subset of $\mathcal{K}(a)$, which is only relevant to the trader $\mathbb{T}$.

D. States

We classify the state variables into two categories, the trader and DeFi states. $S^{\mathbb{T}}$ represents the trader's asset portfolio (cf. Equation 2). S^{DeFi} is the set of all storage variables $\mathbb{T}$ reads from and writes to, for all the DeFi actions in our model (cf. Equation 3). The union S of these two categories is the overall state of our system (cf. Equation 4). Given a strategy with n actions, the state after performing the i^{th} action, where $0 \leq i \leq n$ is denoted as s_i, with the initial state s_0.

$$S^{\mathbb{T}} = \{\mathcal{B}^{\mathbb{T}}(c) : \forall c \in C\} \tag{2}$$

$$S^{\text{DeFi}} = \cup_{\forall a \in A} \mathcal{K}^{\mathbb{T}}(a) \tag{3}$$

$$S = S^{\mathbb{T}} \cup S^{\text{DeFi}} \tag{4}$$

E. Transitions

Our state transition function is $\mathcal{F}^{\mathbb{T}}(s \in S, a \in A, x) \rightarrow S$, outputs the next state if action a with parameter x is performed on state s by trader $\mathbb{T}$. Given a strategy with n actions, where a_i and x_i represents the i^{th} action and parameter respectively, and s_i represents the state after the i^{th} action. Equation 5 shows the state transition process of this strategy, while Equation 6 computes the final state s_n when each action is sequentially applied to s_0.

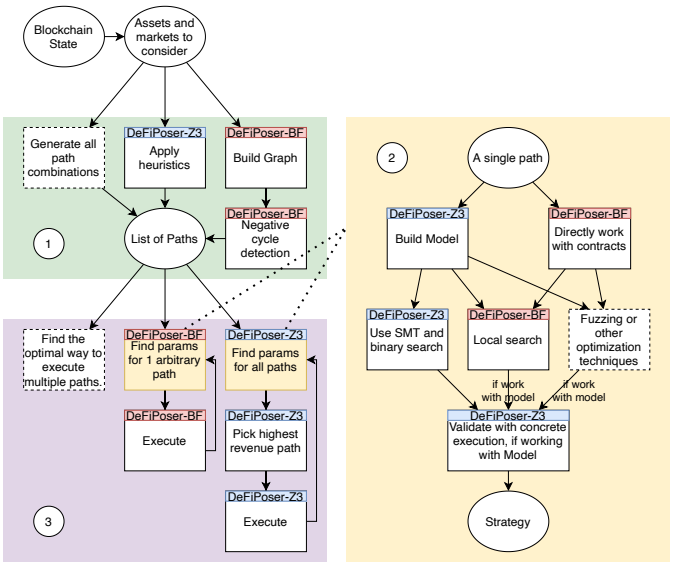


Fig. 3: Technical design choices of DEFIPOSER. DEFIPOSER consists of three components: ① a path pruning component; ② a parameter search component, and ③ a strategy combination/execution component.

$$s_{i+1} = \mathcal{F}^{\mathbb{T}}(s_i, a_{i+1}, x_{i+1}) \tag{5}$$

$$s_n = \mathcal{F}^{\mathbb{T}}(\ldots \mathcal{F}^{\mathbb{T}}(\mathcal{F}^{\mathbb{T}}(s_0, a_1, x_1), a_2, x_2) \ldots) \tag{6}$$

F. Objective

We choose an asset $b \in C$ as our *base* cryptocurrency asset. The objective of the trader $\mathbb{T}$ is to find a strategy, such that the balance of b (cf. Equation 7) is maximized, whereas the portfolio balances of the trader, except for b, remain the same.

$$\begin{aligned} &\text{maximise}_{p \in P} \ \text{obj}(s_0, p) = \mathcal{B}_n^{\mathbb{T}}(b) - \mathcal{B}_0^{\mathbb{T}}(b) \\ &\text{with constraints: } \mathcal{B}_n^{\mathbb{T}}(c) = \mathcal{B}_i^{\mathbb{T}}(c), \forall c \in C \setminus b \end{aligned} \tag{7}$$

G. Base *cryptocurrency asset*

To identify revenue yielding paths, we make the assumption that the trader $\mathbb{T}$ operates in this work on a single *base* cryptocurrency asset. Naturally, this can be extended to multiple *base* currencies to increase potential financial results.

H. DEFIPOSER *Design Choices*

Figure 3 shows the high-level design choices of the DEFIPOSER tools we present in this paper. DEFIPOSER consists of three components: ①, a pruning algorithm to filter potentially profitable paths; ②, a search algorithm which searches parameters to maximize the revenue of a given path, and ③ a strategy combination/execution algorithm, which decides how the found strategies are executed.

Generally speaking, each instantiation of the different components bears its own advantages and disadvantages. For instance, negative cycle detection only searches cyclic paths, whereas pruning with heuristics can search for any path structure. Given a simple path such as a cyclic arbitrage, we find that local search is faster than the SMT solver (cf. Figure 12), but does not provide satisfiability proofs. In the following we present two variants of DEFIPOSER, namely DEFIPOSER-ARB (cf. Section IV) and DEFIPOSER-SMT (cf. Section V).

IV. APPLYING NEGATIVE CYCLE DETECTION TO DEFI ARBITRAGE

Previous works propose negative cycle detection algorithms, such as the Bellman-Ford-Moore algorithm, to find arbitrage opportunities [18]. In these algorithms, the exchange markets are modeled as a directed weighted graph (g). Every negative cycle in the graph then corresponds to an arbitrage opportunity.

A. Negative Cycle Detection to Detect Arbitrage

We adopt the following notations to translate arbitrage detection into a negative cycle detection problem.

Nodes: The set N denotes the collection of nodes. Each node (vertex) represents a different asset ($c \in C$).

Directed edges: The set E denotes the collection of all edges. An edge $e_{i,j}$ that points from asset c_i to c_j represents that there exist a market where the trader $\mathbb{T}$ can sell cryptocurrency asset c_i to purchase cryptocurrency asset c_j.

Spot price: The spot price $p^{\text{spot}}_{i,j}$ for edge $e_{i,j}$ is the approximated best current price a trader $\mathbb{T}$ finds on all DeFi AMM markets, when selling an arbitrarily small amount (close to 0) of a cryptocurrency asset c_i to purchase c_j.

Arbitrage: A path $[c_1 \xrightarrow{a_1} c_2 \ldots c_{k-1} \xrightarrow{a_{k-1}} c_k]$ consists an arbitrage opportunity, if $p^{\text{spot}}_{1,2} \times \ldots \times p^{\text{spot}}_{k-1,k} > 1$.

Edge weight: To apply negative cycle detection algorithms, we use the negative log of price $w_{i,j} = -log(p^{\text{spot}}_{i,j})$ as the weights for edge $e_{i,j}$. An arbitrage opportunity exists if $w_{1,2} + \ldots + w_{i-1,i} < 0$.

Path finding: Our objective is to maximize $\mathbb{T}$'s *base* cryptocurrency asset. An arbitrage cycle, however, may not consist of the *base* asset. Therefore, we convert the arbitrage revenue to the *base* cryptocurrency asset by the end of the execution. More concretely, we find all 'connecting' markets that support the conversion between one of the arbitrage assets and the *base* cryptocurrency asset. We perform the conversion using the 'connecting' markets with the best price.

B. Negative Cycle Detection Algorithms

Negative cycle detection algorithms combine the shortest path algorithm with a cycle detection strategy. Cherkassky *et al.* [18] studied various combinations of shortest path algorithms (Bellman-Ford-Moore [8], [29], [51], Goldfarb-Hao-Kai [35], Goldberg-Radzik [33], etc.) and cycle detection strategies (Walk to the root, Admissible graph search [34], Subtree traversal [44], etc.) and compared their relative performances. A natural question is whether these cycle detection algorithms can be directly applied to find profitable transactions in DeFi.

Algorithm 1: Negative cycle arbitrage detection.

```
Input:
s_0 ← Initial state ; target ← Minimum revenue target
Output: revenue_total
s ← s_0 ; g ← buildGraph(N, E, s) ; revenue_total ← 0
while hasNegativeCycle(g) do
    cycle ← getNegativeCycle(graph)
    p ← getPath(cycle)
    (revenue, s) ← search(p)
    if revenue > target then
        revenue_total ← revenue_total + revenue
    end
    g ← buildGraph(N, E, s)
end
return revenue_total

Function buildGraph(N, E, s ∈ S) is
    # fetch the spot price for each e ∈ E
    # build the graph g; where w_{c_i,c_j} = −log(p^spot_{c_i,c_j})
    return g
end
Function hasNegativeCycle(g) is
    return (Detects a negative cycle?)
end
Function getPath(cycle) is
    p ∈ P connects T's baseasset with cycle.
    return p
end
Function search(p) is
    # find the parameters for path p
    s' ← state after executinng the strategy
    return (revenue, s')
end
```

In bid-ask markets, the price does not change if the trade volume is within the bid/ask size [17]. DeFi AMM exchanges, however, follow a dynamic price based on the trade volume. Intuitively, the bigger the transaction size, the worse the trading price becomes. Hence, our algorithm needs to consider dynamic price changes and update the graph g after every action. On a high level, a Bellman-Ford-Moore inspired algorithm repeatedly performs the following steps: (i) Build the graph g based on the spot prices from the current state $s \in S$; (ii) Detect arbitrage cycles in the graph g (Bellman-Ford-Moore); (iii) Build a path based on the negative cycle, and find the strategy (parameters for the path), finally (iv) Execute the strategy and update the state s. Algorithm 1 presents the details of DEFIPOSER-ARB. To find the parameters for a path, Algorithm 1 gradually increases the amount of *base* assets into the path until there is no increase in revenue.

We present DEFIPOSER-ARB's evaluation in Section VI.

V. DESIGN OF DEFIPOSER-SMT

In this section, we discuss an alternative technique, DEFIPOSER-SMT, to find profitable transactions in DeFi, which is more general when compared to DEFIPOSER-ARB.

More specifically, DEFIPOSER-SMT can operate on non-cyclic strategies, while DEFIPOSER-ARB cannot. We observe that profitable DeFi strategies do not necessarily form a complete cycle. For example, Figure 4a shows the graph for the economic bZx attack (cf. Section VII). The strategy requires the trader to send Ether to edge 1 without receiving any assets in return and to then perform an arbitrage cycle with edge 2 and 3.

A. Choosing an SMT Solver for DEFIPOSER-SMT

To overcome the aforementioned challenges of non-existent cycles, we chose to adopt a theorem prover for DEFIPOSER-SMT's (cf. Figure 1) design. The theorem prover logically formulates what a profitable strategy entails to locate concrete profitable instantiations. We perform systematic path exploration to determine if the model (cf. Section III) satisfies the provided requirements, similar to other model checking systems [13], [16], [36], [43], [45], [48], [56], [57].

