Formalizing double sided auctions in Coq

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

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

Abstract

In this paper we introduce a formal framework for analyzing double sided auction mechanisms in a theorem prover. In a double sided auction multiple buyers and sellers participate for trade. Any mechanism for double sided auctions to match buyers and sellers should satisfy certain properties. For example, fairness, percieved-fairness, individual rationality are some of the important properties. These are important properties and to verify them we need a formal setting. We formally define all these notions in a theorem prover. This provides us a formal setting in which we prove some useful results on matching in a double sided auction. Finally, we use this framework to analyse properties of two important class of double sided auction mechanism. All the properties we discuss in this paper are completely formalized in the Coq proof assistant without adding any axiom 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

Trading is a principal component of all modern economy. Over the century more and more complex instruments (for example, index, future, options etc.) are being introduced to trade in the financial markets. With the arrival of computer assisted trading, the volume and liquidity in the markets has improved significantly. Today all big stock exchanges use computer algorithms (matching algorithms) to match buy requests (demands) with sell requests (supplies) of traders. Computer algorithms are also used by many traders to place orders in the markets. This is known as algorithmic trading. As a result of all this the markets has become complex and large. Hence, the analysis of markets is no more feasible without the help of computers.

A potential trader (buyer or seller) places orders in the markets through a broker. These orders are matched by the stock exchange to execute trades. Most stock exchanges divide the trading activity into three main sessions known as pre-markets, continous markets and post markets. While in the pre-markets session an opening price of a product is discovered through double sided auction. In the continous markets session the incoming buyers and sellers are continously matched against each other on a priority basis. In the post-markets session clearing of the remaining orders is done and a closing price is discovered.

A double sided auction mechanism allows multiple buyers and sellers to trade simultaneously [4]. In double sided auctions, auctioneer (e.g. stock exchange) collects buy and sell requests over a period. 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 period matches these orders based on their limit prices. This entire process is completed using a matching algorithm for double sided auctions.

Designing algorithms for double sided auctions is a well studied topic [5, 8, 6, 9]. A major emphasis of many of these algorithms is to maximize the number of matches or maximize the profit of the auctioneer. Note that an increase in the number of matches increases the liquidity in the markets. A matching algorithm can produce a matching with a uniform price or a matching with dynamic prices. While an algorithm which clears each matched bid-ask pair at a single price is referred as uniform price algorithm. An algorithm which may clear each matched bid-ask pair at different prices is refered as dynamic price algorithm. There are other important properties besides the number of macthes which are considered while evaluating the effectiveness of a matching algorithm. For example, fairness, uniform pricing, individual rationality are some of the relevant features used to compare these matching algorithms. However, it is known that no single algorithm can posses all of these properties [8, 5].

In this paper, we describe a formal framework to analyze double sided auctions using a theorem prover. For this work, we assume that each trader wishes to trade a single unit of the product and all the products are indistinguishable as well as indivisible. We have used the Coq proof assistant to formally define the theory of double sided auctions. Furthermore, we use this theory to validate various properties of matching algorithms. We formally prove some important properties of two algorithms; a uniform price algorithm and a dynamic price algorithm.

2 Modeling double sided auctions

To formalize the notion of matching in a double sided auction we use list data structure. List is also used to define various processes that operate on a matching. However, to express the properties of these processes we need some relations on lists which are analogous to relations on multisets. In this section we formally define these relations which are further used for stating some important results on matching in a double sided auction.

2.1 Bid, Ask and limit price

An auction is a competitive event, where goods and services are sold to the highest bidders. In any double sided auction multiple buyers and sellers place their orders to buy or sell a unit of underlying product. The auctioneer then 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 doesn't want to buy one quantity of the item. The limit price of a sell order (i.e. *ask*), is the price below which the seller doesn't want to sell one quantity of the item. In this work we assume that each bid is a buy request for one unit of item. Similarly each ask is a sell request for one unit of item. If a trader wishes to buy or sell multiple units, he can create multiple bids or asks with different *ids*.

