CS4211 Formal Methods for Software Engineering

Second-price Sealed-bid Auction Process Modeling

Group	Name	Matriculation Number
Group Member #1	Li Ruize	A0078040M
Group Member #2	Lu Tao	A0078062E
Group Member #3	Truong Viet Trung	A0099324X
Group Member #4	Yos Riady	A0099317U

Contents

1. Introduction

In the project, we will use the PAT modeling techniques to model a second-price sealed-bid auction process (SPSB). In SPSB, the aim is to find my optimal bidding strategy over items in order to maximize the desirability of the items I get. In this project, we will define the problem, the model for the auction process, including the system, the bidders' action, and the items, as well as the experiments and observations using our model.

In the following chapters of the report, we will first define the problem by making a few assumptions for the system, followed by the system modelling of all processes and events, constants and variables in our system, then investigating the properties of the model, finally the experiments and observations for different cases using our model.

2. Problem Description

In the second-price sealed-bid auction system, we have some items, some normal bidders, me (another bidder) and the bidding system. Each item is tagged with desirability of each bidder. Each bidder has a fixed amount of balance and they will bid the items with the corresponding desirability. The highest bidder wins but the price paid is the second-highest bid. All normal bidders bid the items with the desirability. While I bid according to my strategy to maximize the desirability of the items I get.

We model the auction process in terms of the following three aspects: the system (including determining the order of bidding items, announcing the winning bidder), the normal bidders' action (bidding based on desirability), my bidding action (bidding based on my strategy).

3. System Modeling

This session will illustrate the general system modeling structure and implementation details of various components of the PAT program our team has delivered.

Session 3.1 and session 3.2 will introduce the basic structure of the modeling and the system assumptions for the process.

Session 3.3 will explain the processes and events implemented according to the general structure of the system modeling.

Session 3.3, 3.4 and 3.5 will introduce the variables, constance in the implemented system and how they are used in the individual events.

3.1. System Structure Design

In the second-price sealed-bid auction process we have modeled, there are four major components to complete the whole process. They are explained as follow:

- System: it is the background system in the auction process, for an auction of a set of items and bidders, it will generate the order of the items.
- Normal Bidders: it is the component that simulates the normal bidders' behavior with their strategy calculation (normal bidders' strategy will be explained in 3.2 system assumptions)
- My Strategy: my strategy simulates our auction bidding behavior which is distinguished from other normal bidders' strategy. (my strategy will be explained in 3.2 system assumptions)
- Seller: it is the component that simulates the auction seller, which includes the functionalities of collecting bidders' bids, finding the winning bidders, calculating the actual price, collecting the money by reducing the the winning bidder's balance and finally assign the desirability to the winning bidder.

3.2. System Assumptions

In the second-price sealed-bid auction process, the general rules is the bidder submit the bid without knowing other bidders' bid, and the highest bidder wins but the price paid is the second- highest bid.

In our system, we have following assumptions made to form the basic conditions of our modeling:

- The order of the for sale items in the auction process is pre-generated by the seller(selling system) and remains to be unknown to anyone but the seller (selling system).
- The items in this auction are known to all the bidders, and each bidder has a desirability scaled from 1 to 10 (1 as the lowest and 10 as the highest desirability) for one item. A bidder may have the same desirabilities for different items, and bidders may have the same desirabilities for the same item.
- 'Normal' bidders will bid on a certain item based on the his or her desirability for this current item, his remaining balance and the total desirability for the remaining items. The equation to calculate the a bidders' bid for a certain item will be:

$$\text{bid} = \frac{\text{bidders desirability for the current item}}{\text{bidders total desirability for the remaining items}} \times \text{remaining balance}$$

- My strategy will first examine my current remaining balance and list out all the possibilities that I could bid. Through the simulation of exhaustion by PAT, we can simulate all the possible results and eventually get the optimal strategy that would result the most desirability for me as a bidder.
- To resolve the conflict of multiple highest bids, all the bidders are implemented in a list format, where all the bidders will be assigned a position/ priority in the list. Once there are multiple highest bids appears, the seller will assign the winner to the bidder who has the highest priority. By default, our strategy will always has the highest priority, it is also can be changed to lowest priority in our system.

3.3. Processes and Events

According to our system structure design the processes and events are illustrated as below:

System(): system process will have an event to generate the order of the auction items.

NormalBidders(): the process for normal biders will have an event to calculate the bids of all the normal bidders according to their remaining balance, the desirability for the current item and their total desirability for the remaining items. Then the bids will be stored in a list for the usage of seller()

my_bid: my_bid will iterating all the values from 1 value to my current remaining balance, through listing all the possible bidding methods, PAT will find the one which get us the highest desirability.

seller(): the seller() process consists three events: findWinner() to find the highest bid in the current round; updateWinner() to first find the second highest bid in the current round and reduce the balance of the winning bidder by the second highest bid, and assign the specific desirability to the winning bidder; nextItem() to finish the current round and go to the next auction item.

3.4. Constants

Constant	Value	Description
NUM_BIDDERS	Integer; subject to change	This constant indicates the number of bidders in the auction process.
NUM_ITEMS	Integer; subject to change	This constant indicates the number of items for sale in the auction process
BIDDER_BALANCE	Integer; subject to change	This constant indicates the starting balance for all the normal bidders.
MY_BALANCE	Integer; subject to change	This constant indicates the starting balance for me as a special bidder.
MY_BIDDER_ID	Integer; subject to change	This constant indicates the bidder ID of mine: all the normal bidders are arranged in a sequence of list. In order to resolve the conflict when multiple bidders offer the same highest bid, we need this constant for the resolving mechanism.
allItemsPurchased	Ture / False	This constant is for the assertion to imply whether the auction process is complete

3.5. Variables

Component	Variable	Description
NormalBidders()	bidder_id	This variable is for iterating through the list of normal bidders for calculating the bid of each normal bidder for the current auction item.
NormalBidders()	bidder_total_desirability	This variable is for the calculation of each bidder’s desirability for the remaining items in order to calculate each bidder’s bid for the current auction item.
NormalBidders()	item_id	This variable is for iterating through the list of items to calculate the remaining desirability of each bidder in order to calculate the their bids for the current auction item.
NormalBidders()	bidPrice [NUM_BIDDERS]	This variable stores the each bidder’s bid for the current auction item after the calculation.
NormalBidders()	desirabilities [NUM_BIDDERS][NUM_ITEMS]	This variable is a 3D matrix that stores all the desirabilities of the auction items for all the normal bidders.
NormalBidders()	bidPrices[NUM_BIDDERS]	This variable stores all the calculated bid prices.
Seller()	my_bid:{1..MY_BALANCE}	This variable is for listing out all the possible bid in my strategy, from minimal 1 value bid to maximal to my remaining balance.
Seller() -> findWinner()	bidder_id	This variable is for iterating through the normal bidder list for finding the highest bid.
Seller() -> findWinner()	currMax	This variable stores the current highest bid. After iterating through the normal bidder list for finding the highest bid.

Seller() -> findWinner()	winningBidder	This variable stores the bidder id of highest bid for the current item.
Seller() -> findWinner()	bidPrice	This variable is for iterating through the normal bidder list for finding the highest bid.
Seller() -> findWinner()	bidPrices[NUM_BIDDERS]	After all the bids are calculated, this variable stores all the bids as a list.
Seller() ->updateWinner	secondHighest_id	This variable stores the bidder who bidded the second highest bid of all the bids.
Seller() ->updateWinner	secondHighestBid	This variable stores the second highest bid of all the bids.
Seller() ->updateWinner	myBalance	This variable keeps track of my current remaining balance.
Seller() ->updateWinner	myTotalDesirabilities	This variable keeps track of the total desirability in the auction process.
Seller() ->updateWinner	balances[NUM_BIDDERS]	This variable keeps track of the remaining balance of all the normal bidders in the current round.
Seller() ->updateWinner	currentItem	This variable will keep track of the current item and after each round, the seller will go to the next item until all the items are sold.

