(In)Correct Smart Contract Specifications

***Abstract*—Poorly specified smart contracts can be vulnerable to attacks on faulty design. Formal methods are currently unable to resolve these vulnerabilities because they are not vulnerabilities of incorrect code, but of incorrect specification. We are thus in need of a paradigm shift in how we specify and verify smart contracts which allows for a rigorous and accurate notion of a contract specification's correctness. We submit that correctness includes appropriate cryptoeconomic behaviors, which are generally out of scope of a contract's specification. We advocate for an approach to contract specification consisting of contract axiomatization and metaspecification, and illustrate with an example.**

***Index Terms*—Formal verification, formal specification, smart contract specification.**

I. Introduction

Poorly specified smart contracts can be vulnerable to attacks on faulty design [43]. Examples of such attacks, typically targeting poor economic or governance design, are alarmingly common, costing the equivalent of billions of US dollars in cryptocurrency losses each year [12], [72].

The nature of these attacks means that they are rarely targeted by formal methods, as faulty smart contracts can be *correct*, only with regards to a faulty specification [15], [52]. Furthermore, many vulnerabilities relating to poor economic or governance design are out of scope of a specification. So, while specifications attempt to target these properties, formal verification engineers can only make informal arguments to justify many design choices as correctly capturing the intended properties and behaviors of the smart contract in question.

We are thus in need of a paradigm shift in how we specify and verify smart contracts which allows for a rigorous and accurate notion of a contract specification's *correctness*, especially with regards to properties intended by it, but ultimately out of its scope. We advocate for an approach to formal contract specification with interactive theorem provers (ITPs), consisting of *axiomatization* and *metaspecification*.

First, we note that for ITP-based verification, a complete specification forms the basis for an axiomatized theory. This is because a specification is a set of propositions which characterize a contract's design and structure. We can thus state these propositions as a specification and study the behavior of arbitrary contracts satisfying that specification. Importantly, to reduce the burden of verifying any given contract, specifications should be minimal [65]. This is contract axiomatization.

From there we formally study the implications of a contract specification via a *metaspecification*. The metaspecification is to a specification what a specification is to an implementation. It consists of *meta properties*, which are properties intended by, but out of scope of, the contract specification, as well as desired properties of the specification itself. For example, we might include desired high-level (*e.g.* cryptoeconomic) properties of a smart contract, or properties of a specification like consistency or completeness [20]. A contract specification's *correctness*, then, depends on whether it conforms to its metaspecification.

We make the specification-metaspecification distinction because the cost of formally verifying software can already be prohibitive, and we wish to address issues of poor specification in formal methods without unnecessarily augmenting the burden of verification on any given smart contract. By treating the specification as a contract axiomatization, we keep it minimal while expanding the formal study of smart contract behavior, adding to the security guarantees of formal methods without increasing the burden of verifying a specific implementation once the specification is formalized.

We present an initial framework of axiomatization and metaspecification within ConCert [7], a Coq-based formal verification tool which models the execution semantics of third-generation blockchains and has custom Coq tactics for proving contract invariants. The framework we propose is able to incorporate properties of specifications in a way not previously done because ConCert models the execution semantics of smart contracts within a formal model of a blockchain. While it is possible to do similar analysis within other formal settings, without the semantics of the underlying blockchain formalized, we argue that any efforts to do so will eventually encounter theoretical limits.

We proceed as follows. In §II, we give historical context to this problem and discuss related work. In §III, we discuss the problem of correct specification. In §IV we discuss our proposed framework of contract axiomatization and metaspecification. In §V, we illustrate with an example financial smart contract. In §VI, we justify this paradigm as a solution to our issue of (in)correct specification. In §VII, we discuss limitations and future work. In §VIII, we conclude.

II. Related Work

Limitations of formal methods are well-established [26], [43], [59]. We know that formal methods cannot guarantee perfect software [34], in part because of theoretical limits of a causal model of a physical process [13], [25]. As such, formal methods should be used in conjunction with other techniques to ensure software security which compensate for limitations inherent to formal methods [14], [36].

Design and formation of a specification has long been considered in the domain of informal techniques, out of the bounds of formal methods [32], [43]. The literature rightly points out that the infrastructure required to reason about software specification is vast. In order to reason about a specification's correctness, one must have a formal model of its execution model and—intractibly for most software—the social and ecological environment in which that software operates and executes, consisting of different types of users as well as society and the natural environment around them [43]. The limitation of formal methods due to the difficulty of forming a *correct* specification has been long recognized [32].

Even so, there have been efforts of varying formality to address the issue of (in)correct specification. Firstly, it is often argued that formalizing a specification alone, due to the precision required, helps ensure a specification's correctness by clarifying details and preventing inconsistencies [14], [46]. Specification languages are also frequently designed to target certain domain-specific properties in order to ease the translation between prose and formal specification [60].

There is also emerging work which attempts to formally justify the correctness of a contract specification. For example, one study tests the strength of a contract specification by mutation testing to identify any pathological yet correct (per the specification) behavior of Ethereum smart contracts [53]. Similarly, the developers of a formally verified stablecoin, Djed, used techniques such as mutation and unit testing to identify potentially pathological behavior of the specification. They then targeted these behaviors with formal verification, in Isabelle and using SMT solvers, to justify the robustness of the contract specification [69].

These are good examples of developers considering the correctness of their specifications, but crucially the properties they proved about these specifications are articulated and proved *ad hoc*. Testing and intuition ground the conceptual framework from which they derive the results to be proved. In particular, they are not derived systematically via a theory of some kind. Without a systematic framework, one has no notion of *completeness*—whether the propositions proved about a specification are sufficient to guarantee it to be correct.