We can express bid as well ask 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 identity. Similarly for an ask a, (sp a) is the limit price and (ida a) is the unique identity of a. 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, now onward 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 prooved 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

In a double sided auction (DSA), the auctioneer collects all the buy and sell requests for a fixed duration. All the buy requests can be assumed to be present in a list B. Similarly, all the sell requests can be assumed to be present in a list A. At the time of auction, the auctioneer matches bids in B against 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 ask in M. Similarly there might be some asks in A which are not matched to any bid in M. The list of bids present in M is denoted as B_M and the list of asks present in M is denoted as A_M. For example, Fig. 1 shows a matching M between list of bids B and list of asks A. While the asks present in A is shown using left brackets and corresponding limit prices. The bids present in B is represented using right brackets and corresponding limit prices. All the matched bid-ask pair in M is then represented using brackets of same colors. Note that the bid with limit price 37 is not present in B_M since it is not matched to any ask in M.

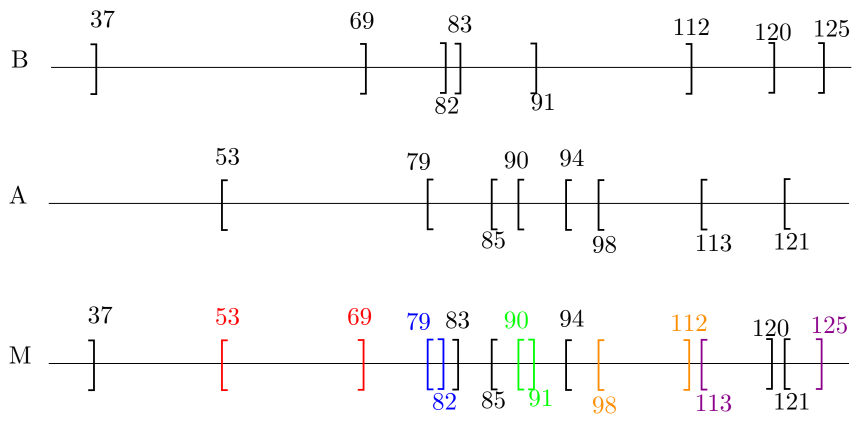


Figure 1 Bids in B and asks in A are represented using right and left brackets respectively. Every matched bid-ask pair in M is shown using brackets of same colors. Bids with limit prices 37, 83 and 120 are not matched to any ask in the matching M.

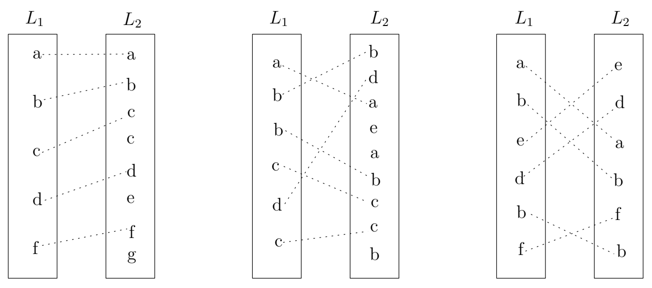
More precisely, for a given list of bids B and list of asks A, M is a matching iff, (1) All the bid-ask pairs in M are matchable, (2) B_M is duplicate-free, (3) A_M is duplicate-free, (4) $B_M \subseteq B$, and (5) $A_M \subseteq A$.

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

While the term *NoDup* B_M in 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.

2.3 Lists, sublist and permutation

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 the following paragraphs we describe three binary relations on lists namely `sublist`, `included` and `perm` which are then used for stating important results on matching in a double sided auction.



(a) sublist (b) included (c) perm

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 no two dotted lines can intersect. (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.

sublist L_1 L_2 **:** The notion of `sublist` is analogous to the subsequence relation on sequences. For the given lists L_1 and L_2 the term `(sublist` L_1 L_2`)` evaluates to `true` if every entry of L_1 is also present in L_2 and they appear in the same succession. For example, in Fig. 2(a) the list L_1 is a `sublist` of L_2 since there is a line incident on each entry of L_1 and no two lines intersect each other.

