BitML for Zenotta

BitML (after "Bitcoin Modelling Language") is both a domain-specific language (DSL) for writing Bitcoin smart contracts and a process calculus with its syntax and semantics. Despite the high level of abstraction, BitML can express most of the contracts proposed so far in the literature. Additionally, since it abstracts from the low-level details of the target blockchain, BitML can be useful for other UTXO-based blockchains like Zenotta.
Examples of smart contracts that can be expressed in BitML include:

- **Direct payments**. User A sends a payment to user B.
- **Multi-sender payments**. Two or more users $A_1, A_2, ..$ join their funds and send a payment to user B.
- **Multi-receiver payments**. User A splits her funds and sends a payment to multiple receivers $B_1, B_2, ...$
- **Authorized payments**. User A sends a payment to user B, provided that another user C gives his authorization.
- **Future payments**. User A sends a payment to user B, who will be able to redeem the funds only after a certain time t in the future.
- **Commitments**. User A commits to a secret s by making her funds redeemable only if she reveals s.

Clever combinations of the simple examples above can be used to create more complex contracts such as escrows and timed commitments.

1 Life cycle of a contract

The life cycle of a contract can be divided into five main phases, which we outline below.

1.1 Design

Any user A can write a contract $\mathcal{C}$ in BitML. To accomplish this, A can use the BitML embedding in Racket, which allows for programming within the DrRacket IDE.

1.2 Distribution

User A distributes $\mathcal{C}$, for example by publishing its source code on a website. Anyone wishing to join the contract must share their public keys, deposits, and commitments with the other participants.

1.3 Formal verification

Any user B can inspect the source code of $\mathcal{C}$ looking for bugs. Moreover, B can use a formal model checker to verify security properties like *liquidity* (i.e., funds never remain frozen in the contract) and other contract-dependent properties. For example, B can utilize the strategy-aware model checker within Maude, which is a model-checking framework based on rewriting logic. To do this, it is necessary to define the syntax and semantics of BitML in Maude. After that, $\mathcal{C}$ must be translated from BitML to Maude. Finally, the model checker takes as input the contract, the properties, and the strategies of the participants, and outputs the results of its formal analysis.

1.4 Compiling

User B can independently compile the source code of $\mathcal{C}$ using the BitML compiler. First, the compiler translates $\mathcal{C}$ into Balzac, which is an abstraction layer over Bitcoin transactions. Then, it translates Balzac transactions into standard Bitcoin transactions.
A crucial result regarding the BitML compiler is a theorem that states its computational soundness. In essence, we are guaranteed that attacks at the computational level are observable at the symbolic level too. In other words, if a security property holds for a contract written in BitML, the same property will hold for the compiled contract as well.
However, if a contract written in BitML is insecure and we are not able to detect its vulnerabilities, the compiled contract will inherit the same vulnerabilities. Moreover, while we are guaranteed that the compiler preserves yes/no properties, there is no proof that the same holds for properties involving probabilities. For example, while we can be sure that a liquid lottery will remain liquid, a theoretical justification that a fair lottery will remain fair seems out of reach for the moment.

1.5 Execution

Any participant P can broadcast the initial transaction T_{init} of the compiled contract to the network. Once T_{init} is published on the blockchain, $\mathcal{C}$ is considered stipulated. At this point, participants can start interacting with $\mathcal{C}$ by publishing admissible transactions, which in turn redeem other previously published admissible transactions. The compiler guarantees a one-to-one correspondence between admissible transactions and valid actions at the symbolic level.

2 BitML extensions

BitML can be enriched with new primitives that allow designing contracts that were not possible before. When this is done, it is essential to specify an extended version of the compiler and provide proof that this extended version preserves its computational soundness.

2.1 Renegotiation and recursion

One of the limitations of the original BitML is that, once a contract has been stipulated, it is not possible to renegotiate its terms. This prevents expressing common financial contracts, where participants need to add funds at run-time. To address this limitation, BitML was extended with a new primitive for contract renegotiation. Examples of contracts that can be expressed with this new primitive include periodic payments and zero-coupon bonds.
The renegotiation primitive allows participants of a contract $\mathcal{C}$ to renegotiate its terms, provided that all participants agree. For example, they can decide to transfer the balance of $\mathcal{C}$ to another contract $\mathcal{D}$, which might implement a different logic and require additional deposits and commitments.
Recursion is obtained when $\mathcal{D} = \mathcal{C}$, i.e. participants agree to run $\mathcal{C}$ again. However, this form of recursion is weak: it is enough that one participant does not cooperate and the recursion is stuck. To obtain a stronger form of recursion one can use covenants, which we describe next.

2.2 Covenants

Covenants are primitives that extend the Bitcoin scripting language, allowing transactions to constrain the scripts of the redeeming ones. They have recently received a lot of attention due to the BIP-119. A non-fungible token is an example of a contract that can be designed with standard covenants, whereas a fungible token would require more powerful covenants that can also inspect sibling and parent transactions.
BitML can be extended with new primitives to express contracts involving covenants. However, it would not be possible to compile those contracts to standard Bitcoin transactions, as Bitcoin does not currently support covenants. Nonetheless, we could use BitML to design contracts to be executed on covenant-aware blockchains like Zenotta.
We highlight that increasing the expressive power of a language does not come without consequences. Specifically, BitML extended with covenants is Turing-complete, since it can simulate a counter machine. As a result, the formal verification of contracts is harder than in the original BitML, as contracts can now encode infinite-state transition systems. Hence, model-checking techniques based on the exploration of the whole state space cannot be applied for certain classes of contracts. However, this is a very controlled way of achieving Turing completeness compared to other approaches.

References

[Atz+18] Nicola Atzei et al. “A formal model of Bitcoin transactions”. In: *International Conference on Financial Cryptography and Data Security*. Springer. 2018, pp. 541–560.

[BCZ18] Massimo Bartoletti, Tiziana Cimoli, and Roberto Zunino. “Fun with Bitcoin smart contracts”. In: *International Symposium on Leveraging Applications of Formal Methods*. Springer. 2018, pp. 432–449.

[BZ18] Massimo Bartoletti and Roberto Zunino. “BitML: a calculus for Bitcoin smart contracts”. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2018, pp. 83–100.

[Atz+19] Nicola Atzei et al. “Developing secure Bitcoin contracts with BitML”. In: *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2019, pp. 1124–1128.

[BZ19] Massimo Bartoletti and Roberto Zunino. “Verifying liquidity of Bitcoin contracts”. In: *International Conference on Principles of Security and Trust*. Springer. 2019, pp. 222–247.

[BLZ20] Massimo Bartoletti, Stefano Lande, and Roberto Zunino. “Bitcoin covenants unchained”. In: *International Symposium on Leveraging Applications of Formal Methods*. Springer. 2020, pp. 25–42.

[BMZ20] Massimo Bartoletti, Maurizio Murgia, and Roberto Zunino. “Renegotiation and recursion in Bitcoin contracts”. In: *International Conference on Coordination Languages and Models*. Springer. 2020, pp. 261–278.

[BLZ21] Massimo Bartoletti, Stefano Lande, and Roberto Zunino. “Computationally sound Bitcoin tokens”. In: *2021 IEEE 34th Computer Security Foundations Symposium (CSF)*. IEEE. 2021, pp. 1–15.