Model Checking Bidding Behaviors in Internet Concurrent Auctions

Haiping Xu and Yi-Tsung Cheng
Computer and Information Science Department
University of Massachusetts Dartmouth
North Dartmouth, MA 02747, USA
Email: {hxu, g_ycheng}@umassd.edu

Abstract

Online auctions have become a quite popular and effective approach in the Internet-based eMarketplace. However, in concurrent online auctions, where multiple auctions for identical items are running simultaneously, bidding behaviors become very complicated and difficult to be verified. Consequently, a shill bidder can easily disguise himself as a legitimate user in order to drive up the bidding price. The goal of this paper is to propose an efficient formal approach to verifying certain bidding behaviors, including shilling behaviors, in order to detect shill suspects in concurrent online auctions. We develop a data-driven auction model based on a model template for concurrent auctions. The auction model can be formally verified using a model checker according to a set of behavioral properties specified in pattern-based linear temporal logic. To illustrate the feasibility and effectiveness of our approach, we use a case study to demonstrate how shill suspects can be efficiently detected.

Keywords: Concurrent online auctions, bidding behaviors, competitive shilling, shill suspects, model checking, pattern-based linear temporal logic (LTL).

1. Introduction

In traditional economic theory, an auction can be used to determine the value of a commodity that is difficult to tag a price. The commodity can be a physical product, such as artwork and antiques; or it can be a virtual product, for example, spectrum licenses and procurement contracts. The most commonly used types of auctions are increasing-price auction (English auction), decreasing-price auction (Dutch auction), first-price sealed-bid auction, and second-price sealed-bid auction (Vickrey auction) [1, 2]. Among them, the English auction becomes the most popular one that is adopted in online auction houses. In an English auction, participants can openly observe other people's bids and then bid against each other. The current bidding price must be higher than the previous one. The auction ends when the auction reaches a point where no one wants to beat the current highest price. Therefore, in an English auction, a bidder can bid multiple times while the bidding price ascends. The seller of the auctioned item can also set a pre-determined reserve price. If the final bidding price is lower than the reserve price, the seller can reserve the right of not selling the auctioned item [3].

The characteristics of multiple bids and ascending bidding price in English auctions have made this type of auctions very popular in online auction houses, for example, eBay, but it also makes shilling behaviors very common. Shill bidding occurs when the seller disguises himself as a legitimate bidder by using a second identity or account solely for the purpose of pushing up the sale price [4]. Particularly, in concurrent online auctions, where multiple auctions for identical items are running simultaneously, the shilling problem becomes even more severe. This is because users' bidding behaviors in concurrent auctions can be very complicated, and as a consequence, a shill bidder may easily hide himself as a legitimate bidder, and put in fake bids once in a while in order to drive up the bidding price.

There are two main kinds of shilling behaviors, namely, the reserve price shilling and the competitive shilling [3]. In reserve price shilling, a seller sets a low reserve price and pretends to be a legitimate bidder to put in bids, in order to drive up the bidding price to his own evaluation of the item. Usually, the lower reserve price the seller sets the cheaper fee he has to pay to the auction house. Thus, the seller can avoid paying higher reserve price fee. On the other hand, in competitive shilling, a seller also pretends to be a legitimate bidder, and constantly monitors the bidding process and puts in fake bids to drive up the bidding price; however, the objective of doing this is to make potential buyers pay extra money to win their bids instead of paying less reserve price fee. In this case, the shill bidder would try his best to avoid winning the auction by not bidding when the auction is close to the end. Although the objectives of these two shilling behaviors are different, their distinction is not always clear. For example, a reserve price shill bidder may still want to drive up the bidding price, even after the bidding price has already reached his own evaluation of the item. Since a reserve price shill has to pay a reserve-price fee, reserve price shills are not usual in real auction marketplaces, e.g., eBay. Meanwhile, the reserve price shilling typically only affects the auction houses; while the competitive shilling may affect all the normal bidders in an auction market. Therefore, the competitive shilling shall cause a greater harm to the auction market than the reserve price shilling. Furthermore, because shilling behaviors involved in concurrent online auctions are more complicated and more difficult to detect than those occurring in a standalone auction, in our research, we focus on studying competitive shilling behaviors in concurrent online auctions.

In this paper, we propose to use model checking techniques [5, 6] to detect shilling behaviors in concurrent online auctions. The model checking approach is a formal method for verifying if a finite state system satisfies certain properties. Using formal methods, we can precisely describe a software system, for example, a concurrent auction system, for the purpose of establishing that the system does or does not exhibit some property, which is itself precisely defined [7]. Thus, a key property of our approach is to derive a formal auction model based on auction data from two concurrent auctions, which paves the way for formal analysis, as seen in earlier work [8]. The formal auction model, which is derived from an auction model template presented in [9], can be verified using the SPIN model checker [6] for certain behavioral properties, which are specified in pattern-based LTL (Linear Temporal Logic) formulae [10, 11]. Since our formal approach is based on analyzing auction data from a limited number of concurrent online auctions, it can be used to efficiently detect shill suspects.

The rest of this paper is organized as follows: Section 2 describes the related work and highlights the relationships to our approach. Section 3 introduces the pattern-based model checking technique. Section 4 first presents a motivating example for shill detection using model checking. Then it introduces a model template, and shows how to build an auction model based on auction data from two concurrent auctions. Section 5 summarizes a set of pattern-based temporal formulae for bidding behavioral properties. Section 6 provides a case study for how to use our approach to detecting shill suspects. Finally, in Section 7, we provide conclusions and our future work.

2. Related Work

There is very little previous work on shill prevention or shill detection for online auctions. Most of the previous work related to shilling behaviors tried to get around the shilling problem by designing sound mechanisms to decrease the incentives to shilling behaviors in online auctions. For example, researchers have proposed reputation mechanisms in online auctions to deter opportunistic behaviors [12, 13, 14]. Since malicious users can easily set up multiple accounts in online environments to disguise themselves as normal users, a good reputation system is necessary to facilitate trust in online auctions, and help other users to identify the trustable and reputable accounts. However, this approach suffers from a few problems. For example, acquainted users can put in good comments for each other, and thus, the reputation system can be easily manipulated [15]. An improved reputation based approach for online auctions is to develop models that characterize sellers according to statistical metrics related to price inflation [16]. However, the proposed approach is based on analyzing large volumes of auction data; thus, it is not efficient in detecting shill bidders.

Additional previous work on prevention or detection of shilling behaviors can be summarized as follows. Wang, Zoltán and Whinston showed that private value English auctions with shill bidding could result in a higher expected seller profit than other auction formats [4]. This explains why in online auction houses like eBay, shilling behaviors have become a serious problem that cannot be ignored. They proposed a commission fee mechanism that suggests the auctioneer charge the seller a commission fee based on the difference between the winning bid and the seller's reserve price. This approach can make shill bidding un-profitable; however, it could also be unfair to sellers' interests, especially when they are not involved in any shill biddings at all.

Chakraborty and Kosmopoulou studied the effect of shill bidding in a common value auction [17]. They described the possible outcomes of an auction where a seller may be able to bid without being detected. They showed that although a seller can increase the price in an auction by shill bidding, he could not benefit from it. This result was based on the assumption that the seller could submit only one bid in the auction. However, in reality, their auction model is not appropriate because a seller may submit multiple bids simultaneously, especially in concurrent online auctions.

Kauffman and Wood used a statistical approach to detecting shilling behaviors and showed how the statistic data of a market would look like if opportunistic behaviors do exist [18]. They also showed how to use an empirical model to test for questionable behaviors. However, their approach suffered from a few problems, such as the need to review multiple auctions over a long period of time [19]. Furthermore, since the statistical approach was based on analyzing large volumes of historical auction data, it was not applicable to directly analyzing any particular auctions where shilling behaviors might be involved. Therefore, their approach is not suitable for detecting new shill bidders or shill bidders who put in fake bids occasionally.

Trevathan and Read designed an algorithm to detect the presence of shill bidding in online auctions [20]. Their approach was based on a set of bidding patterns over a series of auctions held by a particular seller, which can be used to calculate a score to indicate the likelihood that a bidder is a shill. The authors summarized a set of patterns for shilling behaviors, which is useful in detecting shill bidders. However, their approach is not convenient for specifying complex bidding behaviors, such as shilling behaviors occurring in concurrent online auctions, which are more difficult to detect than shilling behaviors appearing in standalone auctions.

There is also some previous work on using model checking techniques to verify certain properties of electronic auction systems in the context of multi-agent systems [21, 22, 23]. Their approaches differ from our approach because they either aimed to verify properties of multi-agent system specification models using agent-based electronic auction systems as examples [21, 22], or attempted to use model checking approach to verifying auction protocols [23]. In contrast, our approach is to automatically generate formal auction models from existing auction data in concurrent online auctions, and verify if an auction bidder has certain bidding behaviors. It is worth noting that some of our ongoing work also involves development of trustworthy agent-based online auction systems [24], and we have attempted to use our model checking approach to detecting malicious agents with shilling behaviors.

