Journal of COMPUTER AND FORENSIC SCIENCES

CFS 2023, Vol. 2, Issue 1
https://doi.org/10.5937/jcfs2-44362
Original research paper

A verifiable model of a minimal market operating sequentially, with price and time discrete

Dragiša D. Žunić[*1]

[1] The Institute for Artificial Intelligence R&D of Serbia

Corresponding author: dragisa.zunic@ivi.ac.rs

Received: April 9, 2023 • Accepted: April 28, 2023 • Published: June 19, 2023

Abstract: This research presents a minimal computational market model, i.e., a model of a trading venue, with sequential order matching, in a declarative style, and proceeds to demonstrate how some fundamental properties can be formally proved. It is a challenging task to formally certify the properties of a fundamental system in any realm of human endeavor, especially of systems with infinite state space. With the recent development of theoretical frameworks based on formal logic, it is now possible (albeit very difficult) to both formalize and reason about an object system in the same language. This research derives from the previous research presented in [1], and represents a simplification to obtain a minimal model. The computational model of a minimal market, presented here in a declarative style, is important from the perspective of both market design and verification.

Keywords: formal logic; market design; financial exchanges; automated reasoning; logical frameworks; computational models.

1. INTRODUCTION

A financial exchange is a platform where buyers and sellers can come together to trade various financial instruments. These instruments include stocks, bonds, commodities, currencies, and derivatives. Financial exchanges provide a central marketplace where market participants can buy and sell financial instruments with each other based on their respective prices. The computational core of a financial exchange is the order matching engine, which handles interaction between buy and sell flows of orders. In order to guarantee trading fairness, exchanges must meet the requirements of regulatory bodies, in addition to numerous general requirements. However, both specifications and requirements are presented in natural language, which hardly qualifies as an adequate method for such a task.

Based on the previous experience, violations frequently originate either from interactions between order types, or from the way matching logic is specified and implemented [2, 3]. Formalization and formal reasoning can play a big role in mitigating these problems. They provide methods to verify properties of complex and infinite state space systems with certainty and have already been applied in fields ranging from hardware design to flight safety and financial contracts [4, 5], with trading systems being considered recently as well [6, 7].

Citation: D. Žunić, "A verifiable model of a minimal market operating sequentially, with price and time discrete," Journal of Computer and Forensic Sciences, 2(1), https://doi.org/10.5937/jcfs2-44362.

A general sequential order matching core has been formalized in [1], followed by proving properties such as: the trade always takes place at either bid or ask; the market is never in a locked or crossed state; and order priority is never violated. Everything is based on being able to declaratively represent an archetypal sequential (matching in one-by-one fashion) trading system, using only symbols and symbol manipulation, which provides a setting for verification, i.e., some form of semi-automated reasoning about the system's properties.

2. LINEAR LOGIC AND LOGICAL FRAMEWORKS

Linear logic (LL) is a logical system that was introduced by Girard in the 1980s [8]. One of the key features of linear logic is that it has a notion of resources, which allows it to capture more accurately certain aspects of computation and reasoning. In other words, linear logic is a resource conscious logic, and formulas are consumed when used to prove a statement. This is achieved by having structural rules of contraction and weakening explicit in the system, and being able to selectively mark formulas intended to be an unbounded resource by the exponential operator !. Intuitionistic linear logic (ILL) is the restriction of linear logic to the intuitionistic fragment. Formulas in (propositional) ILL are composed of the following connectives: $\otimes$ and 1 (multiplicative conjunction and its neutral element), & and $\top$ (additive conjunction and its neutral element), $\oplus$ and 0 (disjunction and its neutral element), $\multimap$ (linear implication), $\to$ (intuitionistic implication), ! (exponential).

The logical framework CLF (Concurrent Linear Framework) [9] is based on a fragment of intuitionistic linear logic. It extends the traditional LF [10] framework with the linear connectives $\multimap$, &, $\top$, $\oplus$, 1 and ! to obtain a resource-aware framework with a satisfactory representation of concurrency. The rules of the system impose a discipline on when the (less deterministic) connectives $\otimes$, 1 and ! are decomposed, thus still retaining enough determinism to allow for the implementation of a logical framework. For simplicity, we present only the logical fragment of CLF needed for our encoding. This is only a small fragment of the logical framework, but the need for $\otimes$ in our encodings indicates that anything less than CLF (e.g., LLF) would be less suitable.

