Towards Satisfactory Web3 Software Engineering

Applying lightweight formal methods and SAT solvers to build better blockchain applications (position talk)

Jan Gorzny, Ph.D.

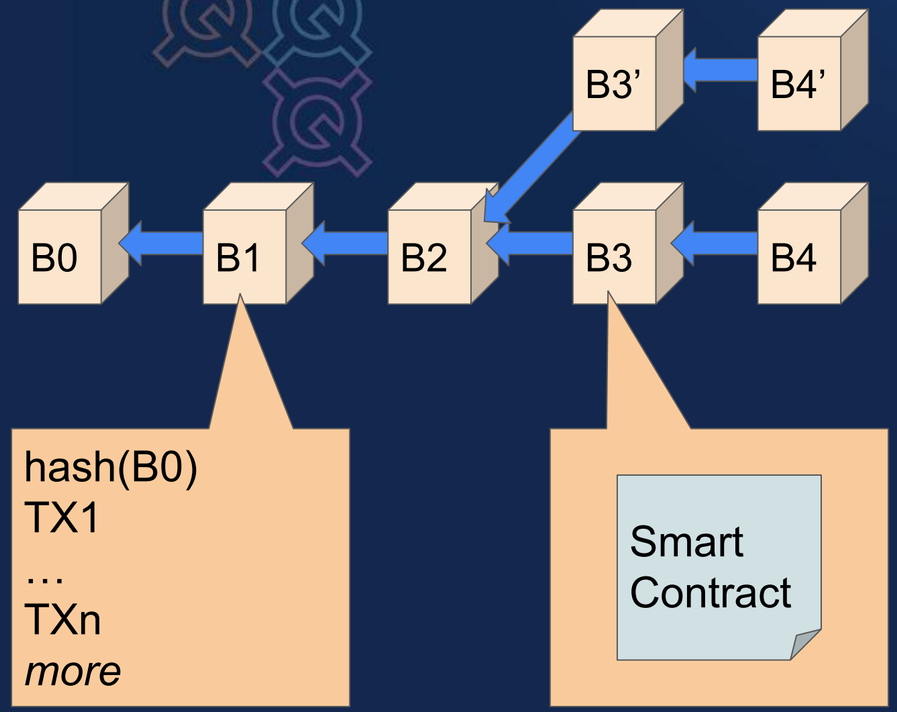
14th Pragmatics of SAT International Workshop 2023, Italy

Blockchains & Web 3

Download these slides using this link!
(Google slides)

Blockchains

- Append-only **distributed ledger**
  - Users interact via transactions
- Blocks record which transactions are included/processed
  - Blocks are determined by some consensus algorithm (e.g., Proof-of-Work or Proof-of-Stake)
- Transactions can invoke **smart contracts**



dApps & Smart Contracts

- Smart contracts form **decentralized applications (dApps)**
  - Only “back-end” is on-chain
- Smart contracts are **immutable**
  - But some tricks exist
- Computation is **metered**
- Language is often novel (e.g. Solidity)

Smart Contract Example

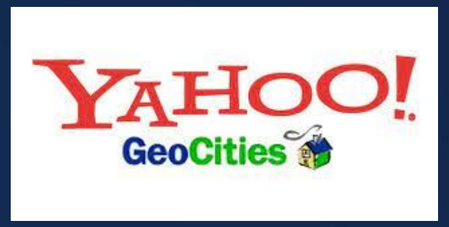
```
// contracts/GLDToken.sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.0;
import
"@openzeppelin/contracts/token/ERC20/ER
C20.sol";

contract GLDToken is ERC20 {
    constructor(
        uint256 initialSupply
    ) public ERC20("Gold", "GLD") {
        _mint(msg.sender,initialSupply);
    }
}
```

Web3

Web 1
1984-2004

Web 2
2004-now?

Web 3
now-future?



Web3 Failures

DealBook / Business & Policy

A Hacking of More Than $50 Million Dashes Hopes in the World of Virtual Currency

Technology

Axie Infinity's Ronin Network Suffers $625M Exploit

It may be the largest exploit in DeFi history.

