BULLETIN of Cherkasy State Technological University

Journal homepage: https://bulletin-chstu.com.ua/en

Vol. 31 No. 3. 2026

UDC UDC 004.415.53
DOI: 10.62660/bcstu/3.2026.38

ISSN: 2306-4412
E-ISSN: 2708-6070

Applying algebraic semantics for smart contracts formalisation and verification

Olha Konnova*

Postgraduate Student
Kherson State University
73000, 27 Universytetska Str., Kherson, Ukraine
https://orcid.org/0000-0001-5590-9527

Abstract. Smart contracts are a key component of blockchain systems, but their immutability after deployment imposes increased requirements for correctness and security. This necessitates the use of formal methods capable of providing mathematically sound verification of contracts at the design stage. The objective of this study was to provide a theoretical substantiation and systematisation of algebraic semantics as a formal foundation for the specification and verification of smart contracts, as well as to demonstrate its capabilities through a comparative analysis with alternative semantic approaches and a practical illustration of its application through the construction and analysis of a formal model. A range of theoretical methods was employed during the work, including a systematic literature review and a comparative analysis of operational, denotational, axiomatic and algebraic semantics. The practical significance of the approach was illustrated through the formalisation of the properties of a crowdfunding contract within the Insertion Modelling System. As a result of the research, a formal algebraic model of a smart contract was constructed, within which the states, transitions and invariants of the system were defined. The feasibility of formalising safety conditions and goal states was demonstrated. Specifically, the reachability of the goal state (successful fundraising) was confirmed and verified through the analysis of a generated execution trace fragment. Furthermore, an example of the successful detection of a deadlock in the contract model was provided. This, in turn, enables the systematic identification of complex behavioural vulnerabilities and the prevention of logical errors during the conceptual design phase. The findings confirm the effectiveness of the algebraic approach to formal verification and demonstrate its suitability for the early detection of logical errors and the enhancement of smart contract reliability

Keywords: blockchain; decentralised applications; formal methods; algebraic specifications; Insertion Modelling System

INTRODUCTION

The development of blockchain technologies has led to the emergence of a new tool for organising digital interactions: smart contracts. Smart contracts are digital agreements that are automatically executed upon the occurrence of predefined conditions. They are written as code and stored on the blockchain, which makes them transparent, secure, and immutable. Modern blockchain systems widely use smart contracts to automate business logic, manage digital assets, and ensure transparent interactions among participants in decentralised systems. Their popularity stems from the ability to automate transactions without intermediaries. However, errors in smart contract code can lead to significant financial losses. The complex logic of such contracts and the impossibility of modifying them after deployment create a high risk of vulnerabilities. Therefore, the correctness of smart contract operation is crucial, as errors in the code can result in financial losses,

Article's History: Received: 13.05.2026; Revised: 24.08.2026; Accepted: 21.09.2026; Published: 07.10.2026

Suggested Citation:
Konnova, O. (2026). Applying algebraic semantics for smart contracts formalisation and verification. *Bulletin of Cherkasy State Technological University*, 31(3), 38-48. doi: 10.62660/bcstu/3.2026.38.

*Corresponding author (okonnova@ksu.ks.ua)

CC BY

disruption of business process logic, and a reduction in platform trust. In this context, the issue of ensuring smart contract reliability and security becomes particularly relevant. Because errors in decentralised systems cannot be fixed with regular updates, traditional testing is often insufficient. Standard methods cannot test every possible scenario, nor can they clearly define how a system will behave. These features necessitate the application of more rigorous and formal approaches that allow for the mathematically proven correctness of contracts at the design stage.

Consequently, formal modelling and verification methods have emerged as a critical solution. These methods enable the construction of precise mathematical models of smart contracts logic, analyse their behaviour, and identify potential vulnerabilities before deployment. According to P. Tolmach *et al*. (2021), formal methods provide mathematical rigor that establishes functional correctness and guarantees safety properties such as deadlock-freedom, atomicity, single-entrancy, and liquidity. The use of such methods is currently considered key to enhancing the security, reliability, and trust in smart contracts. Various formal verification techniques are applied across different stages of smart contract development. Providing a foundational taxonomy, P. Tolmach *et al*. (2021) classify formal modelling into contract-level and program-level approaches. They distinguish between four primary methodologies: theorem proving for infinite-state reasoning using proof assistants like Coq, Isabelle/HOL, and Agda; symbolic execution for detecting vulnerable path patterns in control-flow graphs (CFGs); model checking for exploring finite system states; and program verification, which translates source code into intermediate languages like Boogie or F* to perform deductive reasoning and data-flow analysis.

Addressing practical verification, M. Xiao *et al*. (2024) advocate for an advanced hybrid auditing method that combines static analysis using the ACL2 tool, dynamic simulation via an Ethereum simulator, and formal verification. This formal component employs model checking tools like SPIN and NuSMV, alongside Promela and Linear Temporal Logic (LTL), to systematically verify properties by exploring all possible system states. Similarly applying model checking, R. Xie *et al*. (2024) utilise the PRISM tool to explore finite states and verify probability models specifically for game-theory-based supply chain smart contracts. In the realm of theorem proving, T. Ferariu *et al*. (2025) demonstrate the use of the Agda proof assistant to model state transition systems and formally prove generalised properties, such as Validity, Liquidity, and Fidelity, for UTxO-based smart contracts. Furthermore, R. Davila *et al*. (2025) propose the use of Description Logic (DL) specifically for the design stage, utilising concepts, roles, and individuals to mathematically verify the logical correctness and consistency of a contract's rules.