The majority of the trading system encoding involves clauses in the following shape (for atomic p_i and q_i): $p_1 \otimes \ldots \otimes p_n \multimap \{q_1 \otimes \ldots \otimes q_m\}$. We used an implementation of this framework called Celf[1]. Following the tool's convention, variable names start with an upper-case letter.

3. ELECTRONIC TRADING SYSTEMS

As mentioned in the introduction, real life trading systems, both public exchanges and alternative trading systems, differ slightly in the way they manage orders. However, there is a certain common core that guides all those trading systems and embodies the market logic of trading on an exchange. A detailed account for formalization of a general sequential trading core, in a declarative style, is presented in [1].

1 https://clf.github.io/celf/

The default mode of operation of electronic trading systems globally is the *price/time priority*, although other modes of operation exist. Before we explain the mode of operation and different types of orders, let us introduce basic notions.

An *order* is an investor's instruction to a broker to buy or sell securities (or any asset type traded on a financial exchange), thus we have *buy orders* and *sell orders*. There are two basic types of orders: limit, which is short for limit-price, and market.

A *limit order* has a specific limit price at which it is willing to trade, meaning that it will trade at that price or better. In the case of a limit order to sell, a price limit price p means that the security will be sold at the best available price in the market, but no less than p. And symmetrically for buy limit orders.

A *market order* does not specify the price at which it is willing to trade, and will be immediately (if possible) matched against the best available price for this security.

Besides price, orders have a *quantity*, the amount of securities they are willing to trade. An order is identified by a timestamp, which records the time it enters the trading system. Orders, regardless of the type, are filled eagerly.

Outstanding orders, i.e., those that are waiting to be filled in the trading system, are called *resident orders*. There are two sorted lists that keep track of *active prices*, which are those for which there exists at least one resident limit order, namely *active buy prices* and *active sell prices*. Of particular importance is the maximum value of the active buy prices, called *bid*, and minimal value of active sell prices, called *ask*. The difference between those two prices is the *bid-ask spread*, or simply *spread*.

For each active price, there is a queue of resident orders, sorted by time of arrival (which identifies them uniquely): the order that arrived first is at the front of the queue whilst the last one is last in the queue.

There are numerous models of trading venues, however, there are some standard order types, such as limit, market, and immediate or cancel (IOC) orders, and basic matching rules. The current state of the art in trading venue design (somewhat surprisingly) assumes that orders for a given security enter the trading venue sequentially, one at a time, and they are executed sequentially. An order is filled, or exchanged, when it is successfully matched, regarding the price, against an opposite order (or orders), provided that the quantity of opposite orders was sufficient.

The standard mode of operation is *price/time priority*, which determines how orders are prioritized for execution. Orders are first ranked according to their price; orders of the same price are then ranked depending on when they entered the venue. Other than price/time priority, the most common is the pro-rata matching algorithm, which takes into account the overall volume of the incoming order as well as resident orders at a considered price, thus making the timestamp less important.

Some of the standard regulatory requirements for real world financial trading systems are: order priority is always respected, the system will not illegally prohibit any two orders from trading with each other, no crossed and locked markets (maximum buy price must remain strictly less than the minimum sell price), and transitivity of order ranking (order priority is transitive). It is a problem for financial companies that run trading platforms to guarantee these properties.

4. FORMALIZING A MINIMAL MARKET CORE

We present a computational model for a minimal market operating sequentially (orders are entering the market and are processed in one-by-one fashion). Unlike the general model presented in [1], we consider the following:

- All orders are limit-price[2],
- All orders have unit size quantities.

In this paper, we focus on the computational steps when buy orders are entering. The rules for sell orders are symmetric.

4.1. Properties of the system by design

The paradigm of logical frameworks enables us to think about the object system design, and with that in mind, we can formally prove the desirable properties. However, the object system formalization must be done a priori with those properties in mind.