Finally, there are some verification efforts which take a stronger theoretical approach to smart contract specifications, but these do not reason about deployable or executable code. Consulting and auditing firms such as Gauntlet [2] and 20squares [1] perform statistical, economic, and game-theoretic analysis on contract specifications [11], [31], but crucially such analyses are not present in any setting of formal verification.

Our work is to lay the theoretical foundations for a systematic framework, that can evaluate the correctness of a smart contract specification based on cryptoeconomic analysis, and which brings a high-level approach of cryptoeconomic reasoning into a setting of verification on contracts which can be deployed and executed. The purpose of this work is to improve the efficacy of formal methods against attacks on poor cryptoeconomic design. We do so in ConCert, a Coq-based tool for smart contract verification which models the execution semantics of third-generation blockchains [7] and features verified extraction to multiple third-generation blockchains [6].

Because ConCert models the execution semantics of a blockchain in an ITP, it is a particularly strong choice for reasoning about the properties of contract specifications. In particular, we argue that it is possible to embed a cryptoeconomic theory into ConCert with which one can reason, from first principles, about smart contract behavior and derive properties of the contract metaspecification.

III. The Problem of (In)Correct Specification

Contract specifications almost never feature economic properties, despite the fact that the primary use case for smart contracts is as economic or financial infrastructure. Instead, the specifier goes through an informal translation process from a high-level, informal business or economic specification into a technical specification [61]. This informal process can be erroneous, resulting in a specification that fails to capture the intended cryptoeconomic properties and contract behaviors—in other words, an *incorrect* specification.

Consider the specification of an automated market maker (AMM). From its inception, its design is to facilitate an efficient market, with efficient price discovery [18]. Bonding curves, *e.g.* the first and most fundamental

$$xy = k,$$

were put forward from classical economics as having desirable market properties. However, reading through the specification of an AMM—take for example Dexter2 [8], a Tezos-based AMM, its formal counterparts [19], [40], [48], or a generic formal specification for AMMs using the bonding curve $xy = k$ [71]—does little to convince us that the resulting smart contracts do indeed exhibit the desired high-level, economic properties of an efficient market-maker.

This is because contract specifications tend to be low-level in nature, focusing on contract interface, storage, and functional descriptions of entrypoint functions. High-level, cryptoeconomic properties are assumed to emerge from the specification, but they are difficult to formally justify. For some examples of such properties, properly incentivizing liquidity providers with fees, without disrupting other cryptoeconomic features of the AMM, is a highly complex topic [4], [24], [27], [28], [35], and not at all obvious to be correct from a typical contract specification. Indeed, assurances of desirable cryptoeconomic behavior for AMMs using the $xy = k$ bonding curve were largely provided *after* the original Uniswap contracts were specified and deployed, *e.g.* in [5].

We can see that smart contract developers ubiquitously use an informal translation process, from high-level cryptoeconomic or business logic to a technical sepcification, to specify their smart contracts. In contrast to Uniswap, a resounding success, many instances of contract specification result in catastrophic losses due to incorrect design. Examples include Beanstalk [23], Mango Markets [45], [55], the Spartan Protocol [17], [39], Pancake Bunny [16], [33], [37], [49], and a seemingly countless stream of others [12], [72].

Frustratingly, aside from the benefits of producing a formal specification, formal methods are of no use to resolve these vulnerabilities because they are not vulnerabilities of incorrect code, but of incorrect specification. Since formal methods are an important avenue toward high-assurance software, and are of particular relevance to smart contracts due to contract immutability [11], [61], we are in need of a paradigm shift in how we specify and verify smart contracts in order to adequately address vulnerabilities due to incorrect specification.

Our goal in this and future work is to develop rigorous tools for reasoning about the correctness of smart contract specifications in an ITP-based formal setting. In this paper, we will focus on this problem as it relates to a contract's cryptoeconomic properties. We call a contract specification *correct* if any contract satisfying that specification also exhibits the associated and desired cryptoeconomic properties. The framework that we put forward is one of contract *axiomatization* and *metaspecification*.

IV. Contract Axiomatization and Metaspecification

The essential idea of contract axiomatization and metaspecification is to specify a contract's essential features in its specification (a contract axiomatization) and then to formally study the implications, cryptoeconomic or otherwise, of that specification via the metaspecification. This isolates the minimal conditions that must be true of a contract from the properties and behaviors that necessarily follow, emulating mathematical reasoning. Importantly, this allows us to minimize the size of a contract specification, and thus the burden of formally verifying any particular implementation, while improving our understanding of that contract's cryptoeconomic behavior. Formal specifications remain low-level and technical in nature, but through the metaspecification we are able to express and reason about the high-level, cryptoeconomic properties of the specification.

A. Contract Axiomatization

An effective specification abstracts the essential pieces of a contract's design and interface. It should be *consistent* (unambiguous) and *complete* (fully descriptive of contract behavior) [20], [65]. In particular, we should be able to deduce the outputs of any contract call by the specification, given the inputs. If it is well-defined in an ITP, a formal specification should be able to be stated as a predicate on smart contracts. In ConCert, the contract type is parameterized by a contract's setup, message, state, and error types, and a specification `S` then has the following form:

```
S : forall (C : Contract Setup Msg State
            Error), Prop.
```

The art of specification holds a tension between saying enough, so that implementers do not choose unacceptable implementations, and not saying too much, which can limit the design freedom of the implementer [65]. From the perspective of formal methods, there is further pressure to make the specification as concise as possible, since formal verification is difficult and costly due to the expertise required [66].

For ITP-based verification, we can see right away that a specification, a list of propositions we might hope to prove about a particular implementation, mimics the practice of axiomatization in mathematical theories [44]. For example, in mathematics a *group* is defined by a set of axioms: it is a set, with an associative operation, an identity element, and inverses [70]. Given a set with an operation, one can prove or disprove whether or not that set conforms to the axioms of a group by proving the operation to be associative, demonstrating inverses, and producing the identity.