Much of the recent research prioritises practical vulnerability scanning and tool-assisted verification, while comparatively less attention has been paid to the development of rigorous, compositional, and mathematically uniform semantic frameworks. M. Krichen *et al*. (2022) point out that without unified semantic foundations and precise formal specifications, it is difficult to abstract smart contract behaviour away from low-level implementation details. To address this gap, formal semantic approaches are highly valuable. These approaches allow smart contracts to be represented as strictly mathematical models, ensuring clarity, composability, and the ability to prove correctness across all levels of abstraction. Formal semantics provide a precise and unambiguous definition of the behaviour of program constructs, which is key to understanding and verifying smart contracts. The aim of this article was to systematise and theoretically substantiate algebraic semantics as a framework for the formalisation and verification of smart contracts, achieved through: a comparative analysis of four formal semantic paradigms and their applicability to smart contract specification; and illustrative formal specifications constructed using the Insertion Modelling System (IMS). The proposed approach demonstrated how algebraic semantics can ensure precision and unambiguity in smart contract specification and contribute to improving the reliability and security of decentralised systems.

MATERIALS AND METHODS

This study was based on a set of theoretical research methods used to analyse formal semantic approaches for the specification and verification of smart contracts in blockchain systems. The methodology included a literature review, comparative analysis, and formal modelling using algebraic approach within the IMS. At the first stage, a literature review was conducted to identify modern approaches to smart contract verification. Scientific publications from Scopus and Web of Science databases were used, as well as conference papers in the fields of formal methods and distributed systems. Special attention was given to works representing key semantic approaches: operational semantics (Jiao *et al*., 2020; Liu *et al*., 2024; Xu *et al*., 2025), denotational semantics (Marmsoler & Brucker, 2024), and axiomatic semantics (Dharanikota *et al*., 2021; Alhabardi *et al*., 2022). The selection criteria required that each source either: introduces or extends a formal semantic framework applicable to smart contracts, or demonstrates a concrete tool or case study implementing such a framework on a blockchain platform.

In the second stage, the study used a comparative analysis method. An evaluation structure was created to compare four semantic approaches: operational, denotational, axiomatic, and algebraic. The comparison included several aspects: the core formal framework used in each approach, primary application goal in

the context of smart contracts, the representative verification tools, and typical verification use cases. Each semantic approach was evaluated according to these criteria based on information from the reviewed literature. These specific criteria were selected to establish a clear mapping between theoretical formalisms and their practical relevance, allowing for an objective assessment of how well each paradigm serves smart contract developers and security auditors. This method helped to compare the strengths and limitations of each approach in a structured way.

The next stage involved formal modelling to demonstrate the use of algebraic semantics. To illustrate the practical applicability of the proposed approach, several illustrative specifications of smart contract behaviour were developed using the Insertion Modelling System. IMS is an environment for modelling and analysing the behaviour of distributed, multi-component, and multi-agent systems, enabling the formal specification of agents, environments, and their multi-level interactions (Letichevsky *et al.*, 2016). Within IMS, an agent is represented as a labelled transition system whose states are defined up to bisimulation or trace equivalence. The key characteristic of an agent is its ability to change its state in response to interactions with the environment and other agents. The environment is treated as a specific type of agent that possesses an insertion function (Letychevskyi *et al.*, 2019).

The formal description of system dynamics in IMS is based on behaviour algebra, which serves as a mathematical framework for specifying interactions between agents and environments. Behaviour algebra is defined as a two-sorted universal algebra <U, A>, where *U* denotes the set of behaviours and *A* denotes the set of actions. The behaviour algebra includes several basic operations: prefixing (a·u), where an action a is followed by behaviour u; nondeterministic choice (u + v), which is associative, commutative, and idempotent; and parallel (||) and sequential (;) composition of behaviours. In addition, three terminal constants are defined: Δ (successful termination), 0 (deadlock), and ⊥ (divergence), which represents partially defined behaviour or an infinite loop without observable actions (Letychevskyi *et al.*, 2019). IMS is aimed at symbolic modelling of systems with very large or infinite state spaces, supports algebraic specifications of behaviour, and implements specialised algorithms for reducing interleaving and reachability checking, etc. Overall, the combination of these methods – literature review, comparative analysis, and formal modelling – provided both a theoretical basis and a practical framework for studying algebraic semantics in smart contract verification.

RESULTS AND DISCUSSION

Formalisation enables a smart contract to be represented within a mathematical or logical model, which is a necessary prerequisite for robust automated verification. In the practice of formalising programming languages and smart contracts, three classes of semantics are the most widespread and extensively studied: operational, denotational, and axiomatic. Each offers its own set of mathematical tools and approaches to analysis and verification. The first result of this study is a structured comparison of formal semantic paradigms evaluated against criteria established in the Materials and Methods section. A brief characterisation of each paradigm is provided below to ground the comparative evaluation. Operational semantics defines the meaning of programs by describing the steps of their execution on an abstract machine. It can be presented in the form of small-step (structural operational semantics), which describes individual transitions between states, or big-step (natural semantics), which defines the correspondence between the program and the final result of its execution (Plotkin, 2004). This approach is widely used for verification, modelling of programming languages, and formalisation of smart contracts with tools such as the K-framework. An important area of research is the creation of executable operational semantics for smart contract development languages. Another significant approach to formalisation is denotational semantics, which defines the meaning of programs by mapping language constructs into mathematical objects (e.g., domains, functions) (Stoy, 1986). This approach separates the program from its execution process and allows for the abstract analysis of program properties. In the context of smart contracts, denotational approaches are less common than operational ones, yet they appear in works aiming to provide a rigorous logical–mathematical characterisation of contract behaviour independently of the execution mechanism.

Axiomatic semantics defines the meaning of programs through a system of logical statements about their behaviour (Winskel, 1993). This approach focuses on the logical relationships between preconditions (conditions that must be true before execution) and postconditions (conditions that will be true after execution). The most common form is the Hoare triple {P} S{Q}, which states that if precondition {P} is true when program S is run and S is terminated, then postcondition {Q} will be true. This method enables the proof of programme correctness with respect to specifications and is widely used in the verification of safety-critical systems. This approach enables formal proofs that a programme satisfies given specifications without requiring its actual execution.

