Formal verification of trading in financial markets

Suneel Sarswat
Tata Institute of Fundamental Research, India
suneel.sarswat@gmail.com

Abhishek Kr Singh
Tata Institute of Fundamental Research, India
abhishek.uor@gmail.com

Abstract

We introduce a formal framework for analyzing trades in financial markets. An exchange is where multiple buyers and sellers participate to trade. These days, all big exchanges use computer algorithms that implement double sided auctions to match buy and sell requests and these algorithms must abide by certain regulatory guidelines. For example, market regulators enforce that a matching produced by exchanges should be *fair*, *uniform* and *individual rational*. To verify these properties of trades, we first formally define these notions in a theorem prover and then give formal proofs of relevant results on matchings. Finally, we use this framework to verify properties of two important classes of double sided auctions. All the definitions and results presented in this paper are completely formalized in the Coq proof assistant without adding any additional axioms to it.

2012 ACM Subject Classification Information systems → Online auctions; Software and its engineering → Formal software verification; Theory of computation → Algorithmic mechanism design; Theory of computation → Computational pricing and auctions; Theory of computation → Program verification; Theory of computation → Automated reasoning

Keywords and phrases Coq, formalization, auction, matching, financial markets

Digital Object Identifier 10.4230/LIPIcs..2019.

1 Introduction

In this paper, we introduce a formal framework for analyzing trades in financial markets. Trading is a principal component of all modern economies. Over the past few centuries, more and more complex instruments are being introduced for trade in the financial markets. All big stock exchanges use computer algorithms to match buy requests (demands) with sell requests (supplies) of traders. Computer algorithms are also used by traders to place orders in the markets.[1] With the arrival of computer assisted trading, the volume and liquidity in the markets have increased drastically and as a result, the markets have become more complex.

Software programs that enable the whole trading process are extremely complex and have to meet high efficiency criterion as they operate on massive amounts of data and in real time. Furthermore, to increase the confidence of traders in the markets, the market regulators set stringent safety and fairness guidelines for these software. Traditionally, to meet such criteria, software development has extensively relied on testing the programs on large data sets. Although testing is helpful in identifying bugs, it cannot guarantee the absence of bugs. Even small bugs in the trading software can have a catastrophic effect on the overall economy. An adversary might exploit a bug to his benefit and to the disadvantage of other genuine traders. These events are certainly undesirable in a healthy economy.

Recently, there have been various instances [1, 2, 3] of violation of the trading rules by the stock exchanges. For example, in [1], a regulator noted: "NYSE Arca failed to execute a

[1] This is known as algorithmic trading.

certain type of limit order under specified market conditions despite having a rule in effect that stated that NYSE Arca would execute such orders"[2]. This is an instance of a program not meeting its specification. Here the program is a matching algorithm used by the exchange and the regulatory guidelines are the broad specifications for the program. Note that, in most of the cases, the guidelines stated by the regulators are not a complete specification of the program. Moreover, there is no formal guarantee that these guidelines are consistent. These are some serious issues potentially compromising the safety and integrity of the markets.

Recent advances in formal methods in computer science can be put to good use in ensuring safe and fair financial markets. During the last few decades, formal method tools have been increasingly successful in proving the correctness of large software and hardware systems [9, 8, 12, 10]. While Model checking tools have been used for the verification of hardware, the use of Interactive theorem provers have been quite successful in the verification of large software. A formal verification of financial algorithms using these tools can be helpful in the rigorous analysis of market behaviour at large. The matching algorithms used by the exchanges (venues) are at the core of the broad spectrum of algorithms used in financial markets. A formal framework for verifying matching algorithms can also be useful in verifying other algorithms in financial markets.

1.1 An overview of trading at an exchange

An exchange is an organised financial market. There are various types of exchanges: stock exchange, commodity exchange, foreign exchange etc. An exchange facilitates trading between buyers and sellers for the products which are registered at the exchange. A potential trader (buyer or seller) places orders in the markets for a certain product. These orders are matched by the stock exchange to execute trades. Most stock exchanges hold trading in two main sessions: pre-market (or auction session) and continuous market (or regular trading session).

The pre-market session reduces uncertainty and volatility in the market by discovering the opening price of the product. During the pre-market session, an exchange collects all the buy requests (bids) and sell requests (asks) for a fixed duration. At the end of this duration the exchange matches these buy and sell requests at a single price using a matching algorithm. In the continuous market session, the incoming buyers and sellers are continuously matched to each other. An incoming bid (ask), if matchable, is immediately matched to the existing asks (bids). Otherwise, if the bid (ask) is not matchable, it is placed in a priority queue prioritised by price. A trader can place orders of multiple quantity of each product to trade during both the sessions. However, for the analysis of the markets, it suffices to assume that each order is of a single unit of a single product; a multiple quantity order can always be treated as a bunch of orders each with a single quantity and the analysis for a single product will apply for all the products individually. As a result, note that a single trader who places an order of multiple units is seen as multiple traders ordering a single unit each, even in the continuous market. Thus, in both sessions of trade, multiple buyers and sellers are matched simultaneously. A mechanism used to match multiple buyers and sellers is known as the double sided auction [7].