4. Investigated Properties

Given the model elaborated upon in the previous sections, we shall examine the below properties for model correctness and completeness.

4.1. Our strategy arrives at the optimal node/path

Once the auction has finished and all bid items have been purchased, our strategy arrives at the best strategy of bidding. In other words, we maximize the total desirabilities we get.

In PAT, our assertion is expressed as:

```
#assert System() reaches allItemsPurchased with max(myTotalDesirabilities);
```

4.2. No repeat bids of the same item

All the items cannot be sold again when the item has already been sold once. Once an item has been bid and auctioned off to a bidder, that item cannot be re-bid at any future auction rounds.

In PAT, our assertion is expressed as:

```
#assert System() |= [] noRepeatBids;

#assert System() |= [](!currentItemIndexDecreased);
```

4.3. No decrease in bidders' total desirability

A bidder's desirability can only increase by succesfully obtaining an item. A bidder's desirability cannot decrease, ever. It can only stay the same or go up.

In PAT, our assertion is expressed as:

```
#assert System() |= [] noDecreasingDesirabilities;
```

4.4. All items are bid and purchased at the end of the auction

At the end of the auction, all items must have been purchased by some bidder. In other words, each item must have a winning bid/bidder.

In PAT, our assertion is expressed as:

#assert System() reaches allItemsPurchased;

4.5. Bidders' Balance cannot be negative

A winning bidder's balance, once deducted by the submitted second highest bid for the current item, cannot be in the negative.

In PAT, our assertion is expressed as:

#assert System()|= [](!negativeBalanceForBidder);

4.6. Bids must be positive

When a bidder calculates his price/bid for the current item, the calculated value cannot be negative.

In PAT, our assertion is expressed as:

#assert System()|= [](!negativeBidForCurrentItem);

4.7. An item is auctioned off in each bidding round

At the end of each round, there is a winning bid and winning bidder which earns that items' desirability value.

In PAT, our assertion is expressed as:

#assert System()|= [](!noWinnerForCurrentItem);

4.8. Bidders' Desirability for each item must be positive

A bidder's desirability for each item must be a non-negative number.

In PAT, our assertion is expressed as:

```
#assert System()|= [](!negativeDesirabilityForCurrentItem);
```

4.9. Deadlock-free

The program is deadlock free.

In PAT, our assertion is expressed as:

```
#assert System() deadlockfree;
```

4.10. We submit a bid in each auction round

In each auction round, we submit a positive bid for each item currently open for bidding.

In PAT, our assertion is expressed as:

```
#assert System() !=[] (nextItem -> <> my_bid);
```

4.11. Once the auction ends, no more items are auctioned

Once the auction ends and all items have been purchased, no more additional items are bid and no more future rounds occur.

In PAT, our assertion is expressed as:

```
#assert System() |=[] (!repeatedBids);
```

4.12. Divergence Free

A computation is said to diverge if it does not terminate or terminates in an (unobservable state.) PAT's model checker performs Depth-First-Search or Breath-First-Search algorithm to repeatedly explore unvisited states until a terminating state (i.e., a state with no further move, including successfully terminated state) is found or all states have been visited.

In PAT, our assertion is expressed as:

#assert System() divergencefree;

4.13. Terminating (NOT non-terminating)

This is a negative assertion to the previous assertion. Our program terminates.

In PAT, our assertion is expressed as:

#assert System() nonterminating;

4.14. The auction ends only when all items are purchased

Our action only ends under the condition where all items have been bid and purchased.

In PAT, our assertion is expressed as:

#assert System() |= (allItemsPurchased R auctionRunning);

4.15. A winner is picked only after all bidders have submitted their bids

In PAT, our assertion is expressed as:

#assert System() |= (calculatePrices -> X findWinner);

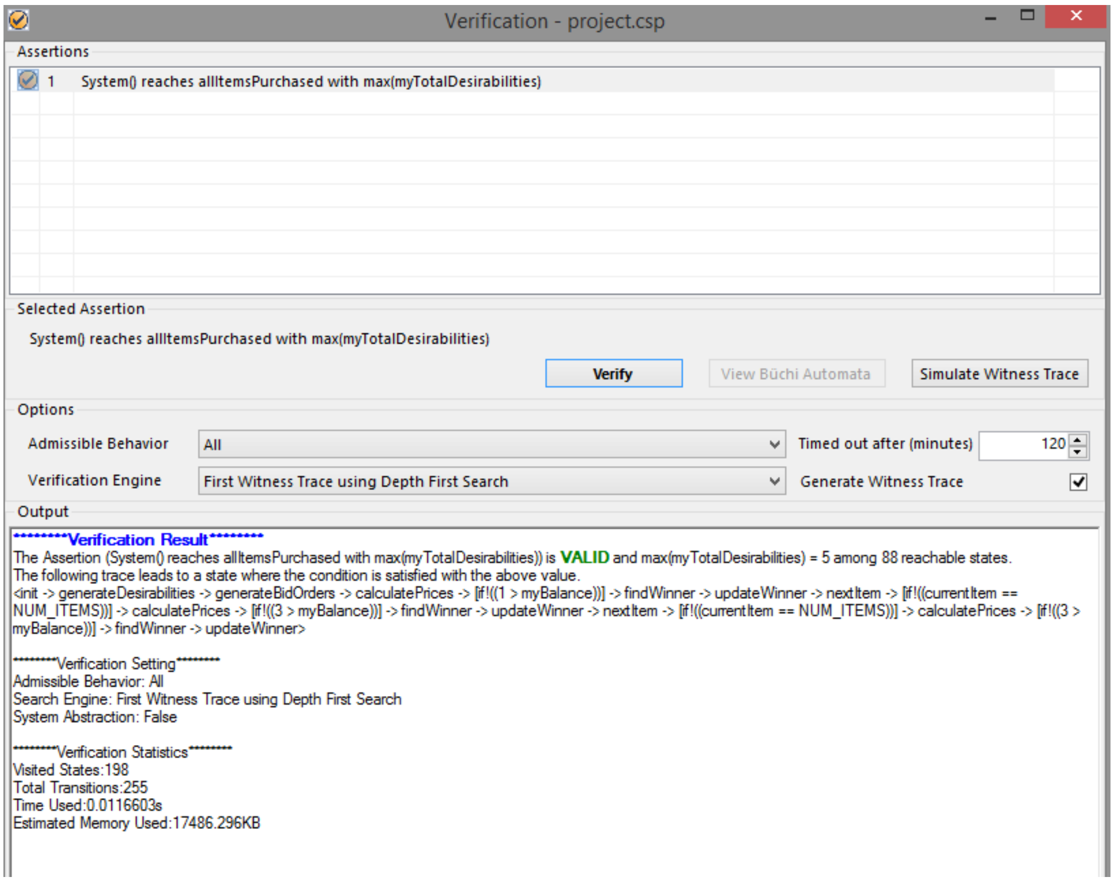
As you can see from the assertion that was used, we have used a variety of assertion methods on the modelling. We have used the most basic reachability assertion to using the more sophisticated LTL assertions with R and X keywords.