We used model checking approach for verifying bidding behavioral properties in concurrent online auctions due to the expressive power of LTL in specifying complex bidding behaviors and the efficiency of our approach in detecting shill suspects. In contrast to other formal methods, such as theorem proving, model checking is completely automatic and fast [25]. Since our formal approach is based on analyzing only a limited number of concurrent auctions, our approach supports efficient search of shill suspects from a large amount of users. In this sense, our approach complements to existing approaches such as statistical approaches that requires analyzing large volume of historical auction data, which is time consuming but may lead to more accurate results [18].

3. Pattern-Based Modeling Checking Technique

3.1 The SPIN Model Checker

There is a wide variety of model checking tools available, such as the SPIN [6], the NuSMV2 [26], Java Pathfinder [27], and the MARIA [28]. Among them, the SPIN model checker represents the most popular one that provides a friendly user interface and accepts model specifications written in PROMELA (PROcess MEta LAnguage) [6, 29]. PROMELA is a language for building verification models that represent an *abstract* of a system, which contains only those aspects that are relevant to the properties one wants to verify [29]. A PROMELA program consists of *processes*, message *channels*, and *variables*. Processes are defined globally; while message channels and variables can be declared either globally or locally within a process. Processes are used to specify system behaviors, and channels and global variables are used to define the environment in which the processes run. Examples and further details about the PROMELA language can be found in references [6, 29].

There are two basic ways to use the SPIN model checking tool in system verification [29]. The first approach is to use the tool to construct verification models that can be shown to have all the required system properties. Such verification models can serve as specification models or high-level design models of a system to be implemented. The second approach is to start from an existing system, and based on the existing system, we build verification models that preserve the system behaviors to be verified. In this case, if the verification models satisfy the required system properties, we can be assured that the existing system also has the required system properties. The approach we proposed in this paper belongs to the second category. Starting from existing online auction systems (i.e., English auction systems) and auction data from specific concurrent auctions, we can automatically generate a formal auction model. Since our generated formal auction model preserves the information about users' bidding activities extracted from the auction data, if the formal auction model can be shown to have certain bidding behavioral properties, the users who participate in the concurrent online auctions must also have such properties.

3.2 LTL and Composition Patterns

The two main types of temporal logic used in model checking are Computation Tree Logic (CTL) and Linear Temporal Logic (LTL). CTL is a branching time logic that is most suitable for applications in hardware verification; while LTL is a linear time logic that is typically used for applications in software verification [29]. The SPIN model checker supports specification of system properties using LTL, which has been proven to have good expressivity and more natural language like statements for verification [30, 31]. LTL consists of only a few logic operators, such as `[]` (always), <> (eventually), `U` (until), `W` (unless, or weak until) and `O` (next). Combining with Boolean operators, i.e., && (and), `||`

(or), `!` (negation), → (logical implication) and ↔ (logical equivalence), LTL is capable of describing many key properties of a concurrent software system.

On the other hand, like many other formal specification and verification methods, writing a LTL formula is not easy and error prone. Even a person who has expertise in using LTL may still have a difficult time in understanding the semantics of a LTL formula, such as `[]((Q && !R && <>R)→(P→ (!R U (S && !R))) U R)`. To solve this problem, Dwyer and his colleagues proposed a pattern-based approach to help software engineers to specify requirements properties without having to worry about the complexity and potential traps [10].

There are quite a few patterns proposed in previous work [10, 11]. Before we present some of the patterns that we use in this paper, we first introduce a notation called *pattern scope*, which represents the extent of a program execution over which the pattern must hold.

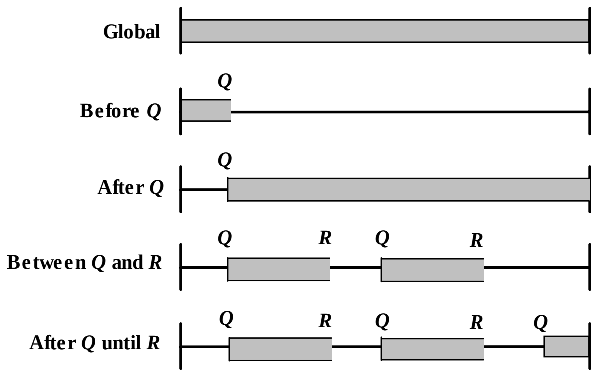


Figure 1: Pattern scopes for pattern-based LTL

Figure 1 is an illustration of pattern scopes for pattern-based LTL adapted from [11]. The capital letters *Q* and *R* stand for events. Every pattern can be assigned with one of the five scopes, in which during the extent of the specified scope, a pattern must hold. It should be clarified that all these pattern scopes be defined as closed-left and open-right. For example, if the scope is "Between *Q* and *R*," then *Q* is included in the scope, but *R* is excluded.

In Table 1, 2 and 3, we list three patterns with different pattern scopes that are used in this paper. For example, in Table 1, we define the *Absence* pattern in pattern scope "Before *R*" as the LTL formula `<>R → (!P U R)`. The formula specifies that during the extent of the starting state and event *R*, event *P* does not occur. Similarly, in Table 2, we define the *Existence* pattern in pattern scope "Between *Q* and *R*" as the LTL formula `[](Q && !R→(!R W (P && !R)).` The formula specifies that during the extent of event *Q* and event *R*, event *P* must occur. For more LTL pattern definitions, refer to previous work [11].

Table 1: Absence pattern (event P does not occur)

Pattern Scope	**Formula**
Globally	`[](!P)`
Before *R*	`<>R → (!P U R)`
After *Q*	`[](Q → [](!P))`
Between *Q* and *R*	`[]((Q && !R && <>R) → (!P U R))`
After *Q* until *R*	`[](Q && !R → (!P W R))`

Table 2: Existence pattern (event P must occur)

Pattern Scope	**Formula**
Globally	`<>(P)`
Before *R*	`!R W (P && !R)`
After *Q*	`[](!Q) || <>(Q && <>P))`
Between *Q* and *R*	`[](Q && !R→(!R W (P && !R)))`
After *Q* until *R*	`[](Q && !R → (!R U (P && !R)))`

Table 3: Precedence pattern (event S precedes event P)

Pattern Scope	**Formula**
Globally	`!P W S`
Before *R*	`<>R → (!P U (S || R))`
After *Q*	`[]!Q || <>(Q && (!P W S))`
Between *Q* and *R*	`[]((Q && !R & <>R) → (!P U (S || R)))`
After *Q* until *R*	`[](Q && !R → (!P W (S || R)))`

4. Modeling Internet Concurrent Auctions

4.1 A Motivating Example

The basic idea of our approach is to automatically generate an auction model based on auction data from two concurrent auctions, and verify if the auction model satisfies certain bidding behavioral properties. We now formally define the concept of concurrent auctions as follows.

Definition 4.1 *Concurrent Auctions*

Let *Auction 0* and *Auction 1* be two auctions running in an online auction system during the time periods of [T_{start0}, T_{end0}] and [T_{start1}, T_{end1}], respectively. *Auction 0* and *Auction 1* are called two *concurrent auctions* if they satisfy the following two conditions: (1) the auctioned items are of the same type and are indistinguishable; (2) the Boolean formula ($T_{start0} \geq T_{end1}$) $\vee$ ($T_{start1} \geq T_{end0}$) evaluates to *false*. Concurrent online auctions *Auction 0* and *Auction 1* are denoted as *Auction 0* || *Auction 1*.

It is easy to show that the operator || for concurrent auctions is symmetric, but *not* transitive, i.e., (1) *Auction 0* || *Auction 1* implies *Auction 1* || *Auction 0*; and (2) *Auction 0* || *Auction 1* and *Auction 1* || *Auction 2* do *not* imply *Auction 0* || *Auction 2*.

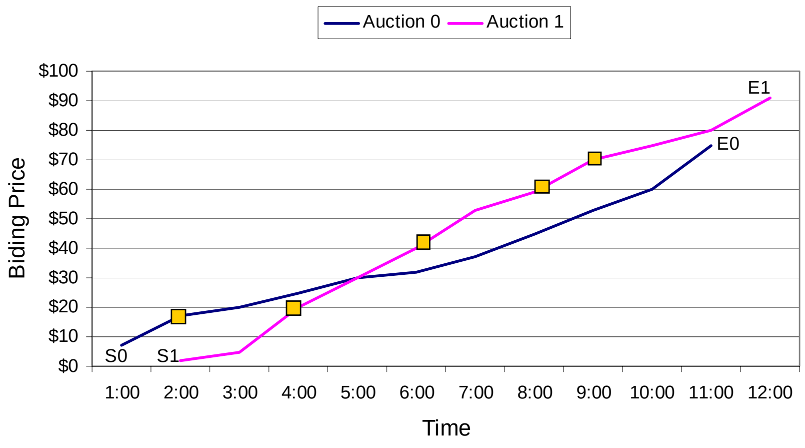


Figure 2: The bidding activities of user *A* in two concurrent auctions

Figure 2 shows an example of two concurrent auctions *Auction 0* and *Auction 1* that are running during the time periods of [1:00, 11:00] and [2:00, 12:00], respectively. To simplify matters, for the rest of the paper, we assume that the auction that starts first is *Auction 0*, and the one that starts later is *Auction 1*. The two curves show the changes of the bidding price over time for the two auctions. The square marks represent user *A*'s bidding activities during the time period [1:00, 12:00]. Now we define two predicates for each of the two auctions, i.e., "Price is lower" and "User *A* bids". If any predicate becomes *true* at a certain point of time in any of the two auctions, it means the event happens at that time. For example, according to Figure 2, at time 4:00, the bidding price is lower in *Auction 1*. Thus, at that time, the predicate "Price is lower" is *true* for *Auction 1*, but it is *false* for *Auction 0*. Similarly, at the same time, since user *A* puts in his bid in *Auction 1*, "User *A* bids" is *true* in *Auction 1*, but it is *false* in *Auction 0*.