More precisely, for any two lists l and s whose elements are of type `T` we have following lemmas specifying the `sublist` relation.

▶ **Lemma 2.** *sublist_intro1 (a:T): sublist l s-> sublist l (a::s).*

▶ **Lemma 3.** *sublist_elim3a (a e:T): sublist (a::l)(e::s)-> sublist l s.*

▶ **Lemma 4.** *sublist_elim4: sublist l s -> (*$\forall$ *a, count a l* $\leq$ *count a s).*

The term (*count a l*) in Lemma 4 represents the number of occurrences of element a in the list l. Note the recursive nature of `sublist` as evident in Lemma 3. 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`).

included L_1 L_2 **:** A list L_1 is `included` in list L_2 if every entry of L_1 is also present in L_2. The notion of `included` is analogous to the subset relation on multisets. In Fig 2(b) the list L_1 is `included` in L_2 since there is a line incident on each entry of L_1. More precisely, we have following lemmas specifying the `included` relation.

▶ **Lemma 5.** *included_intro: (∀ a, count a l ≤ count a s)-> included l s.*

▶ **Lemma 6.** *included_elim: included l s -> (∀ a, count a l ≤ count a s).*

▶ **Lemma 7.** *included_intro3: sublist l s -> included l s.*

Note that if l is `sublsit` of s then l is also `included` in s but not the vice versa. However, if both the lists l and s are sorted based on some ordering on type `T` then l is `sublist` of s whenever l is `included` in s.

▶ **Lemma 8.** *sorted_included_sublist: Sorted l -> Sorted s -> included l s -> sublist l s.*

perm L_1 L_2 **:** A list L_1 is permutation of list L_2 iff L_1 is included in L_2 and L_2 is included in L_1. The notion of permutation for lists is simmilar to the equality in multisets. In Fig 2(c) the list L_1 is perm of list L_2. We have following lemmas specifying the essential properties of the `perm` relation.

▶ **Lemma 9.** *perm_intro: (∀ a, count a l = count a s) -> perm l s.*

▶ **Lemma 10.** *perm_elim: perm l s -> (∀ a, count a l = count a s).*

▶ **Lemma 11.** *perm_sort(e: T-> T-> bool): perm l s -> perm l (sort e s).*

The term (*sort s*) in Lemma 11 represents the list s sorted using an ordering relation `e`. The definition of matching as a list is neccessary for describing processes that operate on it. However, while describing various properties of a matching we can always consider it as a collection. For example, consider the following lemma which states that the property of being a matching is invariant over permutation.

▶ **Lemma 12.** *match_inv: perm M M' -> perm B B' -> perm A A' -> matching_in B A M -> matching_in B' A' M'.*

We use B_M to represent the list of bids from B that are matched in M. Simailarly we use notation P_M to represent a list containing trade prices of matched bid-ask pair in M. While 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 13.** *included_M_imp_included_bids:included* M M' *-> included* B_M $B_{M'}$.

▶ **Lemma 14.** *included_M_imp_included_asks:included* M M' *-> included* A_M $A_{M'}$

▶ **Lemma 15.** *included_M_imp_included_tps:included* M M' *-> included* P_M $P_{M'}$

▶ **Lemma 16.** *sorted_nodup_is_sublistB: Sorted* B *-> Sorted* B' *-> NoDup* B *-> NoDup* B' *->* $B \subseteq B'$ *-> sublist* P_B P'_B .

3 Analysis of Double sided auctions

Usually in a double sided auctions mechanism, the profit of an auctioneer is the difference between the limit prices of matched bid-ask pair. In this work we do not consider analysis of profit for the aiuctioneer. Therefore the buyer of matched bid-ask pair pays the same amout which seller recieves. 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 doesn't want to buy, the trade price for this buyer is expected to be below its limit price. Similarly the limit price for a seller is the price below which he doesn't want to sell, hence the trade price for this seller is expected to be be below its limit price. Therefore it is desired that in any matching the trade price of a bid-ask pair lies between their limit prices. A matching which has this property is called an *indivial rational (IR)* matching. Note that any matching can be converted to an IR matching without altering it's bid-ask pair (See Fig 3).