One of the approaches to the formalisation and modelling of smart contracts is algebraic semantics, which employs algebraic structures to describe the behaviour of software artefacts (Ghezzi *et al.*, 2003). Algebraic semantics relies on the use of algebraic structures for the formal description, specification, and analysis of program behaviour. Its theoretical foundations were established in the works (Goguen *et al.*, 1977) on initial

algebra. The algebraic approach was initially applied to the formalisation of abstract data types and programming languages, where language constructs are represented as terms over a signature, and properties are defined via equations (Ehrig & Mahr, 1985). These foundational contributions established the principle that a software system can be fully characterised by a set of sorts (data types), operations over those sorts, and equational axioms governing their behaviour. It is a principle that proves directly applicable to the domain of smart contracts, where state, transitions, and invariants can each be expressed in exactly these terms. This approach was later generalised within the framework of rewriting logic, which enables the modelling of parallelism, nondeterminism, and complex programme dynamics.

Applying the multi-criteria evaluation, the four primary semantic paradigms were analysed to determine their efficacy in smart contract verification. The evaluation synthesises the foundational ideas with their practical utility, specifically highlighting how available tool support and typical use cases align with the overarching application goals of each paradigm. The result of this comparative analysis is structured in Table 1.

Table 1. Comparison of semantic approaches for smart contract formalisation

	Operational semantics	**Denotational semantics**	**Axiomatic semantics**	**Algebraic semantics**
Core idea	Describes program execution via a sequence of transitions between states	Maps a program to mathematical objects (functions, domains), providing an abstract mathematical interpretation without simulating execution steps	Formalises behaviour using logical assertions (preconditions, postconditions) and uses proof rules to deduce correctness (Hoare Triples: {P} C {Q})	Describes the system using algebraic specifications: sorts, operations, equations, and rewriting rules
Application goal	Simulation and analysis of runtime behaviour to detect execution errors and verify functional correctness at the implementation level	Formal analysis and property-proof, such as function correctness, independent of execution details	Proof of logical correctness of programs through formal specifications, ensuring compliance with pre-/ postconditions for safety and functional soundness	Formalisation of state transition logic of smart contract protocols, executable models for validation, symbolic analysis via rewriting
Primary tools	K-framework, Coq	Isabelle/HOL, HOL4, Coq	Why3, F*	Maude, IMS, CafeOBJ
Typical Use cases	Analysis of low-level behaviour: reentrancy, exceptions, models of concurrent/simultaneous execution, gas-management semantics. Useful for virtual-machine-level auditing and runtime verification	High-level correctness proofs, formalisation of language fragments and compiler correctness, proofs of programme equivalence and non-interference at an abstract level	Proof of invariants (e.g., balance preservation), access control, absence of integer overflows, absence of re-entrancy under given preconditions, modular verification	Modelling of protocols/ state-transition rules, equivalence checking, symbolic execution via rewriting; verification of code compliance with functional requirements through goal state reachability analysis

Source: compiled by the author based on E. Hildenbrandt *et al.* (2018), S. Dharanikota *et al.* (2021), F.F. Alhabardi *et al.* (2022), M. Bartoletti *et al.* (2022), O. Letychevskyi *et al.* (2023), V. Koutavas *et al.* (2024), D. Marmsoler & A.D. Brucker (2024), V. Peschanenko *et al.* (2025), Z. Xu *et al.* (2025)

The analytical results presented in the table reveal a clear functional differentiation among the paradigms. The analysis demonstrates that each approach occupies a distinct niche in the landscape of smart contract verification. Operational semantics excels at low-level, execution-faithful modelling and is well suited to virtual-machine-level auditing. Denotational semantics provides the mathematical foundations for mechanised theorem proving, as demonstrated by tools such as Isabelle/HOL. Axiomatic semantics, meanwhile, enables scalable contract-level reasoning through Hoare-style pre- and postcondition annotations and automated SMT-based verification. In contrast to these approaches, algebraic semantics achieves a distinctive combination: it provides an algebraic specification of system states and transitions, and, when combined with executable frameworks, supports symbolic execution and reachability-based verification without relying on manual, code-level annotations. On the basis of this comparison, the present study proceeds with algebraic semantics as the primary framework for the formal specification and verification of smart contracts. The second result of this study concerns the practical application of algebraic semantics to smart contract verification. To demonstrate this, a formal algebraic model of a crowdfunding smart contract was constructed and analysed using the Insertion Modelling System.

One of the primary challenges in smart contract development is the ambiguity of logic description, which is often presented in natural language. This creates a significant risk of logical errors during implementation and complicates the verification of contract properties. A central practical advantage of the algebraic approach is that it eliminates these ambiguities by enabling mathematically precise specifications. The behaviour of smart contracts is determined by state changes in response to function calls, and algebraic semantics describes these transitions as operations on algebraic structures. In the Insertion Modelling System, a smart contract can be represented as an agent with attributes, which in turn will reflect the global state variables of the smart contract. Furthermore, other entities that interact with the contract and modify its attributes (such as users or external addresses) can also be modelled as distinct agents. To illustrate these theoretical concepts throughout the subsequent analysis, a crowdfunding smart contract is used as a running example. In the IMS syntax, this contract can be formally defined as an agent with the following attributes:

```
Contract:obj (
fundingGoal: int, // The minimum amount of tokens required to unlock withdrawals
totalRaised: int, // The cumulative balance of all contributed tokens
deadline: int, // The timestamp when the fundraising period officially closes
fundsWithdrawn: Boolean, // A flag indicating if the funds have already been moved
balance: int // The current token balance of the smart contract
)
```

The values of agent attributes can be modified through actions. In the IMS, actions describe the operations that agents can perform to alter their attributes. Each agent action defines a basic protocol, which is a Hoare triple $\alpha \rightarrow < P > \beta$, where α is a precondition that defines the state when the protocol can be applied, β is a postcondition that defines the transition of the system to a new state, and P is a process that illustrates this transition (Letychevskyi *et al.*, 2019). In the proposed crowdfunding smart contract model, two primary actions are defined. The contributeToken action represents the process of transferring funds to the contract, capturing the accumulation of contributions from participants. The withdrawFunds action models the transfer of the collected funds to the project owner. For example, the withdrawFunds function can be formally represented as an action in the IMS syntax as follows:

```
withdrawFunds(amount) = (
(totalRaised >= fundingGoal && !fundsWithdrawn) ->
<'withdrawal of funds'>
 fundsWithdrawn == 1;
 contract.balance = contract.balance - amount;
 user.balance = user.balance + amount
)
```