Our model requires the SMT solver (such as MathSat [14], Z3 [21], or Coral [54]) to support floating-point arithmetic because we adopt the theory of real numbers (cf. Section III). We encode the state transition model in three major steps: (i) Encode the initial state as a predicate; (ii) iteratively apply state transition actions, and encode the resulting states after each action as predicates. Then, (iii) convert the objective function into a set of constraints to ensure that the value of the trader portfolio increases by Z, and translate the constraints into predicates. Note that we rely on an optimization algorithm (cf. Algorithm 3) to find the highest possible Z. The optimization process requires solving the same SMT problem with different initializations of Z (cf. Appendix C for an example).

B. Path Pruning

One bottleneck of model checking is the combinatorial path explosion problem. We, therefore, prune the paths by applying the following heuristics. Note that heuristics may prune profitable strategies, and DEFIPOSER-SMT is therefore only a best-effort tool.

Heuristic 1: A profitable strategy must consist of more than one action. That is because, given an initial state S_0, a strategy with only one action will not increase the balance of the *base* cryptocurrency asset while keeping the balance of all other cryptocurrency assets unchanged.

Heuristic 2: A strategy must start with a sequence of entering actions. An entering action is defined as any action which takes the *base* cryptocurrency asset as input.

Heuristic 3: A strategy must end with a sequence of exiting actions. An exiting action is defined as any action that outputs the *base* cryptocurrency asset. Recall that the objective of the trader is to maximize the amount of *base* assets held.

Heuristic 4: Apart from the entering actions, an action must depend on at least one previous action. Conceptually, this is to avoid a strategy to contain actions that do not interact with any other actions. Given two actions $a_i, a_j \in A$, we define that a_i and a_j are independent actions, iff. there is no intersection between $\mathcal{K}^{\mathbb{T}}(a_i)$ and $\mathcal{K}^{\mathbb{T}}(a_j)$ (cf. Equation 8). In other words, the execution of a_i does not affect the execution results of a_j, no matter what concrete state is given.

$$a_i \perp\!\!\!\perp a_j \iff \mathcal{K}^{\mathbb{T}}(a_i) \cap \mathcal{K}^{\mathbb{T}}(a_j) = \emptyset \tag{8}$$

Recall that $\mathcal{K}(a)$ denotes the set of smart contract storage variables an action a reads from and writes to, and $\mathcal{K}^{\mathbb{T}}(a)$ denotes a subset of $\mathcal{K}(a)$, which is relevant to the trader $\mathbb{T}$. As an example of independence, we assume a_1 transacts c_1 to c_2 using a constant product market $M1$ with liquidity $L1^{c_1}$ and $L1^{c_2}$, and a_2 transacts c_1 to c_3 using another constant product market $M2$ with liquidity $L2^{c_1}$ and $L2^{c_3}$. Equation 9 shows the storage variables a_1 and a_2 reads from and writes to. a_1 and a_2 are not independent, as they both read/write variable $\mathbb{T}.c_1$. Therefore, Heuristic 4 does not prune the path containing a_1 and a_2.

$$\begin{aligned}\mathcal{K}^{\mathbb{T}}(a_1) &= \{M1.L1^{c_1}, M1.L1^{c_2}, \mathbb{T}.c_1, \mathbb{T}.c_2\}\\ \mathcal{K}^{\mathbb{T}}(a_2) &= \{M2.L2^{c_1}, M2.L2^{c_3}, \mathbb{T}.c_1, \mathbb{T}.c_3\}\end{aligned} \tag{9}$$

Heuristic 5: An action cannot be immediately followed by another reversing action (i.e., a mirroring action) on the same DeFi market. For instance, if a_1 transacts c_1 to c_2, and a_2 converts c_2 to c_1 on the same market, then heuristic 5 will prune all paths that contain a_1, a_2.

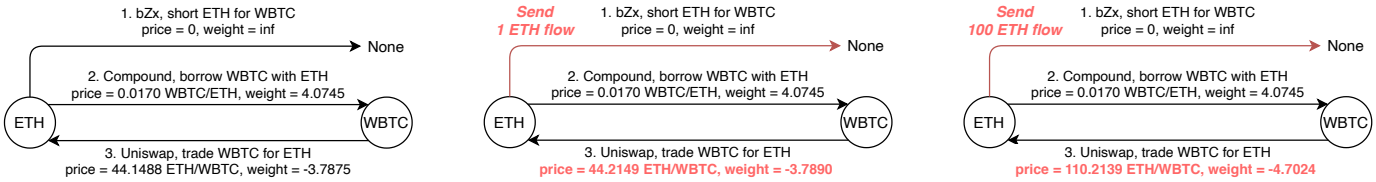
Heuristic 6: A path cannot include any branching. For example, a path of 5 actions $[c_1 \xrightarrow{a_1} c_2 \xrightarrow{a_2} c_4, c_1 \xrightarrow{a_3} c_3 \xrightarrow{a_4} c_4, c_4 \xrightarrow{a_5} c_1]$ is composed of two paths, $[c_1 \xrightarrow{a_1} c_2 \xrightarrow{a_2} c_4 \xrightarrow{a_5} c_1]$ and $[c_1 \xrightarrow{a_3} c_3 \xrightarrow{a_4} c_4 \xrightarrow{a_5} c_1]$ (cf. Figure 5a). In our work, we choose the more profitable path, and discard the other, because both paths affect the asset c_4. In a future work, it might be interesting to attempt to extract profit from both paths in an effort to maximize the revenue.

Heuristic 7: A path must not include any loops. For example, a path $[c_1 \xrightarrow{a_1} c_2 \xrightarrow{a_2} c_3 \xrightarrow{a_3} c_2 \xrightarrow{a_4} c_3 \xrightarrow{a_5} c_1]$ consists of a loop between c_2 and c_3. This path is composed of two sub-paths, namely $[c_1 \xrightarrow{a_1} c_2 \xrightarrow{a_2} c_3 \xrightarrow{a_5} c_1]$ and $[c_1 \xrightarrow{a_1} c_2 \xrightarrow{a_4} c_3 \xrightarrow{a_5} c_1]$ (cf. Figure 5b). We again chose the more profitable path, and discard the other for simplicity. We leave it to future work to optimize the potential gain.

The efficiency of path pruning can be evaluated across two dimensions: *(i)* the number of paths that are pruned, and, *(ii)* the reduction in revenue resulting from the heuristic pruning. To address the former, we show the reduction of the number of paths due to the heuristics in Table II and discuss these results further in Section VI-B. Regarding the latter, because we cannot quantify the optimal revenue due to the combinatorial explosion of the search space, we, unfortunately, see no avenue to quantify the reduction in revenue caused by the heuristics.

C. DEFIPOSER-SMT Revenue Optimizer

SMT solvers validate if any initialization of the free variables would satisfy the requirements defined. One requirement we specify is to increase the *base* cryptocurrency asset balance by a fixed amount. To find the maximum satisfiable revenue, we chose to use the following optimization algorithm (cf. Algorithm 3). At a high level, to identify a coarse upper and

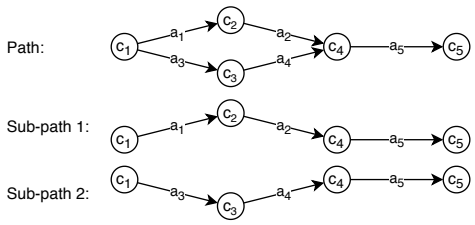


(a) Initial state graph. Cycle weight sum $= 4.07 - 3.79 = 0.28$

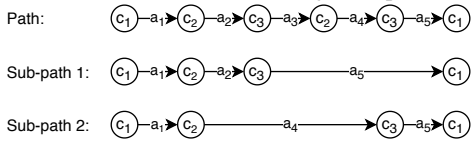
(b) Graph after shorting 1 ETH. Cycle weight sum $= 4.07 - 3.79 = 0.28$

(c) Graph after shorting $1,000$ ETH. Cycle weight sum $= 4.07 - 4.70 = -0.63$

Fig. 4: Directed weighted graph for the economic bZx attack on the Ethereum block $9,462,687$. Shorting ETH for WBTC on bZx does not return assets to the trader $\mathbb{T}$, and the action, therefore, does not point to any cryptocurrency assets. Graph 4a has no arbitrage opportunity on the WBTC/ETH market ($0.0170 \times 44.1488 = 0.75 \leq 1$). In Graph 4b and 4c, the weights change (ETH/WBTC price) after the trader increases the flow (in ETH) to the bZx market because bZx's price depends on the Uniswap price. The graph is hence dynamic [15], i.e., the weights need to be updated after each action. The action encoding of DeFiPoser-SMT models the bZx's price dependence on Uniswap. Note that the bZx attack does not violate **Heuristic 6**, because action 1 does not return any asset nor forms a sub-path (cf. Figure 15).



(a) Heuristic 6 - Branching example



(b) Heuristic 7 - Looping example

Fig. 5: Example of branching and looping paths.

lower revenue bound, this algorithm first attempts to solve, given multiples of 10 for the trader revenue. Given these bounds, we perform a binary search to find the optimal value.

D. *Comparing* DeFiPoser-SMT *to* DeFiPoser-ARB

Table I summarizes our comparison between DeFiPoser-SMT and DeFiPoser-ARB. While arbitrage opportunities appear plentiful, DeFiPoser-ARB cannot capture non-cyclic transactions such as the bZx case. Because DeFiPoser-SMT can encode any arbitrary strategy as an SMT problem, we argue that it is a more generic tool, as long as the underlying SMT solver can find a solution fast enough. We would like to stress again that both tools DeFiPoser-ARB and DeFiPoser-SMT do not provide optimal solutions. DeFiPoser-ARB greedily searches for arbitrage and extracts revenue as each opportunity arises. To show that DeFiPoser-ARB does not find optimal solutions, we provide the following example at block $9,819,643$. Here, DeFiPoser-SMT finds two opportunities:

Strategy 1 $[ETH \xrightarrow{Bancor} BNT \xrightarrow{Bancor} MKR \xrightarrow{Uniswap} ETH]$ with 0.20 ETH of revenue.

Strategy 2 $[ETH \xrightarrow{Uniswap} BAT \xrightarrow{Bancor} BNT \xrightarrow{Bancor} MKR \xrightarrow{Uniswap} ETH]$ with 0.11 ETH of revenue.

DeFiPoser-SMT will only execute strategy 1. DeFiPoser-ARB, however, finds and executes strategy 2 first to extract 0.11 ETH. After executing strategy 2 and updating the graph, strategy 1 is no longer profitable. Therefore, DeFiPoser-ARB only extracts a revenue of 0.11 ETH in this block. Note that DeFiPoser-SMT provides proof of satisfiable/unsatisfiable revenue targets for each considered path. However, DeFiPoser-SMT remains a best-effort tool because the heuristics prune paths that may be profitable. Contrary to DeFiPoser-ARB, DeFiPoser-SMT does not merge paths.

E. *Limitations*

We elaborate on a few limitations of our work.