We can make an analogy, where the axioms defining a group are the analogue to a specification, and any particular group is analogous to a specification-compliant implementation. Indeed, in ITP-based verification, these are in actuality the same practice: a specification is a list of propositions (axioms), and an implementation is a well-defined mathematical object which may or may not satisfy those propositions.

We might, then, resolve the tension of specification in an ITP-based formal setting as mathematicians do: study, refine, and minimize the required axioms (specification) by proving theorems about the axioms and studying their formal implications. For this, we have the metaspecification.

B. Metaspecification

Given a specification, its *metaspecification* is a set of properties either of the specification itself or of the implications of that specification. They are typically properties intended by, but out of scope of, the specification—hence their name *meta properties*. For example, one can prove the specification to be consistent by providing a specification-compliant implementation [65]. Proving completeness is more nuanced, as we need to justify that the specification (axiomatization) correctly captures the intended behaviors.

To do so, we study the implications of a specification. Given a specification `S`, stated as a predicate on contracts, we consider an arbitrary contract `C` and a proof

```
C_conforms_to_S : S(C).
```

By assuming only the witness `C_conforms_to_S` in our context, any theorems we prove apply to all contracts satisfying the specification `S`.

A specification's correctness depends on whether it actually exhibits the intended meta properties. As we will see by example in the upcoming section, a metaspecification can include desired, high-level cryptoeconomic properties. Importantly, proving properties via the metaspecification does not add to the burden of verifying any given implementation, since by definition contracts conforming to a correct specification automatically inherit all the properties of the metaspecification.

For example, consider the standard specification of an ERC20 token contract [50], [62], which defines contract storage, interface, and functionality for a basic token contract. In addition to this standard, every token contract has an associated *tokenomics*, which includes rules governing minting, burning, token issuance or buy-backs, maximum supply, *etc.* [29] A token contract's tokenomics are essential to its correct functionality, since tokens typically attempt to capture value of some kind or regulate the functionality of some other smart contract, *e.g.* the governance tokens for a DAO [64], or the LP tokens for an AMM [67].

Within the framework of axiomatization and metaspecification, we can study a token contract specification by formalizing it as a predicate `P` on contracts and then stating and proving properties relevant to its tokenomics. The specification minimally includes specific rules governing minting and burning, including a maximum supply of tokens (if any). The metaspecification then might include some set of game-theoretic or incentive-based rules governing minting and burning hold, *e.g.* as articulated in [9], proving that the token contract conforms to some given tokenomics. Since the specification languages for ITP-based verification can state arbitrary properties, in principle we could state and attempt to prove anything we wish [54].

Indeed, we might wish to formalize a theory of DeFi and AMMs, a formal counterpart of previous work on the subject by Bartoletti *et al.* [10] and Angeris *et al.* [5]. Bartoletti *et al.* formally derive and prove desirable, high-level, economic properties of AMMs via a labelled state transition system. This work targets the so-called *arbitrage problem*, formally proving that the pricing functions of Uniswap-style AMMs respond, from an economic point of view, appropriately to market actions by rational arbitrageurs. In particular, this is a property explicitly aimed for by the earliest AMM specifications (*e.g.* [18]) for the sake of market efficiency, but to our knowledge has never actually featured in an AMM's specification.

By way of an illustrating example in the following section, our argument is that ITP-based formal methods should consider smart contracts analogously to axiomatized, mathematical objects. Returning to the mathematical analogy, in mathematics, like in formal specification, a set of axioms must be consistent, in that they do not imply a contradiction, and complete, in that they correctly characterize the intended mathematical phenomenon [51]. That the group axioms are correct is confirmed by the emergent behavior of groups, explored mathematically through the resulting theory. Importantly, the axioms of groups were carefully chosen to say enough to capture the intended mathematical structure without overspecifying—precisely the same tension exhibited in specification. To this end we proceed with an example of a formalized AMM specification and metaspecification.

V. Example: Formalizing Structured Pools

We illustrate the process and utility of axiomatization and metaspecification with a specific AMM contract. We have formalized[1] the specification and metaspecification of a structured pool contract [56], an AMM designed to pool and trade tokenized carbon credits [57]. We also verify the formal specification to be correct with regards to a metaspecification consisting of cryptoeconomic properties. Aside from showing the AMM specification to exhibit desirable, high-level cryptoeconomic properties, we also show that parts of the formal specification can only be derived in reference to the metaspecification.

A. The Formal Specification, or Contract Axiomatization

The structured pool specification, given in mathematically precise detail in [56], is an AMM specification split in three parts: contract storage, interface, and entrypoint functions. The first two are type specifications, which we handle in Coq by way of typeclasses. The last are functional specifications, which we can write using pre- and post-conditions. The formal specification can then be summarized into a predicate on an arbitrary contract `C`,

```
is_structured_pool : forall C, Prop.
```

A proof of `is_structured_pool` indicates that the storage, interface, and entrypoint functions of `C` all conform to the specification.

1) Storage: According to the specification, contract storage must contain the following data: exchange rates for each constituent token (used for pooling and trading rates), the quantity of each token held in the pool, the address of the pool token contract, and the number of outstanding pool tokens. We can specify this by using a Coq typeclass, requiring that the storage type of a structured pool contract have functions which reveal each of these data points.

```
Class State_Spec (T : Type) := {
    (* the exchange rates *)
    stor_rates : T → FMap token exchange_rate ;
    (* token balances *)
    stor_tokens_held : T → FMap token N ;
    (* pool token data *)
    stor_pool_token : T → token ;
    (* number of outstanding pool tokens *)
    stor_outstanding_tokens : T → N ;
 }.
```

Listing 1. The formal typeclass characterizing the storage type.

