Towards a B-Method Framework for Smart Contract Verification: The Case of ACTUS Financial Contracts

Zakaryae Boudi[1][0000-0002-0813-6434] and Toub Mohamed[1][0000-0002-4996-5351]

FeverTokens, 55 rue de la Boétie, 75008, Paris, France
{boudi,toub}@fevertokens.io
https://www.fevertokens.io

Abstract. The increasing use of advanced smart contract structures in finance necessitates rigor and scalability in ensuring their correctness. Traditional auditing methods fall short in providing comprehensive security, but formal verification offers a robust and scalable approach to constructing secure-by-design models and implementing smart contracts. In this paper, we introduce a B-method framework for modeling and verifying smart contracts based on the ACTUS standard for financial instruments. We start by converting ACTUS specifications into B-method constructs to bring a systematic approach to model, analyze, and verify financial contracts' implementations within the blockchain context.

Keywords: B-method · ACTUS standard · Smart Contracts · Formal Verification · Financial Instruments · Blockchain · Security.

1 Introduction

Smart contracts have gained significant attention in recent years as a means of automating transactions and conveying ownership on blockchain platforms. They are digital contracts that self-execute when certain conditions are met, allowing for the automation of complex business processes and the elimination of intermediaries [8, 3]. However, the complexity and potential for errors in smart contract code pose a significant challenge in ensuring their correct functionality [16, 12]. Given the immutable and decentralized nature of blockchain, errors in smart contracts can have dire consequences, such as the loss of funds or inability to access assets.

In response to these challenges, we advocate for the use of formal verification techniques to mathematically substantiate the correctness of smart contracts. These methods enable the mathematical substantiation of a smart contract's correctness [4, 11, 10, 7]. In software engineering, the B method is highly regarded [5], offering a formal specification language alongside tools for analyzing system behaviors and affirming key properties like safety and liveness [14]. Our selection of the B method is predicated on its comprehensive modeling capabilities, robust toolset, an active user community, and its proven track record in developing safety-critical software [13, 1].

This paper presents a novel methodology for the formal verification of smart contracts within the financial domain, specifically those based on the ACTUS standard for financial instruments. The ACTUS standard represents a holistic framework for algorithmically defining financial contracts using uniform data structures and deterministic functions [2]. Our approach includes the application of transformation rules for converting Actus specifications into B-method constructs to enable seamless and comprehensive formal verification process for any corresponding smart contract implementations.

The organization of this paper reflects the step-by-step approach taken to implement the B-method framework for modeling and verifying ACTUS-based financial contracts. We begin by introducing the challenges of scalability and standardization in smart contracts and the rationale behind FeverTokens' open-source Package-Oriented Framework in Section 2. Section 3 provides an overview of the ACTUS standard and its significance in representing financial contracts. Section 4 delves into the preliminary B-method framework for financial smart contracts, demonstrating the process of transforming ACTUS specifications into B-method structures.

2 Smart contracts scalability and the rationale behind the FeverTokens open-source Package-Oriented Framework

In the blockchain sphere, the scalability of smart contracts is a critical factor, especially as their complexity and applicability increase across various sectors. FeverTokens confronts this issue with its pioneering open-source Package-Oriented Framework, aimed at augmenting the scalability and flexibility of smart contracts [9]. This framework adopts a modular, package-oriented architecture, enabling developers to construct smart contracts as individual, updatable modules. This method enhances development efficiency and provides significant adaptability and scalability improvements.

The essence of the FeverTokens framework is its emphasis on functional scalability, allowing smart contracts to seamlessly integrate new functionalities while ensuring the integrity and security of the blockchain. Its open-source model promotes a collaborative development atmosphere, inviting community engagement and contributions, which fosters a robust and varied ecosystem of smart contract modules and encourages continuous innovation.

A key advantage of the framework is its capacity to facilitate advanced integration of established standards, particularly in the realm of financial instruments like ACTUS and Common Domain Model (CDM) by the ISDA [6]. The modular architecture of the framework is strategically designed to support the creation of libraries consisting of packages that correspond to specific standardized financial instruments. This feature is particularly beneficial for builders and institutions engaged in tokenization systems.