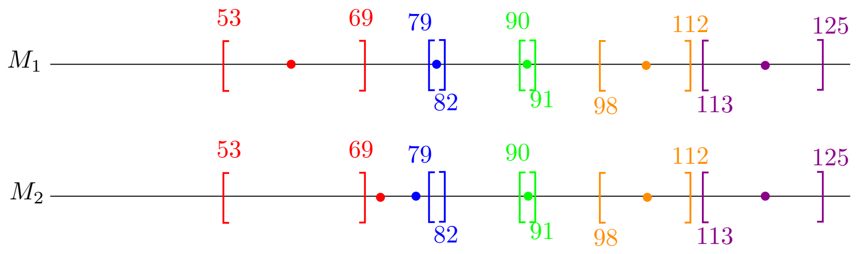


Figure 3 The colored dots represent the trade prices at which the corresponding matched bid-ask pairs are traded. While the matching M_2 is not IR since some dots lie outside the corresponding matched bid ask-pair. The matching M_1 is IR because trade prices for every matched bid-ask pair lie inside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

The number of matched bid-ask pairs produced by any matching algorithm is crucial in the design of a double sided auction mechanism. Increasing the number of matched bid-ask pairs increases the liquidity in market. Therefor, 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 number of matched bid-ask pairs than M.

▶ **Definition 17.** `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 pair must be assigned different trade prices. However, different prices for the same product in the same market simultaneausly leads to dissatisfaction amongst some of the traders. A machanism which clears all the matched bid-ask pairs at same trade price is called a *uniform matching*. It is also known as percived-fairness (cite). In many situation it is not possible to produce an IR matching which is maximum and uniform at the same time. For example in Fig 4 a maximum matching of size two is possible but any uniform matching of size more than one in not possible.

3.1 Fairness

A bid with higher limit price is more competitive campared to bids with lower limit prices. Similaly an ask with lower limit price is more competitive campared to asks with higher limit prices. In a campetitive market, like double sided auction, it is neccesory to priortise more

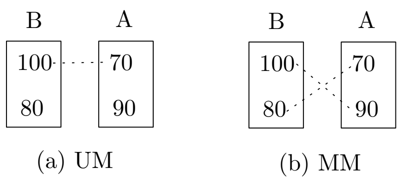


Figure 4 In this figure two bids with limit prices 100 and 80 respectively are matched against two asks of limit price 70 and 90. There is only one matching M_2 of size two possible and it is not uniform.

competitive traders for matching. A matching which priortise competitive traders is a fair matching. Consider the following predicates `fair_on_bids` and `fair_on_asks` which can be used to describe a fair matching.

▶ **Definition 18.** *fair_on_bids M B:=* $\forall$ *b b', In b B* $\wedge$ *In b' B -> b* $>$ *b' -> In b'* B_M *-> In b* B_M*.*

▶ **Definition 19.** *fair_on_asks M A:=* $\forall$ *s s', In s A* $\wedge$ *In s' A -> s* $<$ *s' -> In s'* A_M *-> In s* A_M*.*

▶ **Definition 20.** *Is_fair M B A:= fair_on_asks M A* $\wedge$ *fair_on_bids M B.*

Here, the predicate `fair_on_bids M B` denotes that the matching M is fair for the list of buyers B. Similarly, the predecate `fair_on_asks M A` assures that the matching M is fair for the list of sellers A. A matching which is fair on both the traders (i.e. B and A) is expessed using the predecate `Is_fair M B A`.