In double sided auctions, an auctioneer (e.g. exchanges) collects buy and sell requests over a period of time. Each potential trader places the orders with a limit price: below which a seller will not sell and above which a buyer will not buy. The exchange at the end of this time

[2] The New York Stock Exchange and the Archipelago Exchange merged together to form NYSE Arca, which is an exchange where both stocks and options are traded.

period matches these orders based on their limit prices. This entire process is completed using a double sided auction matching algorithm. Designing algorithms for double sided auctions is well studied [13, 17, 14]. A major emphasis of many of these studies have been to either maximize the number of matches or maximize the profit of the auctioneer. In the auction theory literature, the profit of an auctioneer is defined as the difference between the limit prices of matched bid-ask pair. However, most exchanges today earn their profit by charging transaction costs to the traders. Therefore, maximizing the number of matches increases the profit of the exchange as well as the liquidity in the markets. There are other important properties, like fairness, uniformity and individual rationality, besides the number of matches which are considered while evaluating the effectiveness of a matching algorithm. However, it is known that no single algorithm can possess all of these properties simultaneously [17, 13].

1.2 Related work

There is no prior work known to us which formalizes financial algorithms used by the exchanges. Passmore and Ignatovich in [15] highlight the significance, opportunities and challenges involved in formalizing financial markets. Their work describes in detail the whole spectrum of financial algorithms that need to be verified for ensuring safe and fair markets. Matching algorithms used by the exchanges are at the core of this whole spectrum.

On the other hand, there are quite a few works formalizing various concepts from auction theory [6, 11, 16]. Most of these works focus on the Vickrey auction mechanism. In Vickrey auction, there is a single seller with different items and multiple buyers with valuations for every subsets of items. Each buyer places bids for every combination of the items. At the end of bidding, the aim of seller is to maximise total value of the items by suitably assigning the items to the buyers.

1.3 Our contribution

In this work, we formally define various notions from auction theory relevant for the analysis of trades in financial markets. We define notions like bids, asks and matching in the Coq proof assistant. The dependent types of Coq turn out to be very useful in giving concise representation to these notions, which also reflects their natural definitions. After preparing the basic framework, we define important properties of matching in a double sided auction: fairness, uniformity, maximality and individual rationality. These properties reflect various regulatory guidelines for trading. Furthermore, we formally prove some results on the existence of various combinations of these properties. For example, fairness and maximality can always be achieved simultaneously. These results can also be interpreted as consistency proofs for various subsets of regulatory guidelines. We prove all these results in the constructive setting of the Coq proof assistant without adding any additional axioms to it. These proofs are completed using computable functions which computes the actual instances (certificate). We also use computable functions to represent various predicates on lists. Finally, we use this setting to verify properties of two important classes of matching algorithms: dynamic price and uniform price algorithms.

In Section 2, we formally define the theory of double sided auctions. In Section 3, we define and prove some important properties of matching algorithms in double sided auctions. In particular we present a dynamic price matching algorithm which produces a maximum as well as a fair matching. In Section 3.3, we describe a uniform price matching algorithm used for price discovery in financial markets. Moreover, we prove that it produces a matching which is maximal among all possible uniform matchings. We summarise the work in Section 4 with an overview of possible future works. The Coq formalization for notions and proofs in this paper is available at [4].

2 Modeling double sided auctions

An auction is a competitive event, where goods and services are sold to the most competitive participants. The priority among participating traders is decided by various attributes of the bids and asks (e.g. price, time etc). This priority can be finally represented by ordering them in a sequence. Sequences are best represented using the list data structure in the Coq standard library [5].

2.1 Bid, Ask and limit price

In any double sided auction multiple buyers and sellers place their orders to buy or sell a unit of an underlying product. The auctioneer matches these buy-sell requests based on their *limit prices*. While the limit price for a buy order (i.e. *bid*) is the price above which the buyer does not want to buy the item, the limit price of a sell order (i.e. *ask*) is the price below which the seller does not want to sell the item. If a trader wishes to buy or sell multiple units, he can create multiple bids or asks with different *ids*.

We can express bids as well asks using records containing two fields.

```
Record Bid: Type := Mk_bid { bp:> nat;    idb: nat }.
Record Ask: Type := Mk_ask { sp:> nat;    ida: nat }.
```

For a bid b, (bp b) is the limit price and (idb b) is its unique identifier. Similarly for an ask a, (sp a) is the limit price and (ida a) is the unique identifier of a. Note that the limit prices are natural numbers when expressed in the monetary unit of the lowest denomination (like cents in USA). Also note the use of coercion symbol `:>` in the first field of *Bid*. It declares *bp* as an implicit function which is applied to any term of type *Bid* appearing in a context where a natural number is expected. Hence from now on we can simply use b instead of (bp b) to express the limit price of b. Similarly we can use a for the limit price of an ask a.