By aligning with standards like ACTUS and CDM, the framework ensures that the financial instruments modeled within it are compliant with global financial regulations and practices. This compliance is crucial for institutions looking to leverage blockchain technology for financial applications. The availability of these standardized packages within the framework not only simplifies the development process for builders but also enhances the reliability and interoperability of financial instruments across different platforms and systems. The modular approach, therefore, plays a pivotal role in bridging traditional financial models with the innovative capabilities of blockchain technology, offering a seamless and compliant pathway for the tokenization and management of financial assets.

Leveraging the Diamond standard (ERC-2535) [15], originally introduced by the Ethereum community as a solution to smart contract size limitations, the FeverTokens framework adds a substantial engineering layer to transform this standard into a tool for functional scalability. This enhancement standardizes facet and diamond structures within the framework, facilitating the seamless packaging and integration of modular smart contract components. This structure not only simplifies the management of smart contracts but also enables effective scaling in terms of version control, operational oversight, deployment, and upgrades. FeverTokens is actively developing infrastructure to efficiently manage these packages, addressing the challenges associated with scaling. The nature of this architecture underlines the importance of rigorous security verification processes. Given the increased frequency of updates and the inherent complexity of this system, verification efforts must be multi-dimensional, encompassing individual packages, the overall diamond structure, and governance aspects.

Traditional auditing methods, while reliable, become less feasible and more costly in this expanded scope. Consequently, automated formal verification mechanisms are crucial for ensuring safe-by-design development, enabling the framework to maintain robust security standards despite its scalability and modular complexity.

3 Overview of the Actus standard and financial contracts

The ACTUS (Algorithmic Contract Types Unified Standards) standard is a pivotal development in the digital representation of financial contracts. Financial contracts, essentially legal agreements between parties for the exchange of future cash flows, are defined unambiguously through a set of contractual terms and logic. This clarity allows for their mathematical description and digital representation as machine-readable algorithms. The advent of distributed ledger and blockchain technologies, particularly the use of smart contracts, has opened up novel possibilities for these natively digital financial contracts.

Financial contracts typically follow established cash flow exchange patterns. Examples include bullet loan contracts with fixed principal payments and variable interest payment schedules, and amortizing loans where principal is paid back in portions. The ACTUS standard, through its taxonomy, organizes financial contracts based on their distinct cash flow patterns, covering a wide array of financial instruments like shares, options, swaps, and credit enhancements. The deterministic nature of these financial contracts is key. They define a set

Table 1. Meta-structure of ACTUS financial contracts

Key Actus Element	**Description**
Contract Attributes	These represent the legal terms of a financial contract and define the exchange of cash flows.
State Variables	These describe the state of a contract at a specific point in time. Examples include the outstanding Notional Principal and the applicable Nominal Interest Rate.
Contract Events	These are scheduled or unscheduled events that occur at specific times during the contract's lifetime. They mark points where cash flows are exchanged or where the states of the contract are updated.
State Transition Functions (STFs)	These define the transition of states from a pre-event state to a post-event state when a certain event occurs. STFs are specific to each event type and contract type.
Payoff Functions (POFs)	These define how the cash flow for a certain event is derived from the current states and contract terms. Payoff Functions are specific to each event and contract type.

of rules and conditions under which, given any external variables, the cash flow obligations can be unambiguously determined. For example, in a fixed-rate loan, the obligations are explicitly defined, whereas in a variable rate loan, the rules for rate determination are set in advance, enabling clear derivation of future obligations. This deterministic approach forms the foundation of the ACTUS standard, providing a technology-agnostic, standardized description of financial contracts' cash flow obligations.

The integration of ACTUS with formal methods in the context of smart contracts offers numerous benefits. In the case of ACTUS-based smart contracts, they ensure that the contracts behave as intended, especially crucial given the financial implications and complex interactions involved. By using formal methods, developers can prove the correctness of these contracts against their specifications, thereby reducing the risk of errors and vulnerabilities. This approach is particularly beneficial in financial contexts and will play a crucial role in the safe and effective implementation of ACTUS standards in smart contracts.