Unlike the uniform matching a fair matching can always be achieved without compromising the the size the matching. We can accopmlish 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 `fair_on_bids` matching from a given matching M and a list of bids B, both sorted in decresing order bid prices (See Fig 5). The function `make_FOB` is a recursive function and it replaces the largest bid in M with the largest bid in B. Since at any moment the largest bid in B is bigger than the largest bid in M, the new bid-ask pair is still matchable. Note that `make_FOB` doesn't change any of the ask in M and due to recursive nature of `make_FOB` on B, a bid is not repeated in the process of replacement. This ensure that the new B_M is duplicate-free. Once a matching is modified to a fair matching on bids, we use similar function `make_FOA` on this matching to produce a fair on ask matching. Hence the final result is a fair matching.

For the function `make_FOB` we have following lemma prooving it fair on bids.

▶ **Lemma 21.** *mfob_fair_on_bid M B: (Sorted M) -> (Sorted B) -> sublist* P_{B_M} P_B *-> fair_on_bids (Make_FOB M B) B.*

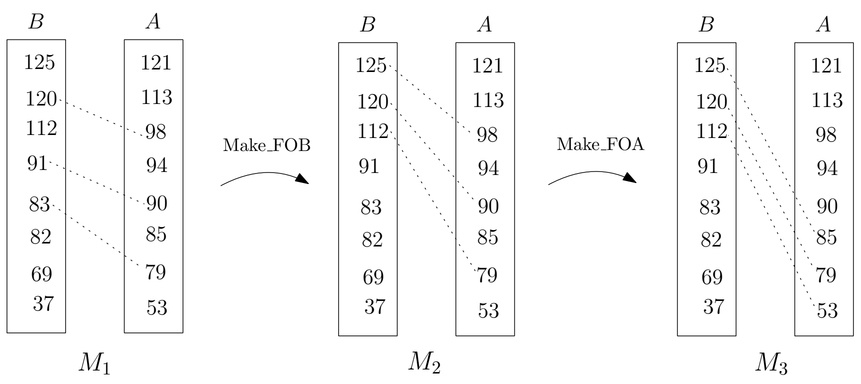


Figure 5 The dotted lines in this figure respresent matched bid-ask pairs in matching M_1, M_2 and M_3. In the first step function `make_FOB` operates on M_1 recursively. At each step it picks the top bid-ask pair, say (b, a) in M_1 and replaces the bid b with most competitive bid available in B. The 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.

▶ **Lemma 22.** *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 23.** *exists_fair_matching (Nb: NoDup B)(Na: NoDup A): matching_in B A M -> (∃ M', matching_in B A M' ∧ Is_fair M' B A ∧ |M| = |M'|).*

Proof of Theorem 23 depends on Lemma 22 and Lemma 21. Furthermore, Lemma 22 and Lemma 21 can be proved using induction on M.