Core properties. The computational market model presented here is designed to have the following required properties:

- The market is never in a crossed or locked state (at any given moment, bid is strictly less than ask);
- If the trade occurred, it happened at either bid or ask price (no trade occurs at a price other than the current bid or ask);
- Order priority is always respected (no computation step ever violates the order priority; given arbitrary two orders o_1, o_2, the following holds: if o_1 has a higher priority than o_2, then o_1 is filled before o_2);
- Order priority is transitive (given any three limit orders o_1, o_2 and o_3, the following holds: if o_1 has higher priority than o_2 and o_2 has a higher priority than o_3, then o_2 has higher priority than o_3);

Notice that the properties do not speak about players' strategies. These properties are core in the sense that they refer to the rules of the game, that is, the game itself. Once we extend the model to include the players participating, then these properties extend to that model as well. Namely, they can be stated in the form "Regardless of the players' strategies, the following holds...".

Additional properties. If we transition to a model that matches orders in batches, using a single market clearing price for each batch, and even allow more parallelism in the matching procedure[3], we find ourselves in a world that, besides the core properties, has numerous additional advantages from the perspective of economics and actual financial

2 Limit orders constitute the market in the price/time priority mode. Unlike, for example, market orders, which cannot become resident orders as they are either filled or canceled. Once we have the fundamentals, it is not difficult to extend the model with different order types, such as market orders, immediate-or-cancel, fill-or-kill, etc.

3 At the level of fundamental design, the key is to have some parallelism, together with discrete time and price.

markets. Namely, the following additional properties – as presented in the research introducing the frequent batch auctions model by Budish et al. in [2, 3]) – are satisfied:

- Competition on speed transformed into competition on price (pure time priority at entry is now closer to price/time priority);
- Race to the bottom in speed-advantage eliminated;
- Sniping stopped (a tax on a liquidity provision, hurting investors);
- Enhanced liquidity;
- Narrower bid-ask spread;
- A computationally simpler market (structured computational trace);
- Reestablished some form of the efficient market hypothesis.

4.2. Formalization: a minimal market core with unit-size orders

The big picture is provided by a typical depth chart representing the current state of the resident market, where we may notice L, M, and R segments. The two piles of resident orders are those orders that, at the time of entry, were not marketable and thus were stored in the market and not executed (filled). The pile on the left are instructions to buy, whereas the pile on the right are instructions to sell. The most competitive buy price, at this particular moment in time, is denoted as B (bid price), whereas the most competitive sell price is S (ask price). B and S divide the market into three segments, namely L, M and R, in that order. Segment M is the segment corresponding to the bid-ask spread, which in some cases may be non-existent, namely when bid and ask prices touch. See Image 1.

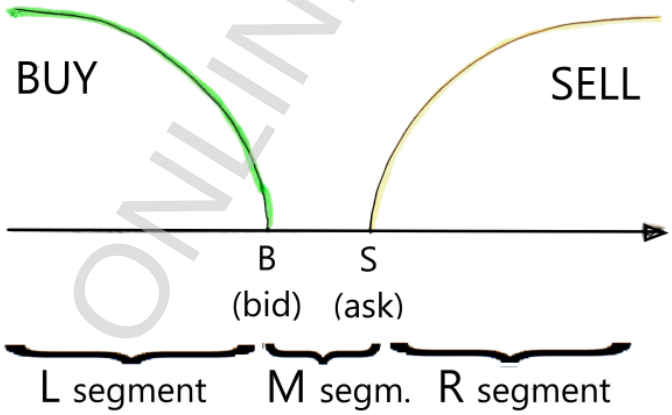

Image 1: *Market view (depth chart). Resident buy and sell orders are displayed. Market's bid and ask and consequently L, M, and R market segments are presented.*

The formalization initiates the model via the begin fact, creating facts actPrices(buy, nil) and actPrices(sell, nil) which, in what follows, will keep track of active prices (active in a sense that there is at least one resident order stored on that price). A fact time(z) is also initiated to keep track of the system time, with each computation step increasing the time counter.

$$\text{begin} \multimap \{\text{actPrices(buy, nil)} \otimes \text{actPrices(sell, nil)} \otimes \text{time(z)}\}$$

The trading system is represented by the following linear predicates: priceQ(A, P, Q), actPrices(A, L), time(T), where action A can be buy or sell. In this paper, we focus more on buy limit orders.

For an action A and a price P, the queue Q in priceQ(A, P, Q) contains all resident orders with those attributes. Due to how orders are processed, the queue is sorted in ascending order of timestamp. Price queues are never empty, we only maintain price queues for active prices. For an action A, the list L in actPrices(A, L) contains the active prices available in the market, i.e., all the prices at which there is something offered. Note that the bid price is the maximum of L when A is buy and the ask price is the minimum when A is sell. The time is represented by the fact time(T) and increases as the state changes.