Since equality for both the fields of *Bid* as well as *Ask* is decidable (i.e. `nat: eqType`), the equality on *Bid* as well as *Ask* can also be proved decidable. This is achieved by declaring two canonical instances `bid_eqType` and `ask_eqType` which connect *Bid* and *Ask* to the `eqType`.

2.2 Matching in Double Sided Auctions

All the buy and sell requests can be assumed to be present in list B and list A respectively. At the time of auction, the auctioneer matches bids in B to asks in A. We say a bid-ask pair (b, a) is *matchable* if $b \geq a$ (i.e. $bp\ b \geq sp\ a$). Furthermore, the auctioneer assigns a trade price to each matched bid-ask pair. This process results in a matching M, which consists of all the matched bid-ask pairs together with their trade prices. We define matching as a list whose entries are of type `fill_type`.

```
Record fill_type: Type:=  Mk_fill {bid_of: Bid;  ask_of: Ask;  tp: nat}
```

In a matching M, a bid or an ask appears at most once. Note that there might be some bids in B which are not matched to any asks in M. Similarly there might be some asks in A which are not matched to any bids in M. The list of bids present in M is denoted by B_M and the list of asks present in M is denoted by A_M. For example, Fig. 1 shows a matching

M between list of bids B and list of asks A. Note that the bid with limit price 37 is not present in B_M since it is not matched to any ask in M.

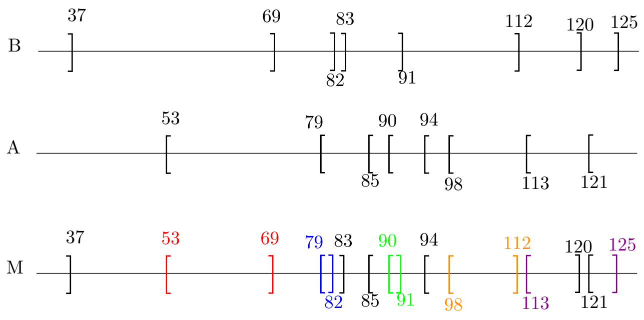

Figure 1 Bids in B and asks in A are represented using right and left brackets respectively along with their limit prices. Every matched bid-ask pair in M is shown using brackets of same colors. Bids with limit prices 37, 83 and 120 are not matched to any ask in the matching M.

More precisely, for a given list of bids B and list of asks A, M is a matching iff, (1) All the bid-ask pairs in M are matchable, (2) B_M is duplicate-free, (3) A_M is duplicate-free, (4) $B_M \subseteq B$, and (5) $A_M \subseteq A$.

▶ **Definition 1.** *matching_in B A M := All_matchable M* $\wedge$ *NoDup* B_M $\wedge$ *NoDup* A_M $\wedge$ $B_M \subseteq B$ $\wedge$ $A_M \subseteq A$.

The term *NoDup* B_M in the above definition indicates that each bid is a request to trade one unit of item and the items are indivisible. We use the expression $B_M \subseteq B$ to denote the term `(Subset` B_M B`)`. It expresses the fact that each element in the list B_M is also present in the list B. While the predicates `NoDup` and `Subset` are sufficient to express the notion of a matching, we need more definitions to describe the properties of matching in double sided auctions. In Fig 2 we describe three binary relations on lists namely `sublist`, `included` and `perm` which are useful in stating some intermediate lemmas leading to important results on matching. For example, consider the following lemma which states that the property of being a matching is invariant under permutation.

▶ **Lemma 2.** *match_inv: perm M M' -> perm B B' -> perm A A' -> matching_in B A M -> matching_in B' A' M'.*

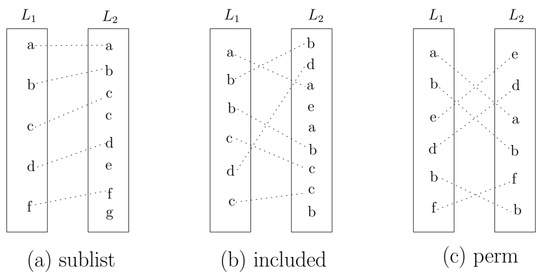

Figure 2 The dotted lines between the entries of lists confirm the presence of these entries in both the lists. (a) If L_1 is `sublist` of L_2 then every entry of L_1 is also present in L_2 and they appear in the same succession. (b) A list L_1 is `included` in L_2 if every entry in L_1 is also present in L_2. (c) Two lists L_1 and L_2 are permutation of each other if each entry has same number of occurrences in both L_1 and L_2.

Note that the notion of permutation for lists is analogous to the equality in multisets. More precisely, we have the following lemmas specifying the `perm` relation.