To illustrate the basic idea of our approach, we use an example to show how to write a pattern-based LTL formula for a certain bidding behavioral property. For instance, we want to detect the following shilling behavior:

While two auctions with the same type of auctioned items are running concurrently, a shill bidder might put bids in the auction with higher bidding price rather than the one with lower bidding price in order to drive up the price in one auction.

Since *Auction 0* starts first and also ends first (as shown in Figure 2), we need to verify the following property: after "start of *Auction 1*" until "end of *Auction 0*", does "(User *A* bids in *Auction 0* && Price is lower in *Auction 1*) or (User *A* bids in *Auction 1* && Price is lower in *Auction 0*) become *true*?" The formula can be composed using the *Existence* pattern with "After *Q* until *R*" scope. If we use "S1" to represent "start of *Auction 1*", "E0" to represent "end of *Auction 0*," "P" to represent "User *A* bids in *Auction 0* && Price is lower in *Auction 1*", and "S" to represent "User *A* bids in *Auction 1* && Price is lower in *Auction 0*", the LTL formula can be written as `([](S1 && !E0 -> (!E0 U(P && !E0)))) || ([](S1 && !E0 -> (!E0 U(S && !E0))))`. From Figure 2, we can see that the shilling behavior specified above has occurred four times (at time 2:00, 6:00, 8:00 and 9:00). Thus, the verification result for this LTL formula must be *valid*.

4.2 Preprocessing the Auction Data

The first step to generate an auction model is to preprocess the auction data. As shown in Figure 3, this task is accomplished by a module component called *Preprocessor*, which extracts numeric data from two concurrent auctions and preprocess the auction data through the following three steps.

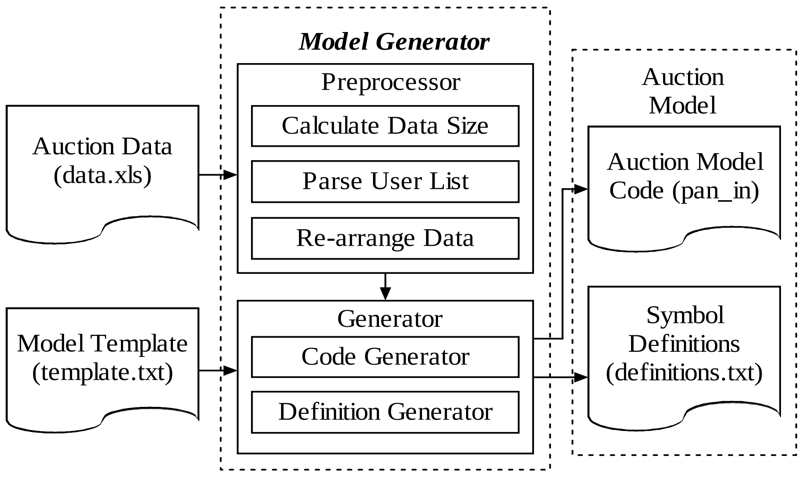


Figure 3: The generation of an auction model

1. *Calculate Data Size*: Calculate the number of bidding actions in each of the two auctions.
2. *Parse User List*: Parse the user names and store them in a file.
3. *Re-arrange Data:* According to the bidding time, interleave the bidding activities happened in the two concurrent auctions.

After the auction data has been preprocessed, it is passed to a module component called *Generator*. As shown in Figure 3, the model generator takes the auction model template and the preprocessed auction data to produce a specific auction model that consists of an auction model coded in PROMELA and an LTL symbol definition file. The auction model coded in PROMELA and the LTL symbol definition file will be passed to the SPIN model checker for property verifications.

4.3 The Auction Model Template

The auction model template we developed is based on English auctions, which is written in PROMELA language. The template allows us to generate different auction models according to different extracted numeric auction data from concurrent online English auctions systems, for example, the eBay auction house.

As shown in Table 4, in the auction model template, we first define the global variables (line 1~19), which are initialized using values extracted from the auction data in the `init` procedure (line 56~67). These global variables can be used to define symbols to compose LTL formulae. The symbols for composition of LTL formulae are defined in the symbol definition file, which is described in Section 4.4. In line 27~28, we define the local variables that can only be used by the model checker (we do not show the definitions of local variables in Table 4 due to space limitation).

The code between line 25~54 specifies the state transitions of the bidding process. When each auction round starts, all flags are cleared (line 33). Then according to different bidding cases, the model runs differently. For example, if the bidding case is “1”, it means that a bidder placed a bid in *Auction 0*, and at the same time, no one was bidding in *Auction 1*. To handle this case, we first set up the flags for *Auction 0*, and then we update all the old values of the relevant variables from the previous bid in *Auction 0* to the new values that represent the current bid (line 36~44). Similarly, if the bidding case is “2”, which means a bidder placed a bid in *Auction 1*, and at the same time, no one was bidding in *Auction 0*, we should set up the flags and update the relevant variables defined for *Auction 1* accordingly.

Table 4: Auction model template code

```
/* definitions of global variables */
int finalRound = ...;     // total number of rounds
byte biddingCase[...];    // sequence of bidding cases
byte flag0[...];          // flag for special events in auction0
int reservePrice0 = ...;  // reserve price set by seller
int currentHighestBid0 = ...;  // current highest bid in auction0
int previousHighestBid0 = ...; // previous highest bid in auction0
int increment0[...];      // increment of the bidding price
bit startPoint0 = 0;      // auction0 has not yet started
bit reservePoint0 = 0;    // reserve price has not yet reached
bit endPoint0 = 0;        // auction0 has not yet ended
...

typedef Auction {
  int dataSize;           // number of bids in the auction
  int timeInterval[...];  // time interval between two bids
  byte userIDs[...];      // user who places the bid
  int bidAmount[...];     // the amount of the bid
};

Auction auction0, auction1;
int timeElapse0, timeElapse1;
int roundCount = 0;

proctype ModelChecker() {

  /* definitions of local variables */
  ...
  checkingState:
  do
  ::(roundCount < finalRound)->   // auctions not completed
    d_step{                       // indivisibly code fragment
      ...                         // clear all flags
      if
      ::(biddingCase[roundCount]==1)-> // bidding case 1
        if
        ::(flag0[roundCount]==1)-> startPoint0 = 1;
        ::(flag0[roundCount]==2)-> reservePoint0 = 1;
        ::else -> skip;
        fi;
        increment0[userID0] = auction0.bidAmount[index0] –
           currentHighestBid0;    // increment of bidding price
        ...

      /* code for bidding case 2-9 */
      ...
      fi;
      roundCount++;
    }
    :: else -> goto endState;
    od;
  endState: skip;
}

init {
  bidSeq[0] = ...;                // set bidding cases
  ...
  auction0.dataSize = ...;        // set number of bids in auction0
  auction0.timeInterval[0] = ...; // set time interval for two bids
  auction0.userIDs[0] = ...;      // set user who places the bid
  auction0.bidAmount[0] = ...;    // set the amount of the bid
  ...
  flag0[0] = ...;                 // set flag for special events
  ...
  run ModelChecker();             // run the model checking process
}
```

Since each auction can be in a state of “bidding”, “not biding” and “end”, we have nine combinations for two concurrent online auctions. In Table 5, we list nine different biding cases, and each case represents a combined state for two concurrent online auctions. Note that if an auction is in the “not bidding” state, then no one is bidding in that auction while the auction has either not started or not yet finished.

Table 5: List of nine bidding cases

Auction / Bidding Case	**Auction 0**	**Auction 1**
Case 1	Bidding	Not Bidding
Case 2	Not Bidding	Bidding
Case 3	Not Bidding	Not Bidding
Case 4	Bidding	Bidding
Case 5	Bidding	End
Case 6	Not Bidding	End
Case 7	End	Not Bidding
Case 8	End	Bidding
Case 9	End	End

4.4 Symbol Definitions for LTL

In order to verify properties specified in LTL formulae, we need to define symbols that can be used in formula composition. Before we show the symbol definitions, we first define a few key notions as follows.

Definition 4.2 *Reserve Price*
Although sellers are not required to place reserve prices for auctioned items, the reserve price can be generalized as a parameterized value that is close to the final auction price. For our model checking purpose, we define the *reserve price* in an auction as a value that equals to 80 percent of the final auction price.

Definition 4.3 *Overbid*
In an auction, a bid is called an *overbid* if the price difference between the current bid and the previous bid is big enough, which is defined as 2 percent of the final auction price with a minimum of 10 dollars. An overbid in an auction is considered as a bid in large increment from the previous bid.

Definition 4.4 *Deliberate Bid*

In an auction, a bid is called a *deliberate bid* if the time gap between the current bid and the previous bid is long enough, which is defined as over 7200 seconds or 2 hours. A deliberate bid implies a deliberate decision made by a bidder.

