Analysis of Evidence Using Formal Event Reconstruction

Joshua James[1,*], Pavel Gladyshev[1], Mohd Taufik Abdullah[2], and Yuandong Zhu[1]

[1] Centre for Cybercrime Investigation University College Dublin, Belfield, Dublin 4, Ireland
[2] Department of Computer Science Faculty of Computer Science and Information Technology, Putra University of Malaysia, 43400 Serdang, Selangor, Malaysia
`Joshua.James@ucd.ie, Pavel.Gladyshev@ucd.ie, mtaufik@fsktm.upm.edu.my, Yuandong.Zhu@ucd.ie`

Abstract. This paper expands upon the finite state machine approach for the formal analysis of digital evidence. The proposed method may be used to support the feasibility of a given statement by testing it against a relevant system model. To achieve this, a novel method for modeling the system and evidential statements is given. The method is then examined in a case study example.

Keywords: Digital Forensics, Event, Reconstruction, State Machine, Automata, Evidence Modeling.

1 Introduction

A sound forensic analysis is expected to rely on credible scientific theory that explains why and how expert conclusions follow from the available evidence [6]. While advancements have been made in the formalization of the digital investigation process, analysis still remains largely ad-hoc. This paper adds to a body of work that attempts to formalize the digital investigation process, and specifically works to extend the finite state machine (FSM) theory [3][4][6].

As stated in [4], "many digital systems, such as digital circuits, computer programs, and communication protocols can be described mathematically as finite state machines. A finite state machine can be viewed as a graph whose nodes represent possible system states, and whose arrows represent possible transitions from state to state". Likewise, Carrier claims, "modern computers are FSMs with a large number of states and complex transition functions"[2]. By utilizing this fact, finite state machine models that represent the computations of a system may be formally defined. It is then possible to test scenarios in terms of the model to see if given situations are computationally possible.

* Research funded by the Science Foundation Ireland (SFI) under Research Frontiers Programme 2007 grant CMSF575 and conducted using equipment funded by the Higher Education Authority of Ireland under the Research Equipment Renewal Grant Scheme.

1.1 Formal Methods of Investigation

A number of methods for the formalization of the digital investigation process have previously been proposed, such as Semantic Integrity Checking [9], Multiple Log Analysis [1], Temporal Logic of Security Actions [8], Computer History Model approach [3], Causal Relation of Events [11], and previous Finite State Machine modeling approaches [4].

Semantic Integrity Checking [9] involves the analysis of redundant data objects that must exist in a system. Using found inconsistencies in the redundant data it is possible to hypothesize attack scenarios.

[1] propose a method of model checking where sets of logs are formally modeled. Events within the model are expressed as a "term of algebra" with properties being expressed as formulas. Suspected events may then be expressed as a pattern in the logic formula.

Temporal Logic of Security Actions (S-TLA) is a Logic-Based Language for Digital Investigations [8], which represents the system using state-based logic. By combining pre-defined generic scenario fragments with the created system model, undesirable states may be found from which evidence may be derived. From the combination of recurring scenario fragments and found evidence, formulation of possible event scenarios can occur.

Carrier [3] proposed a computer history model that groups the system into primitive (lowest level) and complex (causing multiple primitive or complex events) events. An investigator may then follow the scientific method to formulate hypotheses of events. These hypotheses may be tested against the created computer history model in order to support or refute the hypothesis.

Willassen also uses a hypothesis-based approach to construct models of possible adjustments to clocks, called 'Clock Hypothesis' [11]. This method uses the notion of a causal connection between events, and uses observations of timestamps to formulate hypotheses that may be tested against these connections for consistency.

The Finite State Machine approach proposed by [6] models the system as an FSM whose transitions may be back-traced from the state the system was found. Witness observations are used to restrict the possible transitions of the system. This restricted back-tracing tests each possibility in the model to find possible transitions leading back from the current state, to the suspect event.

1.2 Contribution

One of the limitations of the finite state machine approach is the exponential growth in the number of recovered incident scenarios as backtracing advances into the past. This paper addresses this limitation by proposing a novel approach to formally defining the system model. An algorithm is proposed to convert the FSM model of the system into a deterministic finite automaton (DFA) that encodes the set of system computations as a set of strings. Witness statements are then formally defined as restrictions on strings accepted by the DFA.