▶ **Lemma 3.** *perm_intro: (∀ a, count a l = count a s) -> perm l s.*

▶ **Lemma 4.** *perm_elim: perm l s -> (∀ a, count a l = count a s).*

The term (*count a l*) in Lemma 3 represents the number of occurrences of element a in the list l. In proving various properties of a matching M we very often base our arguments solely on the information present in B_M, A_M and P_M. Therefore it is useful to have lemmas establishing the interaction of B_M, A_M and P_M with above mentioned relations on lists.

▶ **Lemma 5.** *included_M_imp_included_bids:included* M M' *-> included* B_M $B_{M'}$.

▶ **Lemma 6.** *included_M_imp_included_asks:included* M M' *-> included* A_M $A_{M'}$

The notion of included in above lemmas is similar to subset relation on multisets. We have the following lemmas specifying the exact behaviour of `included` relation.

▶ **Lemma 7.** *included_intro: (∀ a, count a l* $\leq$ *count a s)-> included l s.*

▶ **Lemma 8.** *included_elim: included l s -> (∀ a, count a l* $\leq$ *count a s).*

In this work, we come across various processes whose input and output are lists. We need the `sublist` relation, which is similar to the subsequence relation, to properly specify the behaviour of these processes.

▶ **Lemma 9.** *sublist_intro1 (a:T): sublist l s-> sublist l (a::s).*

▶ **Lemma 10.** *sublist_elim3a (a e:T): sublist (a::l)(e::s)-> sublist l s.*

Note the recursive nature of `sublist` as evident in Lemma 10. It makes inductive reasoning easier for the statements which contain `sublist` in the antecedent. However, this is not true for the other relations (i.e. `included` and `perm`).

3 Analysis of Double sided auctions

In this, work we do not consider analysis of profit for the auctioneer. Therefore the buyer of a matched bid-ask pair pays the same amount which the seller receives. This price for a matched bid-ask pair is called the trade price for that pair. Since the limit price for a buyer is the price above which she does not want to buy, the trade price for this buyer is expected to be below its limit price. Similarly the trade price for the seller is expected to be above its limit price. Therefore in any matching it is desired that the trade price of a bid-ask pair lies between their limit prices. A matching which has this property is called an *individual rational (IR)* matching. Note that any matching can be converted to an IR matching without altering its bid-ask pair (See Fig 3).

The number of matched bid-ask pairs produced by a matching algorithm is crucial in the design of a double sided auction mechanism. Increasing the number of matched bid-ask pairs increases liquidity in the market. Therefore, producing a maximum matching is an important aspect of double sided auction mechanism design. For a given list of bids B and list of asks A we say a matching M is a maximum matching if no other matching M' on the same B and A contains more matched bid-ask pairs than M.

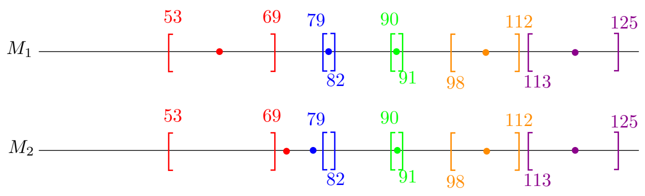

Figure 3 The colored dots represent the trade prices at which the corresponding matched bid-ask pairs are traded. While the matching M_2 is not IR since some dots lie outside the corresponding matched bid ask-pair. The matching M_1 is IR because trade prices for every matched bid-ask pair lie inside the interval. Note that the matching M_1 and M_2 contains exactly the same bid-ask pairs.

▶ **Definition 11.** *Is_MM M B A := (matching_in B A M) ∧ (∀ M', matching_in B A M' → |M'| ≤ |M|).*

In certain situations, to produce a maximum matching, different bid-ask pairs must be assigned different trade prices (Fig 4). However, different prices for the same product in the same market simultaneously leads to dissatisfaction amongst some of the traders. A mechanism which clears all the matched bid-ask pairs at same trade price is called a *uniform matching*. It is also known as *perceived-fairness*.

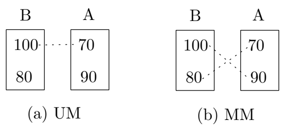

Figure 4 In this case the only individually rational matching of size two it is not uniform.

3.1 Fairness

A bid with higher limit price is more competitive compared to bids with lower limit prices. Similarly an ask with lower limit price is more competitive compared to asks with higher limit prices. In a competitive market more competitive traders are prioritised for matching. A matching which prioritises more competitive traders is called a *fair* matching.

▶ **Definition 12.** *fair_on_bids M B:= ∀ b b', In b B ∧ In b' B -> b > b' -> In b' B_M -> In b B_M.*

▶ **Definition 13.** *fair_on_asks M A:= ∀ s s', In s A ∧ In s' A -> s < s' -> In s' A_M -> In s A_M.*

▶ **Definition 14.** *Is_fair M B A:= fair_on_asks M A ∧ fair_on_bids M B.*

Here, the predicate `fair_on_bids M B` states that the matching M is fair for the list of buyers B. Similarly, the predicate `fair_on_asks M A` states that the matching M is fair for the list of sellers A. A matching which is fair on bids as well as ask is expressed using the predicate `Is_fair M B A`.