State dependency: In this study, we focus on block-level state dependencies (cf. Appendix G), i.e., we consider a state to only change when a new block is mined. In practice, a DeFi state can change several times within the same blockchain block (as several transactions can trade on a DeFi platform within a block). Our assumption hence may cause us to not consider potentially profitable trades. An alternative approach to study state dependency, which we leave to future work, is to perform a transaction-level analysis. Such an analysis would assume that the trader observes the peer-to-peer network layer of the Ethereum network. Based on the information of transactions in the memory pool (the pool of unconfirmed transactions), the transaction order and state changes in the next block could be estimated ahead of the block being mined.

Scalability: One problem of DeFiPoser is the combinatorial path explosion. To mitigate this problem, heuristics reduce the path space, which only needs to be executed once. For every new block, DeFiPoser can parallelize the parameter search process to find the most profitable paths. A limitation of negative cycle detection is that it has to search for negative cycles before starting to search parameters. The graph needs to be updated after executing every strategy. This is difficult to parallelize and limits the system's real-time capability,

	DeFiPoser-ARB	DeFiPoser-SMT
Path generation	Bellman-Ford-Moore, Walk to the root; No acyclic paths	Pruning with heuristics; Any paths within the heuristics
Path selection	Combines multiple sub-paths	Selects the highest revenue path
Manual DeFi modeling	Not required	Required
Captures non-cyclic strategies	No	Yes (e.g., bZx)
Optimally chosen parameters	No	Yes (subject to inaccuracy of binary search)
Maximum Revenue	81.31 ETH (32,524 USD)	22.40 ETH (8,960 USD)
Total Revenue (over 150 days)	4,103.22 ETH (1,641,288 USD)	1,552.32 ETH (620,928 USD)
Lines of code (Python)	300	2,300

TABLE I: High-level comparison between DeFiPoser-ARB and DeFiPoser-SMT.

especially when there are multiple negative cycles, or the cycle length is long.

Manual Modeling and Code Complexity: DeFiPoser-ARB only needs to be aware of the spot price of each market and treats the underlying smart contracts and exchange protocols as a black box while greedily exploring opportunities. DeFiPoser-SMT, however, requires the manual translation of the objective function into an SMT problem. This requires to encode the state transitions into a group of predicates (cf. Appendix C). The modeling process not only increases the code complexity (cf. Table I) but also causes inaccuracies in the found solutions and therefore requires a validation process through, e.g., concrete execution.

Approximated Revenue: To avoid double-counting revenue when a profitable path exists over multiple blocks, we apply a state dependency analysis and only exploit paths with a state change (cf. Section G). However, DeFiPoser's reported revenue is not accurate because: (i) We work on historical blockchain states. In practice, the profitability of DeFiPoser will be affected by the underlying blockchain's network layer; (ii) For simplicity within this work, we assume that DeFiPoser does not change other market participants' behavior. In practice, other traders are likely to monitor our activity and adjust their trading strategy accordingly.

Multiple Traders: Within this work, we only consider a single trader using DeFiPoser. Zhou *et al.* [61] simulated the outcome of competing transactions from several traders under a reactive counter-bidding strategy. We believe that those results translate over to MEV when multiple traders (specifically non-miners) compete over DeFiPoser transactions. Zhou *et al.* [61]'s results suggest that the total revenue will be divided among the competing traders.

VI. Experimental Evaluation

To query the Ethereum blockchain, we set up a full archive Geth[1] node (i.e., a node which stores all intermediate blockchain state transitions) on a AMD Ryzen Threadripper 3990 X Processor (4.3 GHz, 64 cores), 4x2 TB NVMe SSD RAID 0 and 256 GB RAM. We perform the concrete execution with a custom py-evm[2], which can fork the Ethereum blockchain at any given block height. To simplify our experimental complexity, we do not consider trades which yield below 0.10 ETH (40 USD) and are aware that this potentially reduces the resulting financial gain.

We select 96 actions from the Uniswap, Bancor, and MakerDAO, with a total of 25 assets (cf. Table III and IV in Appendix). To enable action chaining, all considered assets trade on Uniswap and Bancor, while SAI and DAI are convertible on MakerDAO. The total value of assets on the three platforms sums up to 3.3 billion USD, which corresponds to 82% of the total USD value locked in DeFi as of May 2020.

Both DeFiPoser-ARB and DeFiPoser-SMT apply dependency-based state reduction. Stationary blockchain states are identified and skipped to avoid redundant computation and double counting of revenue.

A. DeFiPoser-ARB

We translate the 25 assets and 96 actions into a graph with 25 nodes and 94 edges. Each node in the graph represents a cryptocurrency asset. For each edge $e_{i,j}$ pointing from asset c_i to c_j, we find all markets with asset c_i as input, and output asset c_j. Each edge's weight is derived using the highest price found among all supporting markets, or 0 if there is no market. We then follow Algorithm 1 to greedily extract arbitrage revenue as soon as one negative cycle is found. We use the BFCF (Bellman-Ford-Moore, Walk to the root) algorithm to find negative cycles, which operates in $O(|N^2| \cdot |E|)$. For each arbitrage opportunity, DeFiPoser-ARB gradually increases the input parameter (amount of *base* cryptocurrency asset) until the revenue ceases to increase.

B. DeFiPoser-SMT

We translate DeFi states into Z3 [21] as constraints on state symbolic variables (cf. Section III). We symbolically encode all variables using floats instead of integers because the EVM only supports integers. Most DeFi smart contracts express floats as integers by multiplying floats with a large factor. Division and power are, therefore, estimated using integer math. This practice may introduce a bias in our state and transition functions. Due to such model inaccuracies, we proceed to concrete execution (i.e., real-world smart contract execution on the EVM) to avoid false positives and validate our result.

An exhaustive search over the total action space is infeasible. Therefore, we apply path pruning (cf. Section V-B) to discard irrelevant paths.

Path Discovery and Pruning: The 96 DeFi actions (cf. Table IV in Appendix) result in 9.92×10^{149} possible paths

[1] https://github.com/ethereum/go-ethereum
[2] https://github.com/ethereum/py-evm

Path length	Before	After
2	9,120	2
3	857,280	90
4	79,727,040	466
5	7,334,887,680	42
Total	7,415,481,120	600

TABLE II: Results of path pruning after applying the heuristics from Section V-B. In total, 600 paths remain, with the majority (77.67%) consisting of 4 actions. For each path length, the heuristics remove at least 99.98% of the strategies.

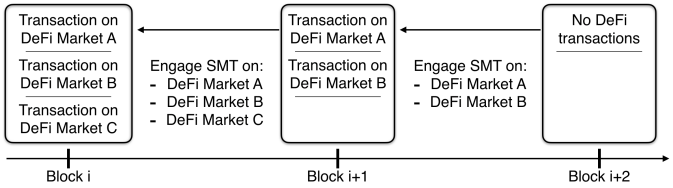


Fig. 6: We consider each blockchain block as an independent state representation of the DeFi platform markets. Only if a DeFi market changes in state, we need to re-engage the SMT solver for the affected paths only.

without repeating actions, which is an impractical space to evaluate. Table II hence illustrates the impact of our heuristics on paths of various lengths. We observe a significant reduction of at least 99.98% per path length of the total number of considered paths, resulting in only 600 remaining paths. The majority of paths (77.67%) consist of 4 actions, while the shortest paths count 2 actions, and the longest 5 actions. Although we do not enforce a constraint on the maximum number of actions, all paths with more than 5 actions failed to pass our heuristics.

Action Dependency: In the following, we present a concrete example of determining the dependency between two actions. The first action a_{Uniswap} transacts ETH to SAI using the Uniswap SAI market. The second action a_{Bancor} transacts BNT to SAI using the Bancor SAI contract. Equation 10 and Equation 11 show the relevant storage variables, respectively. These two actions are not independent, as they both modify the trader's balance in the SAI contract.

$$\mathcal{K}^{\mathbb{T}}(a_{\text{Uniswap}}) = \{\text{<UniswapSAI>.ETH,}\ \text{<SAI>.balance of UniswapSAI,}\ \text{<Trader>.ETH,}\ \text{<SAI>.balance of trader}\} \quad (10)$$

$$\mathcal{K}^{\mathbb{T}}(a_{\text{Bancor}}) = \{\text{<BNT>.balance of BancorSAI}\ \text{<SAI>.balance of BancorSAI}\ \text{<BNT>.balance of trader}\ \text{<SAI>.balance of trader}\} \quad (11)$$

Dependency-based Blockchain State Reduction: If a DeFi state does not change across a number of blockchain blocks, the same SMT solver computation is not re-engaged (cf. Figure 6). Algorithm 2 specifies the algorithm we apply to automate the dependency-based blockchain state reduction. Figure 15 in the Appendix shows a timeline analysis of the state dependencies for all considered assets. We observe that ETH experiences the most state changes with over 950,000 blocks (36.76%), followed by DAI (14.62%).

Algorithm 2: Block state dependency analysis.

Input:
$p = (a_1, a_2, \ldots) \in P \leftarrow$ Path ; $b \leftarrow$ Block number
Output: Has a state change
foreach $a \in p$ **do**
  foreach $s \in \mathcal{K}^{\mathbb{T}}(a)$ **do**
    if *fetch(s, b)* $\neq$ *fetch(s, b - 1)* **then**
      return *True*
    end
  end
end
return *False*

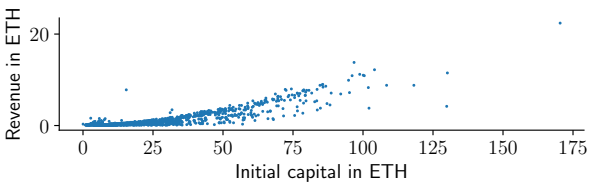
Function *fetch(s, b)* **is**
  return (*Concrete value for storage variable address s on block b*)
end

C. DeFiPoser-ARB *and* DeFiPoser-SMT*: Revenue*

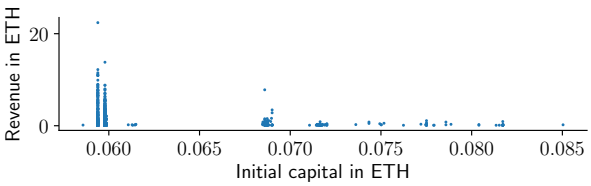
We validate both DeFiPoser designs on past blockchain data from block 9,100,000 to block 10,050,000, over a total of 150 days. We visualize the distribution of traders' revenue for DeFiPoser-SMT in Figure 8. DeFiPoser-SMT found 13,317 strategies consisted of 2 to 5 actions. In total DeFiPoser-SMT yields a total of 1,552.32 ETH (620,928 USD), and we observe that the most profitable strategies consist of 3 actions, where the highest revenue yielded amounts to 22.40 ETH (8,960 USD). Similarly, Figure 9 visualizes the distribution of traders' revenue for DeFiPoser-ARB. Recall that DeFiPoser-ARB greedily combine multiple paths into a single strategy. We observe that the revenue increases as the number of paths increases, with the highest revenue amounting to 81.31 ETH (32,524 USD). In total, DeFiPoser-ARB finds 2,709 strategies and yields 4,103.22 ETH (1,641,288 USD).