2) Interface: The interface consists of at least three entrypoints: `POOL`, `UNPOOL`, and `TRADE`. These are for pooling liquidity, withdrawing (unpooling) liquidity, and trading individual carbon credits, respectively. We formalize the payload data for each entrypoint into three types:

- `pool_data`, the payload type for `POOL`,
- `unpool_data`, the payload type for `UNPOOL`,
- `trade_data`, the payload type for `TRADE`, and
- `other_entrypoint`, an abstract type representing one, many, or no additional entrypoints.

The typeclass characterizing the interface then requires that each of these types are legitimate payload types.

```
Class Msg_Spec (T : Type) :=
 build_msg_spec {
   pool : pool_data → T ;
   unpool : unpool_data → T ;
   trade : trade_data → T ;
   (* any other potential entrypoints *)
   other : other_entrypoint → option T ;
}.
```

Listing 2. The typeclass characterizing the interface type.

We also require that these be exhaustive, simulating the interface as an inductive type.

[1] For the full formalization, see https://tinyurl.com/fincert-structured-pool.

```
Definition msg_destruct contract :=
  forall (m : Msg),
  (exists p, m = pool p) ∨
  (exists u, m = unpool u) ∨
  (exists t, m = trade t) ∨
  (exists o, Some m = other o).
```

Listing 3. The payload of any legitimate contract call is the image of one of: `pool`, `unpool`, `trade`, or `other`.

3) Entrypoint Functions: Entrypoint functions are characterized with functional specifications. There are twenty-four properties of the full entrypoint specification, encoded as propositions. Some of the key properties are:

1) `pool_increases_tokens_held`, which states that a successful call to `POOL` increases the tokens pooled,
2) `unpool_decreases_tokens_held`, which states that a successful call to `UNPOOL` decreases the tokens pooled,
3) `trade_pricing_formula`, which specifies the formula used to price trades, and
4) `trade_update_rates_formula`, which specifies how exchange rates update in response to trades.

The last two are parameterized by functions that calculate trades and update exchange rates, respectively `calc_delta_y` and `calc_rx'`. This is all we need to fully specify the AMM in question, but there are two ambiguities in the formal specification which can only be clarified by the metaspecification.

The first relates to how trades are priced, specified in the pricing formula of `trade_pricing_formula`. As is typical in prose specifications of AMMs that price trades along a convex curve, or indeed for any financial contract involving mathematical calculations, the structured pool specification does arithmetic in rational or real numbers. However, any implementation necessarily uses arithmetic with natural numbers which estimate rational or real numbers (typically at 6 or 9 decimal points of precision) [58]. We must decide, then, whether to estimate from above, below, or some combination of the two depending on the context. At the heart of the question is how to estimate the calculations in such a way that all the desired cryptoeconomic behaviors of the contract are satisfied. This is thus a question for the metaspecification.

The second is the functional specification of the `other` entrypoint, which is a placeholder in the specification for any entrypoints other than the three explicitly specified. The structured pool specification allows for other entrypoints, but none that fundamentally change the functionality of the contract. However, this is only an intuitive requirement, difficult to formalize. From the specification it is not obvious what functionality is and is not permitted of any other entrypoints. We must restrict the `other` entrypoint so that any additional entrypoints, whether they be to add a governance layer or something more inocuous like an entrypoint for updating metadata, do not sabotage the contract's correct cryptoeconomic behavior. Again, we can answer this within the context of the metaspecification, enabling us to give a precise functional specification of the `other` entrypoint.

B. The Formal Metaspecification

The metaspecification consists of six cryptoeconomic properties derived from previous work which elucidate desirable economic behavior of AMMs [3], [5], [10], [68]. The properties we have formalized here are those proved in the original, informal AMM specification [56], and are designed to justify the AMM specification to be cryptoeconomically correct. Informally, these are:

1) *Demand sensitivity*: in a trade, the relative price of the token traded in decreases, and that of the token being traded out increases, simulating the principle of supply and demand from classical economics.
2) *Nonpathological prices*: the price of an asset can never reach zero or go negative.
3) *Swap rate consistency*: trading cost must be nonnegative, so that it is impossible to make a sequence of calls to the `TRADE` entrypoint and output more in assets than were traded in initially.
4) *Zero-impact liquidity change*: pooling or unpooling tokens (depositing or withdrawing liquidity) must not affect trade prices.
5) *Arbitrage sensitivity*: if the price of a token differs on an external AMM from this one, a rational arbitrageur will either equalize the prices by trading, or drain the structured pool of that token.
6) *Pooled consistency*: the total value of the outstanding pool tokens is equal to the value of the pool.

Together, these properties are designed to encapsulate the intended cryptoeconomic behavior for this AMM [56]. In particular, demand and arbitrage sensitivity target the desired property that the AMM facilitate an efficient (*i.e.* price-finding) market. Swap rate consistency ensures that there are no arbitrage opportunities internal to the AMM itself. Pooled consistency and nonpathological prices are the invariants of the contract state, while the rest pertain to specific entrypoint functions.

To illustrate, see the formalized statements of properties 1 (nonpathological prices) and 6 (pooled consistency) in the following two figures. The correspondence between Coq code and prose is illustrated in the comments.

```
Theorem nonpathological_prices bstate caddr :
  (* Forall reachable states with
     our contract at the address caddr, *)
  reachable bstate →
  env_contracts bstate caddr =
  Some (contract : WeakContract) →
  (* ... where contract state is cstate, *)
  exists (cstate : State),
  contract_state bstate caddr = Some cstate ∧
  (* For a token t_x in T and rate r_x, *)
  forall t_x r_x,
  (* if r_x is the exchange rate of t_x,
   then r_x > 0 *)
  FMap.find t_x (stor_rates cstate) =
  Some r_x → r_x > 0.
```

Listing 4. The formalization of Property 2, Nonpathological Prices.

```
Theorem pooled_consistency bstate caddr :
   reachable bstate →
   env_contracts bstate caddr =
   Some (contract : WeakContract) →
   exists (cstate : State),
   contract_state bstate caddr = Some cstate ∧
   (* The sum of all the constituent,
      pooled tokens, multiplied by
      their value in terms of pooled tokens,
      always equals the total number of
      outstanding pool tokens. *)
   suml (tokens_to_values
     (stor_rates cstate)
     (stor_tokens_held cstate)) =
   (stor_outstanding_tokens cstate).
```

Listing 5. The formalization of Property 6, Pooled Consistency.