5. Experimentation and Observation

5.1. Different item orders

Run 1:

Variable	Value
Balance	6
Number of bidders	4 (including us)
Number of items	3
Item order	1, 2, 3
Bidder 1 desirability	1, 2, 3
Bidder 2 desirability	2, 3, 1
Bidder 3 desirability	3, 1, 2
Our desirability	1, 2, 3



1 System() reaches allItemsPurchased with max(myTotalDesirabilities)

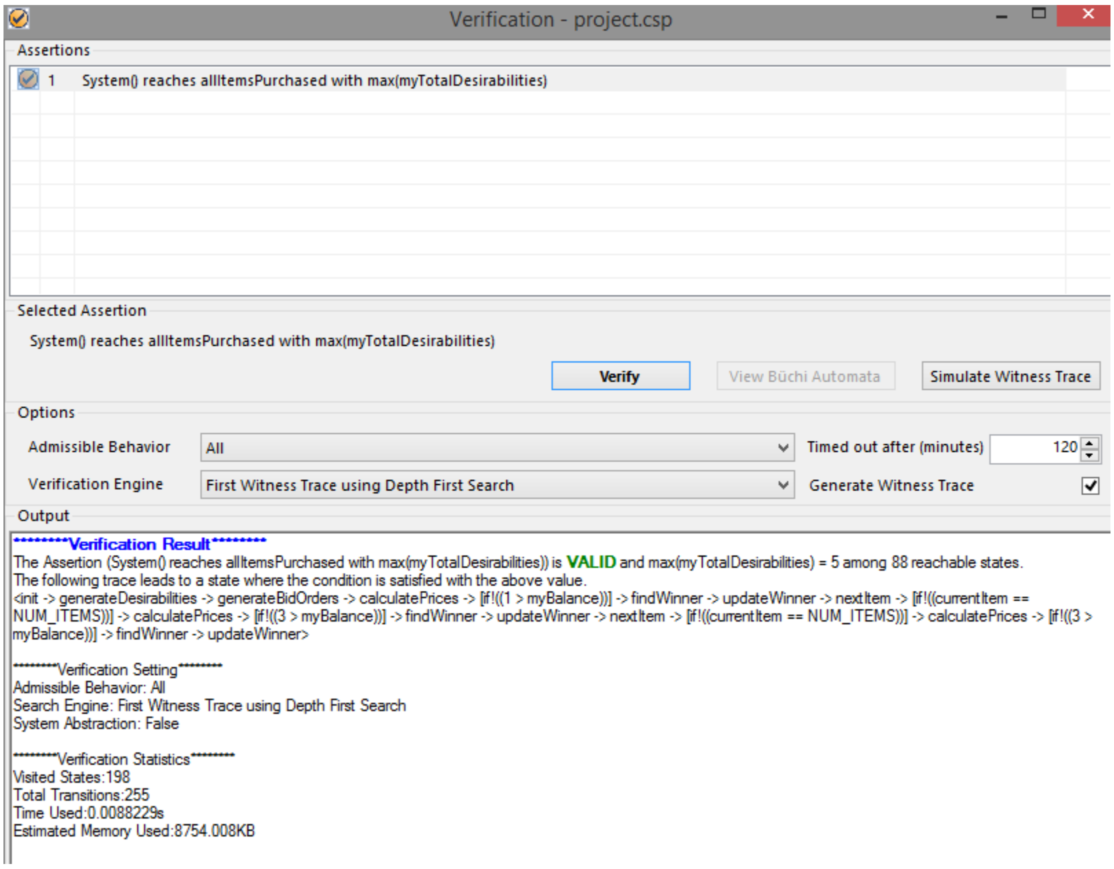
System() reaches allItemsPurchased with max(myTotalDesirabilities)

Admissible Behavior: All
Verification Engine: First Witness Trace using Depth First Search
Timed out after (minutes): 120
Generate Witness Trace: ✓

```
********Verification Result********
The Assertion (System() reaches allItemsPurchased with max(myTotalDesirabilities)) is VALID and max(myTotalDesirabilities) = 5 among 88 reachable states.
The following trace leads to a state where the condition is satisfied with the above value.
<init -> generateDesirabilities -> generateBidOrders -> calculatePrices -> [if!((1 > myBalance))] -> findWinner -> updateWinner -> nextItem -> [if!((currentItem ==
NUM_ITEMS))] -> calculatePrices -> [if!((3 > myBalance))] -> findWinner -> updateWinner -> nextItem -> [if!((currentItem == NUM_ITEMS))] -> calculatePrices -> [if!((3 >
myBalance))] -> findWinner -> updateWinner>

********Verification Setting********
Admissible Behavior: All
Search Engine: First Witness Trace using Depth First Search
System Abstraction: False

********Verification Statistics********
Visited States:198
Total Transitions:255
Time Used:0.0116603s
Estimated Memory Used:17486.296KB
```

Run 2:

Variable	Value
Balance	6
Number of bidders	4 (including us)
Number of items	3
Item order	1, 3, 2
Bidder 1 desirability	1, 2, 3
Bidder 2 desirability	2, 3, 1
Bidder 3 desirability	3, 1, 2
Our desirability	1, 2, 3