TECH / SECURITY / POLICY

Wormhole cryptocurrency platform hacked for $325 million after error on GitHub

TECH · CRYPTOCURRENCY

Hackers stole $182 million in crypto. Here's how Beanstalk Farms is hoping to stay afloat

BY CHRIS MORRIS

April 19, 2022 at 11:51 AM EDT

See also

- Atzei et al. 2016
- Lee et al. 2022

Formal Methods & Web3

- Use of **theorem provers**, **bounded model checkers**, **symbolic execution**, **static analysis**, and **linters** to improve code quality was presented at major conferences (e.g. DevCon 2022, DevCon 2019)

- Applied to platform level concepts like consensus algorithms (e.g., Verma et al. 2020, Tholoniat and Gramoli 2022) and dApp code (e.g. Park et al. 2018, Bhargavan et al. 2016)

7

- Still **difficult** and **expensive**
  - Decentralized Autonomous Organizations (DAOs) and startups can't afford the time and expertise

Lightweight Formal Methods

- To reduce costs, there is a community focus on so-called “lightweight formal methods” that:
  - **sacrifice full verification for usability**
  - often only use tools which are **powered by state-of-the-art SAT solvers** to find bugs in code
  - often available as push-button tools that **focus on the detection of well-known classes of bugs**

- Exploit the limited nature of blockchain execution (recall, **metered** execution), which restricts the state space for dApps

- Tools that can often prevent the more egregious errors from being repeated on blockchains now exist and **are being used!**

Position

Lightweight formal methods are critical to web3 development.

- The blockchain community **has already embraced** some lightweight formal methods
- Lightweight formal methods **fit with the ethos** of both independent and institutional Web3 developers
- There are (SAT-based) techniques that **are not yet utilized** that could have prevented errors in real blockchain systems
- **There are more web3 domains** which are fit for the next phase of lightweight formal methods

Lightweight Formal Methods

Lightweight Formal Methods

Lightweight Formal Methods (Jackson, 2012):

> *"[software] models are developed incrementally, driven by the modelers perception of which aspects of the software matter most, and of where the greatest risks lie, and automated tools are exploited to find flaws as early as possible."*

Does not discourage powerful tooling, but instead emphasizes that its use be focused on particular areas of concern.

Lightweight Formal Methods

- Lightweight formal methods may not be sufficient to make claims about the entire system, but are **useful for detecting local bugs** and **confirming particular behaviors** of a system

- These methods may view correctness of the system as the satisfaction of several properties, some or all of which can be **modeled**, **automatically checked**, and used to **guide implementations**
  - Can be used to **rapidly prototype systems at the specification level**, saving development time

- These techniques trade coverage and total confidence for usability but often do not take as long to employ.
  - These techniques are also **less costly** than “full” formal methods
  - Are often **more accessible** to average developers, reducing the need for expensive in-house experts or consultants

Lightweight Formal Methods & Web 3

Blockchain developers have embraced lightweight formal methods in the form of **symbolic execution** for dApp source code

- They often pay special attention to unintended **reentrancy** (*a function is reentrant if it can be re-entered before its initial execution finishes*)

Tools (di Angelo et al. 2019 provide a good survey of 27 tools):

- EthBMC (Frank et al. 2020)
- Manticore (Mossberg et al. 2019)
- Oyente (Luu et al. 2016)
- Slither (Feist et al. 2019)

SAT & Web3

- Under the hood, these tools employ SAT solvers like **Z3** and **CVC5** to find bugs
- These approaches are **powerful** and **general**
  - **Metering** of execution and storage helps!
- Fall short of showing program correctness

SAT: Fit for Web3

- These tools are a **perfect fit** for blockchain developers
  - These tools are valuable: symbolic execution can **catch more bugs than testing alone**
  - These tools are simple: they **do not require a substantial investment** from the development team. Easily accessible by DAOs, start-ups, and institutions

- These tools are **integrated and developed at hackathons**, improving their capabilities and accessibility
  - These **decentralized improvements** to these tools are necessary to keep up with the growing complexity of dApps and Solidity itself

Building More Secure Systems through Lightweight Formal Methods with SAT solvers

The Future of SAT in Web3

- Push-button analyzers are **not silver bullet solutions** to improve dApp code quality. dApp developers also rely on
  - **Tests** that help ensure that at least individual functions and “happy path” scenarios work as intended
  - **Code audits** (which include manual code review); seen as a prerequisite to deployment (due to **immutability**)

- Audits are the **last line of defense**[17] against incorrect code
  - Typically, audits are solicited only **near the end** of the project’s development

Can Web3 developers do better?