1.3 Organization

The remainder of this paper is comprised of four sections. In the first section an informal overview of the method is given. The second section explains how to derive a DFA representative of the computations of the system. The third section shows how automata intersection using both the system and witness statement models can be used to test validity of the given witness statement. This process is illustrated in a given case study. Finally, considerations of the strengths and weaknesses of the proposed method will be given.

2 Representing the System

Suppose that the system under investigation is modeled as a finite state machine. An example of a very simple state machine is given in Figure 1. For a given Finite State Machine each transition can be encoded as a triple (state1, event, state2). For example, in Figure 1 there are two transitions: A-1->B and B-1->B.

They can be encoded as [A, 1, B] and [B, 1, B], respectively. A sequence of transitions may be defined as a computation. An example of which is the sequence 'A-1->B-1->B' that may be expressed as ([A, 1, B][B, 1, B]).

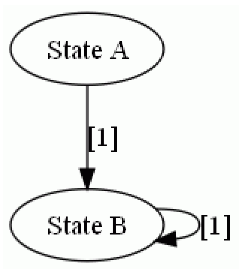

Fig. 1. Simple Finite State Machine

Assuming that a computation can start in any state, the entire set of possible computations of the FSM in Figure 1 is as follows:

A = { [A, 1, B]
[A, 1, B] [B, 1, B]
[A, 1, B] [B, 1, B] [B, 1, B]
[A, 1, B] [B, 1, B] [B, 1, B] [B, 1, B]
...
[B, 1, B]
[B, 1, B] [B, 1, B]
[B, 1, B] [B, 1, B] [B, 1, B]
... }

2.1 Witness Statements

Witness statements can be viewed as restrictions on possible computations of the FSM [6]. These restrictions can be expressed as regular expressions [10], or patterns over the sequences of transition triples. For example, an observation that the FSM from Figure 1 started in 'State A', can be written as a pattern:

[A, 1, B]* (†)

which corresponds to all possible sequences of triples that begin with [A, 1, B]:

```
B = {   [A, 1, B]
        [A, 1, B] [A, 1, B]
        [A, 1, B] [B, 1, B]
        [A, 1, B] [A, 1, B] [A, 1, B]
        [A, 1, B] [A, 1, B] [B, 1, B]
        [A, 1, B] [B, 1, B] [A, 1, B]
        [A, 1, B] [B, 1, B] [B, 1, B]
        ...                              }
```

Observe that not all of the above sequences correspond to valid computations of the FSM in Figure 1. The subset of computations of the FSM from Figure 1 that obey the restriction † can be obtained by intersecting sets **A** and **B**:

```
A ∩ B = { [A, 1, B]
          [A, 1, B] [B, 1, B]
          [A, 1, B] [B, 1, B] [B, 1, B]
          ...                          }
```

Although sets **A** and **B** are infinite, it is possible to construct finite automata that represent them. Automata intersection can then be used to construct the automaton that represents intersection of the two sets.

3 Constructing an Automata Representation of the System Model

In this section the system is represented as its corresponding Finite State Machine *M*. From *M* a Deterministic Finite Automaton (DFA) is derived that accepts the computations of *M* as a set of strings. Automata representations of witness statements are also briefly discussed.

3.1 Finite State Machine Model of the System

Following [6], a finite state machine is defined as a triple $M = (Q, \Sigma, \delta)$, where
Q is the finite set of all possible states
Σ is the finite set of all possible events
$\delta: Q \times \Sigma \to Q$ is a transition function that returns the next state.

3.2 Automaton Model of the System

Given M, the proposed algorithm produces a DFA M_I that represents the set of computations of M encoded as sequences of triples. Essentially this can be thought of as a DFA that accepts the computations that M performs. Formally M_I is defined as $M_I = (Q_I, \Sigma_I, \delta_I, g, F)$, where

- Q_I is the finite set of all possible states
- Σ_I is the newly created finite set of all possible events
- $\delta_I : Q_I \times \Sigma_I \to Q_I$ is a transition function that determines the next state of M_I
- g is the start state where $g \notin Q$
- F is the set of accepting states

Automaton M_I is said to accept a given string of triples s if s corresponds to a sequence of transitions in M_I starting in g and ending in one of the accepting states. Accepting states are shown in figures as double-lined ovals.