Storing orders. The computation when storing takes place (adding to the resident market) is presented in Fig. 1. An order is stored when its limit price P is such that it cannot be exchanged against an opposite resident orders. Namely, when P < ask in the case of a buy order, and when P > bid in the case of a sell order. The rules distinguish whether there are pre-existing resident orders at that price in the market or not.

(L) limit/empty:

$$\text{order(limit, buy, P, ID, N} = 1, \text{T)} \otimes \text{actPrices(buy, L)} \otimes$$
$$\text{maxP(L, B)} \otimes \text{less-or-eq(P, B)} \otimes \text{notInList(L, P)} \otimes \text{insert(L, P, LP)} \otimes \text{time(T)}$$
$$\multimap \quad \{\text{priceQ(buy, P, expListP([ID, N} = 1, \text{T], nilP))} \otimes \text{actPrices(A, LP)} \otimes \text{time(s(T))}\}$$

(L) limit/queue:

$$\text{order(limit, buy, P, ID, N} = 1, \text{T)} \otimes \text{actPrices(buy, L)} \otimes$$
$$\text{maxP(L, B)} \otimes \text{less-or-eq(P, B)} \otimes \text{inList(L, P)} \otimes \text{priceQ(buy, P, PQ)} \otimes$$
$$\text{expListP(PQ, [ID, N} = 1, \text{T], PQ}') \otimes \text{time(T)}$$
$$\multimap \quad \{\text{actPrices(buy, L)} \otimes \text{priceQ(buy, P, PQ}') \otimes \text{time(s(T))}\}$$

(M) limit/empty:

$$\text{order(limit, buy, P, ID, N} = 1, \text{T)} \otimes \text{actPrices(buy, L)} \otimes \text{actPrices(sell, L}') \otimes$$
$$\text{maxP(L, B)} \otimes \text{minP(L}', \text{S)} \otimes \text{nat-great(P, L)} \otimes \text{nat-less(P, S)} \otimes$$
$$\text{insert(L, P, LP)} \otimes \text{time(T)}$$
$$\multimap \quad \{\text{priceQ(buy, P, expListP([ID, N} = 1, \text{T], nilP))} \otimes \text{actPrices(buy, LP)} \otimes \text{time(s(T))}\}$$

Figure 1: *Storing of the incoming unit-size buy limit-orders in the market (in segments L and M).*

The first two rules describe how the incoming buy limit-order (when it cannot be filled) is stored, depending on whether the pre-existing queue exists (in which case that price is already active) or not. In the latter case, a new queue is created (and the corresponding price is activated). The third rule describes storing of orders in the middle segment (between bid and ask), which is by definition empty and therefore there is never a pre-existing queue. So the incoming order will be stored, and the corresponding price added to the list of active prices. By construction, this price always updates the bid or ask, depending onwhether the incoming order was buy or sell order.

Filling orders. The computation when filling takes place (executing an incoming order against the most competitive opposite resident order) is presented in Fig. 2. A limit order is filled (against the most competitive opposite order) when its limit price P satisfies P ≤ bid, in the case of sell orders, or P ≥ ask for buy orders.

(R) limit/1:

$$\begin{aligned} & \text{order(limit, buy, P, ID, N} = 1, \text{T}) \otimes \text{actPrices(sell, L}') \otimes \\ & \text{minP(L}', \text{S}) \otimes \text{great-or-eq(P, S)} \otimes \text{priceQ(sell, S, consP([ID}', \text{N}' = 1, \\ & \text{T}'], \text{nilP}))\otimes \\ & \text{remove(L}', \text{S, L}'') \otimes \text{nat-equal(N} = 1, \text{N}' = 1) \otimes \text{time(T}) \\ \multimap \quad & \{\text{actPrices(sell, L}) \otimes \text{time(s(T}))\} \end{aligned}$$

