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
Email: {hxu, g_ycheng}@umassd.edu

Abstract

Online auctions have become a quite popular and effective approach in the Internet-based e-Marketplace. In concurrent auctions, where multiple auctions for identical items are running simultaneously, users' bidding behaviors become very complicated. This situation motivates shilling behaviors, in which a seller disguises himself as normal bidders in order to drive up the bidding price and make the winning bidder pay more for an auctioned item. The goal of this paper is to propose a formal approach to verifying bidding behaviors, and especially, detecting shilling behaviors in concurrent online auctions. We develop a model template for concurrent auctions and derive auction models based on auction data from two concurrent auctions. The auction model can be formally verified using the SPIN model checker for certain behavioral properties, which are specified in pattern-based LTL (Linear Temporal Logic) formulas. To illustrate the feasibility and effectiveness of our approach, we provide a case study to show how possible shill bidders can be detected.

Keywords: Concurrent auctions, Bidding behaviors, Competitive shilling, 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. So, 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, the 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 can be very complicated in concurrent auctions, and as a consequence, a shill bidder may easily hide himself as a normal 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 the case of the reserve price shilling, a seller sets a low reserve price and pretends to be normal bidders 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 the case of the competitive shilling, a seller also pretends to be normal bidders, 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 very 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. Notice that normally the reserve price shilling only affects the auction houses; while the competitive shilling affects all the normal bidders in the auction market. It is obvious that the competitive shilling causes a greater harm to the auction market than the reserve price shilling. In addition, because

shilling behaviors involved in concurrent online auctions are much more difficult to detect than shilling behaviors occurring in a standalone auction, in this paper, we focus on studying competitive shilling behaviors in concurrent online auctions.

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 [5, 6, 7]. Since malicious users can easily set up multiple accounts in online environments to disguise themselves, 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 [8].

Additional previous work on prevention or detection of shilling behaviors can be summarized as follows. Wang and Whinston showed that private value English auctions with shill bidding can result in a higher expected seller profit than other auction formats [4]. This explains why in online auction houses like the eBay, shilling behaviors have become a very 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 [9]. 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 cannot 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, especially in concurrent auctions.

Kauffman and Wood used a statistical approach to detecting shilling behaviors and showed that how the statistic data of a market would look like if opportunistic behaviors do exist [10]. 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 [11]. Furthermore, since the statistical approach was based on analyzing large amount of historical auction data, it could not directly work for any particular auctions where shilling behaviors were involved. Therefore, this approach is not suitable for detecting new shill bidders or shill bidders who put in fake bids occasionally.

In this paper, we propose to use model checking techniques [12, 13] 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 [14]. 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 [15]. The formal auction model, which is derived from an auction model template presented in [16], can be verified using the SPIN model checker [13] for certain behavioral properties, which are specified in pattern-based LTL (Linear Temporal Logic) formulas [17, 18]. Since our approach is based on auction data from any particular concurrent auctions, it can be used to efficiently detect shilling behaviors and suggest any shill suspects.

The rest of this paper is organized as follows: Section 2 introduces the pattern-based model checking technique. Section 3 first presents a motivation 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 4 describes a pattern-based model checking tool that facilitates the tasks of preprocessing the auction data, composing LTL formulas, and invoking the SPIN model checker to verify the specified property. Section 5 provides a case study for how to use our approach to detect shilling behaviors. Finally, in Section 6, we provide conclusions and our future work.

2 Pattern-Based Modeling Checking Technique

2.1 The SPIN Model Checker

There is a wide variety of model checking tools available, such as the SPIN [13], the NuSMV2 [19], Java Pathfinder [20] and the MARIA [21]. 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) [22, 13]. 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 [22]. 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 [22, 13].

There are two basic ways to use SPIN in system verification [22]. 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 automatically generate a formal auction model. If the generated formal auction model can be shown to have certain bidding behavioral properties, the concurrent auctions must also have such properties.

2.2 LTL and Composition Patterns

The SPIN model checker supports specification of system properties using Linear Temporal Logic (LTL), which is a formal method to specify temporal relationships of statements [23, 24]. LTL has been proven to have good expressivity and more natural language like statements for verification. LTL consists of only a few logic operators, such as *G* (always), *F* (eventually), *U* (until), *W* (unless, or weak until) and *O* (next). Combining with Boolean operators, i.e., `&&` (and), `||` (or), `!` (negation), $\rightarrow$ (logical implication) and $\leftrightarrow$ (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 [17].

There are quite a few patterns proposed in previous work [17, 18]. 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.

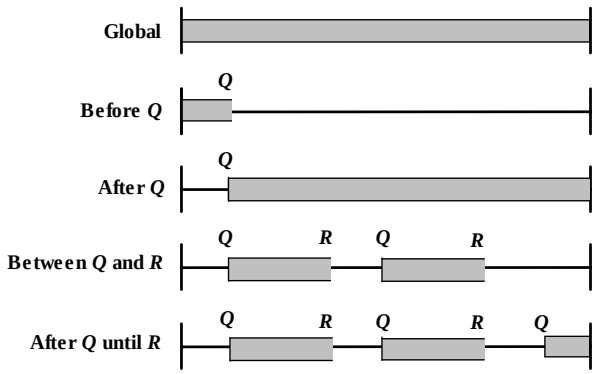


Figure 1: Pattern scopes for pattern-based LTL

Figure 1 is an illustration of pattern scopes adapted from [18]. 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 are 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 [18].

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)))`

3 Modeling Internet Concurrent Auctions

3.1 A Motivation 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 3.1 *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*. Two 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 both symmetric and transitive, i.e., (1) *Auction 0* || *Auction 1* implies *Auction 1* || *Auction 0*; and (2) *Auction 0* || *Auction 1* and *Auction 1* || *Auction 2* imply *Auction 0* || *Auction 2*.