3.3 Algorithm for Constructing M_I

Given M:

1. Define Σ_I as a finite set of triples where each triple $\varphi = (q, e, q_2)$.
   $\Sigma_I = \{ \varphi \mid \varphi = (q, e, q_2)\}$, where
   - $q \in Q$
   - $e \in \Sigma$
   - $q_2 = \delta(q, e)$
2. Define Q_I as containing the set Q as well as a generic start-state g, where g is not in the set Q.
   Define: $Q_I = Q \cup \{g\}, g \notin Q$
3. Define a transition function δ_I where for each transition in δ there also exists a corresponding transition in δ_I from q to q_2 and from g to q_2.
   Define: $\delta_I : Q_I \times \Sigma_I \to Q_I$
   such that: $\forall e \in \Sigma, \forall q, q_2 \in Q$:

$$\begin{gathered} ((q, e), q_2) \in \delta \\ \Leftrightarrow \\ (q, (q, e, q_2), q_2) \in \delta_I \wedge \\ (g, (q, e, q_2), q_2) \in \delta_I \end{gathered}$$

4. Define the set of accepting states F_I as the set of states Q.
   Define: $F_I = Q$

Figure 2 shows the automaton M_I constructed from the finite state machine depicted in Figure 1. It accepts sequences of triples that correspond to valid sequences of transitions of the given finite state machine. Consider, for example, sequence [A,1,B][B,1,B]. It moves the automaton from its start state g into state B via transition g–[A,1,B]->B, and then from state B back into state B via B-[B,1,B]->B. The sequence is accepted because B is an accepting state.

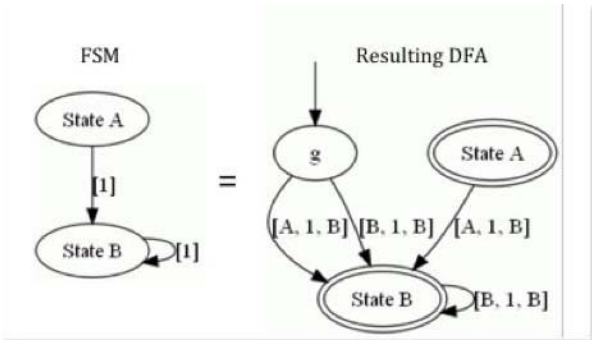

Fig. 2. DFA that represents the computations of the given FSM

The constructed automaton also rejects sequences of triples that do not correspond to valid sequences of transition of the finite state machine from Figure 1. Consider the sequence [A,1,B][A,1,B]. It moves the automaton from the start state *g* into state B via *g*-[A,1,B]->B, but then there is no transition leading out of state B via [A,1,B] at which point the sequence [A,1,B][A,1,B] is rejected.

3.4 Witness Statements

Witness statements may be represented as patterns over the sequences of transition triples. Patterns can be converted into automata using standard algorithms (see [7]). Given below are some examples of automata built for simple statements. These examples use some simplifying shorthand notation, which keeps automata graphs clear from multiple parallel arcs. In particular, '*?*' is used in transition labels to stand for an arbitrary value. For example, instead of drawing multiple parallel arcs for all possible transitions whose label begins with A, a single arc is drawn and labeled [A, *?*, *?*] (Fig. 3):

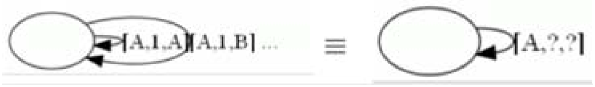

Fig. 3. Representing arbitrary with '?'

Another shorthand used is the '¬' (negation) sign placed before a state name in a triple. A label that contains a '¬' before a state name stands for all possible labels that *do not* contain the said state in that position of the triple (Fig. 4).