4 A preliminary B-method framework for financial smart contracts under the Actus standard

In the ACTUS framework, financial contracts are defined using a meta-structure comprising several key elements: Contract Attributes, State Variables, Contract Events, State Transition Functions, and Payoff Functions (Table 1). The contribution of this paper lies in translating this complex meta-structure into a B-method project structure.

We encapsulate the ACTUS components within the B-method's formal constructs, facilitating not only the rigorous formalization of financial contracts but also ensuring their correctness and reliability through mathematical proof. Each ACTUS component –attributes, state variables, events, STFs, and POFs– is translated into a B-method construct, enabling a systematic and precise development of financial contracts (Figure 1). Figure 1 reflects the mapping of the ACTUS specification onto a B-method structure where the Contract Attributes are implemented in the `contract_head` machine, while utility functions are implemented in separate modules (`utility.mch` and `env.mch`). The `contract_main` contains the key variables and operations that define the core functionality of the contract, with `contract_main_i` implementing the operations, encompassing the state transition and payoff logic.

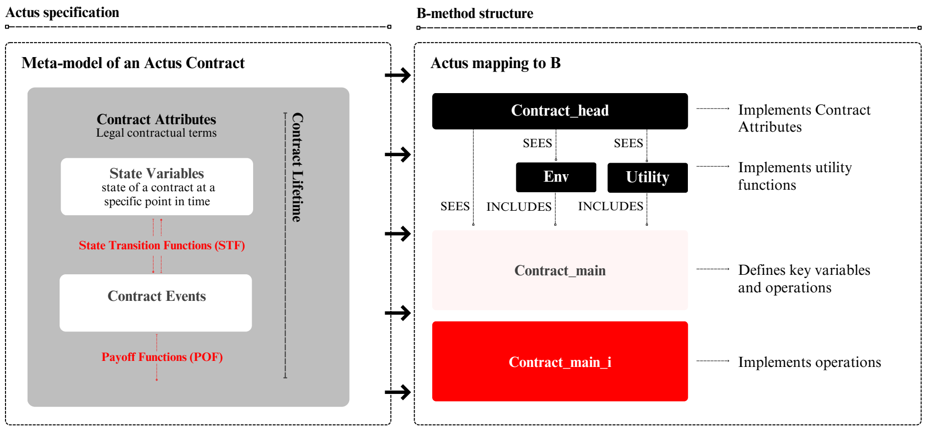

Fig. 1. Actus meta-model mapping to B-method structure.

In the context of formalizing financial contracts using the B-method, we set a head machine that serves as the cornerstone for this contract model, encapsulating the foundational constants and properties that govern the contract's behavior throughout its lifecycle (Figure 2).

The head machine delineates a set of concrete constants –immutable parameters that define the financial contract's structure. The properties of the head machine establish the exact values for these constants, such as a notional amount of 100 units and an interest rate of 5%.

On the one hand, env machine within the B-method model functions as the dynamic context for the contract (Figure 3). It sees the head machine constants, and it manages the stateful aspects of the contract's lifecycle. The OPERATIONS section defines actions that can be performed on the contract's state, such as updating the contract performance, setting payoff and state transition statuses, and progressing the date to simulate the passage of time. On the other hand, the utility machine provides essential computational operations that serve

```
MACHINE
    head
SETS
    Contract_Performance_States = {Performing, Delinquent, Default, Terminated, Matured};
    Event_states = {NA,missed,done}
CONCRETE_CONSTANTS
    NT,
    IPNR,
    MD,
    SD,
    PF_states,
    Year_convention
PROPERTIES
   NT = 100 &
   IPNR = 5 &
   MD = [01,09,2040] &
   SD = [01,09,2009] &
   PF_states = Contract_Performance_States &
   Year_convention = 365
END
```