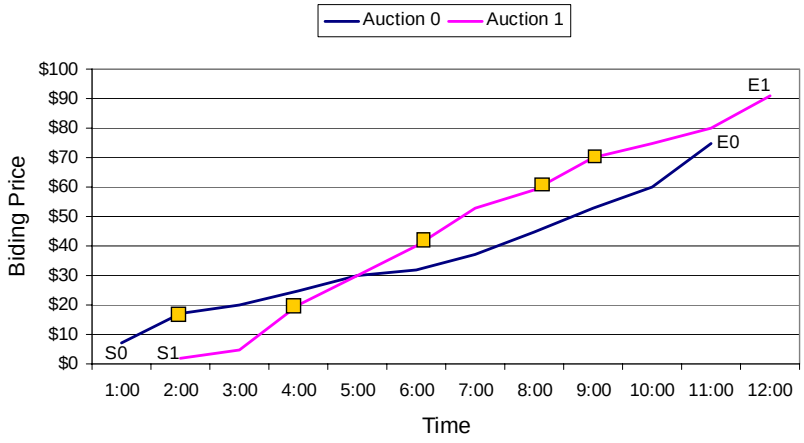


Figure 2: The bidding activities of user *A* in two concurrent auctions

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, we assume that the auction that starts first is always *Auction 0*, and the one that starts later is always *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 that the event happens at that time. For example, according to Figure 2, at time 4:00, the bidding price is lower in *Auction 0*. Thus, at that time, the predicate "Price is lower" is *true* for *Auction 0*, but it is *false* for *Auction 1*. Similarly, at the same time, since user *A* puts in his/her 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 are running concurrently, a shill bidder may bid in the auction with higher bidding price rather than the one with lower bidding price.

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 for four times (at time 2:00, 6:00, 8:00 and 9:00). Thus, the LTL formula for this behavioral property must be *valid*.

3.2 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.

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:* interleave the bidding activities of the two auctions according to the bidding time.

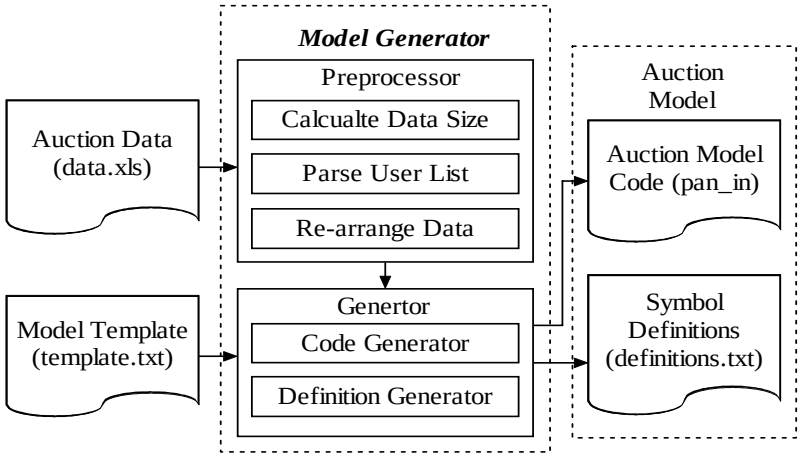


Figure 3: The generation of an auction model

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 auction data after preprocessed to produce a specific auction model that consists of the auction model code in PROMELA and an LTL symbol definition file. The auction model code and the LTL symbol definition file will be passed to the SPIN model checker for property verifications.

3.3 The Auction Model Template

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

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 formulas. The symbols for composition of LTL formulas are defined in the symbol definition file, which is described in Section 3.4. 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, when the bidding case is "0", it means that a bidder placed a bid in *Auction 0*; while 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 "1", which means a bidder placed a bid in *Auction 1*; while 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.

In Table 5, we list 8 different biding cases for two concurrent online auctions. Since each auction can be in a state of "bidding", "not biding" and "end", we have 9 combinations. However, the case "not bidding, not bidding" can be excluded because when this happens, no actions need to be taken in the auction model.

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]==0)-> // bidding case 0
        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 1-7 */
      ...
      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
}
```

Table 5: List of eight bidding cases

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

3.4 Symbol Definitions for LTL

In order to verify properties specified in LTL formulas, 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 3.1 *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 (e.g., over 10 dollars). An overbid in an auction is considered as a bid in large increment from the previous bid.

Definition 3.2 *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 (e.g., over 7200 seconds or 2 hours). A deliberate bid implies a deliberate decision made by a bidder.

Table 6 lists a few symbol definitions that are used in the case study 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 `bid00`, 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 `bid016`.

Table 6: Symbol definitions for LTL formulas