The Future of SAT in Web3

Can web3 developers do better?

In some cases, **Yes!**

- 15-line Ethereum token? ❌
- Cross-chain protocols (a.k.a. “bridges”), side-chains, rollups, and DeFi? ✅

Check correctness at the **specification level**

- **Feature interactions** study; SAT solvers can find these (e.g. Tsuchiya et al 2002)
- **Software product lines** and software synthesis (e.g. Xiang et al. 2018)

Specification-level lightweight formal methods can also **overcome scaling** issues of applying SAT solvers directly to code

- These artifacts can guide test cases and be shared with code auditors

The Future of SAT in Web3

Can web3 developers do better?

In some cases, **yes**!

SAT based **automated test generation** (e.g. Yan and Zhang 2006)

- Testing is generally used well in Web3, but not always!
- Integration into various development frameworks possible

The Future of SAT in Web3

Can SAT do better in Web3?

SAT by far is most used in verification of code on contracts as part of those tools mentioned earlier.

Performance may be improved by answering some of these questions:

- How do you model the system correctly? Can it be improved?
- How can bitwise operations be handled?
- How can formulas be modelled?
- How can bytecode be parsed & represented?
- Which bugs can be found? How many transactions can be modelled?

The Future of SAT in Web3

- Where can SAT solvers be applied as part of other tools?
- Can you add SAT to dynamic analysis? Is that a meaningful question?
- Can you apply SAT to the cryptographic functions which are typically only modelled as a black-box?
- How can we explore larger state spaces efficiently? What is too large?
- What domain-specific benchmarks exist? Are they updated?
- On what ecosystems do we have tooling?

and the list goes on...

The Future of SAT in Web3

Zero-knowledge proofs (in Web3)

- Translate functions to system of equations over a finite field that are satisfiable iff the function is computed correctly

Massive amount of polynomials that cannot be verified by hand (2^19 degree polynomials, and about that many constraints).

We can use SAT (SMT) to analyse these systems for some properties (e.g., **unique satisfiability**).

Requires **new theories**: finite field solver for SMT (e.g. Ozdemir et al. 2023)

- Other ways to do this? Other implementations, or other solvers?
- Symmetry breaking? Domain specific tricks?

Requires **new theories**: finite field solver for SMT (e.g. Ozdemir et al. 2023)

- Also more clever encodings than these?

Advice A_0	Advice A_1	Advice A_2	Fixed F_0	Fixed F_1	Fixed F_2
⋮	⋮	⋮	000	000	000
⋮	⋮	⋮	000	001	001
$a_{0,i}$	$a_{1,i}$	$a_{2,i}$	⋮	⋮	⋮
⋮	⋮	⋮	111	111	000

Table 3
A lookup in a Halo2 circuit table with constraints on the fixed columns (only the relevant columns are shown). In this case, the entry of F_2 at row i is the bit-wise exclusive or function (XOR) applied to the values of row i in F_0 and F_1. In this case, we want the i row of the advice columns A_0, A_1, and A_2 to be a valid XOR for entries which have three bits. We add the following constraint to the solver: $(a_{0,i} = 000 \wedge a_{1,i} = 000 \wedge a_{2,i} = 000) \vee (a_{0,i} = 000 \wedge a_{1,i} = 001 \wedge a_{2,i} = 001) \vee \ldots \vee (a_{0,i} = 111 \wedge a_{1,i} = 111 \wedge a_{2,i} = 000)$.

Conclusion

- The blockchain community **has already embraced** lots of SAT tools!
- Lightweight formal methods **are accessible**
- SAT **should be applied to more areas** of software engineering
- **Technical challenges** are also present for these tools themselves

Thank you for listening!


@jgorzny


@jgorzny


jan@quantstamp.com


@quantstamp

Download these slides using this link! (Google slides)

See my SMT 2023 talk in Rome for analyzing Halo2 circuits tomorrow!