Fig. 2. PAM example: the head machine.

```
MACHINE
 env
SEES
 head
VARIABLES
 Contract_performance,
 Date,
 IP_event_tracker,
 MD_event_tracker,
 IP_payment_tracker,
 STF_IP_status,
 POF_IP_status
INVARIANT
 Contract_performance : PF_states &
 Date : NATURAL &
 IP_event_tracker : seq(Event_states) &
 MD_event_tracker :seq(Event_states) &
 IP_payment_tracker : seq(seq(NATURAL)) &
 POF_IP_status : BOOL &
 STF_IP_status : BOOL
INITIALISATION
 Contract_performance := Performing ||
 Date := 1 ||
 IP_event_tracker := []||
 MD_event_tracker := [] ||

...
```

```
 IP_payment_tracker := [[0,0,0]] ||
 POF_IP_status := FALSE ||
 STF_IP_status := FALSE
OPERATIONS
 SET_Contract_performance (ii) =
 PRE
 ii : PF_states
 THEN
 Contract_performance := ii
 END;
 SET_POF_IP_status (ii) =
 PRE
 ii : BOOL
 THEN
 POF_IP_status := ii
 END;
date <-- GET_Date =
 BEGIN
 date := Date
 END;
 BULK_SET_IP_event_tracker (ii) =
 PRE
 ii : seq(Event_states)
 THEN
 IP_event_tracker := ii
 END;
...
```

```
SET_IP_event_tracker (ii,date) =
 PRE
 ii : Event_states & date : NATURAL1
 THEN
 IP_event_tracker(date) := ii
 END;
 SET_IP_payment_tracker (ii,date) =
 PRE
 ii : seq(NATURAL) & date : NATURAL1
 THEN
 IP_payment_tracker(date) := ii
 END;
 MONTH_STEP_Eventdate =
 PRE
 Date : NATURAL
 THEN
 Date := Date + 1
 END
END
```

Fig. 3. PAM example: excerpt of the env machine

various other components of the model. It acts as a library of functions that can be invoked to perform specific calculations needed for the contract's processing.

The `main_i` implementation refines the abstract main machine and serves as the executable part of the PAM contract within the B-method framework. It leverages the definitions and constants from the head machine and the dynamic behavior specified in the env machine, as well as the computational functions provided by the utility machine. The `main_i` operations collectively simulate the contract's progression through time, calculate payments, and track events, embodying the dynamic aspects of the contract's execution (Figure 4).

```
IMPLEMENTATION main_i
REFINES main
SEES
 head
IMPORTS
 env, utility
CONCRETE_VARIABLES
 Nt,
 Accr,
 Schedule_AD
INVARIANT
 Nt : INTEGER &
 Accr : INTEGER &
 Schedule_AD : POW(INTEGER)
INITIALISATION
 Nt := NT;
 Accr := 0;
 Schedule_AD := {}
OPERATIONS
 IP_EVENT (state) =
  BEGIN
   VAR date IN
    date <-- GET_Date;
    SET_IP_event_tracker (state,date)
   END
  END;

 ...
```

```
  Init_trackers =
   BEGIN
    VAR md,init_ip_event_tracker IN
     md <-- calculate_months(SD(3), SD(2),MD(3), MD(2));
     init_ip_event_tracker := {ee | ee : NATURAL1 & ee <= md}*{NA};
     BULK_SET_IP_event_tracker (init_ip_event_tracker)
    END
   END;
  Step =
   BEGIN
    VAR date IN
     date <-- GET_Date;
     IF date+1 : Schedule_AD
     THEN
      SET_IP_event_tracker(done,date+1);
      VAR md,dcf,ip IN
       md <-- calculate_months(SD(3), SD(2),MD(3), MD(2));
       dcf <-- calculate_day_count_fraction_ACT (date,md,Year_convention);
       ip := [Nt * IPNR * dcf(1),dcf(2), Nt * IPNR * dcf(3)];
       SET_IP_payment_tracker(ip,date+1)
      END;
      MONTH_STEP_Eventdate
     ELSE
      SET_IP_event_tracker(missed,date+1);
      SET_IP_payment_tracker([0,0,0],date+1);
      MONTH_STEP_Eventdate
     END
    END
   END
END
```

Fig. 4. PAM example: excerpt of the `main_i` implementation

5 Conclusion and Perspectives

This paper has outlined a B-method framework tailored for the verification of smart contracts, with the ACTUS standard serving as a cornerstone for financial instruments. Looking ahead, we envision the application of this framework to the real case of Bond & Swap instruments, inviting both the financial industry and the research community to contribute to the evolution of this framework.