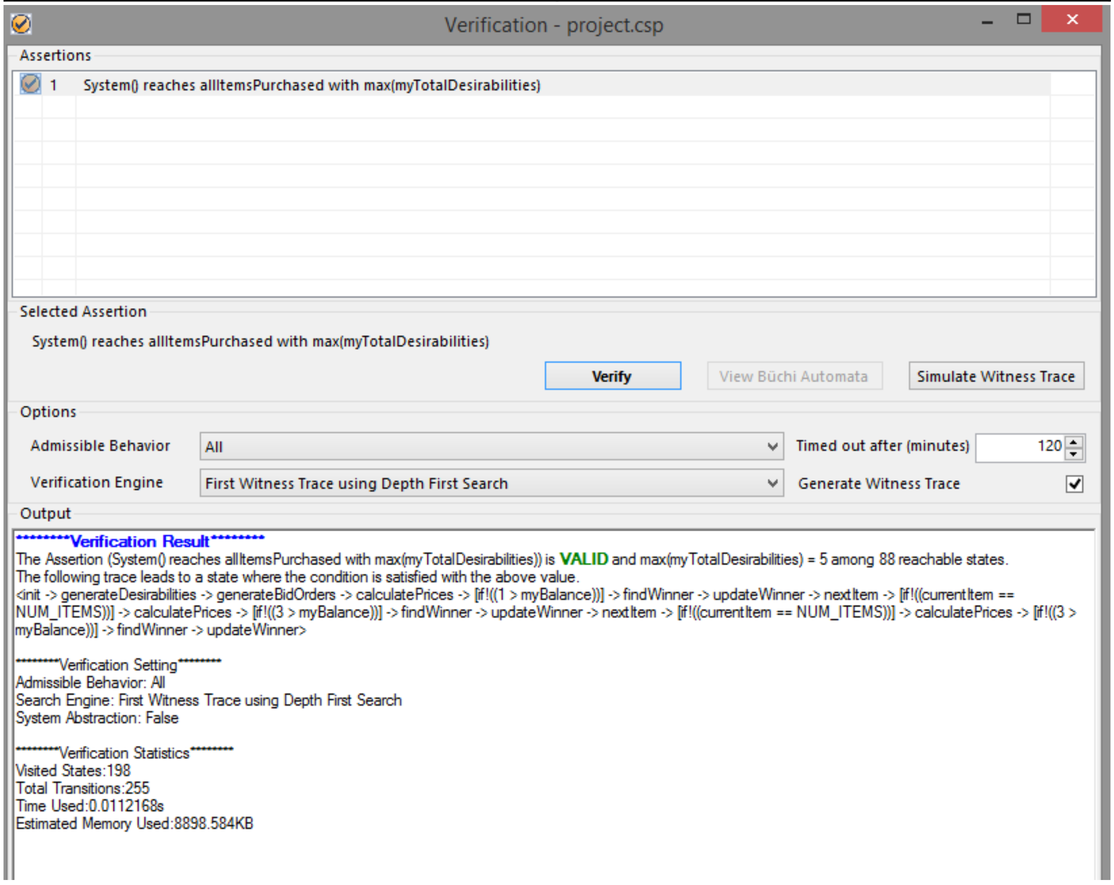

Verification - project.csp
Assertions
1 System() reaches allItemsPurchased with max(myTotalDesirabilities)
Selected Assertion
System() reaches allItemsPurchased with max(myTotalDesirabilities)
Verify
View Büchi Automata
Simulate Witness Trace
Options
Admissible Behavior
All
Timed out after (minutes)
120
Verification Engine
First Witness Trace using Depth First Search
Generate Witness Trace
Output

```
*********Verification Result*********
The Assertion (System() reaches allItemsPurchased with max(myTotalDesirabilities)) is VALID and max(myTotalDesirabilities) = 5 among 88 reachable states.
The following trace leads to a state where the condition is satisfied with the above value.
<init -> generateDesirabilities -> generateBidOrders -> calculatePrices -> [if!((1 > myBalance))] -> findWinner -> updateWinner -> nextItem -> [if!((currentItem ==
NUM_ITEMS))] -> calculatePrices -> [if!((3 > myBalance))] -> findWinner -> updateWinner -> nextItem -> [if!((currentItem == NUM_ITEMS))] -> calculatePrices -> [if!((3 >
myBalance))] -> findWinner -> updateWinner>

*********Verification Setting*********
Admissible Behavior: All
Search Engine: First Witness Trace using Depth First Search
System Abstraction: False

*********Verification Statistics*********
Visited States:198
Total Transitions:255
Time Used:0.0088229s
Estimated Memory Used:8754.008KB
```

Run 3:

Variable	Value
Balance	10
Number of bidders	4 (including us)
Number of items	3
Item order	1, 3, 2

Bidder 1 desirability	1, 2, 3
Bidder 2 desirability	2, 3, 1
Bidder 3 desirability	3, 1, 2
Our desirability	1, 2, 3



1 System() reaches allItemsPurchased with max(myTotalDesirabilities)

System() reaches allItemsPurchased with max(myTotalDesirabilities)

Admissible Behavior: All
Timed out after (minutes): 120
Verification Engine: First Witness Trace using Depth First Search
Generate Witness Trace: ✓

```
*********Verification Result*********
The Assertion (System() reaches allItemsPurchased with max(myTotalDesirabilities)) is VALID and max(myTotalDesirabilities) = 5 among 88 reachable states.
The following trace leads to a state where the condition is satisfied with the above value.
<init -> generateDesirabilities -> generateBidOrders -> calculatePrices -> [if!((1 > myBalance))] -> findWinner -> updateWinner -> nextItem -> [if!((currentItem ==
NUM_ITEMS))] -> calculatePrices -> [if!((3 > myBalance))] -> findWinner -> updateWinner -> nextItem -> [if!((currentItem == NUM_ITEMS))] -> calculatePrices -> [if!((3 >
myBalance))] -> findWinner -> updateWinner>

*********Verification Setting*********
Admissible Behavior: All
Search Engine: First Witness Trace using Depth First Search
System Abstraction: False

*********Verification Statistics*********
Visited States:198
Total Transitions:255
Time Used:0.0112168s
Estimated Memory Used:8898.584KB
```

<u>Observation:</u>

Without any other changes, change in items order does not change the maximum desirabilities found from our strategy.

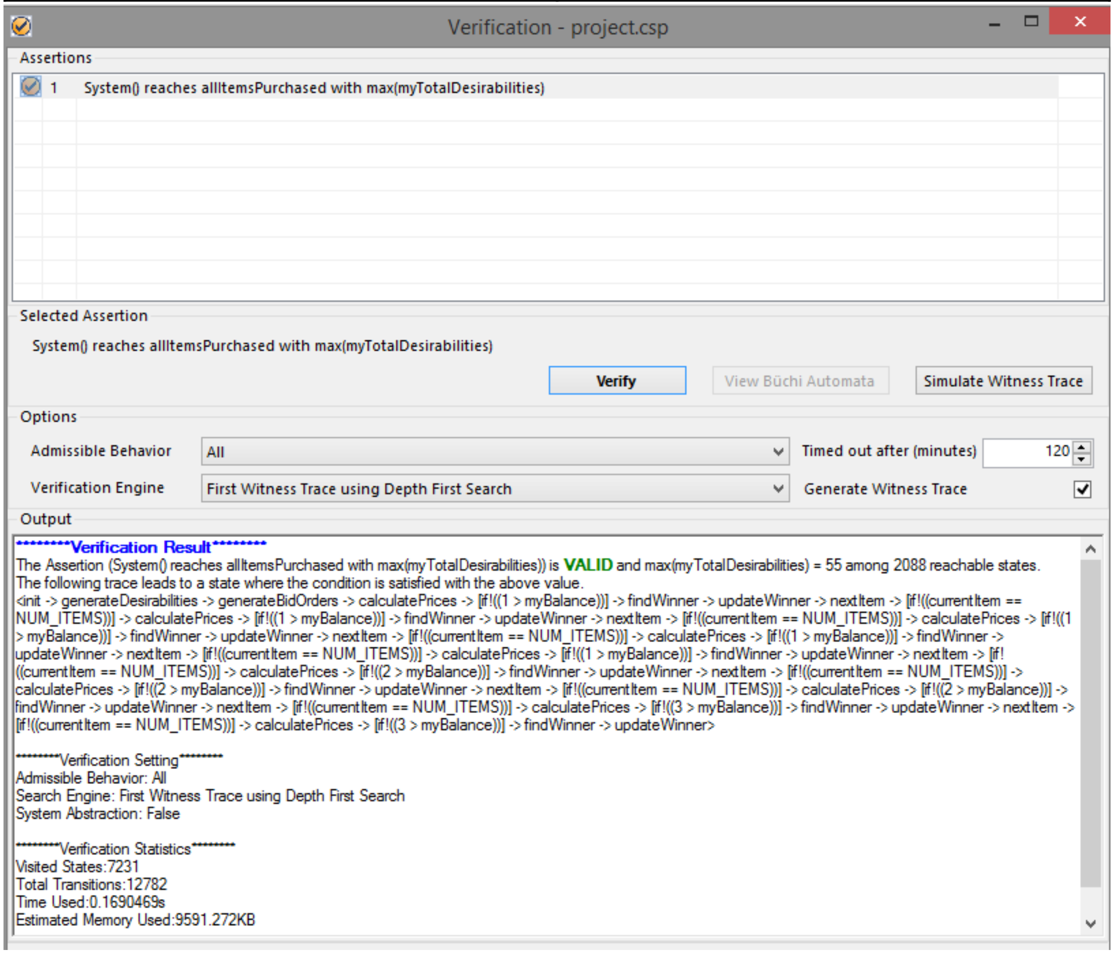
<u>Explanation:</u>

Even if we change the order of the item, the amount that the second highest bidder bid for the item will not change, then to win those items we still need to bid at least the same amount as the second highest bidder.