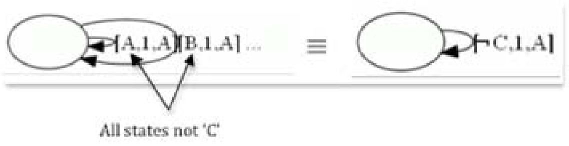

Fig. 4. Denoting all possible labels that do not start with state 'C'

In the following examples '*?*' and '¬' sometimes appear in the same arc label. The first example shown in Figure 5 is the automaton that accepts computations of the system model that start in the state 'A':

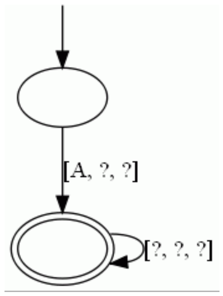

Fig. 5. Automaton accepting computations starting in state 'A'

Another example shown in Figure 6 is the automaton that accepts computations that end in the state 'B':

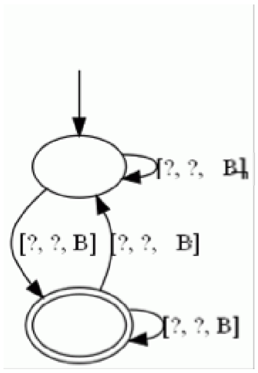

Fig. 6. Automaton that accepts computations ending in state 'B'

Figure 7 shows an example of an automaton that accepts only those computations of the system model that do not contain event '1':

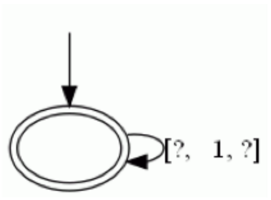

Fig. 7. Automata that accepts computations that do not contain event '1'

These three examples are sufficient for the case study discussed in the next section. More complex witness accounts can be formulated as patterns or regular expression [10] and then converted into finite automata.

4 Analysis of the Evidence

This section illustrates how to derive the feasibility of a given witness statement by using the proposed method. The printer analysis case explored in [6] will be examined.

4.1 The Case (Adopted from [6])

The local area network at ACME Manufacturing consists of two personal computers and a networked printer. Its two users, Alice (A) and Bob (B), share the cost of running the network. Alice, however, claims that she never uses the printer and should not be paying for the printer consumables. Bob disagrees; he says that he saw Alice collecting printouts. The system administrator, Carl, has been assigned to investigate this dispute.

To get more information about how the printer works, Carl contacted the manufacturer. According to the manufacturer, the printer works as follows:

1. When a print job is received from the user it is stored in the first unallocated directory entry of the print job directory.
2. The printing mechanism scans the print job directory from the beginning and picks the first active job.
3. After the job is printed, the corresponding directory entry is marked as "deleted", but the name of the job owner is preserved.

The manufacturer also noted that

1. The printer can accept only one print job from each user at a time.
2. Initially, all directory entries are empty.

After that, Carl examined the print job directory. It contained traces of two of Bob's print jobs, and the rest of the directory was empty:

job from B (deleted)
job from B (deleted)
empty
empty
empty
...

4.2 Informal Analysis

Carl reasons as follows: If Alice never printed anything, only one directory entry must have been used because the printer accepts only one print job from each user. However, two directory entries have been used and there are no other users except Alice and Bob. Therefore, it must be the case that both Alice and Bob submitted their print jobs at the same time. The trace of Alice's print job was overwritten by Bob's subsequent print jobs.

Given the printer model, it would be possible to test Carl's reasoning process, and explore other possibilities Carl's informal reasoning may have missed.

4.3 Defining the Printer Model

Carl's observation claims that there were traces of print jobs in only the first two printer queues. Since the remaining queues were observed as being empty, there is no other relevant information that could be derived from them. Because of this, the investigation and modeling can be defined in terms of only the first two. This helps to reduce the scope of possibilities, and reduces the complexity required to model the system.