```
1     // the bidding activities of the users
2     #define bid00  (increment0[0] > 0)
3     #define bid01  (increment0[1] > 0)
4     ...
5     // users' bidding in large increments
6     #define overBid00 (increment0[0] > ...)
7     #define overBid01 (increment0[1] > ...)
8     ...
9     // definition of deliberate bids
10    #define deliBid0  (timeElapse0 > ...)
11    #define deliBid1  (timeElapse1 > ...)
12    // start of auctions
13    #define start0  (startPoint0==1)
14    #define start1  (startPoint1==1)
15    // the events that the reserve price has been reached
16    #define reserve0 (reservePoint0==1)
17    #define reserve1 (reservePoint1==1)
18    // end of auctions
19    #define end0 (endPoint0==1)
20    #define end1 (endPoint1==1)
21    // the condition that the bidding price in one auction
22    // is lower than that in another one
23    #define p0Lower
24       ((currentHighestBid0-previousHighestBid1) < 0)
25    #define p1Lower
26       ((currentHighestBid1-previousHighestBid0) < 0)
```

Symbol definitions can be used to ease the task of writing LTL formulas. For example, the predicate "User 5 bids in *Auction 0* && Price is lower in *Auction 1*" can be written as `(bid05 && p1Lower)`. In practical use, we can also develop a different set of symbol definitions for our convenience.

3.5 The Model Checking Process

After the auction model has been created and the LTL formulas 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 [22]. 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*.

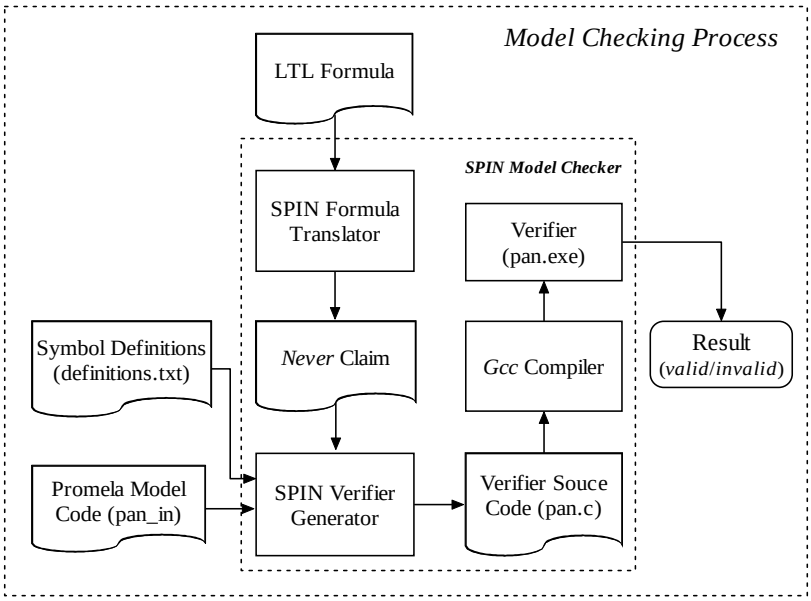


Figure 4: Model checking process

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 *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.

4 Tool Support for Model Checking Bidding Behaviors

To facilitate the process of model checking bidding behaviors in concurrent online auctions, we developed a 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. Our pattern-based model checking tool consists of the following features:

- *Auction Model Generation*: to preprocess the auction data and generate the auction model.
- *Formula Composition*: to design pattern-based LTL formulas for concurrent auctions.
- *Formula Generation*: to automatically generate the LTL formulas based on the formula design.
- *Result Display*: to invoke the SPIN model checker and display the model checking result.

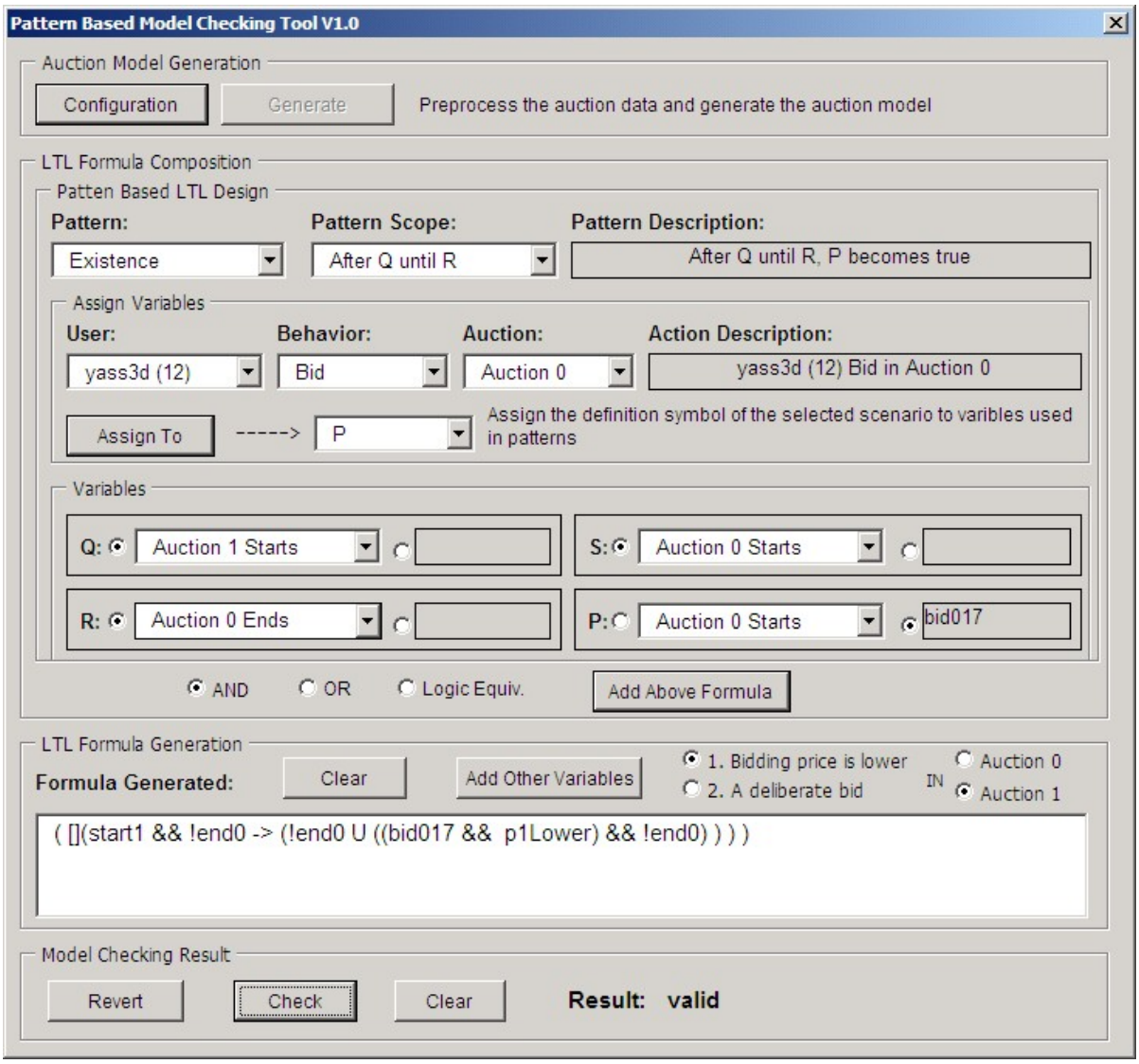


Figure 5: The user interface of the pattern-based model checking tool

Figure 5 is a snapshot of the user interface of the pattern-based model checking tool. The first step to use our model checking tool is to preprocess the auction data and generate the auction model. The tool can automatically read auction data from an MS Excel file. Before preprocess the auction data, we need to click on the “Configuration” button to set up a few parameters. As shown in Figure 6, we need to setup the starting bidding price and the seller’s reserve price for each of the auctions. We also need to setup the boundary values for overbids and deliberate bids. All these values will be substituted into the symbol definition file for generation of the auction model.

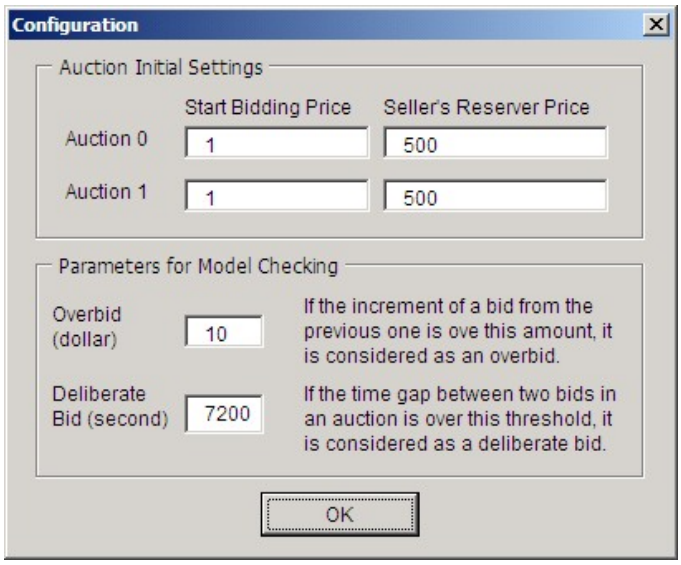


Figure 6: Configuration for the auction model

After the configuration is done, we can press the "Generate" button to invoke the auction model generator to preprocess the auction data and generate the auction model, which consists of the auction model code and the symbol definitions in PROMELA. The next step for model checking bidding behaviors is to compose LTL formulas that specify user's behaviors in the concurrent auctions. To compose an LTL formula, we need to first select an LTL pattern as well as the pattern scope, and set up the required Boolean variables. For example, in Figure 5, we select the "Existence" pattern with pattern scope "After *Q* until *R*," and we assign "*Auction 1* starts" to *Q* and "*Auction 0* ends" to *R*. We further choose to verify the bidding behavior of the user "yass3d" in *Auction 0*, so we assign "yass3d Bid in *Auction 0*" to *P*, which is denoted as `bid017`. The model checking tool also supports insertion of additional Boolean variables, such as `p0Lower`, `p1Lower`, `deliBid0` and `deliBid1`, by clicking on the button "Add Other Variables."