We visualize in Figure 7 the revenue generated by DeFiPoser-SMT and DeFiPoser-ARB as a function of the initial capital. If a trader owns the *base* asset (e.g., ETH), most strategies require less than 150 ETH. Only 10 strategies require more than 100 ETH for DeFiPoser-SMT, and only 7 strategies require more than 150 ETH for DeFiPoser-ARB. This capital requirement is reduced to less than 1.00 ETH (400 USD) when using flash loans (cf. Figure 7 (b, d)).

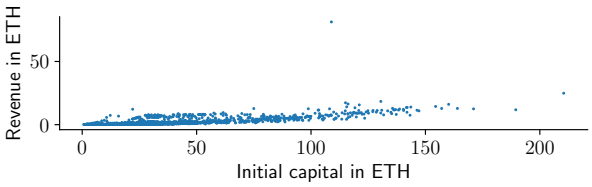
Figure 10a shows how our concrete execution validation over 150 days yields consistent revenue for both tools. The concrete execution estimates a weekly revenue of 191.48 ETH (76,592 USD) for DeFiPoser-ARB and 72.44 ETH (28,976 USD) for DeFiPoser-SMT. For DeFiPoser-SMT, our validation estimates a total revenue



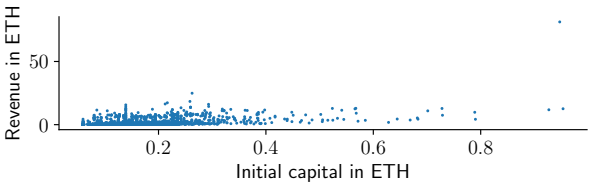
(a) DEFIPOSER-SMT without flash loans.



(b) DEFIPOSER-SMT with flash loans.



(c) DEFIPOSER-ARB without flash loans.



(d) DEFIPOSER-ARB with flash loans.

Fig. 7: Revenue as a function of the initial capital, in ETH with and without flash loans for DEFIPOSER-ARB (total of $2,709$ found strategies) and DEFIPOSER-SMT (total of $1,556$ found strategies).

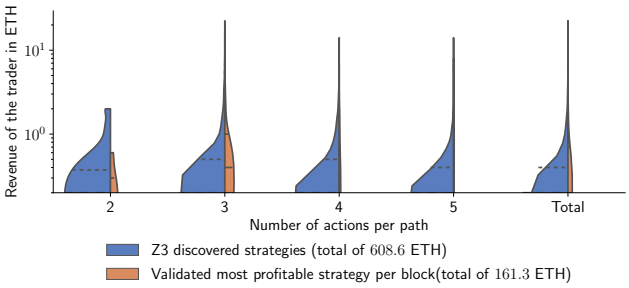


Fig. 8: Analytical distribution of the trader's revenue. The majority of the most profitable strategies consist of 3 actions. Because DEFIPOSER-SMT requires manual modeling, the revenues discovered by Z3 are not accurate, and thus the number of discovered strategies (yellow) are less than the profitable strategies (blue). We use concrete execution to validate the strategies from Z3.

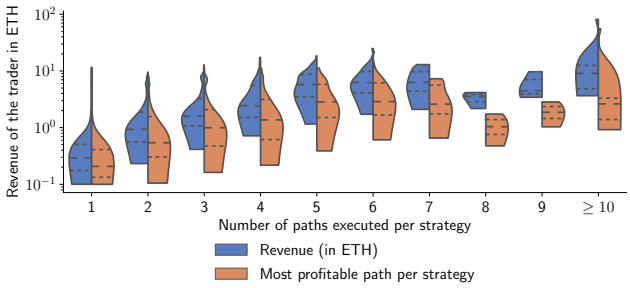


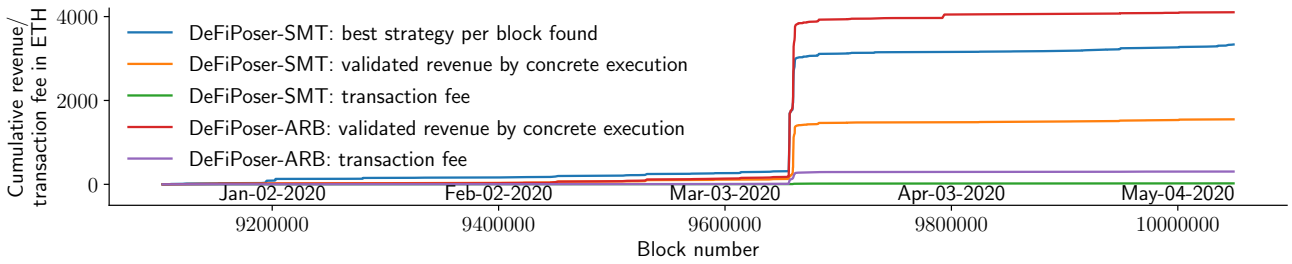
Fig. 9: Distribution of the trader's revenue using DEFIPOSER-ARB. We observe that the revenue increases as the number of paths increases. We also visualize the distribution of the most profitable sub-path (orange) for every strategy. Intuitively, the more paths DEFIPOSER-ARB try to combine, the higher the revenue.

of 1,552.32 ETH (620,928 USD) out of 3,577.14 ETH (1,430,856 USD) (i.e., 40% of the Z3 indicated revenue is validated in practice).

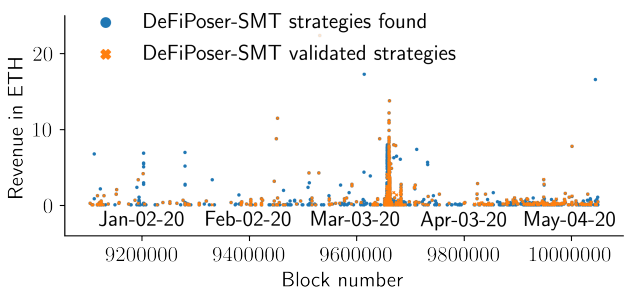
Cost Analysis: The trader's principal costs are the blockchain transaction fees (e.g., gas in Ethereum), which remain below the revenue yielded by the strategies we validated (cf. Figure 11). Note that a trading strategy may fail if the underlying market state changes before its execution. Therefore, we assume that the trader adopts the gas price of 32 GWei, which is highly volatile, but the recommended fast transaction gas price at the time of writing. Summarizing, the execution of all strategies costs less than 0.05 ETH, which warrants all strategies to be profitable.

Performance Analysis: Our tools must find trades within the average Ethereum block time of 13.5 ± 0.12 seconds [9] to be applicable in real-time. Assuming a network propagation latency of roughly three seconds towards miners in the blockchain P2P network [22], our tools must generate transactions within at most 10.5 seconds. Figure 12 shows the detailed execution speed of DEFIPOSER-ARB and DEFIPOSER-SMT on an AMD Ryzen Threadripper 3990 X Processor (4.3 GHz, 64 cores) CPU. For new block states, we measure a total average computing time of 6.43 seconds and 5.39 seconds per block, respectively.

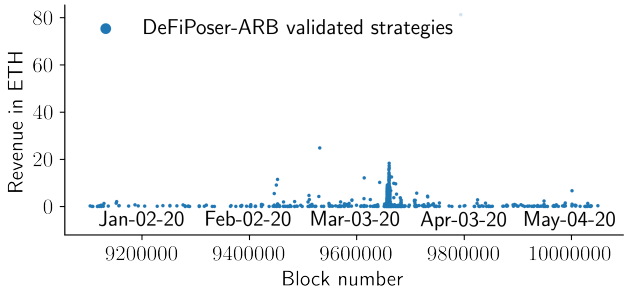
We further group the strategies detected by DEFIPOSER-ARB based on the number of negative cycles and compare the respective analysis time (cf. Figure 13). We find that DEFIPOSER-ARB exceeds our estimated time limit ($13.5 - 3 = 10.5$ seconds) when exploiting more than 6 cycles. The higher the total number of negative cycles, the more likely DEFIPOSER-ARB misses the most profitable opportunity.



(a) Cumulative revenue over time (150 days), found by DEFIPOSER-SMT and DEFIPOSER-ARB, and validated via concrete execution.



(b) Strategies detected and validated by DEFIPOSER-SMT.



(c) Strategies validated by DEFIPOSER-ARB.

Fig. 10: Revenue and transaction fees analysis over time, measured in blocks.

VII. PROFITABLE TRANSACTIONS AND BLOCKCHAIN SECURITY

In this section, we show that DEFIPOSER-SMT is capable of identifying the economic bZx attack from February 2020 [53] and provide forensic insights into the event. Given optimal adversarial strategies provided by an MDP, we then quantify whether an MEV opportunity will cause a rational miner to create a blockchain fork.

A. Economic bZx Attack

On the 15th of February, 2020, a trader performed a pump and arbitrage attack on the margin trading platform bZx[3]. The core of this trade was a pump and arbitrage involving four DeFi platforms atomically executed in one single transaction. As a previous study shows, this trade resulted in in 4,337.62 ETH (1,735,048 USD) loss from bZx loan providers, where the trader gained 1,193.69 ETH (477,476 USD) in total [53].

Attack Window: To gain deeper insights into this DeFi composability event, we extend DEFIPOSER-SMT with two additional actions: *(i)* borrow WBTC with ETH on compound finance; *(ii)* short ETH for WBTC on bZx. We replayed DEFIPOSER-SMT on historical blockchain data by starting at the creation of the bZx's margin short smart contract (cf. Figure 16). Surprisingly, the bZx attack window lasted for 69 days until it was openly exploited. DEFIPOSER-SMT finds that the attack yielded the highest revenue of 2,291.02 ETH (916,408 USD) at block $9,482,670$, which is about one day before the attack occurred.

B. MEV, an MDP and Optimal Adversarial Strategies

The economic bZx attack revenue exceeds the average Ethereum block reward[4] by a factor of $874\times$. After bZx, the other most profitable validated strategies found by DEFIPOSER-ARB and DEFIPOSER-SMT exceed the block reward by a factor of $31\times$ and $8.5\times$ respectively. In this section, we quantify the value at which an MEV-aware miner would exploit an MEV opportunity by forking the blockchain.

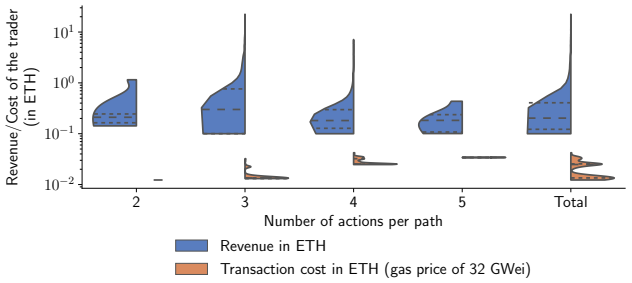
Markov Decision Process: A Markov Decision Process is a single-player decision process that allows identifying the optimal strategies for an encoded decision problem. In this work, we adopt the state transition and reward matrix of the PoW double-spending MDP of Gervais *et al.* [31]. Note that the MDP we use does not consider uncle rewards.