This action is mathematically represented as a formal transition between two states (S_0, S_1).

$$S_0 \text{ —— withdrawFunds (amount)} \longrightarrow S_1$$

In this case, S_0 is the contract state before the operation's execution (e.g., user balances before the operation), and S_1 denotes the new state following execution (e.g., updated user balances and a locked withdrawal flag). By formalising each contract action as an algebraic expression with a precisely defined effect on system state, the model eliminates the ambiguity inherent in textual specifications. This provides the foundation for all subsequent verification steps described below. In the context of smart contracts, it is essential not only to provide a formal description of system behaviour but also to specify constraints that guarantee its safe execution. Such constraints are formalised as invariants (safety conditions) and goal states, which define the correct functioning logic of the contract. Safety conditions describe the invariants of a system that must remain true in all reachable states of the model. Violation of at least one of these conditions means that the system has entered an unsafe state, i.e., something has happened that contradicts the requirements for correctness or safety. In formal models of smart contracts, these properties can reflect aspects such as balance preservation, constancy of the total number of tokens, etc. For example, in a smart-contract model that uses accounts to track users' balances, a fundamental safety condition requires that no account may ever have a negative balance:

$$\forall account: balances[account] \geq 0$$

In the syntax of the IMS, this can be expressed as:

```
Forall(i:int){0<i<=countAccount &&
            listAccounts(i).balance>=0
    }
```

A violation of this invariant (for instance, if a balance becomes negative following a fund transfer operation) indicates the presence of a logical error or vulnerability (e.g., integer overflow/underflow, incorrect state update, etc.). In traditional Solidity programming, such conditions are checked via tests or static analysis, which does not guarantee full coverage of all execution scenarios. The algebraic approach allows for a mathematical proof that any operation altering the balance (e.g., transferring funds between accounts) preserves this property. This guarantees behavioural consistency regardless of the programming language. Goal states define the desired or final configuration of the system in which the specified conditions are met, or certain functional goals are achieved. They are used for the formal description of a system's resultant behaviour. Achieving a goal state signifies the system's transition to a new mode of operation or the completion of a specific phase of its lifecycle. In theoretical terms, a goal state corresponds to the reachability property, which guarantees that within the model there exists at least one sequence of transitions leading from an initial state to

a desired (correct) result. For example, in a smart contract for crowdfunding, the goal state can be defined as achieving the desired amount of funds raised:

totalRaised >= fundingGoal

In the IMS, demonstrating that a goal state is reached relies on dynamic verification techniques, primarily through the generation of execution traces using a concrete trace generator (CTG) or a symbolic trace generator (STG). If the condition is met, the generation of that specific trace is immediately interrupted, the trace is recorded, and the system confirms the goal has been successfully reached. This results in a recorded trace that describes the exact path of transitions and intermediate states through which the goal state is achievable. For instance, for the crowdfunding goal state, the concrete trace terminates exactly at the moment the condition is satisfied, confirming that the state is reachable:

```
text ' fundingGoalState detected';
all: condition FINITE /*$
contract.totalRaised == 110 &&
contract.fundingGoal == 100 &&
contract.fundsWithdrawn == 0 &&
contract.balance == 110 &&
user.balance == 0
```

Simultaneously, if the model allows continued execution, the system generates a separate SuccessfulTermination trace that captures the complete execution path of the contract. This trace can subsequently be analysed to observe further system behaviour, such as the execution of the withdrawFunds protocol. Once the goal condition (totalRaised >= fundingGoal) is satisfied and the fundsWithdrawn flag remains false, the precondition for withdrawal is met, enabling the execution of the corresponding basic protocol. The relevant fragment of the trace can be represented as follows:

```
User#user,Contract#contract:condition
BASIC_PROTOCOL   /*$withdrawFunds_   _user_
(User#user)$*/;

contract.totalRaised == 110 &&
contract.fundingGoal == 100 &&
contract.fundsWithdrawn == 1 &&
contract.balance == 0 &&
user.balance == 110
```

As shown in the trace, once the goal condition is satisfied, the system proceeds with the execution of the withdrawFunds protocol. The trace explicitly captures the sequence of interactions between the user and the contract, as well as the resulting state transformation. The trace not only confirms the reachability of the goal state but also provides a detailed and verifiable account of the execution path leading to it. The successful generation of a finite trace therefore validates the feasibility of achieving the contract's primary functional objective within the defined model. In contrast to safety conditions, which restrict the system from reaching forbidden or inconsistent states, goal states specify the desirable outcomes that the system is expected to reach. Such a formal definition makes it possible to prove that, for any sequence of user actions, there exists a path in the state space that leads to a goal state, which means that the system satisfies the liveness property. A practical application of this concept is demonstrated in the verification of smart contracts generated by artificial intelligence (Peschanenko *et al.*, 2025). In this approach, functional requirements are first defined as acceptance criteria using the Gherkin language and then formalised as goal states within the IMS. The verification process involves using a depth-first search algorithm to prove that the algebraic model of the generated Solidity code inevitably reaches this goal state. Reaching the goal state confirms that the AI-generated code strictly adheres to the initial tokenomics specifications and functions correctly.