(R) limit/2:

$$\begin{aligned} & \text{order(limit, buy, P, ID, N} = 1, \text{T}) \otimes \text{actPrices(sell, L}') \otimes \\ & \text{minP(L}', \text{S}) \otimes \text{great-or-eq(P, S)}\otimes \\ & \text{priceQ(sell, S, consP([ID}', \text{N}' = 1, \text{T}'], \text{consP([ID1, N 1} = 1, \text{T 1}], \text{L}))) \otimes \\ & \text{nat-equal(N} = 1, \text{N}' = 1) \otimes \text{time(T}) \\ \multimap \quad & \{\text{actPrices(sell, L}') \otimes \text{priceQ(sell, S, consP([ID1, N 1} = 1, \text{T 1}], \text{L})) \otimes \\ & \text{time(s(T}))\} \end{aligned}$$

Figure 2: *Filling of incoming unit-size buy limit-orders against an opposite resident order (computation in segment R).*

The first rule covers the case when the most competitive opposite order in the market is the last in the price-queue (most competitive by definition always sits at ask, i.e., S). Thus, we need to remove the current ask price from the active price list (this effectively defines the new ask price). The second rule covers the case when the most competitive opposite is not the last in that price-queue, and there is no need to update the current ask price or the active price list.

5. TOWARDS THE VERIFICATION OF THE TRADING SYSTEM PROPERTIES

As we have seen in the previous section, the market (exchange, or trading venue) is a dynamic system consisting of two incoming flows of buy and sell orders, which interact with the opposite pool of resident orders – changing the current state of the system. This interaction gives birth to the computation of the financial exchange.

Using a declarative style formalization, we are able to check that this combination of matching rules does not violate desired trading system properties. In particular, we show that the system is never in a crossed or locked market state. A trading system enters a crossed or locked state if a bid price (the most competitive buy price) becomes greater or equal to ask price (the current most competitive sell price), respectively. Intuitively, bid and ask prices are oscillating, i.e., increase or decrease a bit. If a system is designed correctly, it will be impossible to have a state where bid = ask, or bid > ask. Clearly, this is because if there was a chance to fill an incoming order, it would have been done a priori, at the time of entry (an incoming order that can be matched is executed immediately upon arrival, it is not stored).

The minimal market is specified via five state transition rules for incoming buy orders, and symmetrically five rules for incoming sell orders. To prove this property, we need to inspect the rules that change the list of active prices in the way to expand them[4].

Other than that, a fundamental property that could be proved is that the trade, at any given moment, takes place exclusively at either bid or ask. The current version of Celf does not yet support automated meta-reasoning, so the proof is developed by hand. We provide a rough sketch of the formal proof with the intent to demonstrate how this relies on simple induction, when in real life – based on standard testing – it is next to impossible to certify an object system for these properties.

Definition 5.1 (No crossed or locked market) We say that the system satisfies "No crossed or locked property" if, at all times, the maximum buy price in the market is strictly less than the minimum sell price.

5.1. Proving that the market is never in a crossed or locked state

To prove that *no crossed or locked market* property is maintained, we need to show that the maximum of the list of active prices (for resident buy orders) is less than the minimum of active prices for resident sell orders. This is shown by induction on the reachable states; for each relevant state change, we check if the property is maintained. The bid and ask prices are potentially updated only if a new active price is added to the list of buy or sell active prices. Thus, we need to show that whenever this addition takes place, the resulting lists do not violate the property.

Note that if an order is added to the market, there are no matching orders that it could have been exchanged with. Note also that we only need to worry about those rules that rewrite L into some L′. By analyzing those, we observe that the new L′ is computed by the predicate insert(L, X, L′).

4 If a rule contracts the list of active prices, then bid and ask are moving away from each other, which does not lead to a potential locked or crossed scenario.

Theorem 5.1 The *no crossed or locked market* property holds in all states.

For every state that is characterized by actPrices(buy, BP), actPrices(sell, SP), maxP(BP, X) and minP(SP, Y), it is the case that X < Y.

Proof. The proof goes by induction and case analysis of the state transitions.

Base case: The system comes to life via the beginning fact which generates the initial state. The facts actPrices(buy, L) and actPrices(sell, L) are generated initially using nilN for L. Since maxP(nilN, infinity) and minP(nilN, z), and z < infinity, we have that the property holds for the initial state.

IH: We assume that for a given list of active prices BP, SP, it holds that if actPrices(buy, BP), actPrices(sell, SP), maxP(BP, X) and minP(SP, Y), then X < Y.

Using this hypothesis, we proceed to prove the property.