To our knowledge, these types of economic properties do not feature in any other contract specifications, informal or formal, but as we have pointed out they are critical to evaluating the correctness of the specification with respect to our cryptoeconomic intent. That they are formally verified to be true of the structured pool specification assures us that the design itself is correct. Importantly, any contract satisfying the functional specification of §V-A also satisfies these economic properties without requiring any further proofs.

Furthermore, the two ambiguities in the formal specification of §V-A can only be clarified in the context of the specification's cryptoeconomic properties. These are: verifying the pricing formulae to be correct using natural-number arithmetic, rather than rational or real numbers; and formally specifying minimal requirements on any additional entrypoints such that the economic properties of the metaspecification are not violated. We expound on both.

1) Rational to natural-number arithmetic: The aspects of the informal specification [56] which require the metaspecification due to the fact that smart contracts use natural-number, rather than rational, arithmetic are these: first, how trades are priced, and second, how token exchange rates are updated by trades. Both of these implicitly use properties of rational numbers which are not true of natural numbers: that between 0 and any positive rational number r, there are infinitely many rational numbers, and that every nonzero rational number has an inverse. See in particular Section 3 and Figure 1 of the structured pool specification [56], which specifies how trades are to be calculated.

Because any implementation necessarily uses natural numbers for arithmetic, in the formal, functional specification of the trade and exchange rate functions we must decide which properties of rational arithmetic must be preserved in our formalization into natural-number arithmetic. In the structured pool's formal specification, this resulted in seven formal properties on the abstract functions `calc_delta_y` and `calc_rx'` which are, respectively, the functions that price trades and update token exchange rates. These include theoretical bounds on trade slippage, exchange rates, and that there be no theoretical upper bound on the output of trades. The specification allows for any pricing and rate-updating formulae which conform to those seven formal properties.

```
(* ... *)
(* specification of calc_rx', calc_delta_y *)
update_rate_stays_positive ∧
rate_decrease ∧
rates_balance ∧
rates_balance_2 ∧
trade_slippage ∧
trade_slippage_2 ∧
arbitrage_lt ∧
arbitrage_gt ∧
(* ... *)
```

Listing 6. An exerpt of the formal specification of a structured pool contract consisting of the required properties of `calc_rx'` and `calc_delta_y`.

Importantly, this shows that the solutions to issues such as rounding errors in calculating trades and exchange rates have solutions from within a *cryptoeconomic* context. This is particularly relevant considering recent costly attacks due to rounding error in smart contracts, *e.g.* DFX Finance [38] and KyberSwap [21].

2) Specifying the `other` *entrypoint:* The metaspecification also governs the behavior of any additional entrypoints, such as one for a governance mechanism or something more inocuous like for updating metadata. Two properties—nonpathological prices and pooled consistency—are high-level invariants of the contract, in contrast with the other properties of the metaspecification which are entrypoint-specific. In particular, they are the only invariants on contract state, so they dictate the admissible behavior of any additional entrypoint: We retain the desired cryptoeconomic behavior of our AMM so long as no additional entrypoint does not push prices to a nonpositive value, or make the total value of outstanding pool tokens unequal to the value of the pool. For example, a specification that requires that any additional entrypoints not alter rates, token balances, or outstanding pool tokens satisfies the metaspecification, though the metaspecification may allow for more varied entrypoint behavior.

```
(* ... *)
(* specification of all other entrypoints *)
other_rates_unchanged C ∧
other_balances_unchanged C ∧
other_outstanding_unchanged C ∧
(* ... *)
```

Listing 7. An exerpt of the formal specification of a structured pool contract consisting of the required properties of any additional, unspecified entrypoint.

VI. (In)Correct Contract Specifications

From our example we can observe various benefits to formalizing contract specifications and metaspecifications.

1) Formally specifying the high-level properties intended by the specification gives the benefits of clarity and rigor inherent to formalization, analogous to the benefits of formalizing a specification on an implementation.
2) The metaspecification can inform, and evolve with, the specification, just as the specification does with an implementation.

3) Choices inevitably made when formalizing a specification can be proved correct with reference to a metaspecification.
4) Once formalized, the metaspecification adds to the security guarantees of the formal specification without increasing the burden of formally verifying any particular implementation, since an implementation proved correct with regards to the specification inherits the properties of the metaspecification without requiring additional proof.

In particular, the metaspecification achieves our goal to develop rigorous tools for reasoning about the correctness of smart contract specifications in an ITP-based formal setting: It forces us to formalize the contract specification as a standalone mathematical object, and then to clearly and formally articulate the intended properties of the specification and contract design. Its efficacy will depend on the sophistication with which we are able to state and verify cryptoeconomic properties, which leads us to current limitations and future work.

VII. Limitations and Future Work

The example given here is preliminary and illustrative. In order to more fully realize these benefits we should lay stronger foundations from which to derive desirable cryptoeconomic properties of smart contracts. We might also consider similar work in other formal settings.