Definition 4.5 *Underbid*

In an auction, a bid is called an *underbid* if the price difference between the current bid and the previous bid is small enough, which is defined as less than 3 dollars. An underbid in an auction is considered as a bid in small increment from the previous bid.

Definition 4.6 *Aggressive Bid*

In an auction, a bid is called an *aggressive bid* if the time gap between the current bid and the previous bid is small enough, which is defined as less than 60 seconds. An aggressive bid implies a quick response in driving up the bidding price.

Table 6 lists a few symbol definitions that are used for formula composition described in Section 5. We define most of the terms to be self-explanatory. For example, `start0` (`start1`) denotes the start of *Auction 0* (*Auction 1*); while `end0` (`end1`) denotes the end of *Auction 0* (*Auction 1*). Similarly, the symbol `reserve0` (`reserve1`) denotes that the reserve price of *Auction 0* (*Auction 1*) is reached. However, for a term like `bid0_0`, the first "`0`" denotes *Auction 0*, and the second "`0`" denotes the bidding behavior of *User 0*. Thus, if a user numbered `16` bids in *Auction 0*, then this event should be represented by the symbol `bid0_16`.

Table 6: Symbol definitions for LTL formulae

```
// the bidding activities of the users
#define bid0_0  (increment0[0] > 0)
#define bid0_1  (increment0[1] > 0)
...
// users' bidding in large increments
#define overBid0_0 (increment0[0] > ...)
#define overBid0_1 (increment0[1] > ...)
...
// users' bidding in small increments
#define underBid0_0 (increment0[0] < ...)
#define underBid0_1 (increment0[1] < ...)
...
// definition of deliberate bids
#define deliBid0  (timeElapse0 > ...)
#define deliBid1  (timeElapse1 > ...)
// definition of aggressive bids
```

```
#define aggrBid0  (timeElapse0 < ...)
#define aggrBid1  (timeElapse1 < ...)
// start of auctions
#define start0  (startPoint0==1)
#define start1  (startPoint1==1)
// the events that the reserve price has been reached
#define reserve0 (reservePoint0==1)
#define reserve1 (reservePoint1==1)
// end of auctions
#define end0 (endPoint0==1)
#define end1 (endPoint1==1)
// the condition that the bidding price in one auction
// is lower than that in another one
#define p0Lower
   ((currentHighestBid0-previousHighestBid1) < 0)
#define p1Lower
   ((currentHighestBid1-previousHighestBid0) < 0)
```

Symbol definitions can be used to ease the task of writing LTL formulae. For example, the predicate "User 5 bids in *Auction 0* && Price is lower in *Auction 1*" can be written as `(bid0_5 && p1Lower)`. Note that based on Definition 4.2-4.6, the set of symbol definitions can be automatically generated.

4.5 The Model Checking Process

After the auction model has been created and the LTL formulae have been designed, the model checking process becomes straightforward. The model checking process can be automated using the SPIN model checker. As shown in Figure 4, the SPIN formula translator first translates the LTL formula into a *never* claim, which is used to match behaviors that should *never* occur. With the *never* claim, the verification system could flag it as an error if the full behavior specified in the claim could be matched by any feasible system execution [29]. The *never* claim is written in PROMELA code, so it can be appended to the system definition file in the SPIN verifier generator. When the model is running, the claim process is executed at each step of the system. As soon as the property specified in the claim is violated, the system terminates and indicates that the error behavior occurred; otherwise, the LTL formula will evaluate to *valid*.

The SPIN verifier generator generates the model verifier source code based on the auction model in PROMELA code and the system definition file appended by the *never* claim. The verification source code is then compiled into an executable file using a *gcc* compiler. By running the executable model verifier, we can get the model checking result, which is either *valid* or *invalid*. An *invalid* result indicates that during the verification process, the model verifier encountered errors. In other words, the auction model we developed violates the property specified in the LTL formula. In contrast, if the result is *valid*, it indicates that the behavioral property we specified is satisfied by the auction model.

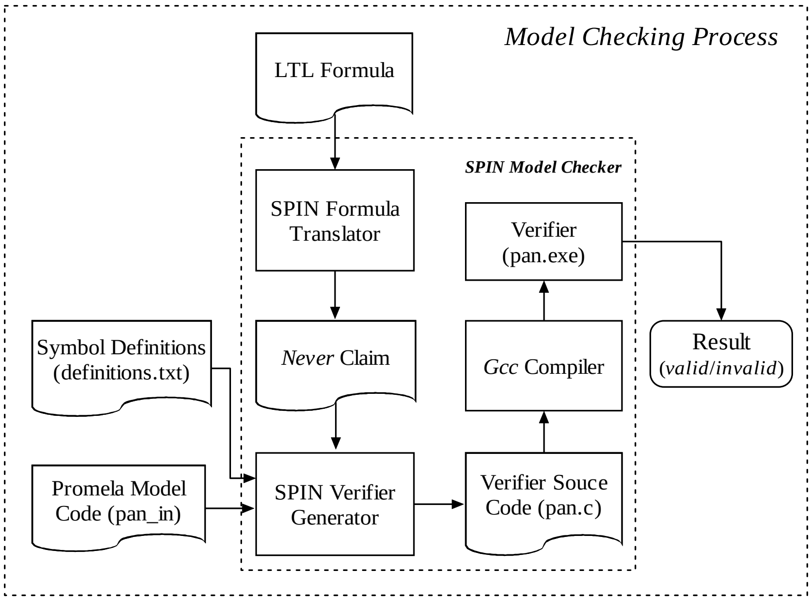


Figure 4: Model checking process

5. Bidding Behavior Predicates

To facilitate the process of model checking bidding behaviors and detecting shill suspects in concurrent online auctions, we developed a toolkit that can be used to calculate a user's accumulated points for being a shill bidder, called *S-Points*. The toolkit is based on our previous implementation of the pattern-based model checking tool that can help to automate the model checking process for data processing, LTL formula design and invocation of the SPIN model checker [32]. The calculation of *S-Points* is according to a set of bidding behavior predicates that denote either *positive* or *negative* indications for shill bidders. A *positive* indication of shilling behaviors is a bidding pattern that indicates a user is likely a shill; while a *negative* indication of shilling behaviors is a bidding pattern that indicates a user is *not* likely a shill.

Most of the previous work only checks positive indication of shilling behaviors in detecting shill bidders. We argued that negative indications of shilling behaviors are equally important in determining the likelihood of a shill bidder by providing evidence that the user is *not* likely a shill. In the following, we summarize the bidding behavior predicates for both positive and negative indications of shilling behaviors, and their associated temporal formulae that are used in our toolkit.

Predicate 1: User bids in an auction only after the reserve price in that auction is reached.
Pattern Used: Precedence, globally.
Formula 1.1 User bids in *Auction 0* only after the reserve price in *Auction 0* is reached.
`(!bid0_i W reserve0)` i.e., `(!bid0_i U reserve0)||([]!bid0_i)`
Formula 1.2 User bids in *Auction 1* only after the reserve price in *Auction 1* is reached.
`(!bid1_i W reserve1)` i.e., `(!bid1_i U reserve1)||([]!bid1_i)`
Explanation: A user bids only after the reserve price is reached in an auction is most likely a *last-minute* bidder or called a bid sniper, who only cares about if he can win the auction. Thus, a user with this kind of behavior (when the formula evaluates to be *valid*) is *not* likely a shill bidder. The predicate `bid0_i` in Formula 1.1 refers to the bidding activities of the user numbered `i` in *Auction 0*, which is defined in the symbol definition file. Similarly, the predicate `bid1_i` in Formula 1.2 refers to the bidding activities of the user numbered `i` in *Auction 1*. Since the SPIN does not directly support the temporal operator `W` (unless) in a LTL formula, the resulting formula needs to be converted to a LTL formula without using the `W` operator. The conversion can be automatically done in our toolkit according to the *valid* formula of "`p W q <=> (p U q || []p)`." Thus, for the following temporal formulae, we always use the `W` operator directly when needed.

Predicate 2: User bids in one auction only after another auction ends.
Pattern Used: (1) Existence, after *Q*; (2) Precedence, globally.
Formula 2.1: User bids in *Auction 0* only after *Auction 1* ends.
`([](!end1) || <>(end1 && <>bid0_i))) && (!bid0_i U end1)||([]!bid0_i)`
Formula 2.2: User bids in *Auction 1* only after *Auction 0* ends.
`([](!end0) || <>(end0 && <>bid1_i))) && (!bid1_i U end0)||([]!bid1_i)`
Explanation: Each of the above formula consists of two parts. For example, in Formula 2.1, the first part of the formula `([](!end1) || <>(end1 && <>bid0_i)))` is used to test if a user bids in *Auction 0* after *Auction 1* ends, i.e., the event of the user's bidding activity in *Auction 0* exists after *Auction 1* ends (denoted by `end1`). The second part of the Formula 2.1 `(!bid0_i U end1)||([]!bid0_i)` is used to test if the user bids in *Auction 0* only after *Auction 1* ends, i.e., the event of `end1` precedes *any* event of the user's bidding activity in *Auction 0*. A user with such behavior only bids when an auction is close to the end, and focuses on bidding one auction at a time.

This kind of bidding behavior is very normal, though the bidder may have to bid on an item with higher price using his bidding strategy.