The LTL formula can be automatically generated in the "Formula Generated" textbox by clicking on the "Add Above Formula" button. When the LTL formula is ready to be verified, click on the "Check" button, the SPIN model checker will be invoked and the result will be displayed.

5 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 detect shilling behaviors 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 the eBay with the title of auctioned items as "HP/COMPAQ PRESARIO LAPTOP CD-RW BURNER DVD WIRELESS." Both auctions are held by the same seller and the detailed descriptions of the auctioned items are shown in Table 7.

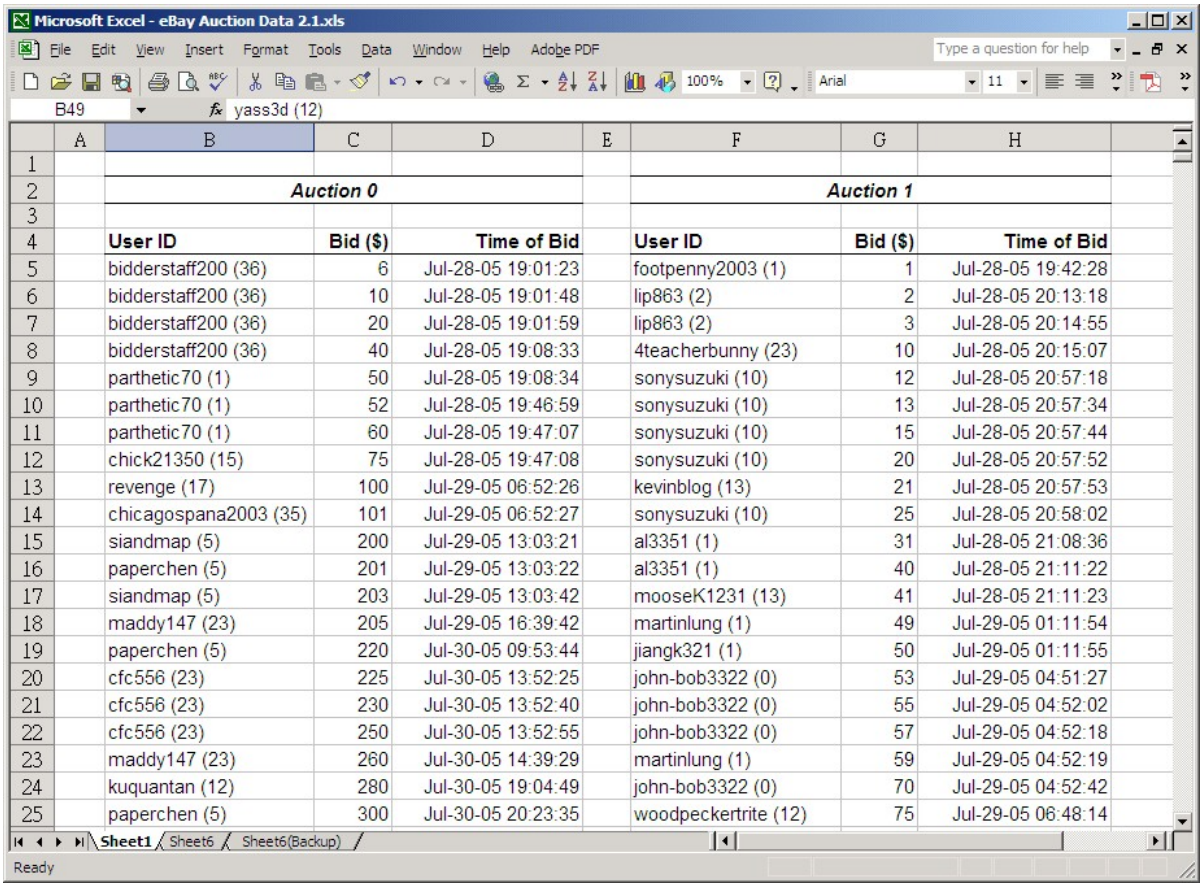
Table 7: Descriptions of the auctioned items

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 raw auction data for the two concurrent auctions is listed in Figure 7. To protect the privacy of the users, we changed all user IDs. In addition, we made the following adjustments on the 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 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.

Since the reserve price for each auction is not shown in the eBay, we assume the reserve prices for both auctions are $500. The value of $500 is very close to both winning bids, which are $630 and $620, but it still gives us good price ranges (from $500 to $630 and from $500 to $620) for our bidding behavior verification purpose.

In the following experiments, we set the price difference of 10 dollars for overbid and the time gap of 7200 seconds or 2 hours for deliberate bid. The reason we set 10 dollars as the boundary is based on our observation that most bid increments are less than this number. Similarly, we set the 2 hours boundary for deliberate bid because 2 hours is a reasonable period of time for a bidder to make a deliberate decision. In practical, these two values should be defined and adjusted according to auction administrator's experiences and observations.



Microsoft Excel - eBay Auction Data 2.1.xls

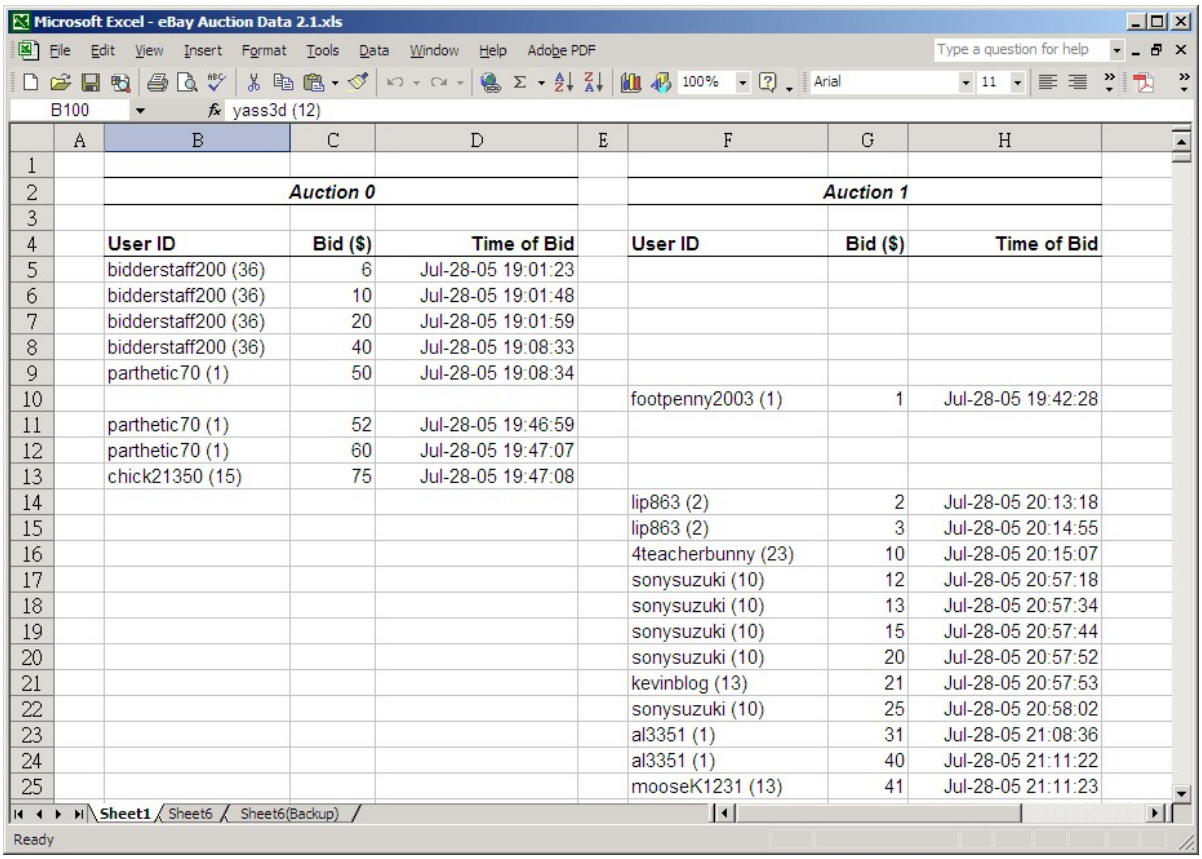
B49 yass3d (12)

	Auction 0			Auction 1		
	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 7: Auction data from two concurrent auctions before preprocessing

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.

After the auction data is preprocessed using our model checking tool, the auction data is re-arranged as shown in Figure 8. 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 9, 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 $500 as the reserve price for both auctions, the small filled circles (denoted as R0 and R1 in Figure 9) represent the events of “*Auction 0* reaches the reserve price” and “*Auction 1* reaches the reserve price.”



Microsoft Excel - eBay Auction Data 2.1.xls

	Auction 0			Auction 1		
	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 8: Auction data from two concurrent auctions after preprocessing

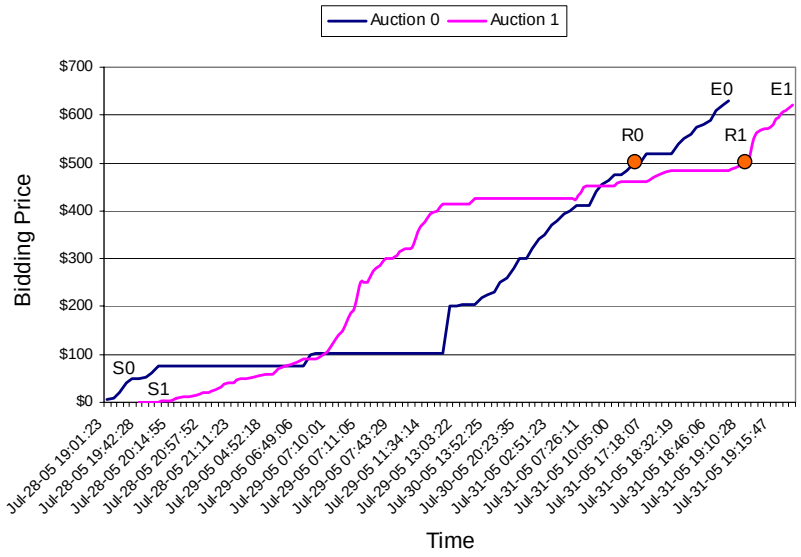


Figure 9: Overlapping style of the two concurrent auctions

We now use our model checking tool to verify the following behavioral properties that are related to shilling behaviors in concurrent online auctions.

Property 1: User bids in an auction only after the reserve price in that auction is reached, i.e., the user only bids in *Auction 0* after R0 or only bids in *Auction 1* after R1.

Pattern Used: Precedence, globally.