A. Formal Theories of DeFi and AMMs

We mentioned before that substantial work has already been done to develop theories of DeFi and AMMs. The metaspecification of this paper was informed by the work of Angeris *et al.* [3], [5], Bartoletti *et al.* [10], and Xu *et al.* [68] to characterize the desirable cryptoeconomic properties of AMMs which price trades along a convex curve. However, rather than rigorously deriving them from within a theory if DeFi and AMMs embedded into ConCert, we formalized the statements of the metaspecification ourselves. The process of metaspecification could be made more rigorous if we had a formalized theory of DeFi and AMMs from which to derive our desired cryptoeconomic properties.

The cited studies are not the only attempts to systematically study the cryptoeconomic behavior of blockchains and smart contracts. There is a growing literature on cryptoeconomics more generally, *e.g.* [41], [42], [63]. We are hopeful that the growing literature will provide strong, theoretical foundations of cryptoeconomics which can be applied to specification design and verification.

To aid in the rigorous formation of contract metaspecifications, we hope to start from first principles and develop a Coq-native cryptoeconomic theory in ConCert, which fomralizes token and AMM primitives as abstract specifications [10], and operations for owning, transferring, and trading resources [15]. This is doable in ConCert because it models the semantics of a blockchain embedded in Coq, and so arbitrary theories can be constructed, including those that reason about the cryptoeconomic incentives relating to the blockchain itself. From such a theory we could make a formal study of cryptoeconomics, and provide strong foundations for contract metaspecifications.

B. Extensions to other Formal Settings

Most ITP-based smart contract verification tools only provide an embedding of the smart contract language [30], [47], [60]. There would be no issue in stating the meta properties that we gave in §V in these settings, since we did not meaningfully draw on the semantics of the blockchain to derive the statements. However, since we know that the incentives of block producers have an effect on the cryptoeconomic security of blockchains [22], any verification pipeline which does not model the semantics of an executing blockchain has inevitable limitations with regards to the cryptoeconomic meta properties of a smart contract that it is able to state and verify. Future work might also include making a formal study of the limits of these language embeddings with regards to contract meta properties.

VIII. Conclusion

Poorly specified smart contracts are vulnerable to attacks on faulty design for which formal methods typically have no answer. We are in need of a paradigm shift in how we specify and verify contracts so that we can rigorously consider a contract specification's correctness.

We propose a framework for formal specification in interactive theorem provers consisting of contract axiomatization and metaspecification. This framework treats contracts as well-defined mathematical objects, and contract specifications as the axiomatization of a mathematical theory. Our aim was to increase the expressiveness and rigor of ITP-based formal methods, enabling the expression and verification of meta properties.

We illustrated with an example, formal specification of an AMM. Not only were we able to describe high-level, cryptoeconomic properties that target market efficiency and arbitrage, but we showed that a metaspecification can shed light on choices made in the formalization of the specification and justify their correctness.

We hope that this work leads to a more rigorous and formal understanding of the cryptoeconomic properties of smart contracts, which in turn can help us mitigate the near-ubiquitous cryptoeconomic vulnerabilities in contract design.

References

[1] 20squares. https://20squares.xyz/. Accessed December 2023.

[2] Gauntlet. https://www.gauntlet.xyz/. Accessed December 2023.

[3] Guillermo Angeris, Akshay Agrawal, A. Evans, T. Chitra, and Stephen P. Boyd. Constant Function Market Makers: Multi-Asset Trades via Convex Optimization. 2021.

[4] Guillermo Angeris, Tarun Chitra, and Alex Evans. When Does The Tail Wag The Dog? Curvature and Market Making. *Cryptoeconomic Systems*, 2(1), June 2022.

[5] Guillermo Angeris, Hsien-Tang Kao, Rei Chiang, Charlie Noyes, and Tarun Chitra. An Analysis of Uniswap markets. *Cryptoeconomic Systems*, 0(1), April 2021.

[6] Danil Annenkov, Mikkel Milo, Jakob Botsch Nielsen, and Bas Spitters. Extracting smart contracts tested and verified in Coq. In *Proceedings of the 10th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2021, pages 105–121, New York, NY, USA, January 2021. Association for Computing Machinery.

[7] Danil Annenkov, Jakob Botsch Nielsen, and Bas Spitters. ConCert: A smart contract certification framework in Coq. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2020, pages 215–228, New York, NY, USA, January 2020. Association for Computing Machinery.

[8] b. Dexter2 Specification. https://gitlab.com/dexter2tz/dexter2tz/-/blob/master/docs/informal-spec/dexter2-cpmm.md. Accessed November 2023.

[9] Julian Barreiro-Gomez and Hamidou Tembine. Blockchain token economics: A mean-field-type game perspective. *IEEE Access*, 7:64603–64613, 2019.

[10] Massimo Bartoletti, James Hsin-yu Chiang, and Alberto Lluch-Lafuente. A Theory of Automated Market Makers in DeFi. In Ferruccio Damiani and Ornela Dardha, editors, *Coordination Models and Languages*, Lecture Notes in Computer Science, pages 168–187, Cham, 2021. Springer International Publishing.

[11] Chaïmaa Benabbou and Önder Gürcan. A survey of verification, validation and testing solutions for smart contracts. In *2021 Third International Conference on Blockchain Computing and Applications (BCCA)*, pages 57–64. IEEE, 2021.

[12] Beosin. Global Web3 Security Report 2022. https://medium.com/Beosin_com/beosin-global-web3-security-report-2022-7aa2e4bb13. Accessed November 2023.

[13] Daniel M Berry. Formal methods: the very idea: Some thoughts about why they work when they work. *Science of computer Programming*, 42(1):11–27, 2002.

[14] J.P. Bowen and M.G. Hinchey. Ten commandments of formal methods. *Computer*, 28(4):56–63, 1995.