Unlike the uniform matching, a fair matching can always be achieved without compromising the size of matching. We can accomplish this by converting any matching into a fair matching without changing its size. For example, consider the following function `make_FOB`.

```
Fixpoint Make_FOB (M:list fill_type) (B: list Bid):=
    match (M,B) with
    |(nil,_) => nil
    |(m::M',nil) => nil
    |(m::M',b::B') => (Mk_fill b (ask_of m) (tp m))::(Make_FOB M' B')
  end.
```

The function make_FOB produces a fair_on_bids matching from a given matching M and a list of bids B, both sorted in decreasing order of bid prices (See Fig 5). Note that make_FOB doesn't change any of the ask in M and due to the recursive nature of make_FOB on B, a bid is not repeated in the process of replacement. Hence the new B_M is duplicate-free. Once we get a fair matching on bids, we use similar function make_FOA to produce a fair matching.

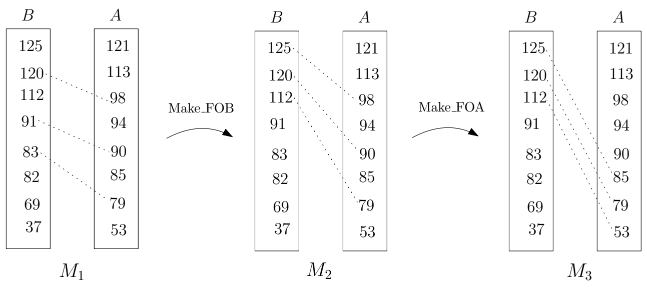

Figure 5 The dotted lines in this figure represent matched bid-ask pairs in matching M_1, M_2 and M_3. In the first step function make_FOB operates on M_1 recursively. At each step it picks the top bid-ask pair, say (b, a) in M_1 and replaces the bid b with a most competitive bid available in B. The result of this process is a fair_on_bids matching M_2. In a similar way the function make_FOA changes M_2 intro a fair on ask matching M_3.

More precisely, for the function make_FOB and make_FOA we have the following lemmas proving it fair on bids and fair on asks respectively.

▶ **Lemma 15.** *mfob_fair_on_bid M B: (Sorted M) -> (Sorted B) -> sublist P_{B_M} P_B -> fair_on_bids (Make_FOB M B) B.*

▶ **Lemma 16.** *mfob_fair_on_ask M A: (Sorted M) -> (Sorted A) -> sublist P_{A_M} P_A -> fair_on_asks (Make_FOA M A) A.*

▶ **Theorem 17.** *exists_fair_matching (Nb: NoDup B)(Na: NoDup A): matching_in B A M -> ($\exists$ M', matching_in B A M' $\wedge$ Is_fair M' B A $\wedge$ |M| = |M'|).*

Proof of Theorem 17 depends on Lemma 16 and Lemma 15. Furthermore, Lemma 16 and Lemma 15 can be proved using induction on the size of M.

3.2 Maximum Matching

The liquidity in any market is a measure of how quickly one can trade in the market without much cost. One way to increase the liquidity is to maximize the number of matched bid-ask pairs. In the previous section we have seen that any matching can be changed to a fair matching without altering its size. Therefore, we can have a maximum matching without compromising on the fairness of the matching. In this section we describe a matching which is fair as well as maximal. For a given list of bid B and list of ask A, a maximum and fair matching can be achieved in two steps. In the first step we apply function produce_MM which produces a matching which is maximal and fair on bids. In the next step we apply make_FOA to this maximum matching to produce a fair matching (See Fig 6).

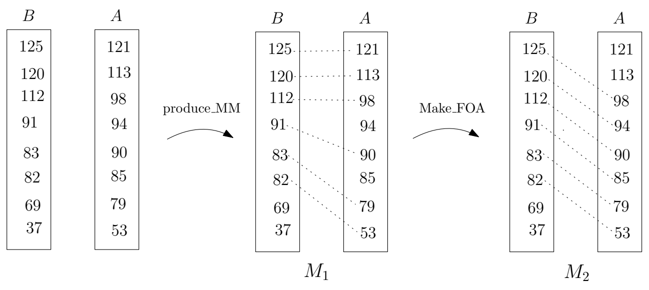

Figure 6 In the first step, the function `produce_MM` operates recursively on the list of bids B and list of asks A. At each iteration `produce_MM` selects a most competitive available bid and then pairs it with the largest matchable ask. The output of this function is fair on bids since it doesn't leave any bid from top. In the second step, the function `make_FOA` converts M_1 into fair matching M_2.

```
Fixpoint produce_MM (B:list Bid) (A: list Ask): (list fill_type) :=
  match (B, A) with
  |(nil, _) => nil
  |(b::B', nil) => nil
  |(b::B', a::A') =>  match (a <= b) with
     |true => ({|bid_of:= b; ask_of:= a; tp:=(bp b)|})::(produce_MM B' A')
     |false => produce_MM B A'
    end
  end.
```