Predicate 3: User places a deliberate overbid before the reserve price in that auction is reached, but does not bid at all thereafter.

Pattern Used: (1) Existence, before *R*; (2) Absence, after *Q*.

Formula 3.1: User places a deliberate overbid before the reserve price in *Auction 0* is reached, but does not bid at all thereafter.

```
((!reserve0 W ((overBid0_i && deliBid0) && !reserve0))) && ([](reserve0 ->
[](!bid0_i)))
```

Formula 3.2: User places a deliberate overbid before the reserve price in *Auction 1* is reached, but does not bid at all thereafter.

```
((!reserve1 W ((overBid1_i && deliBid1) && !reserve1))) && ([](reserve1 ->
[](!bid1_i)))
```

Explanation: This property implies that a user places an overbid to stimulate the bidding when he notices that it has been a while since the previous bid was placed, and he stops bidding after the bid reaches the reserve price. This behavior is highly suspicious for shill biddings. Note that a true value of `deliBid0` means that in *Auction 0*, the time interval between the currently placed bid and the previous bid is longer than the limit we have set, so the bid is considered as a deliberate bid. The formula `(overBid0_i && deliBid0)` specifies that user *i* places a deliberate overbid in *Auction 0*.

Predicate 4: User always places aggressive underbids in either of the concurrent auctions.

Pattern Used: Universality, Between *Q* and *R*.

Formula 4.1: User always places aggressive underbids in *Auction 0*.

```
[]((start0 && !end0 && <>end0) → ((underBid0_i && aggrBid0) U end0))
```

Formula 4.2: User always places aggressive underbids in *Auction 1*.

```
[]((start1 && !end1 && <>end1) → ((underBid1_i && aggrBid1) U end1))
```

Explanation: This property implies that a user always places small bids aggressively to outbid legitimate bids in order to drive up the final auction price, which is also a typical shilling strategy. This formula uses the *Universality* pattern, which can be derived from the *Absence* pattern by substituting `!P` with `P` in each of the formulae in Table 1.

Predicate 5: User wins either of the auctions in two concurrent auctions.

Pattern Used: Existence, Before *R*.

Formula 5.1: User wins *Auction 0*.

```
!end0 W ((winningBid0 && bid0_i) && !end0)
```

Formula 5.2: User wins *Auction 1*.

```
!end1 W ((winningBid1 && bid1_i) && !end1)
```

Explanation: If a shill accidentally wins an auction, the auction will have to be repeated with cost to the seller [20]. Thus, a typical shill will try his best to avoid winning an auction, and a bidder who actually wins an auction is *not* likely a shill.

Predicate 6: During the overlapping of two concurrent auctions, user bids in an auction that has higher bidding price.

Pattern Used: Existence, Between *Q* and *R*.

Formula 6.1: During the overlapping of *Auction 0* and *Auction 1*, user bids in *Auction 0* that has higher bidding price. Since we assume the auction that starts first is always `Auction 0`, `start1` always does *not* precede `start0`. Therefore, we only need to consider two cases, i.e., `end1` does *not* precede `end0` (`F6_1_1`) and `end0` does *not* precede `end0` (`F6_1_2`).

```
F6_1_1: !(!end0 W end1) -> ([](start1 && !end0 -> (!end0 W ((bid0_i &&
p1Lower) && !end0))))
F6_1_2: !(!end1 W end0) -> ([](start1 && !end1 -> (!end1 W ((bid0_i &&
p1Lower) && !end1))))
```

The complete Formula 6.1 is `(F6_1_1 || F6_1_2).`

Formula 6.2: During the overlapping of *Auction 0* and *Auction 1*, user bids in *Auction 1* that has higher bidding price. Similarly, we have two cases for formula 6.2: `end1` does *not* precede `end0` (`F6_2_1`) and `end1` does *not* precede `end0` (`F6_2_2`).

```
F6_2_1: !(!end0 W end1) -> ([](start1 && !end0 -> (!end0 W ((bid1_i &&
p0Lower) && !end0))))
F6_2_2: !(!end1 W end0) -> ([](start1 && !end1 -> (!end1 W ((bid1_i &&
p0Lower) && !end1))))
```

The complete Formula 6.2 is `(F6_2_1 || F6_2_2).`

Explanation: During the overlapping time of the two auctions, any user who does not bid in the auction that has lower bidding price could be suspicious. The purpose of bidder's such bidding behavior could be simply to drive up the auction price without caring about winning the auction with a good price. Since a shill bidder would try to avoid bidding after the reserve price is reached in order to avoid winning the auction, we can further narrow down the pattern scope to make the result more accurate. The new scope is during the overlapping of two concurrent auctions, but before the earlier reserve price has been reached. This case is described in *Predicate 7*.

Predicate 7: During the overlapping of two concurrent auctions, but before the earlier reserve price has been reached, user bids in auctions that has higher bidding prices.

Pattern Used: (1) Existence, Between *Q* and *R*; (2) Precedence, globally.

Formulae 7.1: During the overlapping of *Auction 0* and *Auction 1*, but before the earlier reserve price has been reached, user bids in *Auction 0* that has higher bidding price. Similar to Formula 6.1, here we need to consider two cases: `reserve1` does *not* precede `reserve0` (`F7_1_1`) and `reserve0` does *not* precede `reserve1` (`F7_1_2`).

```
F7_1_1: !(!reserve0 W reserve1) -> ([](start1 && !reserve0 -> (!reserve0 W
((bid0_i && p1Lower) && !reserve0))))
F7_1_2: !(!reserve1 W reserve0) -> ([](start1 && !reserve1 -> (!reserve1 W
((bid0_i && p1Lower) && !reserve1))))
```

The complete Formula 7.1 is: `(F7_1_1 || F7_1_2).`

Formula 7.2: During the overlapping of *Auction 0* and *Auction 1*, but before the earlier reserve price has been reached, user bids in *Auction 1* that has higher bidding price. Similarly, we have two cases: `reserve1` does *not* precede `reserve0` (`F7_2_1`) and `reserve0` does *not* precede `reserve1` (`F7_2_2`).

```
F7_2_1: !(!reserve0 W reserve1) -> ([](start1 && !reserve0 -> (!reserve0 W
((bid1_i && p0Lower) && !reserve0))))
F7_2_2: !(!reserve1 W reserve0) -> ([](start1 && !reserve1 -> (!reserve1 W
((bid1_i && p0Lower) && !reserve1))))
```

The complete Formula 7.2 is: `(F7_2_1 || F7_2_2).`

Explanation: This property is similar to Property 6, but with narrowed pattern scope. It should be noted that any user who satisfies Property 7 must also satisfy Property 6. In this case, the user with such bidding behavior will actually receive points twice towards his *S-Points.* However, this is reasonable because a user who satisfies Property 7 is highly suspicious for being a shill bidder.

To calculate the *S-Points* of a user, we define the following formulae to accumulate the user's *S-Points* according to various temporal formulae that specify either positive or negative indications of shilling behaviors.

point(*U*, 0).

point(*U*, *P*+1) :- *positive*-*indication*(*U*, *PF*), *point*(*U*, *P*).

point(*U*, *P*-1) :- *negative*-*indication*(*U*, *NF*), *point*(*U*, *P*).

Here, *positive*-*indication*(*U*, *PF*) is a predicate that denotes user *U* is a possible shill according to temporal formula *PF*, *negative*-*indication*(*U, NF*) is a predicate that denotes user *U* is *not* likely a shill according to temporal formula *NF*, and *point*(*U*, *P*) is a predicate that denotes user *U*'s accumulated *S*-*Points* is *P*. User *U* has initially 0 *S*-*Points*. The corresponding *S*-*Point* of each temporal formula described above is listed in Table 7. Note that in Table 7, a "P" or "N" indication of a temporal formula denotes that the formula is positive or negative indication of shilling behaviors, respectively.

Table 7: Usage of temporal formulae in calculating user's S-Points

Formula	1.1	1.2	2.1	2.2	3.1	3.2	4.1	4.2	5.1	5.2	6.1	6.2	7.1	7.2
Indication	N	N	N	N	P	P	P	P	N	N	P	P	P	P
S-Point	-1	-1	-1	-1	1	1	1	1	-1	-1	1	1	1	1

According to Table 7, a user's *S*-*Points* can be calculated by accumulating the *S*-*Point* based on the verification results of the aforementioned temporal formulae. A user receive an *S*-*Point* (either -1 or 1) associated with a temporal formula if and only if the temporal formula is evaluated to be *valid* for the user. The higher *S*-*Points* a user has, the higher possibility the user is a shill bidder.

6. Case Study: Detection of Shill Bidders

The purpose of model checking bidding behaviors in Internet concurrent auctions is to gain a better understanding of online auctions, and more importantly, it can be used to efficiently detect shill suspects in concurrent online auctions. In this section, we use a case study to show how potential shills can be detected using our model checking approach.