Formulas:

We design the following LTL formulas for each of the auctions such that the event of reaching the reserve price in an auction precedes any bidding activities (denoted as *event*) of a user in that auction.

For *Auction 0*: `(!event W reserve0)` i.e., `(!event U reserve0)||([]!event)`

For *Auction 1*: `(!event W reserve1)` i.e., `(!event U reserve1)||([]!event)`

Events:

For *Auction 0*: `event` should be replaced by `bid06`, `bid014`, `bid016` or `bid017`.

For *Auction 1*: `event` should be replaced by `bid16`, `bid114`, `bid116` or `bid117`.

Notes:

1. User "paperchen", "benniten23", "andy293" and "yass3d" are numbered by the auction model generator as "6", "14", "16" and "17", respectively. Therefore, the `event` in the formulas should be substituted by the corresponding symbols defined in the symbol definition file, such as `bid06` ("paperchen" bids in *Auction 0*), and `bid117` ("yass3d" bids in *Auction 1*).
2. Since the SPIN does not 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 is done according to the *valid* formula of "`p W q <=> (p U q || []p)`."

Now we use our model checking tool to verify the above LTL formula for each of the four users. The model checking results for *Property 1* are shown in Table 8.

Table 8: Model checking results for *Property 1*

Auction / User	Auction 0	Auction 1
paperchen (5)	*invalid*	*invalid*
benniten23 (1)	*invalid*	*invalid*
andy293 (12)	*valid*	*valid*
yass3d (12)	*valid*	*invalid*

Explanation: A user bids only after the reserve price is reached in an auction is most likely a *last-minute* bidder, who only cares about if she or he can win the auction. Thus, a user with this kind of behavior is *not* likely a shill bidder. The result shows that "andy293" bids only after reserve price is reached in both auctions; therefore, he is the least suspicious shill bidder. User "yass3d" is also not likely a suspicious shill bidder for the same reason. For "paperchen" and "benniten23", since both of the results are *invalid*, no conclusions can be drawn in terms of shilling behaviors. The results for "paperchen" and "benniten23" only tell us that both of the users may start to participate in both of the auctions from the beginning.

Property 2: User bids in *Auction 1* only after *Auction 0* ends.

Pattern Used: (1) Existence, after *Q*; (2) Precedence, globally.

Formulas:

We design two formulas for this property. The first one is to test if a user bids in *Auction 1* after *Auction 0* ends, i.e., the event of the user's bidding activity in *Auction 1* exists after *Auction 0* ends (denoted by `end0`). The second one is to test if the user bids in *Auction 1* only after *Auction 0* ends, i.e., the event of `end0` precedes *any* event of the user's bidding activity in *Auction 1*.

Formula 1: `([](!end0) || <>(end0 && <>event))`

Formula 2: `(!event W end0)` i.e., `(!event U end0)||([]!event)`

Events: `event` should be replaced by `bid16`, `bid114`, `bid116` or `bid117`

The model checking results for *Property 2* are shown in Table 9.

Table 9: Model checking results for *Property 2*

Formula / User	Formula 1	Formula 2
paperchen (5)	*invalid*	*invalid*
benniten23 (1)	*invalid*	*invalid*
andy293 (12)	*valid*	*valid*
yass3d (12)	*valid*	*invalid*

Explanation: The results show that the user "paperchen" and "benniten" do not bid in *Auction 1* after *Auction 0* ends. Thus, it is expected that the *Formula 2* for both of these two users evaluates to *invalid*. The results also show that both the user "andy293" and "yass3d" bid in *Auction 1* after *Auction 0* ends; however, only the user "andy293" bids in *Auction 1* only after *Auction 0* ends. This means that user "andy293" starts to bid in *Auction 0*; but more likely, he failed to win the auction. So