[15] Christian Bräm, Marco Eilers, Peter Müller, Robin Sierra, and Alexander J Summers. Rich specifications for ethereum smart contract verification. *Proceedings of the ACM on Programming Languages*, 5(OOPSLA):1–30, 2021.

[16] BscScan.com. Pancake Bunny Exploiter. Address 0x158c244b62058330f2c328c720b072d8db2c612f, 2021.

[17] BscScan.com. Spartan Protocol Exploit. Transaction 0xb64ae25b0d836c25d115a9368319902c972a0215bd108ae17b1b9617dfb93af8, 2021.

[18] Vitalik Buterin. Improving front running resistance of x*y=k market makers - Decentralized exchanges. https://ethresear.ch/t/improving-front-running-resistance-of-x-y-k-market-makers/1281, March 2018. Accessed November 2023.

[19] Raphael Cauderlier. Dexter2 Specification (Mi-Cho-Coq). https://gitlab.com/nomadic-labs/mi-cho-coq/-/blob/dexter-verification/src/contracts_coq/dexter_spec.v. Accessed November 2023.

[20] J Craig Cleaveland. Mathematical specifications. *ACM SIGPLAN Notices*, 15(12):31–42, 1980.

[21] Tim Copeland. Dex protocol kyberswap appears to lose $47 million in possible exploit. https://www.theblock.co/post/264432/dex-protocol-kyberswap-appears-to-lose-47-million-in-possible-exploit. Accessed December 2023.

[22] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash Boys 2.0: Frontrunning in Decentralized Exchanges, Miner Extractable Value, and Consensus Instability. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 910–927, May 2020.

[23] etherscan.io. Beanstalk Exploit. Transaction 0xcd314668aaa9bbfebaf1a0bd2b6553d01dd58899c508d4729fa7311dc5d33ad7.

[24] Alex Evans, Guillermo Angeris, and Tarun Chitra. Optimal Fees for Geometric Mean Market Makers. In *Financial Cryptography and Data Security. FC 2021 International Workshops*, Lecture Notes in Computer Science, pages 65–79, Berlin, Heidelberg, 2021. Springer.

[25] James H Fetzer. Program verification: The very idea. *Communications of the ACM*, 31(9):1048–1063, 1988.

[26] Luciano Floridi. *The Blackwell guide to the philosophy of computing and information*. John Wiley & Sons, 2008.

[27] Robin Fritsch and Roger Wattenhofer. A Note on Optimal Fees for Constant Function Market Makers. *DeFi@CCS*, 2021.

[28] Robin Frtisch, Samuel Käser, and Roger Wattenhofer. The economics of automated market makers. In *Proceedings of the 4th ACM Conference on Advances in Financial Technologies*, pages 102–110, 2022.

[29] Jingxing Gan, Gerry Tsoukalas, and Serguei Netessine. Decentralized platforms: Governance, tokenomics, and ico design. *Management Science*, 2023.

[30] Ikram Garfatta, Kais Klai, Walid Gaaloul, and Mohamed Graiet. A Survey on Formal Verification for Solidity Smart Contracts. In *2021 Australasian Computer Science Week Multiconference*, ACSW '21, pages 1–10, New York, NY, USA, February 2021. Association for Computing Machinery.

[31] Neil Ghani, Jules Hedges, Viktor Winschel, and Philipp Zahn. Compositional game theory. In *Proceedings of the 33rd annual ACM/IEEE symposium on logic in computer science*, pages 472–481, 2018.

[32] Joseph A Goguen. More thoughts on specification and verification. *ACM SIGSOFT Software Engineering Notes*, 6(3):38–41, 1981.

[33] Samuel Haig. PancakeBunny tanks 96% following $200M flash loan exploit. https://cointelegraph.com/news/pancakebunny-tanks-96-following-200m-flash-loan-exploit, May 2021.

[34] Anthony Hall. Seven myths of formal methods. *IEEE software*, 7(5):11–19, 1990.

[35] Lioba Heimbach, Ye Wang, and Roger Wattenhofer. Behavior of Liquidity Providers in Decentralized Exchanges. arXiv:2105.13822, October 2021.

[36] Robert M. Hierons, Kirill Bogdanov, Jonathan P. Bowen, Rance Cleaveland, John Derrick, Jeremy Dick, Marian Gheorghe, Mark Harman, Kalpesh Kapoor, Paul Krause, Gerald Lüttgen, Anthony J. H. Simons, Sergiy Vilkomir, Martin R. Woodward, and Hussein Zedan. Using formal specifications to support testing. *ACM Comput. Surv.*, 41(2), feb 2009.

[37] Igor Igamberdiev (@FrankResearcher). BUNNY Exploit Report. https://twitter.com/FrankResearcher/status/1395196961108774915, May 2021. Accessed November 2023.

[38] Immunefi. Dfx finance rounding error bugfix review. https://medium.com/immunefi/dfx-finance-rounding-error-bugfix-review-17ba5ffb4114. Accessed December 2023.

[39] PeckShield Inc. The Spartan Incident: Root Cause Analysis. https://peckshield-94632.medium.com/the-spartan-incident-root-cause-analysis-b14135d3415f, May 2021. Accessed November 2023.

[40] Runtime Verification Inc. Dexter2 Specification (K Framework). https://github.com/runtimeverification/michelson-semantics/blob/a46be4a542e01b17a93134395c889df1468a067b/tests/proofs/dexter/dexter-spec.md. Accessed November 2023.

[41] Daniel Kirste, Niclas Kannengießer, Ricky Lamberty, and Ali Sunyaev. How automated market makers approach the thin market problem in cryptoeocnomic systems. *arXiv preprint arXiv:2309.12818*, 2023.

[42] Samela Kivilo. *Designing a Token Economy: Incentives, Governance and Tokenomics*. PhD thesis, 06 2023.