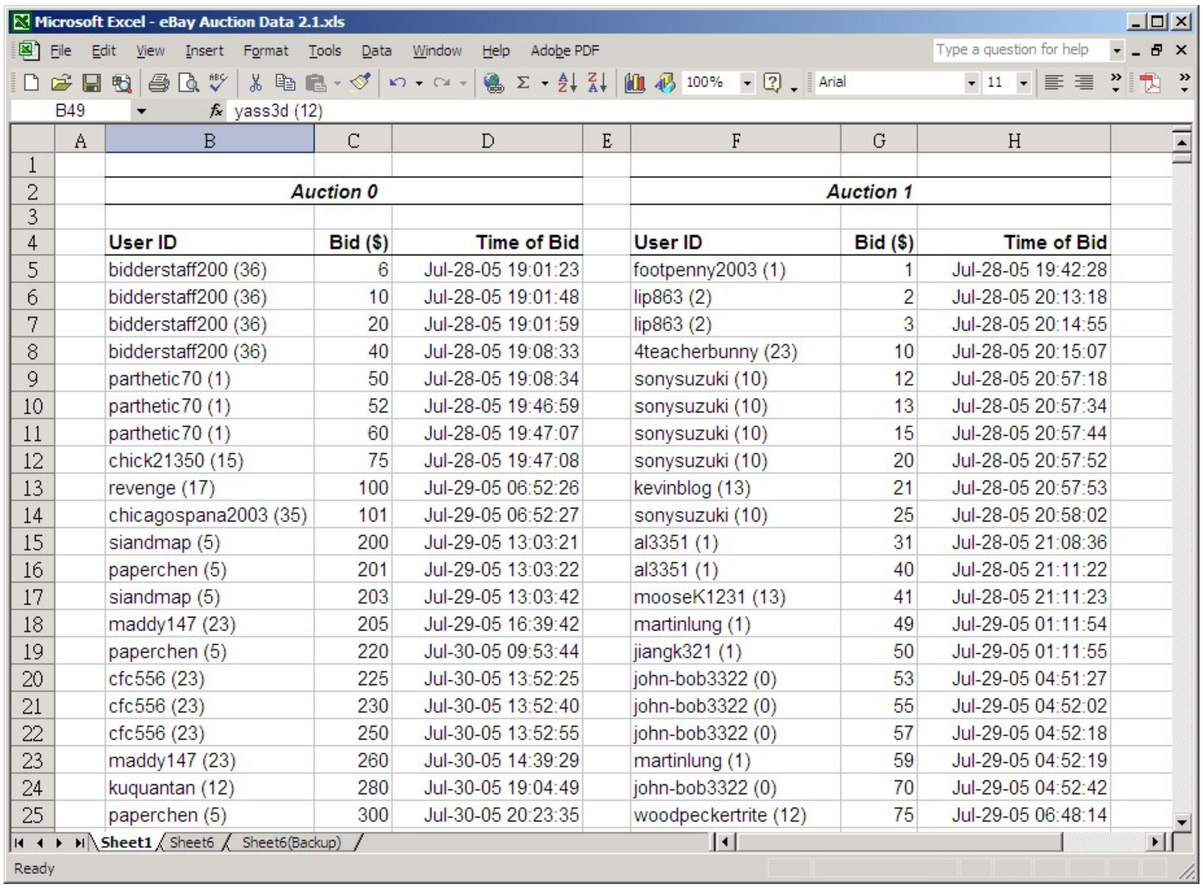
We collected some recent auction data of two concurrent auctions from eBay with the title of auctioned items as "HP/COMPAQ PRESARIO LAPTOP CD-RW BURNER DVD WIRELESS."[1] Both auctions are held by the same seller and the detailed descriptions of the auctioned items are shown in Table 8.

[1] We have applied our approach on various real auction data collected from eBay. The reason we chose this example as our case study is that it represents a typical concurrent auction with potential shill bidders.

Table 8: Descriptions of the auctioned items

Item Specifics – PC Laptops			
Brand:	**Compaq**	Hard Drive Cap:	**60 GB**
Chip Type:	**--**	Screen Size:	**15 inches**
Model:	**--**	OS Included:	**Yes**
Processor Speed:	**1.4 GHz**	Primary Drive:	**CD-RW/DVD Combo**
Memory	**512 MB**	Condition:	**--**

Some of the auction data for the two concurrent auctions is listed in Figure 5. To protect the privacy of the users, we have changed all user IDs. In addition, we have made the following adjustments on the raw auction data. We erased all currency symbols and time zone abbreviations to make them appear simpler. We also rounded up all bidding prices that have decimals because the SPIN tool cannot handle decimals. Note that each user name is associated with a numeric value in parentheses, such as paperchen(5). The number represents the user's *feedback score*, and usually, a higher feedback score is a good sign for better comments and higher rating for the user.



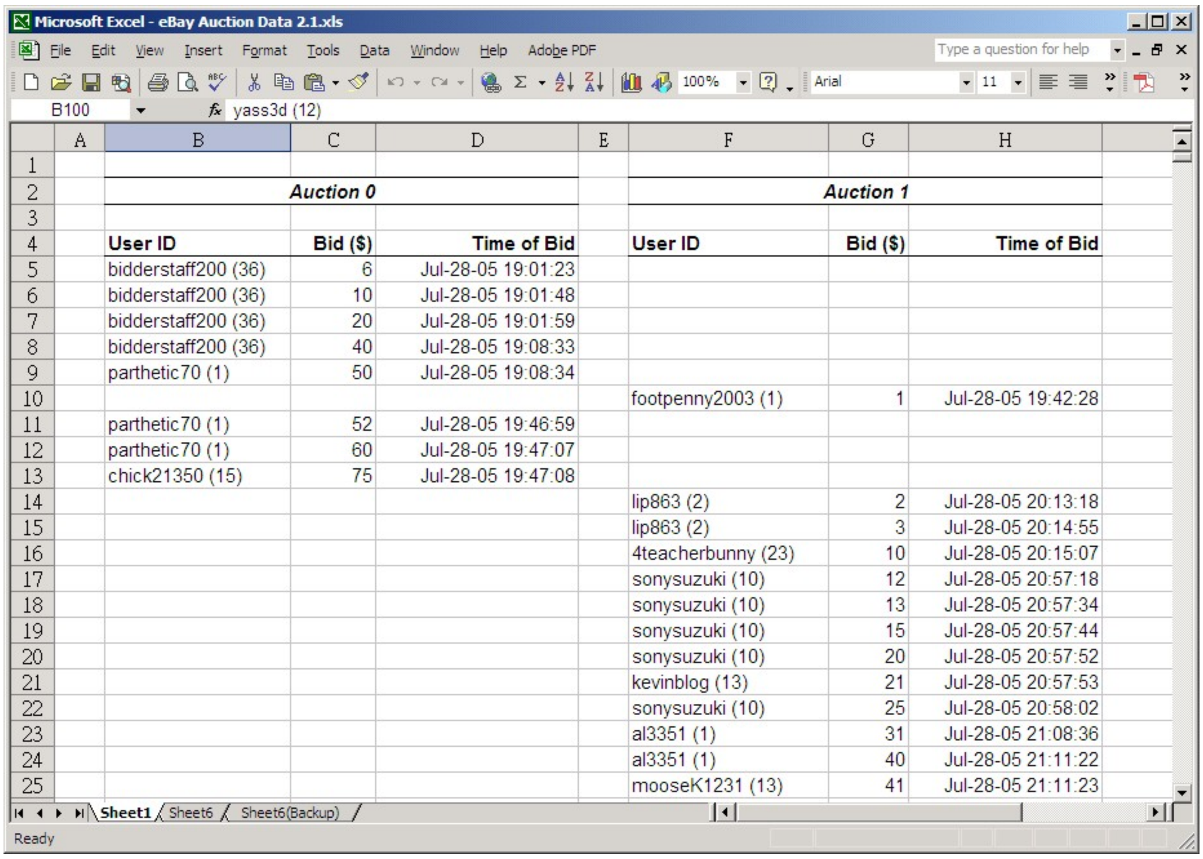
Microsoft Excel - eBay Auction Data 2.1.xls
File Edit View Insert Format Tools Data Window Help Adobe PDF
Type a question for help
100%
Arial
11
B49
yass3d (12)
Sheet1
Sheet6
Sheet6(Backup)
Ready