5.2. Different desirability of other bidders

Run 1:

Variable	Value
Balance	20
Number of bidders	4 (including us)
Number of items	10
Item order	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
Bidder 1 desirability	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
Bidder 2 desirability	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
Bidder 3 desirability	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
Our desirability	1, 2, 3, 4, 5, 6, 7, 8, 9, 10


Verification - project.csp
Assertions
1 System() reaches allItemsPurchased with max(myTotalDesirabilities)
Selected Assertion
System() reaches allItemsPurchased with max(myTotalDesirabilities)
Verify
View Büchi Automata
Simulate Witness Trace
Options
Output

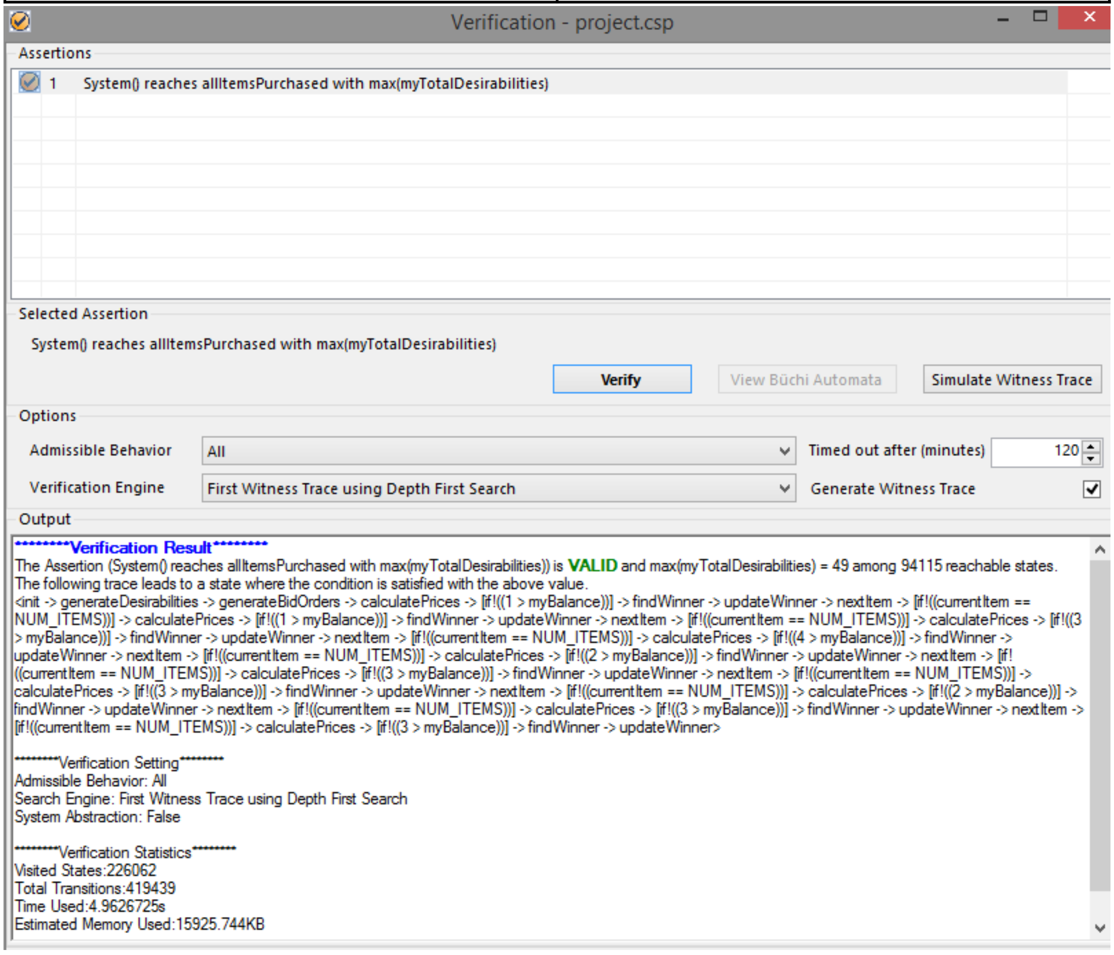
Admissible Behavior: All
Verification Engine: First Witness Trace using Depth First Search
Timed out after (minutes): 120
Generate Witness Trace: ✓

```
*********Verification Result*********
The Assertion (System() reaches allItemsPurchased with max(myTotalDesirabilities)) is VALID and max(myTotalDesirabilities) = 55 among 2088 reachable states.
The following trace leads to a state where the condition is satisfied with the above value.
<init -> generateDesirabilities -> generateBidOrders -> calculatePrices -> [if!((1 > myBalance))] -> findWinner -> updateWinner -> nextItem -> [if!((currentItem ==
NUM_ITEMS))] -> calculatePrices -> [if!((1 > myBalance))] -> findWinner -> updateWinner -> nextItem -> [if!((currentItem == NUM_ITEMS))] -> calculatePrices -> [if!((1
> myBalance))] -> findWinner -> updateWinner -> nextItem -> [if!((currentItem == NUM_ITEMS))] -> calculatePrices -> [if!((1 > myBalance))] -> findWinner ->
updateWinner -> nextItem -> [if!((currentItem == NUM_ITEMS))] -> calculatePrices -> [if!((1 > myBalance))] -> findWinner -> updateWinner -> nextItem -> [if!
((currentItem == NUM_ITEMS))] -> calculatePrices -> [if!((2 > myBalance))] -> findWinner -> updateWinner -> nextItem -> [if!((currentItem == NUM_ITEMS))] ->
calculatePrices -> [if!((2 > myBalance))] -> findWinner -> updateWinner -> nextItem -> [if!((currentItem == NUM_ITEMS))] -> calculatePrices -> [if!((2 > myBalance))] ->
findWinner -> updateWinner -> nextItem -> [if!((currentItem == NUM_ITEMS))] -> calculatePrices -> [if!((3 > myBalance))] -> findWinner -> updateWinner -> nextItem ->
[if!((currentItem == NUM_ITEMS))] -> calculatePrices -> [if!((3 > myBalance))] -> findWinner -> updateWinner>

*********Verification Setting*********
Admissible Behavior: All
Search Engine: First Witness Trace using Depth First Search
System Abstraction: False

*********Verification Statistics*********
Visited States:7231
Total Transitions:12782
Time Used:0.1690469s
Estimated Memory Used:9591.272KB
```

Run 2:

Variable	Value
Balance	20
Number of bidders	4 (including us)

Number of items	10
Item order	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
Bidder 1 desirability	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
Bidder 2 desirability	4, 5, 6, 7, 8, 9, 10, 1, 2, 3
Bidder 3 desirability	7, 8, 9, 10, 1, 2, 3, 4, 5, 6
Our desirability	1, 2, 3, 4, 5, 6, 7, 8, 9, 10


Verification - project.csp
Assertions
1 System() reaches allItemsPurchased with max(myTotalDesirabilities)
Selected Assertion
Verify
View Büchi Automata
Simulate Witness Trace
Options
Output