We observe that conceptually, an MEV opportunity is equivalent to a double-spending opportunity: if an MEV opportunity is mined by an honest miner, and an adversarial miner aims to claim the MEV opportunity, the MEV miner will need to outrun the honest chain with a fork. The MEV miner will hence want to follow the optimal adversarial strategies given by the MDP, which advises whether to fork or not to fork the blockchain depending on the MEV value.

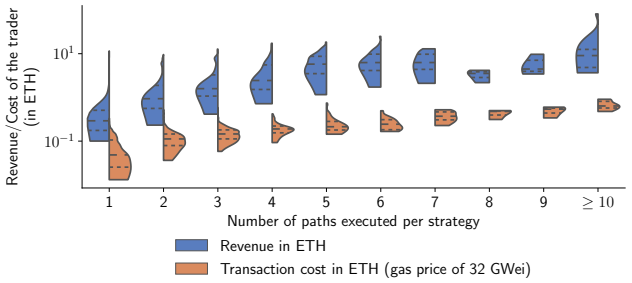
Threat Model: We assume a rational and computationally bounded adversary. Because MDP's are single-player decision problems, we assume the existence of only one adversarial

[3]transaction id: 0xb5c8bd9430b6cc87a0e2fe110ece6bf527fa4f170a4bc8cd032f768fc5219838

[4]At the time of writing, the average Ethereum block reward is 2.62 ETH (https://bitinfocharts.com/ethereum/)



(a) DEFIPOSER-SMT



(b) DEFIPOSER-ARB

Fig. 11: Distribution of revenue and transaction cost based on concrete execution on the EVM for DEFIPOSER-SMT and DEFIPOSER-ARB. The revenue outpaces the transaction costs, which are higher for DEFIPOSER-ARB because the found strategies often consist of more cycles (arbitrage opportunities).

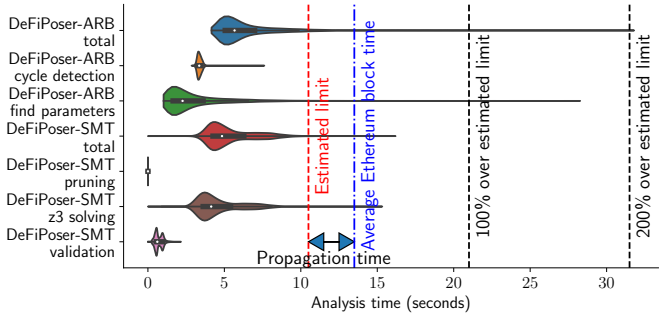


Fig. 12: Analysis time distribution to detect a profitable strategy on DEFIPOSER-SMT and DEFIPOSER-ARB. For most strategies the search and validation process remains below the average Ethereum block time of 13.5 ± 0.12 seconds.

miner willing to exploit MEV. We parametrize the miner with a hash rate $\alpha \in]0, 0.5[$, while the remaining non-MEV miners have a hash rate of $1 - \alpha$. We ignore the existence of eclipse attacks ($\omega = 0$) and assume the weakest possible network propagation parameter of the adversary ($\gamma = 0$). We parametrize the MDP with the stale block rate of the Ethereum blockchain at the time of writing. By crawling the number of uncle blocks (from the block 9.1M to 10.5M), we approximate the stale block rate to $r_s = 5.72\%$. We set the mining costs to match the hash rate of the MEV miner ($c_m = \alpha$).

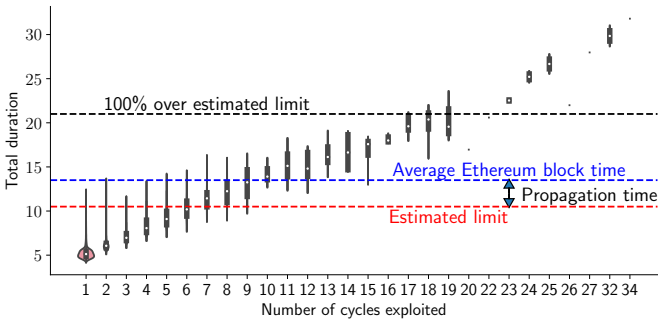


Fig. 13: The analysis time of DEFIPOSER-ARB exceeds our estimated time limit (taking into consideration block time and transaction network propagation to miners), when DEFIPOSER-ARB exploits more than 6 cycles.

Exploit or not exploit MEV? Each time an MEV opportunity arises on the network layer, we assume that the honest miner succeeds in mining the MEV opportunity, and the MEV miner fails to receive the reward initially. The MEV miner, therefore, needs to decide whether to start to mine on a private chain, where he claims the MEV opportunity. Note that the MDP's $exit$ state can only be reached when the MEV miner mined a private chain that is longer than the honest chain ($l_a > l_h$) given $k = 1$ ($l_a > k$). *Depending on the MEV value, the optimal strategy π might advise against forking the chain to attempt to claim the MEV reward.* We quantify the minimal MEV value MEV_v, such that MEV_v is strictly larger than the reward from honest mining (cf. Equation 13). We denote h is the process of mining honestly.

$$P = (\alpha, \gamma, r_s, k, \omega, c_m) \tag{12}$$

$$MEV_v = min\{MEV_v | \exists \pi \in A : R(\pi, P, MEV_v) > R(h, P)\} \tag{13}$$

To solve the MEV MDP for the optimal strategies, we use the code of [31][5] and reparametrize given the current Ethereum stale block rate ($r_s = 5.72\%$). We further set $k = 1$, $\gamma = 0$, $\omega = 0$ and the cut-off value (the maximum length of l_a and l_h) to 20 blocks. Similar to [31], we apply a binary search to find the lowest value for MEV_v in units of block rewards, given a margin of error of 0.1.

Results: We visualize our findings in Figure 14, which shows that for an MEV miner with 10% hash rate, on Ethereum (stale block rate of 5.72%), MEV_v equals to 4. We conclude that in this case, if an MEV opportunity yields at least a reward that is 3 times higher than the block reward, then an MEV miner which follows the optimal strategies will fork the blockchain. A fork of the blockchain deteriorates the blockchain's security as it increases the risks of double-spending and selfish mining [31].

Multiple MEV Miner: Our MDP model does not allow us to draw conclusions on the dynamics under multiple independent MEV miners. We hence can only speculate about the outcome

[5] https://github.com/arthurgervais/pow_mdp

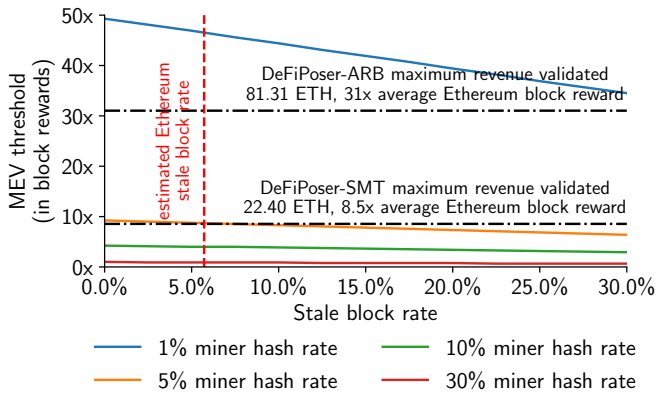


Fig. 14: Minimum MEV value in terms of block rewards to fork a PoW blockchain, given by optimal adversarial strategies of the MDP. For instance, on Ethereum ($r_s = 5.72\%$), a miner with 10% hash rate will engage to fork the chain to exploit an MEV opportunity, if the adversary follows the optimal strategy and the MEV opportunity yields more than 4 block rewards.

and leave a simulation to future work. We can imagine that multiple miners either collaborate to share an MEV profit (which falls back to our MDP game of one adversary), or the miners would compete among each other, which is likely to exacerbate the fork rate and hence further deteriorates the blockchain consensus security.

VIII. Related Work

While the research literature of blockchain span over 10 years, DeFi is a relatively recent area with fewer works.

DeFi: There is a growing body of literature focusing on the security of the DeFi ecosystem. Blockchain front-running in exchanges, games, gambling, mixer, the network layer, and name services is soundly studied [1], [12], [20], [25], [32], [42], [47], [61]. Daian *et al.* [20] demonstrate a thorough analysis of profiting from opportunities provided by transaction ordering. Xu *et al.* [60] presents a detailed study of a specific market manipulation scheme, pump-and-dump, and build a prediction model that estimates the pump likelihood of each coin. Gudgeon *et al.* [37] explore the possibility of a DeFi crisis due to the design weakness of DeFi protocols and present a stress testing framework. Qin *et al.* [53] investigate DeFi attacks through flash loans and how to optimize their profit. We remark that the optimization solution presented in [53] only applies to previously fixed attack vectors, while this work considers the composability of DeFi protocols.

Smart Contract Analysis: Besides the above-mentioned works on DeFi, many studies on the vulnerability discovery of smart contracts are related to our work [10], [13], [16], [19], [36], [38], [39], [41], [43], [45], [48], [56], [57]. Traditional smart contract vulnerabilities examined in related work include, for instance, re-entrancy attack, unhandled exceptions, locked ether, overflow [48]. To the best of our knowledge, no analysis tool has yet considered the problem of a *composability analysis* as we've performed.

Model Checking: Model-checking is another viable method to verify the security of smart contracts. Model-checking examines all possible states in a brute-force manner [7] and performs systematic exhaustive exploration for checking whether a finite transition machine model of a system meets appropriate specifications [13], [16], [36], [43], [45], [48], [56], [57]. One of the main limitations of model-checking is the exponential growth of the number of possible states, resulting in unsolvability for complex contracts.

IX. Conclusions

This paper presents two practical approaches that automatically extract revenue from the intertwined mesh of decentralized finance protocols. The first technique, DeFiPoser-ARB, is well-suited for arbitrage, and the second, DeFiPoser-SMT, can also find acyclic opportunities. When evaluated over a span of 150 days with 96 DeFi actions and 25 cryptocurrency assets, DeFiPoser-ARB and DeFiPoser-SMT are estimated to generate an average weekly revenue of 191.48 ETH (76,592 USD) and 72.44 ETH (28,976 USD), with the highest transaction being 81.31 ETH (32,524 USD) and 22.40 ETH (8,960 USD), respectively.

Our techniques apply to a real-time operation on blockchains with reasonably fast inter-block times (such as Ethereum), with an average search of 6.43 seconds and 5.39 seconds per block for DeFiPoser-ARB and DeFiPoser-SMT, respectively, using a relatively unoptimized implementation. We find that the capital requirements to extract the found revenues are minimal: the majority of strategies produced require less than 150.00 ETH (60,000 USD), without, and less than 1.00 ETH (400 USD) with flash loans.

We quantitatively demonstrate some troubling security implications of profitable transactions on the blockchain consensus. Given optimal adversarial strategies provided by a Markov Decision Process, we quantify the threshold value at which an MEV-aware rational miner will fork the blockchain if the miner does not succeed in claiming an unconfirmed MEV opportunity first. For example, on the current Ethereum network, a 10% hash rate miner will fork the chain if an MEV opportunity exceeds 4 block rewards. As a comparison, the bZx opportunity exceeded the Ethereum block reward by a factor of 874×! Our work hence quantifies the inherent tension between revenue extraction from profitable transactions and blockchain security. We can generally expect trading opportunities highlighted in this paper to expand as the DeFi ecosystem grows and becomes more popular.