he starts to bid in *Auction 1* thereafter. Combined with the model checking results for *Property 1*, we can conclude that "andy293" only bids when an auction is close to the end, and he focuses on bidding one auction at a time. The bidding behavior of "andy293" is very normal, though he may have to bid on an item with higher price using his bidding strategy.

Property 3: User places a deliberate overbid before R0 in *Auction 0* or before R1 in *Auction 1*, but doesn't bid at all after R0 or R1.

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

Formulas:

For *Auction 0*: `((!reserve0 W ((event0) && !reserve0))) && ([](reserve0 -> [](!event1)))` i.e., `(((!reserve0 U ((event0) && !reserve0)) || ([]!reserve0))) && ([](reserve0 -> [](!event1)))`

For *Auction 1*: `((!reserve1 W ((event0) && !reserve1))) && ([](reserve1 -> [](!event1)))` i.e., `(((!reserve1 U ((event0) && !reserve1)) || ([]!reserve1))) && ([](reserve1 -> [](!event1)))`

Events:

For *Auction 0*: `event0` should be replaced by `(overBid06 && deliBid0)`, `(overBid014 && deliBid0)`, `(overBid016 && deliBid0)` or `(overBid017 && deliBid0)`; and `event1` should be replaced by `bid06`, `bid014`, `bid016` or `bid017`.

For *Auction 1*: `event0` should be replaced by `(overBid16 && deliBid1)`, `(overBid114 && deliBid1)`, `(overBid116 && deliBid1)` or `(overBid117 && deliBid1)`; and `event1` should be replaced by `bid16`, `bid114`, `bid116` or `bid117`.

Notice that `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 `(overBid06 && deliBid0)` specifies that user "paperchen" places a deliberate overbid in *Auction 0*. The model checking results for *Property 3* are shown in Table 10.

Table 10: Model checking results for *Property 3*

Auction / User	Auction 0	Auction 1
paperchen (5)	*valid*	*valid*
benniten23 (1)	*invalid*	*invalid*
andy293 (12)	*invalid*	*invalid*
yass3d (12)	*invalid*	*invalid*

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. From Table 10, we found that only "paperchen" had such behavior, so "paperchen" is very likely a shill.

Property 4: Between S1 and E0, user bids in auctions that have higher bidding prices.
Pattern Used: Existence, Between *Q* and *R*.
Formulas:
For *Auction 0* and *Auction 1*: `([](start1 && !end0 -> (!end0 W ((event) && !end0))))`
i.e., `([](start1 && !end0 -> ((!end0 U ((event) && !end0))||([]!end0))))`
Events:
For *Auction 0*: `event` should be replaced by `(bid06 && p1Lower)`, `(bid014 && p1Lower)`, `(bid016 && p1Lower)` or `(bid017 && p1Lower)`.
For *Auction 1*: `event` should be replaced by `(bid16 && p0Lower)`, `(bid114 && p0Lower)`, `(bid116 && p0Lower)` or `(bid117 && p0Lower)`.

Notice that `p1Lower` represents that the bidding price in *Auction 1* is lower than that in *Auction 0*. So the formula `(bid06 && p1Lower)` specifies that "paperchen" bids in *Auction 0* but *Auction 1* has lower bidding price. Also notice that, according to Figure 1 and 9, the pattern scopes of "Between S1 and E0" and "After S1 until E0" are the same for this example, so we should get the same results using either of the scopes. The model checking results for *Property 4* are shown in Table 11.

Table 11: Model checking results for *Property 4*

Auction / User	Auction 0	Auction 1
paperchen (5)	*invalid*	*valid*
benniten23 (1)	*valid*	*valid*
andy293 (12)	*valid*	*invalid*
yass3d (12)	*valid*	*invalid*

Explanation: From Figure 9, we can see that the time period [S1, E0] covers the duration while the two auctions overlap. During the overlapping time, any user who does not bid in the auction that has cheaper bidding price could be suspicious. Thus, from Table 11, we can see that user "benniten23" might *not* be a normal bidder because the model checking results for "benniten23" are *valid* in both of the auctions. For the other three users, they all show this behavior in one auction, but do not have