3.2 Maximum Matching

The liquidity in any market is a meausre of how quickly one can trade in the market without much cost. A highly liquid market boosts the investor's confidence in the market. One way to increase the liquidity in a double sided auction is to maximize the number of matched bid-ask pair. 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 fair as well as maximum. For a given bid B and ask A, a maximum and fair matching can achieved in two steps. In first we have function `produce_MM` which produce a matching which is maximum and fair on bids. In the next step we apply `make_FOA` to this maximum matching to produce a fair on ask matching (See Fig 6).

```
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 the above function 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 pair any bid with

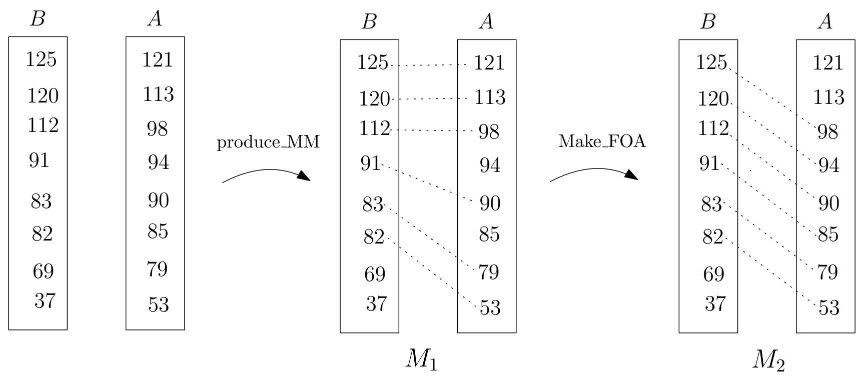


Figure 6 In the first step, the function `produce_MM` operates reccursively on the list of bids B and list of asks A. At each step the function `produce_MM` selects the most competitive available bid and then pairs it with the largest mathchable ask. Note that the output of this function is fair on bid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

more than two asks. This ensures that the list of bids in matching B_M is duplicate-free. Note that the function `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 decresing order by limit prices.

The following theorem states that when the function `produce_MM` is given a list of bids B and list of asks A, both sorted in decreasing order by limit prices, then it produces a maximum matching.

▶ **Theorem 24.** *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`): ∀ *A', |A'| < |A| ->* ∀ *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 need to prove that $|M| \leq |$`produce_MM` B $A|$. We prove this inequality 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 $|M| \leq |$ `produce_MM` B A $|$.
- **Case-2** ($a \leq b$): In this case the function `produce_MM` produces a matching of size $m+1$ where m is the size of matching `produce_MM` B' A'. Hence we need to prove that $|M| \leq m+1$. 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).

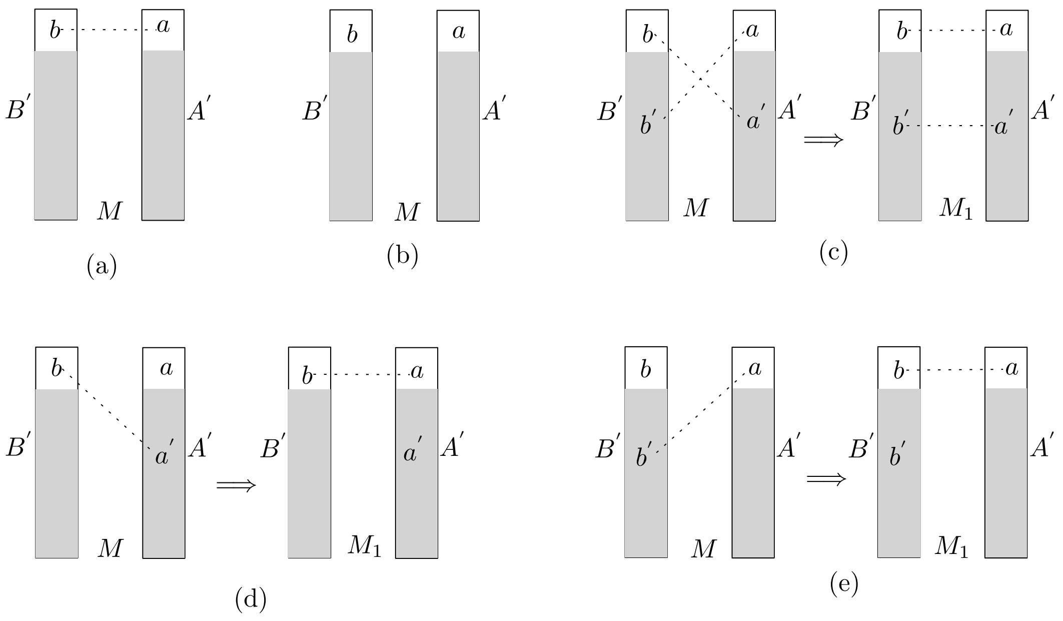


Figure 7 Dotted line expresses the presense of the connected pair in the matching M.

- **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 the membership predicates on lists. Since we have decidable equality on the elements of list all these predicates are also decidable. Hence, we can do case analysis on them without assuming any axiom. ☐

Now that we have prooved maximality property of `produce_MM` one can produce a fair as well as maximum 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 maximum and fair.

▶ **Theorem 25.** *exists_fair_maximum (B: list Bid)(A: list Ask): ∃ M, (Is_fair M B A ∧ Is_MM M B A).*

4 Matching in financial markets

An exchange is an organized financial market. There are various types of exchanges for example stock exchange, commodity exchange, foreign exchange etc. The job of the exchange is to facilitate trading between buyers and sellers for the products which are registered in the exchange. Many exchanges operates during a fix duartion in the day. Some exchanges devides the trading activities into multiple sessions for various reseaons. Many stock exchanges hold trading into two main session; pre-market or auction session and continous market or regular trading session. During the pre-market session an exchange collects all the bids and asks for fix duration and then apply double sided auction mechanism to these orders. At the end of the pre-market session an opening price for the product is deiscovered. During the regular sessions a bid (ask) is matched against the existing asks (bids) immedietly. If the bid (ask) is not matchable it is placed in the priority queue based on their limit price.

The pre-market session reduces uncertenity and volatilty for the regular sessions of trading. To avoid failure on behalf of the traders the exchange must match all the bids or asks withing their limit price. So any matching produced during this session must be indivdual rational. Furthermore, the exchange must be fair towards all the traders. So the matching produced during this session be should be fair matching. One of the most important aspects of the pre-market session is to descover the price of underlying product based on the total demand and supply. This means there must be a unique price at which all the matched bid-ask pairs shuold be traded. So any matching produced during this session should produce a uniform matching.

Most exchanges matches the bids and asks during the pre-market session at eqilibrium price. An equlibrium price is the price at which maximum number of bids can be matched with maximum number of asks. We describe an algorithm which produces an equilibrium price. The algorithm `UM` produces a individual rational matching which is fair and maximum amongst 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 (UM_matching B A))).
Definition UM B A:= replace_column (UM_matching B A) (uniform_price B A).
```