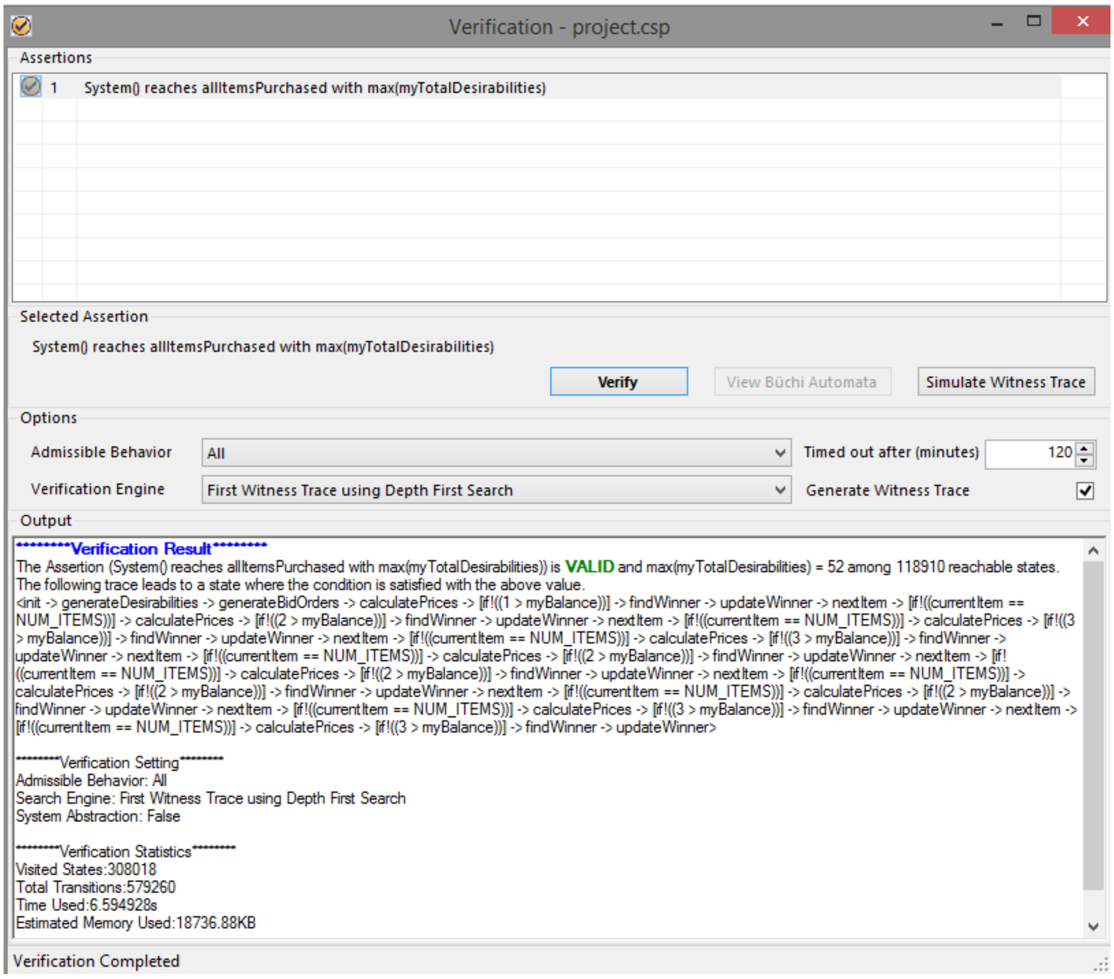
System() reaches allItemsPurchased with max(myTotalDesirabilities)

Admissible Behavior: All
Timed out after (minutes): 120
Verification Engine: First Witness Trace using Depth First Search
Generate Witness Trace ✓

```
*********Verification Result*********
The Assertion (System() reaches allItemsPurchased with max(myTotalDesirabilities)) is VALID and max(myTotalDesirabilities) = 49 among 94115 reachable states.
The following trace leads to a state where the condition is satisfied with the above value.
<init -> generateDesirabilities -> generateBidOrders -> calculatePrices -> [if!((1 > myBalance))] -> findWinner -> updateWinner -> nextItem -> [if!((currentItem ==
NUM_ITEMS))] -> calculatePrices -> [if!((1 > myBalance))] -> findWinner -> updateWinner -> nextItem -> [if!((currentItem == NUM_ITEMS))] -> calculatePrices -> [if!((3
> myBalance))] -> findWinner -> updateWinner -> nextItem -> [if!((currentItem == NUM_ITEMS))] -> calculatePrices -> [if!((4 > myBalance))] -> findWinner ->
updateWinner -> nextItem -> [if!((currentItem == NUM_ITEMS))] -> calculatePrices -> [if!((2 > myBalance))] -> findWinner -> updateWinner -> nextItem -> [if!
((currentItem == NUM_ITEMS))] -> calculatePrices -> [if!((3 > myBalance))] -> findWinner -> updateWinner -> nextItem -> [if!((currentItem == NUM_ITEMS))] ->
calculatePrices -> [if!((3 > myBalance))] -> findWinner -> updateWinner -> nextItem -> [if!((currentItem == NUM_ITEMS))] -> calculatePrices -> [if!((2 > myBalance))] ->
findWinner -> updateWinner -> nextItem -> [if!((currentItem == NUM_ITEMS))] -> calculatePrices -> [if!((3 > myBalance))] -> findWinner -> updateWinner -> nextItem ->
[if!((currentItem == NUM_ITEMS))] -> calculatePrices -> [if!((3 > myBalance))] -> findWinner -> updateWinner>

*********Verification Setting*********
Admissible Behavior: All
Search Engine: First Witness Trace using Depth First Search
System Abstraction: False

*********Verification Statistics*********
Visited States:226062
Total Transitions:419439
Time Used:4.9626725s
Estimated Memory Used:15925.744KB
```

Run 3:

Variable	Value
Balance	20
Number of bidders	4 (including us)
Number of items	10
Item order	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
Bidder 1 desirability	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
Bidder 2 desirability	1, 3, 2, 4, 6, 5, 8, 7, 10, 9
Bidder 3 desirability	10, 9, 8, 7, 6, 5, 4 ,3, 2, 1
Our desirability	1, 2, 3, 4, 5, 6, 7, 8, 9, 10



1 System() reaches allItemsPurchased with max(myTotalDesirabilities)

System() reaches allItemsPurchased with max(myTotalDesirabilities)

Admissible Behavior: All
Timed out after (minutes): 120
Verification Engine: First Witness Trace using Depth First Search
Generate Witness Trace: ✓

```
********Verification Result********
The Assertion (System() reaches allItemsPurchased with max(myTotalDesirabilities)) is VALID and max(myTotalDesirabilities) = 52 among 118910 reachable states.
The following trace leads to a state where the condition is satisfied with the above value.
<init -> generateDesirabilities -> generateBidOrders -> calculatePrices -> [if!((1 > myBalance))] -> findWinner -> updateWinner -> nextItem -> [if!((currentItem ==
NUM_ITEMS))] -> calculatePrices -> [if!((2 > myBalance))] -> findWinner -> updateWinner -> nextItem -> [if!((currentItem == NUM_ITEMS))] -> calculatePrices -> [if!((3
> myBalance))] -> findWinner -> updateWinner -> nextItem -> [if!((currentItem == NUM_ITEMS))] -> calculatePrices -> [if!((3 > myBalance))] -> findWinner ->
updateWinner -> nextItem -> [if!((currentItem == NUM_ITEMS))] -> calculatePrices -> [if!((2 > myBalance))] -> findWinner -> updateWinner -> nextItem -> [if!
((currentItem == NUM_ITEMS))] -> calculatePrices -> [if!((2 > myBalance))] -> findWinner -> updateWinner -> nextItem -> [if!((currentItem == NUM_ITEMS))] ->
calculatePrices -> [if!((2 > myBalance))] -> findWinner -> updateWinner -> nextItem -> [if!((currentItem == NUM_ITEMS))] -> calculatePrices -> [if!((2 > myBalance))] ->
findWinner -> updateWinner -> nextItem -> [if!((currentItem == NUM_ITEMS))] -> calculatePrices -> [if!((3 > myBalance))] -> findWinner -> updateWinner -> nextItem ->
[if!((currentItem == NUM_ITEMS))] -> calculatePrices -> [if!((3 > myBalance))] -> findWinner -> updateWinner>

********Verification Setting********
Admissible Behavior: All
Search Engine: First Witness Trace using Depth First Search
System Abstraction: False

********Verification Statistics********
Visited States:308018
Total Transitions:579260
Time Used:6.594928s
Estimated Memory Used:18736.88KB
```