Another advantage of employing the algebraic approach is the possibility of using symbolic modelling (symbolic execution) methods. Symbolic modelling allows for the analysis of operation execution not for concrete values, but for symbolic variables that describe the set of possible system states. This approach makes it possible to explore all potential sequences of actions and interactions between participants, including those that are difficult or impossible to reproduce using traditional testing methods. Symbolic models cover a much wider range of states and provide a more reliable mechanism for controlling the correctness of system behaviour than classical testing or static analysis tools. Specifically, the IMS supports two directions of analysis: forward symbolic modelling, which involves trace generation from an initial state to a sought property (e.g., a goal state), and backward symbolic modelling, which involves trace generation from a target property back to an initial state (Letychevskyi *et al.*, 2019). As illustrated in Figure 1, backward symbolic modelling is particularly effective for detecting safety violations: it determines whether a path can be constructed backwards from an unsafe state (where the invariant is violated) to an initial state. If such a path exists, a vulnerability is confirmed; otherwise, if no path can be constructed and the analysis results in a deadlock, the system can be considered safe. When combined with symbolic execution, the formal definitions of Safety Conditions and Goal States provide a rigorous framework for verification. Forward symbolic modelling establishes the reachability of goal states by generating execution traces from the initial state until the goal condition is satisfied. Together, these approaches ensure consistency between the contract's logic and its formal specification, enabling systematic reasoning about both safety and correctness. Algebraic semantics is also employed for the detection and prevention of vulnerabilities that arise from errors in logic or incorrect assumptions concerning the behaviour of the environment. Specifically, algebraic semantics

enables the analysis of behavioural vulnerabilities that manifest only during the execution of a sequence of operations or through the interaction of multiple system entities. Such vulnerabilities are difficult to detect via traditional testing; however, within the IMS, they can be formally specified and verified at the level of execution traces (trace-level analysis). The algebraic approach supports the verification of system properties across execution traces rather than restricting analysis to single operations.

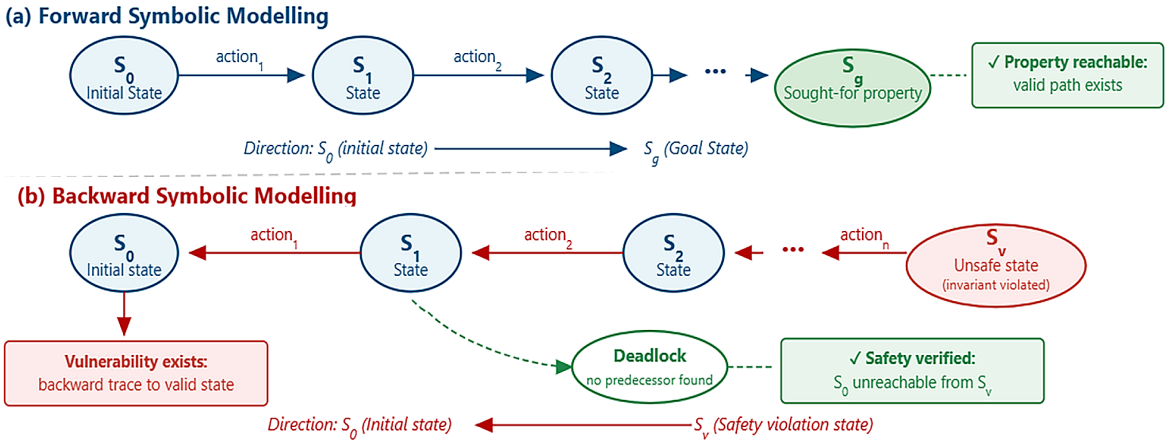


Figure 1. Forward and Backward Symbolic Modelling in the IMS
Source: created by the author based on O. Letychevskyi *et al.* (2019).

Among such vulnerabilities are Trace-level safety/liveness failures – violations of safety or liveness properties that occur within a sequence of states. Typical examples include deadlock (a state in which no further actions are possible) or double-spending, where the same transaction affects the system state more than once, thereby breaking the balance-preservation invariant (Letychevskyi *et al.*, 2019). To demonstrate this analytical capability in practice, a deadlock vulnerability within the crowdfunding smart contract model was examined. Specifically, considered a concrete trace fragment in which the system reaches a state where no basic protocol is applicable. In this specific scenario, the operational deadline has passed, meaning further contributions are no longer allowed. Simultaneously, the total amount collected (totalRaised) remains below the required fundingGoal, meaning the withdrawal of funds is prohibited. Because no refund function exists within the contract's architecture, there is no alternative protocol available. As a result, the preconditions of all available protocols–such as contributeTokens and withdrawFunds–evaluate to false. The corresponding trace fragment explicitly captures this situation by stopping at a terminal state marked as a deadlock:

```
text 'deadlock detected';
all: condition FINITE /*$
 timestamp == 30 &&
 contract.deadline == 30 &&
 contract.fundingGoal == 100 &&
 contract.totalRaised == 45 &&
 contract.fundsWithdrawn == 0 &&
 contract.balance == 45 &&
 user.balance == 0 $*/;
```

Thus, the analysis demonstrates that the smart contract contains a deadlock state where accumulated funds are permanently locked, and no further actions can be executed. This situation reflects a violation of liveness properties, as the system is unable to progress to any meaningful terminal state, such as successful fund withdrawal or refund distribution. From a practical perspective, such behaviour is undesirable, as it results in the loss of asset accessibility and inevitably leads to permanent financial loss for the contributors. Furthermore, identifying this deadlock through formal modelling emphasises the need to include alternative protocols – such as refund mechanisms – to ensure completeness of the system and guarantee that at least one valid transition is available in every reachable state.

While deadlock analysis ensures the system's operational continuity (liveness), formal trace-level verification is also essential for defending against malicious manipulations. To this end, the methodology facilitates the detection of "strategy-based attacks", such as double-spending. In these scenarios, an attacker deliberately manipulates the execution order of actions or environmental conditions to gain an advantage. The algebraic approach allows for the modelling of such scenarios as sets of alternative transitions within the state space, and for proving that no strategy can lead to a violation of the specified invariants. Another important issue is ensuring the correctness and consistency of smart contract behaviour across different platforms (Ethereum, Algorand, Cardano). Since blockchain ecosystems rely on different programming languages – Solidity, Plutus, Rust – and in this case, there is a need for a unified basis for specifications.