	A	B	C	D	E	F	G	H
1								
2		*Auction 0*				*Auction 1*		
3								
4		**User ID**	**Bid ($)**	**Time of Bid**		**User ID**	**Bid ($)**	**Time of Bid**
5		bidderstaff200 (36)	6	Jul-28-05 19:01:23		footpenny2003 (1)	1	Jul-28-05 19:42:28
6		bidderstaff200 (36)	10	Jul-28-05 19:01:48		lip863 (2)	2	Jul-28-05 20:13:18
7		bidderstaff200 (36)	20	Jul-28-05 19:01:59		lip863 (2)	3	Jul-28-05 20:14:55
8		bidderstaff200 (36)	40	Jul-28-05 19:08:33		4teacherbunny (23)	10	Jul-28-05 20:15:07
9		parthetic70 (1)	50	Jul-28-05 19:08:34		sonysuzuki (10)	12	Jul-28-05 20:57:18
10		parthetic70 (1)	52	Jul-28-05 19:46:59		sonysuzuki (10)	13	Jul-28-05 20:57:34
11		parthetic70 (1)	60	Jul-28-05 19:47:07		sonysuzuki (10)	15	Jul-28-05 20:57:44
12		chick21350 (15)	75	Jul-28-05 19:47:08		sonysuzuki (10)	20	Jul-28-05 20:57:52
13		revenge (17)	100	Jul-29-05 06:52:26		kevinblog (13)	21	Jul-28-05 20:57:53
14		chicagospana2003 (35)	101	Jul-29-05 06:52:27		sonysuzuki (10)	25	Jul-28-05 20:58:02
15		siandmap (5)	200	Jul-29-05 13:03:21		al3351 (1)	31	Jul-28-05 21:08:36
16		paperchen (5)	201	Jul-29-05 13:03:22		al3351 (1)	40	Jul-28-05 21:11:22
17		siandmap (5)	203	Jul-29-05 13:03:42		mooseK1231 (13)	41	Jul-28-05 21:11:23
18		maddy147 (23)	205	Jul-29-05 16:39:42		martinlung (1)	49	Jul-29-05 01:11:54
19		paperchen (5)	220	Jul-30-05 09:53:44		jiangk321 (1)	50	Jul-29-05 01:11:55
20		cfc556 (23)	225	Jul-30-05 13:52:25		john-bob3322 (0)	53	Jul-29-05 04:51:27
21		cfc556 (23)	230	Jul-30-05 13:52:40		john-bob3322 (0)	55	Jul-29-05 04:52:02
22		cfc556 (23)	250	Jul-30-05 13:52:55		john-bob3322 (0)	57	Jul-29-05 04:52:18
23		maddy147 (23)	260	Jul-30-05 14:39:29		martinlung (1)	59	Jul-29-05 04:52:19
24		kuquantan (12)	280	Jul-30-05 19:04:49		john-bob3322 (0)	70	Jul-29-05 04:52:42
25		paperchen (5)	300	Jul-30-05 20:23:35		woodpeckertrite (12)	75	Jul-29-05 06:48:14

Figure 5: Auction data from two concurrent auctions before preprocessing

Since the winning bids of the two concurrent auctions (*Auction 0* and *Auction 1*) are \$630 and \$620, according to Definition 4.2, the reserve prices for the two auctions are \$504 and \$496, respectively. We set the price difference for overbid as \$13 and \$12 for *Auction 0* and *Auction 1*, respectively, which are 2% of the final auction prices by Definition 4.3. We also set the price difference for underbid as \$3 according to Definition 4.5. By Definition 4.4 and 4.6, the time gap for deliberate bid and aggressive bid are 7200 seconds and 60 seconds, respectively. In practice, the above parameters can be further adjusted according to auction administrator's experiences and observations.

In the two concurrent online auctions, there are totally 35 users, among which we selected to investigate the following four users: "paperchen", "benniten23", "andy293" and "yass3d". This is because these four users are the only bidders who are involved in both of the two concurrent auctions, and thus, they are more likely to be shill bidders.


Microsoft Excel - eBay Auction Data 2.1.xls
File Edit View Insert Format Tools Data Window Help Adobe PDF
B100
yass3d (12)
Sheet1
Sheet6
Sheet6(Backup)
Ready

	A	B	C	D	E	F	G	H
1								
2		Auction 0				Auction 1		
3								
4		User ID	Bid (\$)	Time of Bid		User ID	Bid (\$)	Time of Bid
5		bidderstaff200 (36)	6	Jul-28-05 19:01:23				
6		bidderstaff200 (36)	10	Jul-28-05 19:01:48				
7		bidderstaff200 (36)	20	Jul-28-05 19:01:59				
8		bidderstaff200 (36)	40	Jul-28-05 19:08:33				
9		parthetic70 (1)	50	Jul-28-05 19:08:34				
10						footpenny2003 (1)	1	Jul-28-05 19:42:28
11		parthetic70 (1)	52	Jul-28-05 19:46:59				
12		parthetic70 (1)	60	Jul-28-05 19:47:07				
13		chick21350 (15)	75	Jul-28-05 19:47:08				
14						lip863 (2)	2	Jul-28-05 20:13:18
15						lip863 (2)	3	Jul-28-05 20:14:55
16						4teacherbunny (23)	10	Jul-28-05 20:15:07
17						sonysuzuki (10)	12	Jul-28-05 20:57:18
18						sonysuzuki (10)	13	Jul-28-05 20:57:34
19						sonysuzuki (10)	15	Jul-28-05 20:57:44
20						sonysuzuki (10)	20	Jul-28-05 20:57:52
21						kevinblog (13)	21	Jul-28-05 20:57:53
22						sonysuzuki (10)	25	Jul-28-05 20:58:02
23						al3351 (1)	31	Jul-28-05 21:08:36
24						al3351 (1)	40	Jul-28-05 21:11:22
25						mooseK1231 (13)	41	Jul-28-05 21:11:23

Figure 6: Auction data from two concurrent auctions after preprocessing

After the auction data is preprocessed using our model checking tool, the auction data is re-arranged as shown in Figure 6. Notice that the data from the two concurrent auctions is interleaved according to the bidding time. Based on the interleaved auction data, we can draw the price-time diagram and show the overlapping style of the two concurrent auctions. As shown in Figure 7, S0/S1 represents the event of "*Auction 0*/*Auction 1* starts" and E0/E1 represents the event of "*Auction 0*/*Auction 1* ends," thus the two concurrent auctions overlap in time period [S1, E0]. Since we set $504 and $496 as the reserve prices for *Auction 0* and *Auction 1*, respectively, the small filled circles (denoted as R0 and R1 in Figure 7) represent the events of "*Auction 0* reaches the reserve price" and "*Auction 1* reaches the reserve price."

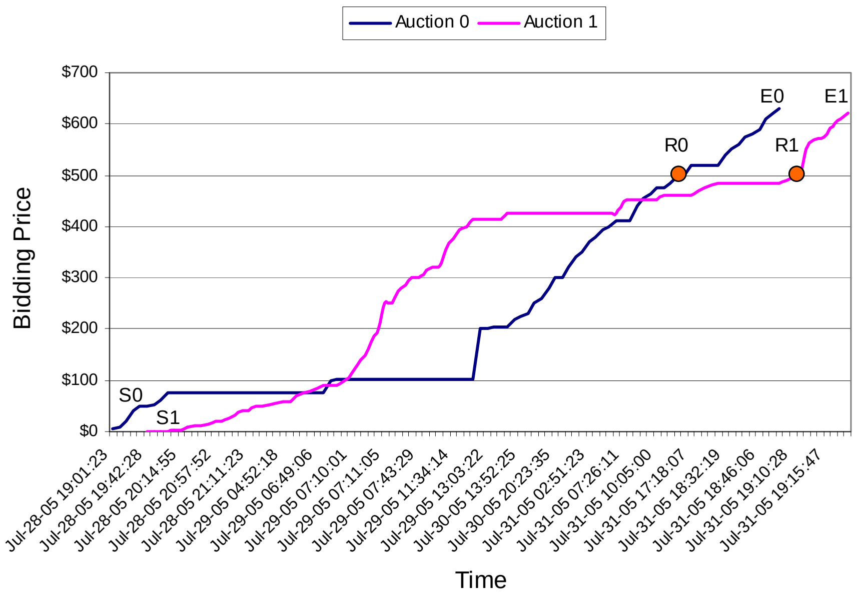


Figure 7: Overlapping style of the two concurrent auctions

We now use our model checking tool to verify the bidding behavioral properties of the four users according to the temporal formulae described in Section 4. The computation only took a few seconds, which is quite efficient for any reasonable sizes of auction data as shown in this case study. The model checking results for all four users are shown in Table 9. Note that the number listed in the brackets immediately following each verification results represents the *S-Point*. The total *S-Points* of a user is the sum of the *S-Point* for each LTL formula.

Table 9: Model checking results for 4 users

User / **Formula**	paperchen (5)	benniten23 (1)	andy293 (12)	yass3d (12)
Formula 1.1	*invalid* [0]	*invalid* [0]	*valid* [-1]	*valid* [-1]
Formula 1.2	*invalid* [0]	*invalid* [0]	*valid* [-1]	*invalid* [0]
Formula 2.1	*invalid* [0]	*invalid* [0]	*invalid* [0]	*invalid* [0]
Formula 2.2	*invalid* [0]	*invalid* [0]	*valid* [-1]	*invalid* [0]
Formula 3.1	*valid* [1]	*invalid* [0]	*invalid* [0]	*invalid* [0]
Formula 3.2	*valid* [1]	*invalid* [0]	*invalid* [0]	*invalid* [0]
Formula 4.1	*invalid* [0]	*invalid* [0]	*invalid* [0]	*invalid* [0]
Formula 4.2	*invalid* [0]	*invalid* [0]	*invalid* [0]	*invalid* [0]
Formula 5.1	*invalid* [0]	*invalid* [0]	*invalid* [0]	*invalid* [0]
Formula 5.2	*invalid* [0]	*invalid* [0]	*invalid* [0]	*invalid* [0]
Formula 6.1	*invalid* [0]	*valid* [1]	*valid* [1]	*valid* [1]
Formula 6.2	*valid* [1]	*valid* [1]	*invalid* [0]	*invalid* [0]
Formula 7.1	*invalid* [0]	*valid* [1]	*invalid* [0]	*invalid* [0]
Formula 7.2	*valid* [1]	*valid* [1]	*invalid* [0]	*invalid* [0]
Total S-Points	4	4	-2	0

Based on the above model checking results, we may draw our conclusions as follows:

1. User "paperchen" has *S-Points* of 4. This user is most likely a shill bidder due to the following two major reasons: (1) user "paperchen" attempts to drive up the bidding price, and stops bidding after the price reached the reserve price. When a certain time has passed after the last bid was placed, the user would try to create a competitive bidding atmosphere by placing overbids (according to the verification results of Formula 3.1 and 3.2); (2) during the time period [S1, R0] (refer to Figure 7), the user places bids in *Auction 1* that has higher bidding price (according to the verification results of Formula 6.2 and 7.2).
2. User "benniten23" also has *S-Points* of 4. The user is very likely a shill bidder because he puts in bids on an item that has higher bidding price in both *Auction 0* and *Auction 1* (according to the verification results of Formula 6.1, 6.2, 7.1 and 7.2). The user has such bidding behavior during the overlapping of the two auctions, especially before R0 has been reached (refer to Figure 7). Thus, the purpose of such activities is very likely to drive up the bidding price.
3. User "andy293" and "yass3d" have *S-Points* of -2 and 0, respectively. Both of the users are *not* likely shills because their bidding behaviors look very normal. Both of them like to bid when the auctions are close to the ends (according to the verification results of Formula 1.1, 1.2). In addition, user "andy293" likes to bid in one auction at a time (according to the verification result of Formula 2.2).