Verification Completed

<u>Observation:</u>

The total desirabilities that we obtains is different every time

<u>Explanation:</u>

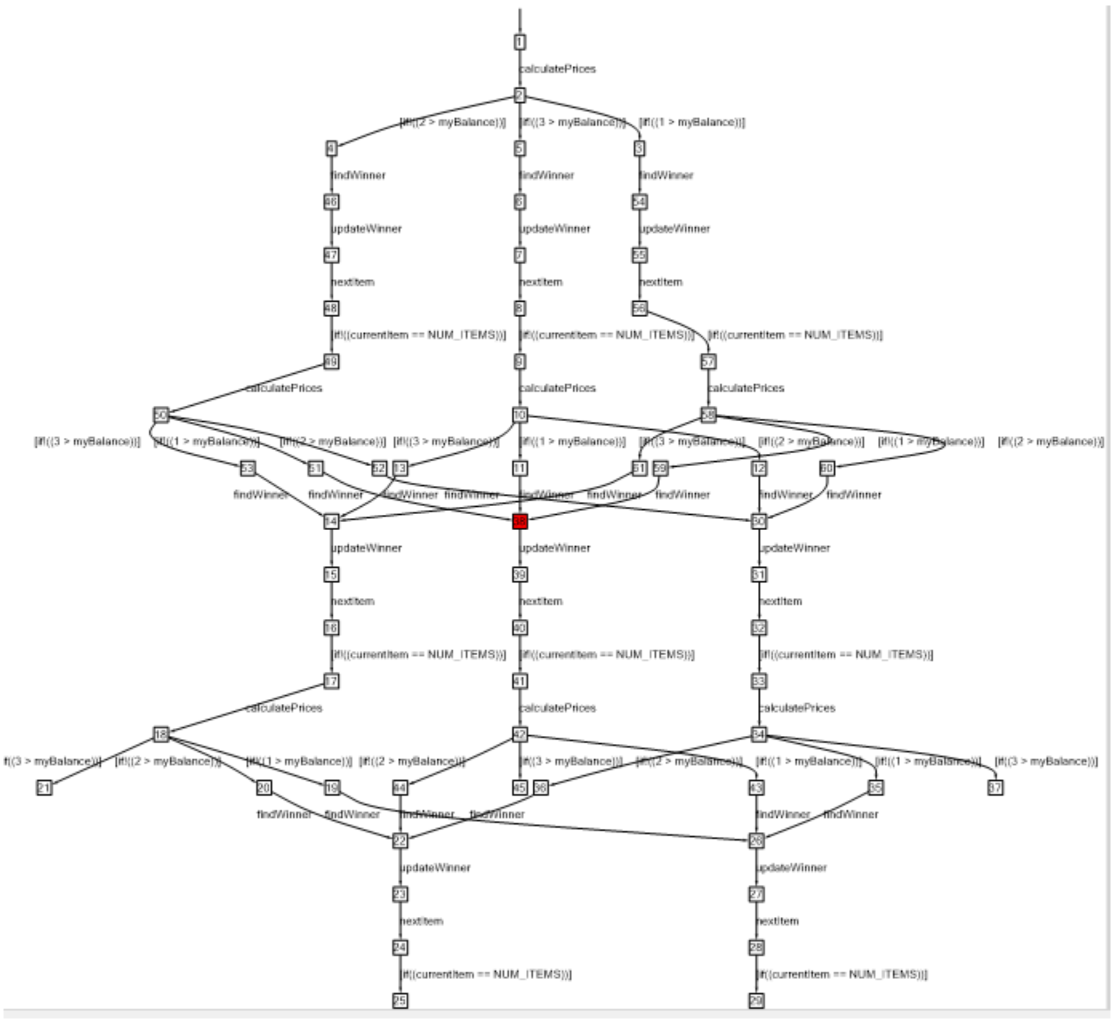
This is because different desirabilities of other bidders will affect the second highest price, and this directly affect how we would have to pay to get one item.

5.3. Different number of items, different number of bidders and different balance

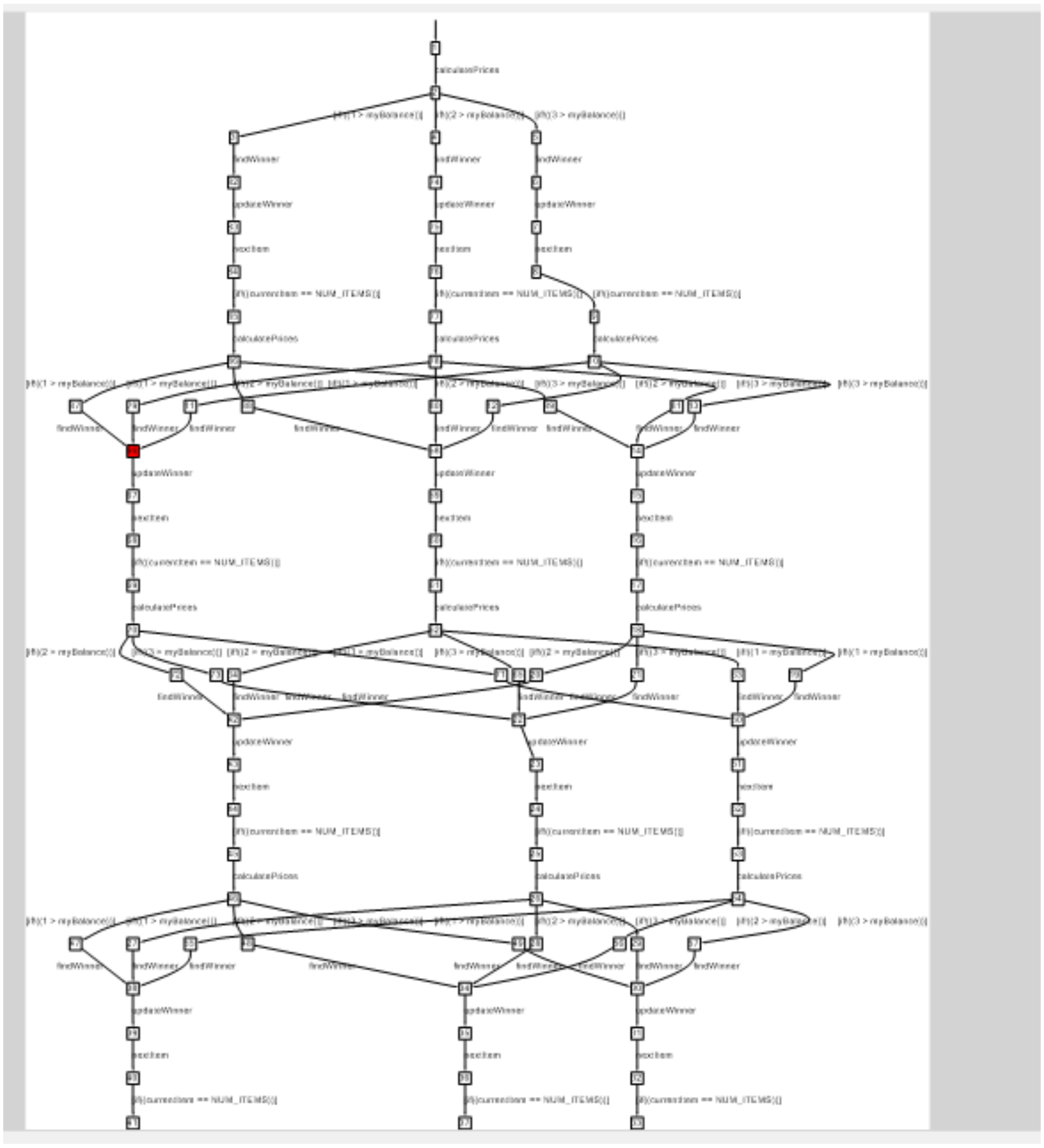
All these do not have specific assumptions and would change our result every time. However, there are common properties of algorithm that we can derive:

<u>Observation 1:</u> Increase in number of items will increase the height of the simulation tree when the program runs as there will be more "rounds".