[43] Ralf Kneuper. Limits of formal methods. *Formal Aspects of Computing*, 9:379–394, 1997.

[44] Barbara Liskov and Stephen Zilles. Specification techniques for data abstractions. In *Proceedings of the international conference on Reliable software*, pages 72–87, 1975.

[45] Shaurya Malwa. How Market Manipulation Led to a $100M Exploit on Solana DeFi Exchange Mango. https://www.coindesk.com/markets/2022/10/12/how-market-manipulation-led-to-a-100m-exploit-on-solana-defi-exchange-mango/, October 2022.

[46] Paul R. McMullin and John D. Gannon. Combining testing with formal specifications: A case study. *IEEE Transactions on Software Engineering*, (3):328–335, 1983.

[47] Yvonne Murray and David A. Anisi. Survey of Formal Verification Methods for Smart Contracts on Blockchain. In *2019 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1–6, June 2019.

[48] Eske Hoy Nielsen, Danil Annenkov, and Bas Spitters. Formalising Decentralised Exchanges in Coq. In *Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 290–302, 2023.

[49] pancakebunny.finance (@PancakeBunnyFin). BUNNY Exploit Report. https://twitter.com/PancakeBunnyFin/status/1395173389208334342, May 2021. Accessed November 2023.

[50] Daejun Park, Yi Zhang, Manasvi Saxena, Philip Daian, and Grigore Roşu. A formal verification tool for ethereum vm bytecode. In *Proceedings of the 2018 26th ACM joint meeting on european software*

engineering conference and symposium on the foundations of software engineering, pages 912–915, 2018.

[51] Charles Parsons. Informal axiomatization, formalization and the concept of truth. *Synthese*, pages 27–47, 1974.

[52] Macauley Peterson. Latest DeFi exploits show audits are no guarantee. https://blockworks.co/news/audits-cannot-guarantee-defi-exploits. Accessed November 2023.

[53] Siraphob Phipathananunth. Using Mutations to Analyze Formal Specifications. In *Companion Proceedings of the 2022 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity*, SPLASH Companion 2022, pages 81–83, New York, NY, USA, December 2022. Association for Computing Machinery.

[54] John Rushby. Theorem proving for verification. In *Summer School on Modeling and Verification of Parallel Processes*, pages 39–57. Springer, 2000.

[55] solscan.io. Mango Markets Exploiter. https://solscan.io/account/CQvKSNnYtPTZfQRQ5jkHq8q2swJyRsdQLcFcj3EmKFfX.

[56] Derek Sorensen. Structured pools for tokenized carbon credits. In *2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–6. IEEE, 2023.

[57] Derek Sorensen. Tokenized carbon credits. *Ledger*, 9, 2024.

[58] Tianyu Sun and Wensheng Yu. A formal verification framework for security issues of blockchain smart contracts. *Electronics*, 9(2):255, 2020.

[59] John Symons and Jack K Horner. Why there is no general solution to the problem of software verification. *Foundations of Science*, 25:541–557, 2020.

[60] Palina Tolmach, Yi Li, Shang-Wei Lin, Yang Liu, and Zengxiang Li. A Survey of Smart Contract Formal Specification and Verification. *ACM Comput. Surv.*, 54(7):148:1–148:38, July 2021.

[61] Palina Tolmach, Yi Li, Shang-Wei Lin, Yang Liu, and Zengxiang Li. A survey of smart contract formal specification and verification. *ACM Computing Surveys (CSUR)*, 54(7):1–38, 2021.

[62] Fabian Vogelsteller and Vitalik Buterin. Erc-20 token standard. https://github.com/ethereum/ercs/blob/master/ERCS/erc-20.md. Accessed December 2023.

[63] Shermin Voshmgir, Michael Zargham, et al. Foundations of cryptoeconomic systems. *Research Institute for Cryptoeconomics, Vienna, Working Paper Series/Institute for Cryptoeconomics/Interdisciplinary Research*, 1, 2019.

[64] Shuai Wang, Wenwen Ding, Juanjuan Li, Yong Yuan, Liwei Ouyang, and Fei-Yue Wang. Decentralized autonomous organizations: Concept, model, and applications. *IEEE Transactions on Computational Social Systems*, 6(5):870–878, 2019.

[65] Jeannette M Wing. A specifier's introduction to formal methods. *Computer*, 23(9):8–22, 1990.

[66] Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui, and John Fitzgerald. Formal methods: Practice and experience. *ACM computing surveys (CSUR)*, 41(4):1–36, 2009.

[67] Jiahua Xu, Krzysztof Paruch, Simon Cousaert, and Yebo Feng. Sok: Decentralized exchanges (dex) with automated market maker (amm) protocols. *ACM Computing Surveys*, 55(11):1–50, 2023.

[68] Jiahua Xu, Krzysztof Paruch, Simon Cousaert, and Yebo Feng. SoK: Decentralized Exchanges (DEX) with Automated Market Maker (AMM) Protocols. *ACM Comput. Surv.*, 55(11):238:1–238:50, February 2023.

[69] Joachim Zahnentferner, Dmytro Kaidalov, Jean-Frédéric Etienne, and Javier Díaz. Djed: A formally verified crypto-backed autonomous stablecoin protocol. In *2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–9. IEEE, 2023.

[70] Hans J Zassenhaus. *The theory of groups*. Courier Corporation, 2013.

[71] Yi Zhang, Xiaohong Chen, and Daejun Park. Formal specification of constant product (xy= k) market maker model and implementation. *White paper*, 2018.

[72] Liyi Zhou, Xihan Xiong, Jens Ernstberger, Stefanos Chaliasos, Zhipeng Wang, Ye Wang, Kaihua Qin, Roger Wattenhofer, Dawn Song, and Arthur Gervais. SoK: Decentralized Finance (DeFi) Attacks, April 2023.