Future work will also include defining an approach for verifying Solidity implementations of ACTUS, exploring the potential for Solidity and bytecode generation directly from B models. This promises a seamless transition from model verification to practical deployment in the blockchain environment. The ultimate goal is to automate and certify the underlying methodology and tooling, ensuring a standard of excellence that facilitates safe-by-design development. By doing so, this work paves the way for robust financial instruments on the blockchain, marked by high levels of security and trust.

Acknowledgments. This research was supported by funding from the Casper Association and Bpifrance. We acknowledge their contributions, which have facilitated the foundational aspects of our study and set the stage for future developments.

References

1. Boudi, Z., Collart-Dutilleul, S., et al.: Safety critical software construction using cpn modeling and b method's proof. In: SESA 2014, Software Engineering and Systems Architecture. p. 4p (2014)
2. Brammertz, W., Mendelowitz, A.I.: From digital currencies to digital finance: the case for a smart financial contract standard. The Journal of Risk Finance **19**(1), 76–92 (2018)
3. Britsiani, N.: Smart contracts: Legal aspects (Sep 2022), https://repository.ihu.edu.gr//xmlui/handle/11544/30033, accepted: 2022-09-30T08:47:30Z
4. Buchs, D., Klikovits, S., Linard, A.: Petri Nets: A Formal Language to Specify and Verify Concurrent Non-Deterministic Event Systems. Foundations of Multi-Paradigm Modelling for Cyber-Physical Systems pp. 177–208 (2020)
5. Butler, M., Körner, P., Krings, S., Lecomte, T., Leuschel, M., Mejia, L.F., Voisin, L.: The first twenty-five years of industrial use of the b-method. In: International Conference on Formal Methods for Industrial Critical Systems. pp. 189–209. Springer (2020)
6. Clack, C.D.: Design discussion on the ISDA Common Domain Model. Journal of Digital Banking **3**(2), 165–187 (2018)
7. Colin, S., Mariano, G.: Coq, l'alpha et l'omega de la preuve pour B? (2009)
8. Cornelius, K.B.: Smart Contracts as Evidence: Trust, Records, and the Future of Decentralized Transactions. In: Hunsinger, J., Allen, M.M., Klastrup, L. (eds.) Second International Handbook of Internet Research, pp. 627–646. Springer Netherlands, Dordrecht (2020)
9. FeverTokens: FeverTokens' open-source Package-Oriented Framework (2023), https://github.com/FeverTokens/ft-package-oriented-framework, accessed: December 2023
10. Hansen, D., Leuschel, M.: Translating B to TLA+ for validation with TLC. Science of Computer Programming **131**, 109–125 (2016)
11. Jensen, K.: Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use. Springer Science & Business Media (1996)
12. Khan, S.N., Loukil, F., Ghedira-Guegan, C., Benkhelifa, E., Bani-Hani, A.: Blockchain smart contracts: Applications, challenges, and future trends. Peer-to-peer Networking and Applications **14**, 2901–2925 (2021)
13. Lahbib, A., Ait Wakrime, A., Laouiti, A., Toumi, K., Martin, S.: An event-B based approach for formal modelling and verification of smart contracts. In: Advanced Information Networking and Applications: Proceedings of the 34th International Conference on Advanced Information Networking and Applications (AINA-2020). pp. 1303–1318. Springer (2020)
14. Treharne, H., Schneider, S.: How to drive a b machine. In: ZB 2000: Formal Specification and Development in Z and B: First International Conference of B and Z Users York, UK, August 29–September 2, 2000 Proceedings 1. pp. 188–208. Springer (2000)
15. Van Vulpen, P., Heijnen, H., Kroon, T., Mens, S., Jansen, S.: Decentralized Autonomous Organization Governance By Upgradeable Diamond Smart Contracts. Available at SSRN 4634762 (2023)
16. Zou, W., Lo, D., Kochhar, P.S., Le, X.B.D., Xia, X., Feng, Y., Chen, Z., Xu, B.: Smart contract development: Challenges and opportunities. IEEE Transactions on Software Engineering **47**(10), 2084–2106 (2019)