At each iteration `produce_MM` generates a matchable bid-ask pair (See Fig 6). Due to the recursive nature of function `produce_MM` on both B and A, it never pairs any bid with more than one asks. This ensures that the list of bids in matching (i.e. B_M) is duplicate-free. Note that `produce_MM` tries to match a bid until it finds a matchable ask. The function terminates when either all the bids are matched or it encounters a bid for which no matchable ask is available. Therefore, the function `produce_MM` produces a matching from a given lists of bids B and a list of asks A, both sorted in decreasing order by there limit prices. The following theorem states that the function `produce_MM` produces a maximum matching when both B and A are sorted in decreasing order by limit prices.

▶ **Theorem 18.** *produce_MM_is_MM(Nb:NoDup B)(Na:NoDup A): Sorted B -> Sorted A -> Is_MM (produce_MM B A) B A.*

Proof: We prove this result using induction on the size of list A.

- Induction hypothesis (`IH`): $\forall$ *A', |A'| < |A| ->* $\forall$ *B, Sorted B -> Sorted A' -> Is_MM (produce_MM B A') B A'.*

Let M be an arbitrary matching on the list of bids B and list of asks A. Moreover, assume that b and a are the topmost bid and ask present in B and A respectively (i.e. $A = (a :: A')$ and $B = (b :: B')$). We prove $|M| \le |$`produce_MM` $B\ A|$ in the following two cases.

- **Case-1** ($b < a$): In this case the limit price of a is strictly more than the limit price of b. In this case the function `produce_MM` computes a matching on B and A'. Note that due to the induction hypotheses (i.e. `IH`) this is a maximum matching for B and A'. Since the limit price of ask a is more than the most competitive bid b in B it cannot be present in any matching of B and A. Therefore a maximum matching on B and A' is also a maximum matching on B and A. Hence we have $|M| \le |$ `produce_MM` $B\ A\ |$.
- **Case-2** ($a \le b$): In this case `produce_MM` produces a matching of size $m + 1$ where m is the size of matching `produce_MM` $B'\ \ A'$. We need to prove that $|M| \ \le\ m+1$. Note that

due to induction hypothesis (i.e. `IH`) the matching `produce_MM` B' A' is a maximum matching on B' and A'. Hence no matching on B' and A' can have size bigger than m. Without loss of generality we can assume that M is also sorted in decreasing order of bid prices. Now we further split this case into the following five sub cases (see Fig 7).

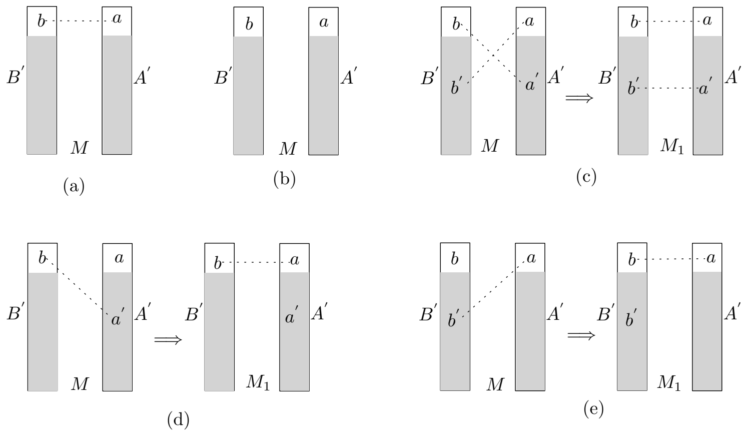

Figure 7 This figure shows all the five sub cases of Case-2 (i.e. when $b \geq a$). The dotted line shows presence of the connected pair in matching M. Both the list of bids B and list of asks A are sorted in decreasing order of their limit prices. Moreover, we assume $B = b :: B'$ and $A = a :: A'$.