In this context, algebraic semantics is an effective tool for formal modelling of contract behaviour, as well as a basis for creating universal models that are independent of the specific implementation language. Formal contract models created as algebraic specifications provide a platform-independent formal basis that can, in principle, support code generation for various blockchain platforms, ensuring consistency between the formal specification and the implementation. The capacity for automatic code generation based on formal models facilitates a "correct-by-construction" methodology (Kourie & Watson, 2012), reducing reliance on post-deployment patching. Unlike traditional development cycles, where code is written manually and subsequently tested for errors, this approach ensures that the implementation is derived directly from a verified model. Consequently, the resulting smart contracts inherit the mathematically proven properties of the algebraic specification, such as safety invariants and functional correctness, thereby eliminating human error during the coding phase.

The obtained results demonstrate that the algebraic approach provides a practically applicable framework for the formal verification of smart contracts. The developed crowdfunding smart contract model confirmed that algebraic specifications enable the precise definition of system behaviour, the formalisation of safety conditions and goal states with trace-based confirmation of reachability, and trace-level detection of a deadlock vulnerability. Furthermore, the formalisation of constraints and system invariants provides a rigorous basis for reasoning about correctness, while symbolic modelling within the IMS supports systematic exploration of the state space and the verification of both safety and liveness properties. The obtained results can be meaningfully interpreted in the broader context of contemporary research on formal methods for smart contract verification. Over the past several years, significant attention has been devoted to the development of operational, denotational, and axiomatic semantic frameworks, each addressing different aspects of correctness, security, and formal reasoning.

A significant body of recent research focuses on operational semantics in order to achieve execution accuracy and equivalence. For example, V. Koutavas *et al*. (2024) formalised the big-step and small-step semantics of the Yul intermediate language, proving operational equivalence and successfully implementing an interpreter with correct optimisations to enhance Ethereum's underlying security. Similarly, Z. Xu *et al*. (2025) introduced a structural operational semantics for Crystality, a smart contract language tailored for parallel EVMs. Formalised in the Coq theorem prover, their model verifies the correctness of token-transfer functions, specifically targeting concurrency and state-partitioning properties. In addition, Y. Liu *et al*. (2024) proposed PropertyGPT, an LLM-driven verification framework that verifies generated properties against a small-step operational semantics of Solidity (KSolidity), checking invariants and pre/postconditions by exploring reachable contract states and using SMT solving. These results are consistent with the conclusions of the present study regarding the importance of formal semantics for ensuring correctness across different abstraction layers. However, the findings also indicate a methodological distinction. While operational semantics provides high fidelity with respect to execution behaviour, it is often closely aligned with specific virtual machines or execution environments. Its step-by-step nature makes high-level property specification less direct and often more complex compared to more abstract semantic frameworks. In turn, the algebraic approach based on the IMS emphasises abstraction from implementation details. By modelling smart contract logic through algebraic structures – states, operations, and invariants – it becomes possible to reason about behaviour independently of a particular platform. This supports the construction of reusable and parameterised models that can be adapted to different blockchain systems.

Denotational approaches, on the other hand, prioritise mathematical rigour and compositional reasoning. In particular, D. Marmsoler & A.D. Brucker (2024) developed a comprehensive denotational semantics framework for Solidity, formalised within the Isabelle/HOL interactive theorem prover. Their research established a logically consistent, executable model that incorporates grammar-based fuzzing and symbolic execution for the formal analysis and verification of Ethereum smart contracts before deployment. Such an approach provides strong mathematical rigour and establishes the foundations for formally verifying the correctness of Solidity contracts at the level of the language's logical constructs. These findings are generally consistent with the present study, particularly regarding the importance of mechanised reasoning for ensuring correctness. Nevertheless, deep embedding into systems such as Isabelle/HOL often requires substantial manual proof-writing effort and advanced expertise in formal verification methods. In contrast, the algebraic framework implemented in IMS allows the modeller to specify system behaviour declaratively in terms of basic protocols, after which the system supports automated exploration of the state space through execution trace generation. This suggests that algebraic semantics may offer a more accessible entry point for formal verification while preserving a sufficient level of rigour.

Axiomatic semantics, heavily reliant on preconditions and postconditions, represents another prevalent approach conceptually aligned with the one proposed in this study. F.F. Alhabardi *et al*. (2022) applied Hoare triples and weakest preconditions to verify Bitcoin Script in the Agda proof assistant. Their work

successfully derives human-readable access-control conditions for standard scripts, such as Pay to Public Key Hash (P2PKH). Additionally, S. Dharanikota *et al*. (2021) developed CELESTIAL, a framework that allows developers to specify Hoare-style functional correctness properties for Solidity smart contracts and formally verify them against an F* model of blockchain semantics. The framework supports the verification of invariants, state-machine transitions, and postconditions, enabling the detection of logical inconsistencies and vulnerabilities, such as arithmetic overflows, prior to deployment. These approaches demonstrate that axiomatic reasoning can be effectively applied to detect common vulnerabilities and ensure functional correctness. The findings of the present study are consistent with these results; however, the key difference lies in the focus of verification. While axiomatic frameworks primarily verify that a concrete implementation satisfies explicitly defined correctness annotations, the algebraic approach in IMS focuses on modelling and analysing the overall state-transition logic of the smart contract system. This makes the approach particularly suitable for identifying structural behavioural vulnerabilities arising from smart contract design. By supporting both high-level behavioural modelling prior to implementation and the analysis of existing contracts, the IMS environment enables a more systematic exploration of protocol reliability.