Acknowledgments

We very much thank the anonymous reviewers and Nicolas Christin for the thorough reviews and helpful suggestions that significantly strengthened this paper. We are moreover grateful to the Lucerne University of Applied Sciences and Arts for generously supporting Kaihua Qin's Ph.D.

REFERENCES

[1] Consensys/0x-review: Security review of 0x smart contracts. https://github.com/ConsenSys/0x-review.

[2] How many stablecoins are there? - cementdao - medium. https://medium.com/cementdao/how-many-stablecoins-are-there-aa39d201ac12.

[3] Bzx network, 2020.

[4] Aave. Aave Protocol. https://github.com/aave/aave-protocol, 2020.

[5] Akropolis. Akropolis Hack Update.

[6] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. A survey of attacks on ethereum smart contracts (sok). In *International conference on principles of security and trust*, pages 164–186. Springer, 2017.

[7] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT press, 2008.

[8] Richard Bellman. On a routing problem. *Quarterly of applied mathematics*, 16(1):87–90, 1958.

[9] Bitinfocharts. Ethereum block time.

[10] Marcel Böhme, Van-Thuan Pham, Manh-Dung Nguyen, and Abhik Roychoudhury. Directed greybox fuzzing. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2329–2344, 2017.

[11] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A Kroll, and Edward W Felten. Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 104–121. IEEE, 2015.

[12] Lorenz Breidenbach, Phil Daian, Florian Tramèr, and Ari Juels. Enter the hydra: Towards principled bug bounties and exploit-resistant smart contracts. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 1335–1352, 2018.

[13] Lexi Brent, Anton Jurisevic, Michael Kong, Eric Liu, Francois Gauthier, Vincent Gramoli, Ralph Holz, and Bernhard Scholz. Vandal: A scalable security analysis framework for smart contracts. *arXiv preprint arXiv:1809.03981*, 2018.

[14] Roberto Bruttomesso, Alessandro Cimatti, Anders Franzén, Alberto Griggio, and Roberto Sebastiani. The mathsat 4 smt solver. In *International Conference on Computer Aided Verification*, pages 299–303. Springer, 2008.

[15] Nitin Chandrachoodan, Shuvra S Bhattacharyya, and KJ Ray Liu. Adaptive negative cycle detection in dynamic graphs. In *ISCAS 2001. The 2001 IEEE International Symposium on Circuits and Systems (Cat. No. 01CH37196)*, volume 5, pages 163–166. IEEE, 2001.

[16] Jialiang Chang, Bo Gao, Hao Xiao, Jun Sun, Yan Cai, and Zijiang Yang. scompile: Critical path identification and analysis for smart contracts. In *International Conference on Formal Engineering Methods*, pages 286–304. Springer, 2019.

[17] James Chen. Bid and ask definition, Sep 2020.

[18] Boris V Cherkassky and Andrew V Goldberg. Negative-cycle detection algorithms. *Mathematical Programming*, 85(2), 1999.

[19] Crytic. Echidna: Ethereum fuzz testing framework, February 2020.

[20] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash Boys 2.0: Frontrunning, Transaction Reordering, and Consensus Instability in Decentralized Exchanges. *arXiv preprint arXiv:1904.05234*, 2019.

[21] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[22] Christian Decker and Roger Wattenhofer. Information propagation in the bitcoin network. In *Conference on Peer-to-Peer Computing*, pages 1–10, 2013.

[23] Value DeFi. MultiStables Vault Exploit Post-Mortem.

[24] dYdX. dYdX. https://dydx.exchange/, 2020.

[25] Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark. Sok: Transparent dishonesty: front-running attacks on blockchain. 2019.

[26] Balancer Finance. Balancer Finance.

[27] Compound Finance. Compound finance, 2019.

[28] Harvest Finance. Harvest Flashloan Economic Attack Post-Mortem.

[29] Lester Randolph Ford Jr and Delbert Ray Fulkerson. *Flows in networks*. Princeton university press, 2015.

[30] The Maker Foundation. Makerdao. https://makerdao.com/en/, 2019.

[31] Arthur Gervais, Ghassan O Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 3–16. ACM, 2016.

[32] Arthur Gervais, Hubert Ritzdorf, Ghassan O Karame, and Srdjan Capkun. Tampering with the delivery of blocks and transactions in bitcoin. In *Conference on Computer and Communications Security*, pages 692–705. ACM, 2015.

[33] Andrew Goldberg and Tomasz Radzik. A heuristic improvement of the bellman-ford algorithm. Technical report, STANFORD UNIV CA DEPT OF COMPUTER SCIENCE, 1993.

[34] Andrew V Goldberg. Scaling algorithms for the shortest paths problem. *SIAM Journal on Computing*, 24(3):494–504, 1995.

[35] Donald Goldfarb, Jianxiu Hao, and Sheng-Roan Kai. Shortest path algorithms using dynamic breadth-first search. *Networks*, 21(1):29–50, 1991.

[36] Neville Grech, Michael Kong, Anton Jurisevic, Lexi Brent, Bernhard Scholz, and Yannis Smaragdakis. Madmax: Surviving out-of-gas conditions in ethereum smart contracts. *Proceedings of the ACM on Programming Languages*, 2(OOPSLA):1–27, 2018.

[37] Lewis Gudgeon, Daniel Perez, Dominik Harz, Arthur Gervais, and Benjamin Livshits. The decentralized financial crisis: Attacking defi, 2020.

[38] Campbell R Harvey. Cryptofinance. 2016.

[39] Jingxuan He, Mislav Balunović, Nodar Ambroladze, Petar Tsankov, and Martin Vechev. Learning to fuzz from symbolic execution with application to smart contracts. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, pages 531–548, New York, NY, USA, 2019. ACM.

[40] Eyal Hertzog, Guy Benartzi, and Galia Benartzi. Bancor protocol. 2017.

[41] Bo Jiang, Ye Liu, and WK Chan. Contractfuzzer: Fuzzing smart contracts for vulnerability detection. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pages 259–269. ACM, 2018.

[42] Harry A Kalodner, Miles Carlsten, Paul Ellenbogen, Joseph Bonneau, and Arvind Narayanan. An empirical study of namecoin and lessons for decentralized namespace design. In *WEIS*. Citeseer, 2015.

[43] Sukrit Kalra, Seep Goel, Mohan Dhawan, and Subodh Sharma. Zeus: Analyzing safety of smart contracts. In *NDSS*, 2018.

[44] Jeff L Kennington and Richard V Helgason. *Algorithms for network programming*. John Wiley & Sons, Inc., 1980.

[45] Johannes Krupp and Christian Rossow. teether: Gnawing at ethereum to automatically exploit smart contracts. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 1317–1333, 2018.

[46] Kyber. Kyber. https://kyber.network/, 2020.

[47] Duc V Le and Arthur Gervais. Amr: Autonomous coin mixer with privacy preserving reward distribution. *arXiv preprint arXiv:2010.01056*, 2020.

[48] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 254–269, 2016.

[49] Makiko Mita, Kensuke Ito, Shohei Ohsawa, and Hideyuki Tanaka. What is stablecoin?: A survey on price stabilization mechanisms for decentralized payment systems. *arXiv preprint arXiv:1906.06037*, 2019.

[50] Amani Moin, Kevin Sekniqi, and Emin Gun Sirer. Sok: A classification framework for stablecoin designs. In *Financial Cryptography*, 2020.

[51] Edward F Moore. The shortest path through a maze. In *Proc. Int. Symp. Switching Theory, 1959*, pages 285–292, 1959.

[52] DeFi Pulse. The DeFi Leaderboard. https://defipulse.com/, 2019.

[53] Kaihua Qin, Liyi Zhou, Benjamin Livshits, and Arthur Gervais. Attacking the defi ecosystem with flash loans for fun and profit. *Financial Cryptography and Data Security (FC)*, 2021.

[54] Matheus Souza, Mateus Borges, Marcelo d'Amorim, and Corina S Păsăreanu. Coral: solving complex constraints for symbolic pathfinder. In *NASA Formal Methods Symposium*, pages 359–374. Springer, 2011.

[55] theblockcrypto. DeFi protocol Origin gets attacked, loses 7 million USD.

[56] Sergei Tikhomirov, Ekaterina Voskresenskaya, Ivan Ivanitskiy, Ramil Takhaviev, Evgeny Marchenko, and Yaroslav Alexandrov. Smartcheck: Static analysis of ethereum smart contracts. In *Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain*, pages 9–16, 2018.

[57] Petar Tsankov, Andrei Dan, Dana Drachsler-Cohen, Arthur Gervais, Florian Buenzli, and Martin Vechev. Securify: Practical security analysis of smart contracts. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 67–82. ACM, 2018.

[58] Uniswap.io, 2018. accessed 12 November, 2019, https://docs.uniswap.io/.

Token	Unique holders	Transfer transactions	Markets trading
SAI	181,223	3,139,071	4
BNT	23,966	2,620,652	144
DAI	68,357	2,155,535	130
BAT	288,970	1,970,176	218
ENJ	52,341	902,471	66
SNT	82,663	868,007	101
KNC	65,018	820,501	73
MKR	20,891	733,845	67
DATA	444,833	588,097	26
MANA	38,276	565,151	77
ANT	22,321	217,657	24
RLC	12,880	209,255	24
RCN	19,831	203,893	24
UBT	10,410	191,153	14
GNO	10,695	170,507	21
RDN	13,842	143,308	16
TKN	5,485	84,912	7
TRST	7,738	71,223	7
AMN	2,593	53,010	3
FXC	2,024	47,906	14
SAN	2,247	36,054	7
AMPL	1,931	31,124	10
HEDG	1,709	30,770	17
POA20	560	26,390	10

TABLE III: Summary of the 24 ERC-20 cryptocurrency assets used in our experiments, ordered by the total number of transfer transactions.

[59] Karl Wüst and Arthur Gervais. Do you need a Blockchain? In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pages 45–54. IEEE, 2018.

[60] Jiahua Xu and Benjamin Livshits. The anatomy of a cryptocurrency pump-and-dump scheme. In *Proceedings of the Usenix Security Symposium*, August 2019.

[61] Liyi Zhou, Kaihua Qin, Christof Ferreira Torres, Duc V Le, and Arthur Gervais. High-frequency trading on decentralized on-chain exchanges. *IEEE Symposium on Security and Privacy*, 2021.

APPENDIX A
SUMMARY OF THE ERC-20 CRYPTOCURRENCY ASSETS

We summarize the 24 ERC-20 cryptocurrency assets in Table III. We observe that for most of the assets, the number of holders and the number of markets increases with the number of transfer transactions.