Each queue can be defined as possibly having the following states:

- E – the queue is completely empty
- A – the queue contains a job from Alice
- B – the queue contains a job from Bob
- Del_A – the queue contains a deleted job from Alice
- Del_B – the queue contains a delete job from Bob

The possible transitions that can act on the queues are:

- Add_A – represents Alice printing
- Add_B – represents Bob printing
- Take – represents deleting the job

The printer, as defined in the case, will be represented as a finite state machine $Printer = (Q, \Sigma, \delta)$ [5] where

- $Q = \{(i,j) | i,j \in \{E, A, B, Del\_A, Del\_B\}\}$
- $\Sigma = \{Add\_A, Add\_B, Take\}$
- $\delta : Q \times \Sigma \rightarrow Q$ is a transition function that determines the next state

By applying the proposed algorithm, the automata model of the printer ($Printer_1$) is obtained which is graphically shown in Figure 10.

4.4 Restriction of the Model

Given this definition of $Printer_1$, all possible combination of states and transitions are represented for the system. Applying known, definite observations can restrict the number of possible events. If the observations come from a trusted source, such as a direct observation by the investigator, then the resulting restricted model can also be trusted to be accurately representative of the actions of the system.

The first known observation is that of the manufacturer who claims the initial state of the printer is (E, E), meaning that both queues were empty. The manufacturer has no knowledge after shipping. An automaton can be defined to represent this observation (Fig. 8):

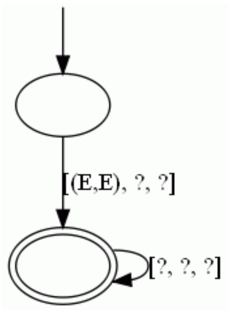

Fig. 8. Observation of the Manufacturer

The second observation is that of Carl, who observed the final state as (Del_B, Del_B), meaning that there are two deleted jobs from Bob in the queue. He has no knowledge of the queues before the investigation (Fig. 9).

These automata can be intersected with *Printer$_1$* to produce an automaton that accepts only computations that are accepted simultaneously by all of the original automata. The restricted model is shown in Figure 11 as *Printer$_{Restricted}$*.

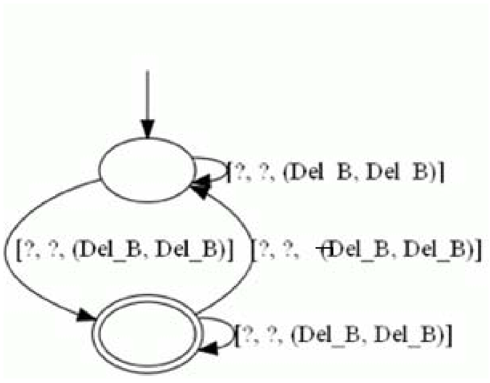

Fig. 9. Observation of Carl

The actual restriction in this instance comes from restricting states other than (E,E) from being the first in the sequence. This statement also sets the only accepting state to (Del_B, Del_B) meaning that only stings ending in this state will be accepted. As shown in Figure 11, *Printer$_{Restricted}$* contains fewer possible transitions than the full, unrestricted printer model in Figure 10.