The predicate actPrices(buy, BP) is rewritten to actPrices(buy, BP′), where insert(BP, P, BP′). This happens as a rule when a new order at price P is added to the market. By construction, this rule is only triggered if P is smaller than X (otherwise, that limit order would have already been exchanged). Since the list BP′ is BP extended with a limit price P, we have two cases:

Case 1. If P ≤ X, X remains the maximum value of BP′, so in the new state we will have actPrices(buy, BP′) and maxP(BP′, X), and therefore X < Y still holds. See rule (L) limit/empty in Figure 1.

Case 2. If X < P < Y, P is in the middle region, between bid and ask, and therefore is the maximum value of newly formed BP′, so in the new state we will have actPrices(buy, BP′) and maxP(BP′, P). But by the reasoning above, P < Y, the new bid is smaller than the ask, and the property still holds. See rule (M) limit/empty in Figure 1.

With this, we are done with the proof.

6. CONCLUSION

We have presented a declarative representation of a sequentially operating archetypal order/matching system. Having in mind that it can be extremely difficult to verify properties of fundamental systems using standard techniques of testing, especially if the system is of the infinite state-space nature, we show that this is straightforwardly done using methods based on formal logic and inductive reasoning. The challenge, however, is in being able to create a model capturing the nature of an object system at the fundamental level, and at the right level of abstraction for a given challenge.

Trading systems are considered as safety-critical systems which must comply with various criteria, including the regulatory requirements. Thus, having method and tools to execute on this is of great importance. We showed here how to formally certify that the system satisfy one of the most important fundamental properties, namely that the market is never in a locked or crossed state, i.e., that during the computation bid remains strictly smaller than ask. This property is obtained by design, therefore, it is important that the systems are carefully implemented.

An interesting challenge for future work is combining this paradigm to detect and prevent prohibited (and disruptive) trading practices and, related to this, critical states such as flash-crashes or, symmetrically, market-bubbles. This may be the path towards the elements of financial forensics, but we need to clearly understand the method to coherently combine machine learning, as an empirical method, with symbolic AI as a theoretical and qualitative method.

Coming from the computational perspective, one of the key future directions is the design of a market model featuring parallelism and concurrency, together with discrete time and price at the level of fundamental design, which represents a step forward in the quest for the right (computationally speaking) market model.

REFERENCES

[1] I. Cervesato, S. Khan, G. Reis, and D. Žunić, "Formalization of automated trading systems in a concurrent linear framework," In Proceeding of Linearity and TLLA, Oxford UK, 2019, pp. 1–15.

[2] E. Budish, P. Crampton, and J. Shim, "Implementation details for frequent batch auctions: slowing down markets to the blink of an eye," *American Economic Review Papers and Proceedings*, Vol. 104, No. 5, pp. 418–424, 2014.

[3] E. Budish, P. Crampton, and J. Shim, "The high-frequency trading arms race: frequent batch auctions as a market design response," *Quarterly Journal of Economics*, Vol. 130, No. 4, pp. 1547–1621, 2015.

[4] P. Bahr, J. Berthold, and M. Elsman, "Certified symbolic management of financial multi-party contracts," In ICFP 2015, Vancouver, Canada, 2015, pp. 315–327.

[5] S. P. Jones, J. M. Eber, and J. Seward, "Composing Contracts: An Adventure in Financial Engineering (Functional Pearl)," In ICFP 2000, 2000, pp. 280–292.

[6] G. O. Passmore and D. Ignatovich, "Formal Verification of Financial Algorithms," In CADE 26, Gothenburg, Sweden, 2017, pp. 26–41.

[7] D. Ignatovich and G. O. Passmore, Case Study: 2015 SEC Fine Against UBS ATS, Aesthetic Integration, Ltd., Technical Whitepaper, 2015.

[8] J. Y. Girard, "Linear Logic," *Theoretical Computer Science*, Vol. 50, pp. 1–102, 1987.

[9] I. Cervesato, K. Watkins, F. Pfenning, and D. Walker, "A Concurrent Logical Framework I: Judgments and Properties," Technical Report CMU-CS-02-101, CMU Pittsburgh, 2003.

[10] R. Harper, F. Honsell, and G. Plotkin, "A Framework for Defining Logics," *J. ACM*, Vol. 50, pp. 143–184, 1993.