APPENDIX B
SUPPORTED DEFI ACTIONS

We summarize the 96 DeFi actions DEFIPOSER supports in Table IV. All considered cryptocurrency assets trade on both the Uniswap and Bancor exchanges. SAI and DAI, in addition, can be converted to each on MakerDAO.

APPENDIX C
SMT ENCODING EXAMPLE

To ease the understanding of the encoding process between the State Transition Model and the SMT problem, we consider in the following a simple strategy with only two actions, and a trader holding two cryptocurrency assets: a *base* cryptocurrency asset c_1, and another cryptocurrency asset c_2.

Uniswap	
From:	**To:**
ETH	AMN
AMN	ETH
ETH	AMPL
AMPL	ETH
ETH	ANT
ANT	ETH
ETH	BAT
BAT	ETH
ETH	BNT
BNT	ETH
ETH	DAI
DAI	ETH
ETH	DATA
DATA	ETH
ETH	ENJ
ENJ	ETH
ETH	FXC
FXC	ETH
ETH	GNO
GNO	ETH
ETH	HEDG
HEDG	ETH
ETH	KNC
KNC	ETH
ETH	MANA
MANA	ETH
ETH	MKR
MKR	ETH
ETH	POA20
POA20	ETH
ETH	RCN
RCN	ETH

Bancor	
From:	**To:**
BNT	AMN
AMN	BNT
BNT	AMPL
AMPL	BNT
BNT	ANT
ANT	BNT
BNT	BAT
BAT	BNT
BNT	DATA
DATA	BNT
BNT	ENJ
ENJ	BNT
BNT	ETH
ETH	BNT
BNT	FXC
FXC	BNT

MakerDAO	
From:	**To:**
DAI	SAI
SAI	DAI

TABLE IV: List of the supported DeFi actions of DEFIPOSER.

Action a_1: Converts x_1 amount of c_1 to c_2, using a constant product market (cf. Section II-A), with liquidity $L1^{c_1}$ for c_1 and $L1^{c_2}$ for c_2 (cf. Equation 14).

$$\text{output amount of } c_2 = L1^{c_2} - \frac{L1^{c_1} L1^{c_2}}{(L1^{c_1} + x_1)} \tag{14}$$

Action a_2: Converts x_2 amount of c_2 back to c_1, using another constant product market, with liquidity $L2^{c_1}$ and $L2^{c_2}$. Based on Heuristic 5 (cf. Section V-B), action a_2 must use another market, because otherwise the conversion becomes a reversing action of a_1, which would result in a zero-sum game with a loss on transaction fees.

Initial state encoding: Equation 15 encodes the state variables with concrete values, which are fetched from the considered blockchain state (e.g., the most recent block). This predicate can also be viewed as the assignment of an initial state.

$$\begin{aligned} &\text{predicate } t_1(\cdot) := \\ &\quad \mathcal{B}_0^{\mathbb{T}}(c_1) = \text{Trader's initial } c_1 \text{ balance} \wedge \\ &\quad \mathcal{B}_0^{\mathbb{T}}(c_2) = \text{Trader's initial } c_2 \text{ balance} \wedge \\ &\quad L1_0^{c_1} = \text{Market 1 initial } c_1 \text{ balance} \wedge \\ &\quad L1_0^{c_2} = \text{Market 1 initial } c_2 \text{ balance} \wedge \\ &\quad L2_0^{c_1} = \text{Market 2 initial } c_1 \text{ balance} \wedge \\ &\quad L2_0^{c_2} = \text{Market 2 initial } c_2 \text{ balance} \end{aligned} \tag{15}$$

Action encoding: The following two predicates encode the

two state transition actions. Equation 16 encodes $\mathcal{F}(s_0, a_1, x_1)$ and Equation 17 encodes $\mathcal{F}(\mathcal{F}(s_0, a_1, x_1), a_2, x_2)$. Simply speaking, predicate t_2 transacts cryptocurrency asset c_1 to c_2, and predicate t_3 converts c_2 back to c_1.

$$
\begin{aligned}
&\text{predicate } t_2(\cdot) := \\
&\quad 0 \leq x_1 \leq \mathcal{B}_0^{\mathbb{T}}(c_1) \wedge \\
&\quad \mathcal{B}_1^{\mathbb{T}}(c_1) = \mathcal{B}_0^{\mathbb{T}}(c_1) - x_1 \wedge \\
&\quad \mathcal{B}_1^{\mathbb{T}}(c_2) = \mathcal{B}_0^{\mathbb{T}}(c_2) + L1_0^{c_2} - \frac{L1_0^{c_1} L1_0^{c_2}}{(L1_0^{c_1} + x_1)} \wedge \\
&\quad L1_1^{c_1} = L1_0^{c_1} + x_1 \wedge \\
&\quad L1_1^{c_2} = \frac{L1_0^{c_1} L1_0^{c_2}}{(L1_0^{c_1} + x_1)} \wedge \\
&\quad L2_1^{c_1} = L2_0^{c_1} \wedge \\
&\quad L2_1^{c_2} = L2_0^{c_2}
\end{aligned}
\tag{16}
$$

$$
\begin{aligned}
&\text{predicate } t_3(\cdot) := \\
&\quad 0 \leq x_2 \leq \mathcal{B}_1^{\mathbb{T}}(c_2) \wedge \\
&\quad \mathcal{B}_2^{\mathbb{T}}(c_1) = \mathcal{B}_1^{\mathbb{T}}(c_1) + L2_1^{c_1} - \frac{L2_1^{c_1} L2_1^{c_2}}{(L2_1^{c_2} + x_2)} \wedge \\
&\quad \mathcal{B}_2^{\mathbb{T}}(c_2) = \mathcal{B}_1^{\mathbb{T}}(c_2) - x_2 \wedge \\
&\quad L1_2^{c_1} = L1_1^{c_1} \wedge \\
&\quad L1_2^{c_2} = L1_1^{c_2} \wedge \\
&\quad L2_2^{c_1} = \frac{L2_1^{c_1} L2_1^{c_2}}{(L2_1^{c_2} + x_2)} \wedge \\
&\quad L2_2^{c_2} = L2_1^{c_2} + x_2
\end{aligned}
\tag{17}
$$

Objective encoding:

We use Z to denote the targeted adversarial revenue. Equation 18 encodes the objective constraints, ensuring that the adversarial cryptocurrency asset portfolio increases in value. Note that we rely on search algorithms (cf. Algorithm 2) to find the highest possible Z. The optimization process requires solving the same SMT problem with different concrete initialization of revenue targets Z (predicate t_4).

$$
\begin{aligned}
&\text{predicate } t_4(\cdot) := \\
&\quad \mathcal{B}_0^{\mathbb{T}}(c_1) >= \mathcal{B}_2^{\mathbb{T}}(c_1) + Z \wedge \\
&\quad \mathcal{B}_0^{\mathbb{T}}(c_2) = \mathcal{B}_2^{\mathbb{T}}(c_2)
\end{aligned}
\tag{18}
$$

Free variables and range: Our model only consists of two free variables (x_1, x_2) for the simple two action paths. For a path of arbitrary length n, the corresponding SMT system consists of n free variables, which are the parameters of each action. As shown in predicate t_2 (cf. Equation 16) and t_3 (cf. Equation 17), the range of free variables are constraint by the amount of $\mathbb{T}$'s cryptocurrency assets.

SMT problem: By following the above procedures, the state transition model we presented in Section III is now encoded as an SMT problem, where we verify if any initialization of the free variables (x_1, x_2) satisfies the requirement of $t_1(\cdot) \wedge t_2(\cdot) \wedge t_3(\cdot) \wedge t_4(\cdot)$.

Number of paths SMT must solve	Number of blocks	Percentage of blocks
0-23	0	0%
24	204,901	21.57%
46	609	0.06%
47	12,201	1.28%
48	57,265	6.03%
50-100	35,771	3.77%
>100	3,897	0.41%
total	314,644	33.12%

TABLE V: After we apply the dependency-based blockchain state reduction we show in this Table the number of paths the SMT solver must solve. 32.71% of the blockchain blocks between $9,100,000$ and $10,050,000$ have less than 100 "state changing" paths, allowing to reduce the SMT computation.

Contract	Count	State change frequency
Uniswap DAI	28,464	27.01%
Bancor ETH	16,466	15.63%
Uniswap UBT	13,623	12.93%
Uniswap MKR	5,984	5.68%
Uniswap SAI	5,195	4.93%
Uniswap BAT	5,090	4.83%
Uniswap KNC	4,141	3.93%
Uniswap DATA	3,546	3.36%
Bancor DATA	2,309	2.19%
Uniswap SNT	2,300	2.18%
Uniswap ANT	1,759	1.67%
Bancor UBT	1,714	1.63%
Bancor ENJ	1,602	1.52%
Uniswap ENJ	1,337	1.27%
Uniswap MANA	1,129	1.07%
Uniswap RLC	1,073	1.02%
Other	9,650	9.16%

TABLE VI: State pruning statistics, showing that the Uniswap DAI contract experiences the highest state change frequency (27.01% of blocks).

APPENDIX D
Z3 PATH PRUNING

Table VI illustrates the state change frequency of the top 15 most frequently changed DeFi markets we consider in this work. The Uniswap DAI market is significantly more active than the other markets, with a state change frequency of 27.01% of the blocks, while the majority (78.72%) of markets experience a frequency below 2% of the blockchain blocks. Note that every market is only involved in a subset of the 600 kept strategies after pruning. For example, only 48 out of the 600 strategies involve the Uniswap DAI market.

APPENDIX E
CONCRETE ENCODING EXAMPLE FOR Z3

In this section, we provide a running example to demonstrate the encoding process of DEFIPOSER-SMT. The example performs an arbitrage at block $9,680,000$, which first converts ETH to BNT on Bancor and then converts BNT back to ETH on Uniswap.