The function `produce_UM` produces bid-ask pairs, `uniform_price` discover the uniform price and finally `UM` produces a uniform matching. The function `produce_UM` is a recursive function which matches the largest bid with the smallest ask at each iteration (See Fig **??**). The function terminates when a bid is not matchable with an ask. Observe that in any individual rational and uniform matching the number of bids above to the trade price is same as the number of asks below to the trade price. Let M be any arbitrary IR and uniform matching of size k with trade price t. Then there are atleast k bids above t and k asks below t. So natuaraaly there are at least k largest bids matchable with k smallest asks. Since UM algorithm pair largest bid with the smallest bid in each iteration then UM must pair at least k bids with k asks. Hence UM produce a matching which of size at least k. Therefore UM produces an IR and fair matching which maximum amongst uniform matchings.

▶ **Theorem 26.** *UM_is_maximal_Uniform (B: list Bid) (A:list Ask): ∀ M: list fill_type, Is_uniform M -> |M| ≤ | (UM B A) |*

5 Conclusion

There has been many attempt in past to design various algorithms for double sided auctions [3]. Fairness, individual rationality and uniformity are some of the critical properties on which effectiveness of these algorithms are evaluated. Theorem provers can be a useful tool in the analysis of these properties [2, 1]. There is an attempt to formalize the financial markets in theorem prover [7]. Our work is the first attempt in formalizing double sided auction in a theorem prover. Formalizing the double sided auction in a theorem proover increases the reliability of the mechanism. Our work in future can be extended for any number of units.

References

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

2 The coq proof assistant, version 87.1, December 15 2017. URL: `https://hal.inria.fr/hal-01673716`.

3 Kaustubh Deshmukh, Andrew V. Goldberg, Jason D. Hartline, and Anna R. Karlin. Truthful and competitive double auctions. *Lecture Notes in Computer Science*, 2461:361–??, 2002.

4 Daniel Friedman. The double auction market institution: A survey. 01 1993.

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

6 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. IFAAMAS, 2013. URL: `http://dl.acm.org/citation.cfm?id=2484920`.

7 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.

8 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.

9 Dengji Zhao, Dongmo Zhang, Md Khan, and Laurent Perrussel. Maximal matching for double auction. In Jiuyong Li, editor, *Australasian Conference on Artificial Intelligence*, volume 6464 of *Lecture Notes in Computer Science*, pages 516–525. Springer, 2010.