this behavior in the other one. To draw more conclusions, we need to narrow down the pattern scope to see if the model checking results become more meaningful. Since a shill bidder would try to avoid bidding after the reserve price is reached to avoid winning the auction, we would be more concerned about such behavior shown up during the time period of [S1, R0] (refer to Figure 9). This case is described in *Property 5*.

Property 5: Between S1 and R0, user bids in auctions that have higher bidding prices.

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

Formulas:

For *Auction 0* and *Auction 1*: `([](start1 && !reserve0 -> (!reserve0 W ((event) && !reserve0))))` i.e., `([](start1 && !reserve0 -> ((!reserve0 U ((event) && !reserve0))||([]!reserve0))))`

Events:

For *Auction 0*: `event` should be replaced by `(bid06 && p1Lower)`, `(bid014 && p1Lower)`, `(bid016 && p1Lower)` or `(bid017 && p1Lower)`.

For *Auction 1*: `event` should be replaced by `(bid16 && p0Lower)`, `(bid114 && p0Lower)`, `(bid116 && p0Lower)` or `(bid117 && p0Lower)`.

Now after narrowing down the pattern scope from [S1, E0] to [S1, R0], the new model checking results for *Property 5* are shown in Table 12.

Table 12: Model checking results for *Property 5*

Auction / User	Auction 0	Auction 1
paperchen (5)	*invalid*	*valid*
benniten23 (1)	*valid*	*valid*
andy293 (12)	*invalid*	*invalid*
yass3d (12)	*invalid*	*invalid*

Explanation: The results show that both the user “andy293” and “yass3d” do not bid in auctions that have higher bidding prices between S1 and R0. Therefore, they are *not* likely shill bidders. However, both the user “paperchen” and “benniten23” have this suspicious behavior in either one or both of the auctions. Thus, they are likely shill bidders.

Based on the above analysis, we summarize our conclusions as follows:

1. User “paperchen” is most likely a shill bidder due to the following two major reasons: (1) user “paperchen” attempted to drive up the bidding price and stopped bidding after the price reached the reserve price; (2) when a certain time has passed after the last bid was placed, he would try to create a competitive bidding atmosphere by placing overbids.
2. User “benniten23” is also likely a shill bidder because he put in bids on an item that has higher bidding price. Such behavior of the user occurred in both concurrent auctions, especially before the reserve prices have been reached. Thus, the purpose of such activities is very likely to drive up the bidding price.
3. Both user “andy293” and “yass3d” 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. In addition, user “andy293” likes to bid in one auction at a time.

The model checking results provide us evidences of shilling behaviors. However, the above analysis is not sufficient to guarantee that both “paperchen” and “benniten23” must be shills. To collect more evidences of their shilling behaviors, we can use the same model checking technique to verify additional properties that combine with other evidences such as user’s IP address, ratings and trading histories. It can be expected that, with more expert knowledge on shilling, our approach can be very effective in practical use for detection of shilling behaviors.

6 Conclusions and Future Work

In this paper, we introduced a model checking approach to verifying bidding behaviors including shilling 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. With our pattern-based model checking tool, we can not only easily write LTL formulas that specify certain bidder behaviors, but also directly detect suspicious shills in concurrent online auctions. The case study, which is based on auction data from existing auction house – the eBay, shows that our approach is feasible and effective. Our approach can be easily extended to support model checking bidding behaviors for more than two concurrent online auctions. For our future work, we will study in more depth on shilling behaviors in online auctions and provide tool supports for detection, prevention and prediction of shill bidders.