The aim of our approach is to detect shill suspects efficiently based on auction data from a number of concurrent auctions, where analysis of large volumes of historical auction data is not required. The model checking results provide us evidences for detecting shill suspects in concurrent online auctions, and our case study shows that shill suspects can be efficiently detected. However, the above analysis is *not* sufficient for drawing a conclusion that both “paperchen” and “benniten23” must be shills. To collect more evidences for a more accurate result, we may further verify additional properties that combine with other evidences such as users’ IP addresses, ratings and trading histories. Details on verifying additional evidence of users are outside the scope of focus for this paper, but can be found in earlier work [16, 18, 20].

7. Conclusions and Future Work

In this paper, we introduced a model checking approach to verifying bidding behaviors including shilling behaviors and normal bidding behaviors in concurrent online auctions. We proposed an auction model template that supports automatic generation of auction models based on auction data from two concurrent online auctions. We also summarized a set of LTL formulae that specify bidding behaviors of users for detection of shill suspects in concurrent online auctions. The case study, which is based on auction data from an existing auction house – eBay, shows that our approach is feasible and efficient. Our approach can be easily extended to support model checking bidding behaviors in more than two concurrent online auctions. For our future work, we will try to combine our approach with other approaches, such as the statistical approaches, to develop a more accurate method for detection, prevention, and prediction of shill bidders.

Acknowledgments: This material is based upon work supported by the Chancellor’s Research Fund and UMass Joseph P. Healey Endowment Grants. We thank Jarrod Trevathan for proof-reading our paper and providing some valuable references. We also thank all anonymous referees for the careful review of this paper and the many suggestions for improvements they provided.

References

1. M. Harkavy, J. D. Tygar, and H. Kikuchi. Electronic Auctions with Private Bids. *Proceedings of the 3rd USENIX Workshop on Electronic Commerce*, September 1998, pp. 61-73.

2. A. Vakali, L. Angelis, D. Pournara. Internet Based Auctions: A Survey on Models and Applications. *ACM SIG on E-commerce Exchanges*, Vol. 2, No. 2, Jun 2001, pp. 5-13.
3. R. J. Kauffman and C. A. Wood. Running up the Bid: Detecting, Predicting, and Preventing Reserve Price Shilling in Online Auctions. *Proceedings of the 5th International Conference on Electronic Commerce*, Pittsburgh, Pennsylvania, 2003, pp. 259-265.
4. W. Wang, H. Zoltán, and A. B. Whinston. Shill Bidding in Multi-Round Online Auctions. *Proceedings of the 35th Hawaii International Conference on System Sciences* (HICSS'02), Hawaii, 2002.
5. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 2000.
6. G. J. Holzmann. The Model Checker SPIN. *IEEE Transactions on Software Engineering*, Vol. 23, No. 5, 1997, pp. 279-295.
7. L. K. Dillon and S. Sanka. Introduction to the Special Issue. *IEEE Transactions on Software Engineering*, Special Issue on Formal Methods in Software Practice, Vol. 23, No. 5, May, 1997, pp. 265-266.
8. H. Xu and S. M. Shatz. A Framework for Model-Based Design of Agent-Oriented Software. *IEEE Transactions on Software Engineering*, January 2003, Vol. 29, No. 1, pp. 15-30.
9. Y. Cheng and H. Xu. A Formal Approach to Detecting Shilling Behaviors in Concurrent Online Auctions. *Proceedings of the 8th International Conference on Enterprise Information Systems (ICEIS 2006)*, May 2006, Paphos, Cyprus, pp. 375-381.
10. M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Property Specification Patterns for Finite-state Verification. *Proceedings of the 2nd Workshop on Formal Methods in Software Practice*, March 1998.
11. M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in Property Specifications for Finite-State Verification. *Proceedings of the 21st International Conference on Software Engineering*, Los Angeles, 1999, pp. 16-22.
12. C. Dellarocas and P. Resnick. Online Reputation Mechanisms: A Roadmap for Future Research. *Summary Report of the First Interdisciplinary Symposium on Online Reputation Mechanisms*, April 26-27, 2003.
13. C. A. Wood, M. Fan, and Y. Tan. An Examination of the Reputation Systems for Online Auctions. *Proceedings of the Workshop for Information Systems and Economics* (WISE 2002), Barcelona, Spain, Dec. 2002.
14. P. Resnick, R. Zeckhauser, E. Friedman, and K. Kuwabara. Reputation Systems. *Communications of the ACM*, Vol. 43, No. 12, December 2000, pp. 45-48.

15. J. H. Dobrzynski. In Online Auctions, Rings of Bidders. *The New York Times*, June 2, 2000.
16. S. Rubin, M. Christodorescu, V. Ganapathy, J. Giffin, et al. An Auctioning Reputation System Based on Anomaly Detection. *Proceedings of the 12nd ACM Conference on Computer and Communication Security*, Alexandria, 2005.
17. I. Chakraborty and G. Kosmopoulou. Auctions with Shill Bidding. *Economic Theory*, Springer, Vol. 24, Issue 2, 2004, pp. 271-287.
18. R. J. Kauffman and C. A. Wood. Running up the Bid: Modeling Seller Opportunism in Internet Auctions. *Proceedings of the Sixth Americas Conference on Information Systems* (AMCIS 2000), M. Chung (ed.), Long Beach, CA, 2000, pp. 929-935.
19. R. Bapna and A. Gupta. Online Auctions: A Closer Look. *The E-Business Handbook*, P. B. Lowry, R. J. Watson, and J. O. Cherrington (eds.), St. Lucie Press, Boca Raton, FL, 2002, pp. 85-98.
20. J. Trevathan and W. Read. Detecting Shill Bidding in Online English Auctions. *Technical Report*, James Cook University, May 2006.
21. M. Huget, M. Esteva, S. Phelps, C. Sierra, et al. Model Checking Electronic Institutions. *ECAI Workshop on Model Checking and Artificial Intelligence (MoChart-2002)*, Lyon, 2002.
22. R. H. Bordini, M. Fisher, C. Pardavila, and M. Wooldridge. Model Checking AgentSpeak. *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2003)*, Melbourne, Australia, July 2003.
23. C. Walton. Model Checking Multi-Agent Web Services. *Proceedings of the 2004 Spring Symposium on Semantic Web Services,* Stanford, California, USA, March 2004.
24. R. J. Patel and H. Xu. A Trustworthy Agent Based Online Auction System. *Technical Report*, Computer and Information Science Department, University of Massachusetts Dartmouth, May 2006.
25. E. Clarke and J. Wing. Formal Methods: State of the Art and Future Directions. *ACM Computing Surveys*, Vol. 28, No. 4, December 1996, pp. 626-643.
26. A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, et al. NuSMV2: An OpenSource Tool for Symbolic Model Checking. *Proceeding of International Conference on Computer-Aided Verification* (CAV 2002), Copenhagen, Denmark, July 27-31, 2002.
27. K. Havelund. Java PathFinder: A Translator from Java to PROMELA. *Proceedings of the 5th and 6th International SPIN Workshops on Theoretical and Practical Aspects of SPIN Model Checking*, Springer-Verlag, 1999, pp. 152.

28. M. Makela. Maria: Modular Reachability Analyser for Algebraic System Nets. In J. Esparza and C. Lakos, editors, *Application and Theory of Petri Nets 2002, 23rd International Conference, ICATPN 2002*, Vol. 2360, *LNCS*, June 2002, pp. 434-444.
29. G. J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addison Wesley, 2003.
30. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, 1992.
31. E. M. Clarke, O. Grumberg, and K. Hamaguchi. Another Look at LTL Model Checking. *Formal Methods in System Design*, Vol. 10, Issue 1, February 1997, pp. 47-71.
32. Y. Cheng. A Formal Approach to Detecting Shilling Behaviors in Concurrent Online Auctions. *Master's Thesis*, Computer and Information Science Department, University of Massachusetts Dartmouth, September 2005.