- **Case-2A** ($M = (b, a) :: M'$) : In this case bid b is matched to ask a in the matching M (see Fig 7 (a)). Note that M' is a matching on B' and A'. Since $|M'| \leq m$ we have $|M| = |M'| + 1 \leq m + 1$.
- **Case-2B** ($b \notin B_M \wedge a \notin A_M$) : In this case neither bid b nor ask a is present in matching M (see Fig 7 (b)). Therefore M is a matching on B' and A'. Hence we have $|M| \leq m < m + 1$.
- **Case-2C** $(b, a') \in M \wedge (b', a) \in M$: In this case we have $(b, a') \in M$ and $(b', a) \in M$ where $a' \in A'$ and $b' \in B'$. We can obtain another matching M_1 of same size as M (see Fig 7 (c)) where $(b, a) \in M_1$ and $(b', a') \in M_1$. Note that all other entries of M_1 is same as M. Therefore we have $M_1 = (b, a) :: M'$ where M' is a matching on B' and A'. Since $|M'| \leq m$ we have $|M| = |M_1| \leq m + 1$.
- **Case-2D**: $(b, a') \in M \wedge a \notin A_M$: In this case we have $(b, a') \in M$ and $a \notin A_M$ where $a' \in A'$. We can obtain another matching M_1 of same size as M (see Fig 7 (d)) where $(b, a) \in M_1$. Therefore we have $M_1 = (b, a) :: M'$ where M' is a matching on B' and A'. Since $|M'| \leq m$ we have $|M| = |M_1| \leq m + 1$.
- **Case-2E**: $(b', a) \in M \wedge b \notin B_M$: In this case we have $(b', a) \in M$ and $b \notin B_M$ where $b' \in B'$. We can obtain another matching M_1 of same size as M (see Fig 7 (e)) where $(b, a) \in M_1$. Therefore we have $M_1 = (b, a) :: M'$ where M' is a matching on B' and A'. Since $|M'| \leq m$ we have $|M| = |M_1| \leq m + 1$.

Note that all the cases in the above proof correspond to predicates which can be expressed using only the membership predicate on lists. Since we have decidable equality on the elements of the lists all these predicates are also decidable. Hence, we can do case analysis on them without assuming any axiom. ☐

Now that we proved the maximality property of `produce_MM` we can produce a fair as well as maximal matching by applying the functions `Make_FOA` and `Make_FOB` to the output of `produce_MM`. More precisely, for a given list of bids B and list of asks A, we have following result stating that there exists a matching which is both maximal and fair.

▶ **Theorem 19.** *exists_fair_maximum (B: list Bid)(A: list Ask): ∃ M, (Is_fair M B A ∧ Is_MM M B A).*

3.3 Matching in financial markets

An important aspect of the pre-market session is to discover a single price (equilibrium price) at which maximum demand and supply can be matched. Most exchanges execute trade during this session at an equilibrium price. Consider the function `UM` which produces an individually rational matching which is fair and maximal among all uniform matchings.

```
Fixpoint produce_UM (B:list Bid) (A:list Ask)  :=
  match (B,A) with
  |(nil, _) => nil
  |(_,nil)=> nil
  |(b::B',a::A') =>  match (a <= b) with
     |false =>nil
     |true  => ({|bid_of:= b; ask_of:= a; tp:=(bp b)|})::produce_UM B' A'
  end
end.
Definition uniform_price B A := bp (bid_of (last (produce_UM B A))).
Definition UM B A:= replace_column (produce_UM B A) (uniform_price B A).
```

The function `produce_UM` produces bid-ask pairs, `uniform_price` computes the uniform price and finally `UM` produces a uniform matching. The function `produce_UM` is recursive and matches the largest available bid in B with the smallest available ask in A at each iteration (See Fig 8). This function terminates when the most competitive bid available in B is not matchable with any available ask in A. The following theorem states that the function `produce_UM` produces a maximal matching among all uniform matchings when the list of bids B is sorted in decreasing order by limit prices and the list of asks A is sorted in increasing order by limit prices.

▶ **Theorem 20.** *UM_is_maximal_Uniform (B: list Bid) (A:list Ask): Sorted B -> Sorted A -> ∀ M: list fill_type, Is_uniform M -> |M| ≤ | (UM B A) |*

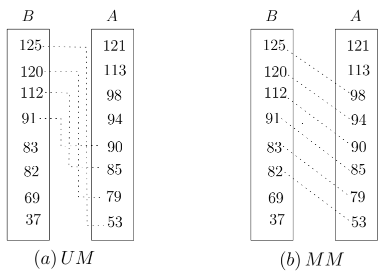

Figure 8 (a) The dotted lines indicate all the bid-ask pair produced by function `produce_UM`. In each iteration function `produce_UM` matches the largest available bid in B with the smallest available ask in A. (b) The dotted lines here indicate a maximum matching for the list of bids B and list of asks A. Note that in this case the matching produced by `UM` is not a maximum matching.

Proof: Let M be any arbitrary IR and uniform matching on the list of bids B and list of asks A where each matched bid-ask pair is traded at price t. We need to prove that $m \leq$

|(UM B A)| where m is the number of matched bid-ask pairs in the matching M. Observe that in any individually rational and uniform matching the number of bids above the trade price is same as the number of asks below the trade price (See Fig 9). Therefore, there are at least m bids above t and m asks below t in B and A respectively.

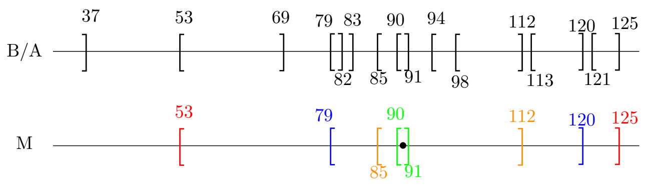

Figure 9 Trade price p for the matching M is shown using a dot that lies between the ask with limit price 90 and bid with limit price 91. Note that since M is individually rational the number of matched asks below the trade price p is same as number of matched bids above the trade price p.