The most direct points of comparison for the present study, however, are found within the algebraic paradigm itself. Building upon verification approaches implemented in the Maude framework (Bae *et al.*, 2022), research by M. Bartoletti *et al*. (2022) applied rewriting logic to formally verify the liquidity of recursive Bitcoin contracts. In their framework, high-level BitML contract behaviours are represented through abstract operational semantics, after which critical properties, such as the absence of permanently frozen funds, are verified using the Maude model checker. Moreover, L. Bellomarini *et al*. (2024) recently expanded upon this formal algebraic foundation to incorporate complex strategic reasoning. By translating BitML specifications into concurrent game structures, they demonstrated how executable mathematical models can detect vulnerabilities stemming from dishonest participant strategies via Alternating-time Temporal Logic (Atl*) model checking. These studies are consistent with the conclusions of the present work and further demonstrate the effectiveness of algebraic modelling for smart contract verification. Just as M. Bartoletti *et al*. (2022) used abstract models to systematically verify contracts, the application of the IMS demonstrated the practical value of algebraic modelling for detecting complex behavioural vulnerabilities before executing any code. Furthermore, the Insertion Modelling System provides practical advantages in terms of usability and scalability. The agent–environment modelling paradigm allows system behaviour to be specified through high-level abstractions, enabling comprehensive analysis of all possible execution paths and component interactions. This approach provides an effective framework for ensuring protocol reliability and is particularly suitable for identifying complex structural issues such as deadlocks, invalid terminal states, and strategy-based attacks.

CONCLUSIONS

This study demonstrated that algebraic semantics provides an effective formal framework for ensuring the correctness and security of smart contracts. The application of the IMS to a crowdfunding smart contract produced several concrete verification results. First, the algebraic specification of the contract, including the formal definition of agent attributes, actions, and state transitions, enabled the elimination of ambiguities and provided a precise formal model of contract behaviour. This model allowed the formal definition of key properties, such as safety conditions (e.g., balance preservation) and reachability conditions defined through goal states. Second, the study demonstrated the effectiveness of trace-level verification within the IMS. It was shown that the algebraic approach supports the analysis of behavioural properties across execution traces rather than isolated operations. In the crowdfunding contract, this made it possible to identify a deadlock scenario in which funds remain permanently locked after the deadline due to the absence of a refund mechanism. The generated execution trace explicitly captured a terminal state in which no protocol was applicable, formally confirming the presence of a design flaw.

These results demonstrate that the algebraic approach enables the detection of liveness violations and behavioural vulnerabilities that are difficult to identify using conventional testing or static analysis techniques. In particular, the combination of algebraic specification with concrete and symbolic trace analysis provides a systematic mechanism for exploring the state space and reasoning about contract behaviour under different execution scenarios. Overall, the findings confirm that algebraic semantics offers a practically applicable approach to smart contract verification, allowing correctness properties to be defined, analysed, and validated. Future research will focus on the automatic generation of deployment-ready smart contract code from formally verified algebraic models created in the Insertion Modelling System. In particular, further work will address the development of a translator for transforming formal specifications into executable smart contract implementations while preserving their verified behavioural properties. Additional research will investigate adapting the translation process to different blockchain platforms and

smart contract programming languages to improve the reliability and security of blockchain systems.

ACKNOWLEDGEMENTS

None.

FUNDING

None.

CONFLICT OF INTEREST

None

REFERENCES

[1] Alhabardi, F.F., Beckmann, A., Lazar, B., & Setzer, A. (2022). Verification of bitcoin script in Agda using weakest preconditions for access control. In *27th international conference on types for proofs and programs (TYPES 2021)*. Wadern: Schloss Dagstuhl – Leibniz Center for Informatics. doi: 10.4230/lipics.types.2021.1.

[2] Bae, K., Escobar, S., & Meseguer, J. (2022). *The maude LTL LBMC tool tutorial*. Retrieved from https://maude.cs.illinois.edu/tools/lmc/manual.pdf.

[3] Bartoletti, M., Lande, S., Murgia, M., & Zunino, R. (2022). Verifying liquidity of recursive bitcoin contracts. *Logical Methods in Computer Science*, 18(1). doi: 10.46298/lmcs-18(1:22)2022.

[4] Bellomarini, L., Favorito, M., & Galano, G. (2024). Strategic reasoning for BitML smart contracts. In *AI4CC-IPS-RCRA-SPIRIT 2024: International workshop on artificial intelligence for climate change, Italian workshop on planning and scheduling, RCRA workshop on experimental evaluation of algorithms for solving problems with combinatorial explosion, and SPIRIT workshop on strategies, prediction, interaction, and reasoning in Italy*. Bolzano: CEUR Workshop Proceedings.

[5] Davila, R., Barcenas, E., & Aldeco-Perez, R. (2025). Smart contracts formal verification: A systematic literature review. *Abstraction & Application*, 50, 46-56. doi: 10.48550/arxiv.2510.17865.

[6] Dharanikota, S., Mukherjee, S., Bhardwaj, C., Rastogi, A., & Lal, A. (2021). Celestial: A smart contracts verification framework. In *Proceedings of the 21st conference on formal methods in computer-aided design – FMCAD 2021* (pp. 133-142). doi: 10.34727/2021/isbn.978-3-85448-046-4_22.

[7] Ehrig, H., & Mahr, B. (1985). *Fundamentals of algebraic specification 1: Equations and initial semantics*. Berlin: Springer. doi: 10.1007/978-3-642-69962-7.

[8] Ferariu, T., Wadler, P., & Melkonian, O. (2025). Validity, liquidity, and fidelity: Formal verification for smart contracts in Cardano. In *6th international workshop on formal methods for blockchains (FMBC 2025)*. Wadern: Schloss Dagstuhl – Leibniz Center for Informatics. doi: 10.4230/oasics.fmbc.2025.6.

[9] Ghezzi, C., Jazayeri, M., & Mandrioli, D. (2003). *Fundamentals of software engineering*. (2nd ed.). New Delhi: Pearson Prentice Hall.

[10] Goguen, J.A., Thatcher, J.W., Wagner, E.G., & Wright, J.B. (1977). Initial algebra semantics and continuous algebras. *Journal of the ACM*, 24(1), 68-95. doi: 10.1145/321992.321997.

[11] Hildenbrandt, E., *et al.* (2018). KEVM: A complete formal semantics of the Ethereum Virtual Machine. In *2018 IEEE 31st computer security foundations symposium (CSF)* (pp. 204-217). Oxford: IEEE. doi: 10.1109/csf.2018.00022.