References

1. M. Harkavy, J. D. Tygar, and H. Kikuchi, "Electronic Auctions with Private Bids," In *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," In *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," In *Proceedings of the 35th Hawaii International Conference on System Sciences* (HICSS'02), Hawaii, 2002.
5. Chris Dellarocas and Paul Resnick, "Online Reputation Mechanisms: A Roadmap for Future Research," *Summary Report of the First Interdisciplinary Symposium on Online Reputation Mechanisms*, April 26-27, 2003.
6. C. A. Wood, M. Fan and Y. Tan, "An Examination of the Reputation Systems for Online Auctions", In *Proceedings of the Workshop for Information Systems and Economics* (WISE 2002), Barcelona, Spain, Dec. 2002.
7. Paul Resnick, Richard Zeckhauser, Eric Friedman, and Ko Kuwabara, "Reputation Systems," *Communications of the ACM*, Vol. 43, No. 12, December 2000, pp. 45-48.
8. J. H. Dobrzynski, "In Online Auctions, Rings of Bidders", *The New York Times*, June 2, 2000.
9. Indranil Chakraborty and Georgia Kosmopoulou, "Auctions with Shill Bidding," *Economic Theory*, Springer, Vol. 24, Issue 2, 2004, pp. 271-287.
10. R. J. Kauffman and C. A. Wood, "Running up the Bid: Modeling Seller Opportunism in Internet Auctions," In *Proceedings of the Sixth Americas Conference on Information Systems* (AMCIS 2000), M. Chung (ed.), Long Beach, CA, 2000, pp. 929-935.
11. Ravi Bapna and Alok Gupta, "Online Auctions: A Closer Look," In *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.
12. Edmund M. Clarke, Orna Grumberg, Doron A. Peled, *Model Checking*, The MIT Press, 2000.
13. G. J. Holzmann, "The Model Checker SPIN," *IEEE Transactions on Software Engineering*, Vol. 23, No. 5, 1997, pp. 279-295.

14. 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.
15. Haiping Xu and Sol 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.
16. Yi-Tsung Cheng and Haiping Xu, “A Formal Approach to Detecting Shilling Behaviors in Concurrent Online Auctions,” In *Proceedings of the 8th International Conference on Enterprise Information Systems (ICEIS 2006)*, May 2006, Paphos, Cyprus.
17. M. B. Dwyer, G. S. Avrunin, and J. C. Corbett, “Property Specification Patterns for Finite-state Verification,” In *Proceedings of the 2nd Workshop on Formal Methods in Software Practice*, March 1998.
18. M. B. Dwyer, G. S. Avrunin, and J. C. Corbett, “Patterns in Property Specifications for Finite-State Verification,” In *Proceedings of the 21st International Conference on Software Engineering*, Los Angeles, 1999, pp. 16-22.
19. A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani and A. Tacchella, “NuSMV2: An OpenSource Tool for Symbolic Model Checking,” In *Proceeding of International Conference on Computer-Aided Verification* (CAV 2002), Copenhagen, Denmark, July 27-31, 2002.
20. K. Havelund, “Java PathFinder: A Translator from Java to PROMELA,” In *Proceedings of the 5th and 6th International SPIN Workshops on Theoretical and Practical Aspects of SPIN Model Checking*, Springer-Verlag, 1999, pp. 152.
21. Marko Makela, “Maria: Modular Reachability Analyser for Algebraic System Nets,” In Javier Esparza and Charles Lakos, editors, *Application and Theory of Petri Nets 2002, 23rd International Conference, ICATPN 2002*, Vol. 2360, *LNCS*, June 2002, pp. 434-444.
22. G. J. Holzmann, *The SPIN Model Checker: Primer and Reference Manual*, Addison Wesley, 2003.
23. Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*, Springer-Verlag, 1992.
24. Edmund M. Clarke, Orna Grumberg and Kiyoharu Hamaguchi, “Another Look at LTL Model Checking,” *Formal Methods in System Design*, Vol. 10, Issue 1, February 1997, pp. 47-71.