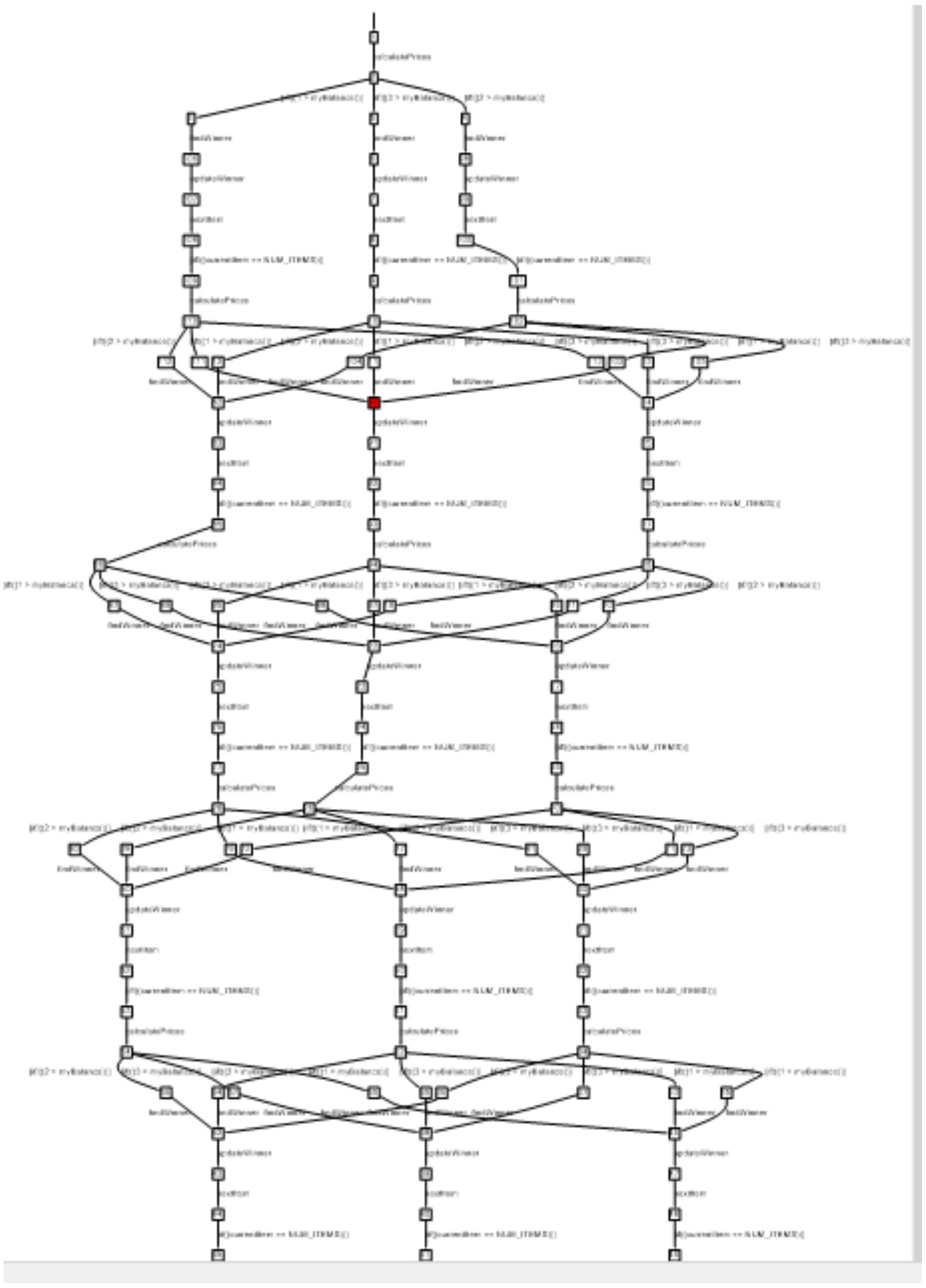
Belows are images of trees for 3, 4 and 5 items (for simplicity we made balance of bidders = 3)



Simulation tree for 3 items



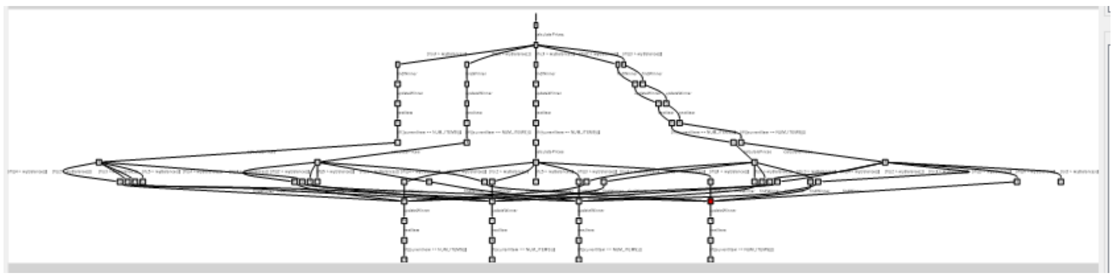
Simulation tree for 4 items

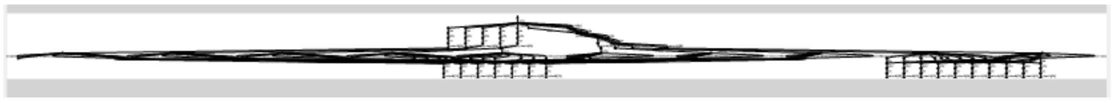


Simulation tree for 5 items

Observation 2: Increase in number of bidders does not create new states because the seller will collect all bids and calculate only once. The running time can increase but not really significant.

Observation 3: Increase in balance make the program run exponentially slower because each round we check bid from 1 to current balance. Belows are graphs for balance 5, 10 and 15 (3 bidders and 2 items)


Balance = 5


Balance = 10

Balance = 15

5.4. Special experiment

Unfair bid:

Case 1: We have more balance to bid than other bidders

Observation: We win more desirabilities since we can spend more bidding

Best case: We win every items

Case 2: We have less balance to bid than other bidders

Observation: We win less desirabilities since we cannot bid as much

Worst case: We win nothing

From this experiment, we decided that we would let we have the same balance to spend as other bidders, since it can generalize the problem and reduce many complexities.

6. Suggestions and Feedback

- Debugging messages are poorly done. For example, we had a hard time debugging the fact that PAT does not support += or -=, the system only provide the information ‘’{’:invalid symbol:’{”
- Resizing windows, especially the graph window is full of bugs, resulting in strange rearrangements and making viewing the graph very difficult. The right visual pane disappears when the window is minimized and maximized again.

7. Conclusion

In summary, using PAT modeling techniques we have successfully modeled the Second-Price Sealed-bid Auction Process Modeling. We have also achieved the goal of our modeling - to find OUR best bidding strategy over items and maximize the desirability of items WE get. We have modeled the auction process, including the system, the bidders’ actions, and the items. We have also made sure that the whole system is functioning correctly by checking for safety properties. In closing, PAT has enabled us to model and verify a problem domain in an efficient way.

References:

- Jun Sun, Yang Liu, Jin Song Dong and Jun Pang. PAT: Towards Flexible Verification under Fairness. The 21th International Conference on Computer Aided Verification (CAV 2009), pages 709-714, June, 2009.

- Jun Sun, Yang Liu, Jin Song Dong and Chun Qing Chen. Integrating Specification and Programs for System Modeling and Verification. The 3rd IEEE International Symposium on Theoretical Aspects of Software Engineering (TASE 2009), pages 127-135, July, 2009.

- Yang Liu, Jun Sun and Jin Song Dong. PAT 3: An Extensible Architecture for Building Multi-domain Model Checkers. The 22nd annual International Symposium on Software Reliability Engineering (ISSRE 2011), pages 190-199, Nov 29 - Dec 2, 2011.