Since at each step the function `produce_UM` pairs the largest bid available in B with the smallest ask available in A it must produce at least m bid-ask pairs. Hence for the list of bids B and list of asks A the function `UM` produces a uniform matching which is of size at least m. ☐

4 Conclusion

Trading activities in today's financial markets are mostly enabled using computer algorithms. These algorithms are extremely large and complex. Matching algorithms used by exchanges (venues) are at the core of this broad range of financial algorithms [15]. To ensure safety and integrity in the markets, the market regulators introduce guidelines specifying different features for these algorithms. Traditional methods of software development, which focus on testing, can not guarantee that these softwares meet the guidelines.

In this work, we develop a formal framework to verify some important properties of the matching algorithms used by exchanges. These algorithms use double sided auctions to match multiple buyers with multiple sellers during different sessions of trading. We use the dependent types of Coq proof assistant to concisely represent various notions from auction theory relevant for the verification of these algorithms. We formally verify two important classes of double sided auctions (uniform price and dynamic price) in this framework.

In this work, we define each bid or ask as a request to trade a single unit of a product and the product is indivisible. In the future this work can be extended to accommodate trades involving multiple units of an item by introducing proper functions to generate bids and asks of single unit from the buy and sell requests of multiple units. Another interesting direction of work is to extend this work for different types of orders (e.g. limit orders, market orders, stop-loss orders, iceberg orders etc) in continuous markets. It would require maintaining a priority queue based on the various attributes of these orders. A formal verification of trading at an exchange will provide a formal foundation that can be used for rigorous analysis of other financial algorithms (e.g. order routing, clearing and settlements etc).

References

1 SEC Charges NYSE for Repeated Failures to Operate in Accordance With Exchange Rules.
2 NYSE to Pay 14 Million dollar Penalty for Multiple Violations.
3 Order in the matter of NSE Colocation.
4 Coq formalization of auctions. `https://github.com/suneel-sarswat/auction`.

5 The Coq Standard Library. `https://coq.inria.fr/library/`.

6 Marco B. Caminati, Manfred Kerber, Christoph Lange 0002, and Colin Rowat. Sound auction specification and implementation. In Tim Roughgarden, Michal Feldman, and Michael Schwarz, editors, *Proceedings of the Sixteenth ACM Conference on Economics and Computation, EC '15, Portland, OR, USA, June 15-19, 2015*, pages 547–564. ACM, 2015.

7 Daniel Friedman. The double auction market institution: A survey. *The double auction market: Institutions, theories, and evidence*, 14:3–25, 1993.

8 Thomas A Henzinger, Ranjit Jhala, Rupak Majumdar, and Grégoire Sutre. Software verification with blast. In *International SPIN Workshop on Model Checking of Software*, pages 235–239. Springer, 2003.

9 J.R. Burch, E.M. Clarke, and K.L. McMillan. Sequential circuit verification using symbolic model checking. In *27th Design Automation Conference*, pages 46–51, 1990.

10 Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, et al. sel4: Formal verification of an os kernel. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 207–220. ACM, 2009.

11 Christoph Lange, Marco B Caminati, Manfred Kerber, Till Mossakowski, Colin Rowat, Makarius Wenzel, and Wolfgang Windsteiger. A qualitative comparison of the suitability of four theorem provers for basic auction theory. In *International Conference on Intelligent Computer Mathematics*, pages 200–215. Springer, 2013.

12 Xavier Leroy. A formally verified compiler back-end. *Journal of Automated Reasoning*, 43(4):363, 2009.

13 R Preston McAfee. A dominant strategy double auction. *Journal of economic Theory*, 56(2):434–450, 1992.

14 Jinzhong Niu and Simon Parsons. Maximizing matching in double-sided auctions. In Maria L. Gini, Onn Shehory, Takayuki Ito, and Catholijn M. Jonker, editors, *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '13, Saint Paul, MN, USA, May 6-10, 2013*, pages 1283–1284, 2013.

15 Grant Olney Passmore and Denis Ignatovich. Formal verification of financial algorithms. In Leonardo de Moura, editor, *Automated Deduction - CADE 26 - 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6-11, 2017, Proceedings*, volume 10395 of *Lecture Notes in Computer Science*, pages 26–41. Springer, 2017.

16 Emmanuel M. Tadjouddine, Frank Guerin, and Wamberto Weber Vasconcelos. Abstracting and verifying strategy-proofness for auction mechanisms. In Matteo Baldoni, Tran Cao Son, M. Birna van Riemsdijk, and Michael Winikoff, editors, *DALT*, volume 5397 of *Lecture Notes in Computer Science*, pages 197–214. Springer, 2008.

17 Peter R. Wurman, William E. Walsh, and Michael P. Wellman. Flexible double auctions for electronic commerce: theory and implementation. *Decision Support Systems*, 24(1):17–27, 1998.