[12] Jiao, J., Kan, S., Lin, S.-W., Sanan, D., Liu, Y., & Sun, J. (2020). Semantic understanding of smart contracts: Executable operational semantics of Solidity. In *2020 IEEE symposium on security and privacy (SP)* (pp. 1695-1712). San Francisco: IEEE. doi: 10.1109/SP40000.2020.00066.

[13] Kourie, D.G., & Watson, B.W. (2012). *The correctness-by-construction approach to programming*. Heidelberg: Springer. doi: 10.1007/978-3-642-27919-5.

[14] Koutavas, V., Lin, Y.-Y., & Tzevelekos, N. (2024). An operational semantics for Yul. *Lecture Notes in Computer Science*, 15280, 328-346. doi: 10.1007/978-3-031-77382-2_19.

[15] Krichen, M., Lahami, M., & Al-Haija, Q.A. (2022). Formal methods for the verification of smart contracts: A review. In *2022 15th international conference on security of information and networks (SIN)*. doi: 10.1109/SIN56466.2022.9970534.

[16] Letichevsky, A., Letychevskyi, O., & Peschanenko, V. (2016). Insertion modeling and its applications. *Computer Science Journal of Moldova*, 24(3(72)), 357-370.

[17] Letychevskyi, O., Peschanenko, V., Poltoratskyi, M., & Konnova, O. (2023). An algebraic approach to the verification of smart contracts in TEAL. *Journal of Information Technologies in Education*, 54, 37-51. doi: 10.14308/ite000774.

[18] Letychevskyi, O., Peschanenko, V., Radchenko, V., Orlovsky, M., & Sobol, A. (2019). Algebraic approach to verification and testing of distributed applications. In *IECC '19: Proceedings of the 1st international electronics communication conference* (pp. 37-43). New York: ACM. doi: 10.1145/3343147.3343159.

[19] Liu, Y., Xue, Y., Wu, D., Sun, Y., Li, Y., Shi, M., & Liu, Y. (2024). PropertyGPT: Llm-driven formal verification of smart contracts through retrieval-augmented property generation. In *Network and distributed system security (NDSS) symposium 2025*. San Diego: Internet Society. doi: 10.48550/arxiv.2405.02580.

[20] Marmsoler, D., & Brucker, A.D. (2024). Isabelle/Solidity: A deep embedding of Solidity in Isabelle/HOL. *Formal Aspects of Computing*, 37(2), article number 15. doi: 10.1145/3700601.

[21] Peschanenko, V., Poltorackiy, M., & Konnova, O. (2025). Verification of smart contract code generated by applying artificial intelligence. *Communications in Computer and Information Science*, 2359. doi: 10.1007/978-3-031-81372-6_1.
[22] Plotkin, G.D. (2004). *A structural approach to operational semantics*. Aarhus: University of Aarhus.
[23] Stoy, J.E. (1986). *Denotational semantics: The Scott-Strachey approach to programming language theory*. Cambridge: MIT Press.
[24] Tolmach, P., Li, Y., Lin, S., Liu, Y., & Li, Z. (2021). A survey of smart contract formal specification and verification. *ACM Computing Surveys*, 54(7), article number 148. doi: 10.1145/3464421.
[25] Winskel, G. (1993). *The formal semantics of programming languages: An introduction*. Cambridge: MIT Press.
[26] Xiao, M., Xu, Y., Li, Z., & Wan, H. (2024). Advanced security auditing methods for Solidity-Based smart Contracts. *Electronics*, 13(20), article number 4093. doi: 10.3390/electronics13204093.
[27] Xie, R., Zhong, X., Chen, X., Xu, S., Yu, H., & Guo, X. (2024). Automatic construction and verification algorithm for smart contracts based on formal verification. *AIP Advances*, 14(11), article number 115304. doi: 10.1063/5.0238456.
[28] Xu, Z., Wang, H., & Sun, M. (2025). Operational semantics for Crystality: A smart contract language for parallel EVMs. *Lecture Notes in Computer Science*, 15841, 303-321. doi: 10.1007/978-3-031-98208-8_18.

Застосування алгебраїчної семантики для формалізації та верифікації смарт-контрактів

Ольга Коннова
Аспірант
Херсонський державний університет
73000, вул. Університетська, 27, м. Херсон, Україна
https://orcid.org/0000-0001-5590-9527

Анотація. Смарт-контракти є ключовим компонентом блокчейн-систем, проте їх незмінність після розгортання зумовлює підвищені вимоги до коректності та безпеки. Це визначає необхідність застосування формальних методів, здатних забезпечити математично обґрунтовану верифікацію контрактів на етапі проєктування. Метою даного дослідження було теоретичне обґрунтування та систематизація алгебраїчної семантики як формальної основи для специфікації та верифікації смарт-контрактів, а також демонстрація її можливостей шляхом порівняння з альтернативними семантичними підходами та практичної ілюстрації її застосування на прикладі побудови та аналізу формальної моделі. У ході роботи було застосовано комплекс теоретичних методів, зокрема системний огляд літератури та порівняльний аналіз операційної, денотаційної, аксіоматичної та алгебраїчної семантик. Практична значущість підходу була проілюстрована через формалізацію властивостей контракту краудфандингу в системі інсерційного моделювання. У результаті дослідження було побудовано формальну алгебраїчну модель смарт-контракту, у межах якої визначено стани, переходи та інваріанти системи. Продемонстровано можливість формалізації умов безпеки і цільових станів контракту. Зокрема, було підтверджено досяжність цільового стану (успішного збору коштів), що було верифіковано через аналіз фрагмента згенерованої траси виконання. Крім того, наведено приклад успішного виявлення взаємного блокування (deadlock) у моделі контракту. Це, у свою чергу, дозволяє систематично виявляти складні поведінкові вразливості та запобігати логічним помилкам ще на етапі концептуального проєктування. Отримані результати підтверджують ефективність алгебраїчного підходу до формальної верифікації та демонструють його придатність до раннього виявлення логічних помилок і підвищення надійності смарт-контрактів

Ключові слова: блокчейн; децентралізовані додатки; формальні методи; алгебраїчні специфікації; система інсерційного моделювання