A. Initial state encoding

The initial state encoding consists of the predicates for both the trader $\mathbb{T}$'s initial balances, as well as the initial states of the underlying platforms.

```
# Trader's initial state.
# We assume the trader holds 1000 ETH at the start.
S0_Attacker[BNT] == 0,
S0_Attacker[ETH] == 1000000000000000000000,
```

```
# Initial states of the underlying platforms.
S0_Uniswap[BNT]_eth == 135368255883939133529,
S0_Uniswap[BNT]_erc20 == 108143877658121296155075,
S0_Bancor[ETH]_erc20 == 109365919812787198371 25,
S0_Bancor[ETH]_erc20_ratio == 500000,
S0_Bancor[ETH]_bnt == 87922490126686956788248921,
S0_Bancor[ETH]_bnt_ratio == 500000,
S0_Bancor[ETH]_fee == 1000,
```

B. Action encoding

We encode the two transition actions as predicates. $P1$ is the input parameter for action 1 (converts ETH to BNT on Bancor), and $P2$ is the input parameter for action 2 (converts BNT to ETH on Uniswap).

```
# converts ETH to BNT on Bancor
P1 > 0,
S1_Bancor[ETH]_bnt > 0,
S1_Attacker[BNT] ==
    S0_Attacker[BNT] +
    (S0_Bancor[ETH]_bnt*
    (1 -
    (S0_Bancor[ETH]_erc20/(S0_Bancor[ETH]_erc20 + P1))**
    (S0_Bancor[ETH]_erc20_ratio/S0_Bancor[ETH]_bnt_ratio))*
    (1000000 - S0_Bancor[ETH]_fee)**2)/
    1000000000000,
S1_Attacker[ETH] == S0_Attacker[ETH] - P1,
S1_Uniswap[BNT]_eth == S0_Uniswap[BNT]_eth,
S1_Uniswap[BNT]_erc20 == S0_Uniswap[BNT]_erc20,
S1_Bancor[ETH]_bnt ==
    S0_Bancor[ETH]_bnt -
    (S0_Bancor[ETH]_bnt*
    (1 -
    (S0_Bancor[ETH]_erc20/(S0_Bancor[ETH]_erc20 + P1))**
    (S0_Bancor[ETH]_erc20_ratio/S0_Bancor[ETH]_bnt_ratio))*
    (1000000 - S0_Bancor[ETH]_fee)**2)/
    1000000000000,
S1_Bancor[ETH]_bnt_ratio == S0_Bancor[ETH]_bnt_ratio,
S1_Bancor[ETH]_erc20_ratio == S0_Bancor[ETH]_erc20_ratio,
S1_Bancor[ETH]_erc20 == S0_Bancor[ETH]_erc20 + P1,
S1_Bancor[ETH]_fee == S0_Bancor[ETH]_fee,
```

```
# converts BNT to ETH on Uniswap
S1_Attacker[BNT] >= P2,
P2 > 0,
S2_Attacker[BNT] == S1_Attacker[BNT] - P2,
S2_Attacker[ETH] ==
    S1_Attacker[ETH] +
    (997*P2*S1_Uniswap[BNT]_eth)/
    (S1_Uniswap[BNT]_erc20*1000 + 997*P2),
S2_Uniswap[BNT]_eth ==
    S1_Uniswap[BNT]_eth -
    (997*P2*S1_Uniswap[BNT]_eth)/
    (S1_Uniswap[BNT]_erc20*1000 + 997*P2),
S2_Uniswap[BNT]_erc20 == S1_Uniswap[BNT]_erc20 + P2,
S2_Bancor[ETH]_bnt == S1_Bancor[ETH]_bnt,
S2_Bancor[ETH]_bnt_ratio == S1_Bancor[ETH]_bnt_ratio,
S2_Bancor[ETH]_erc20_ratio == S1_Bancor[ETH]_erc20_ratio,
S2_Bancor[ETH]_erc20 == S1_Bancor[ETH]_erc20,
S2_Bancor[ETH]_fee == S1_Bancor[ETH]_fee,
```

C. Objective encoding

In this example, we check if it is possible for the trader $\mathbb{T}$ to realize 1 ETH of revenue following this path.

```
# Objective encoding
S2_Attacker[BNT] == 0,
S2_Attacker[ETH] >= 1001000000000000000000
```

Appendix F
Optimizer for the SMT solver

Algorithm 3 shows how the SMT solver can maximize a path's revenue using binary search.

Algorithm 3: Maximize a path's revenue using SMT solver and binary search.

Input:
$p \leftarrow$ Path
$m \leftarrow$ Minimum revenue target
Output: Optimized revenue r
if $\neg$ *isSAT(p, m)* **then**
  return *0*
else
  l $\leftarrow m$
  u $\leftarrow m \times 10$
end
while *isSAT(p, u)* **do**
  l $\leftarrow$ u
  u $\leftarrow$ u $\times$ 10
end
return *binarySearch(p, l, u)*

Function *isSAT(p, r) : bool* **is**
  return *(Is the path p SAT for the revenue r)*
end

Function *binarySearch(p, l, u) : float* **is**
  Binary search SAT solution on path *p*, using the lower bound *l* and upper bound *u* on revenue
  return *(Maximum SAT revenue)*
end

Appendix G
State Dependency

We visualize the state changes in Figure 15. This figure provides an intuition to a trader on how active a particular market is. An asset changes state if a market listing that asset changes state (i.e., a trader trades the asset). ETH experiences the most state changes with over $950,000$ blocks (36.76%). After ETH, we observe that DAI (14.62%) experiences the most frequent state changes over the $950,000$ blocks we crawled. POA20 has the lowest number of state changes (0.08%). For a trader who is not able to position its transactions first in a block, the market activity is relevant because a strategy executed on the POA20 asset has a higher likelihood to succeed than on an active DAI market.

Appendix H
bZx

Figure 16 shows our attack window analysis of the bZx attack using DeFiPoser-SMT.

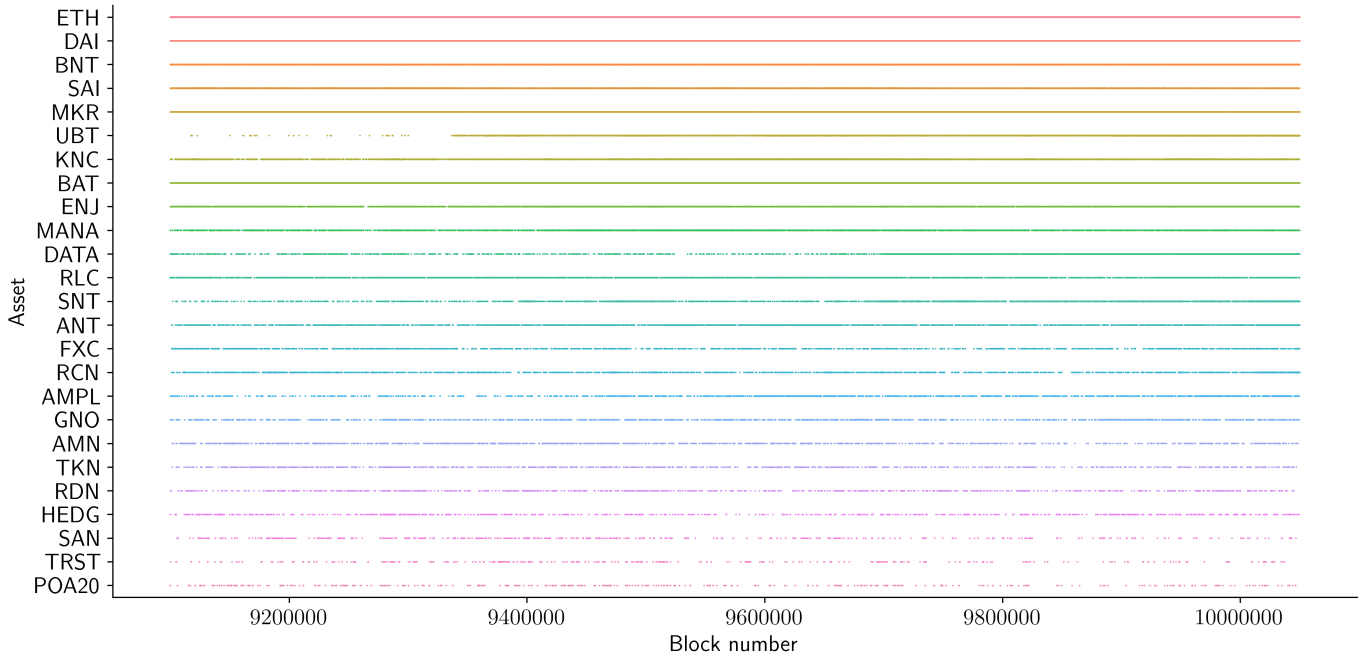


Fig. 15: Timeline analysis of the state changes, over 150 days (950, 000 blocks), where every state change is represented with a colored tick.

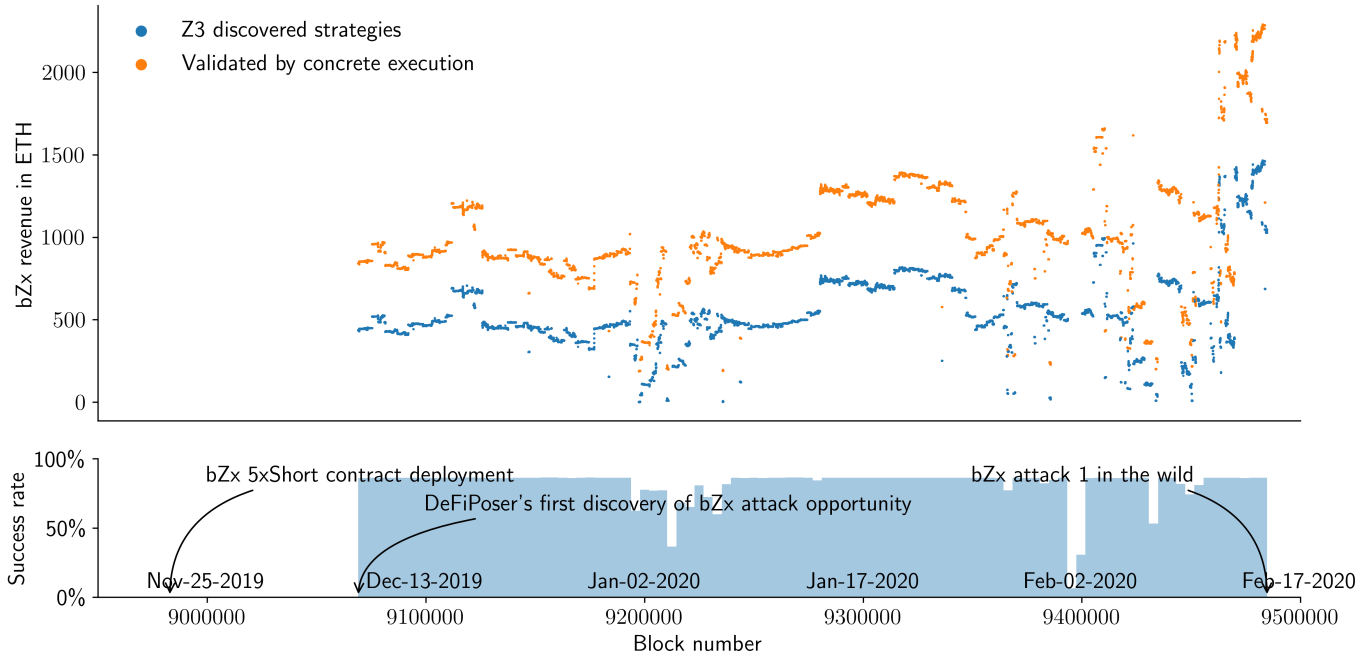


Fig. 16: Attack window analysis of the bZx attack. DeFiPoser-SMT finds the first attack opportunity at block 9, 069, 000 (December 8th 2019). The opportunity lasted for 69 days, until the opportunity was exploited in block 9, 484, 687 (February 15th 2020). We visualize the difference between the profits from Z3 and concrete validation, along with the success rate (using block bin sizes of 100) of a Z3 strategy passing concrete validation. Note that the bZx loan interest rate formula is conservatively simplified in the encoding process, which explains why the Z3 anticipated revenue is lower than the concrete execution yield.