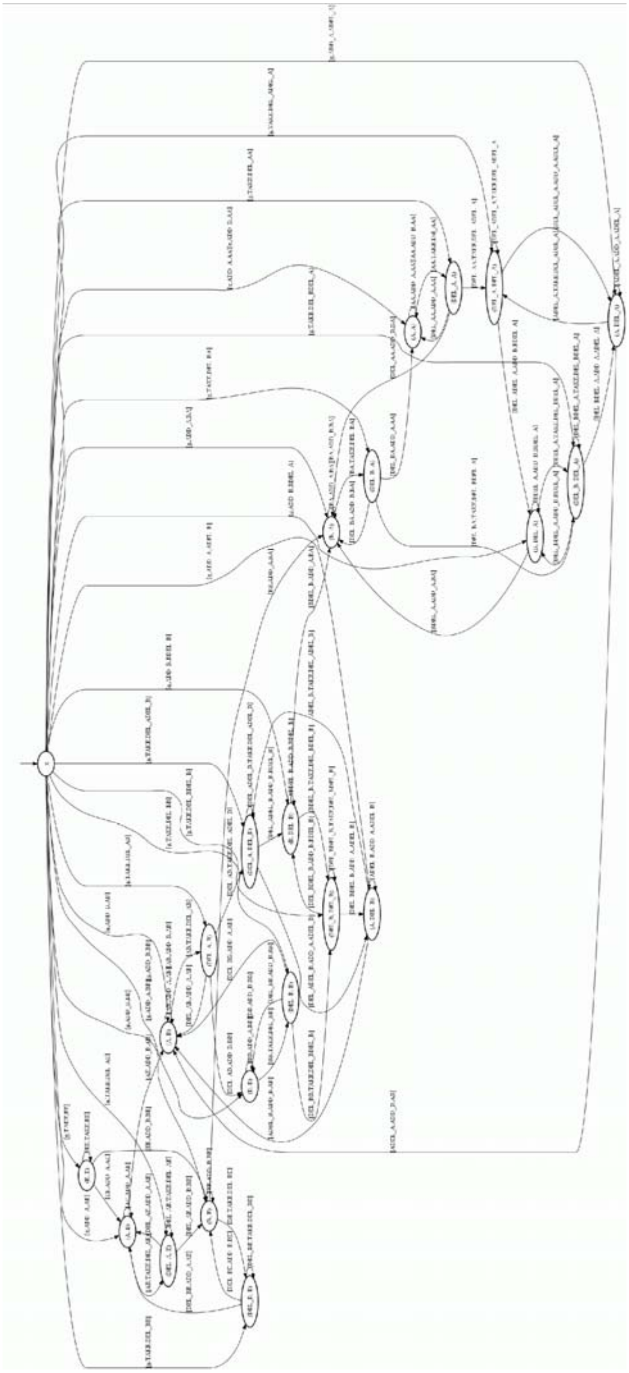

Fig. 10. Automata model of the Printer (*$Printer_1$*)

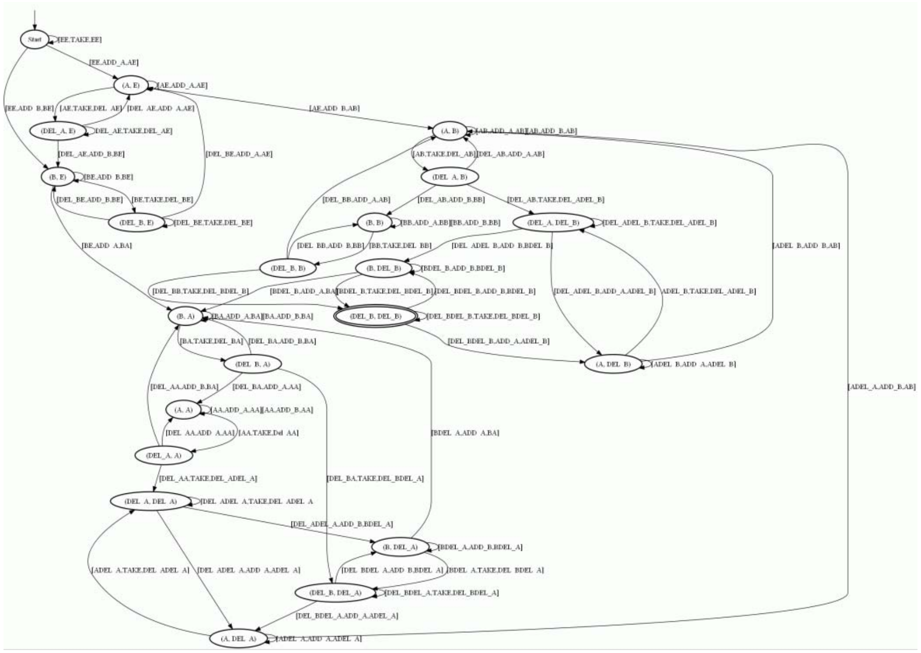

Fig. 11. *Printer₁* intersected with Manufacturer and Carl automata (*Printer*$_{Restricted}$)

Next the witness statement for Bob is created. His statement is simply that both he and Alice used the printer. Essentially this puts no restriction on the constructed system. It is a single-state automaton that accepts triples containing events 'Add_A', 'Add_B', and 'Take'.

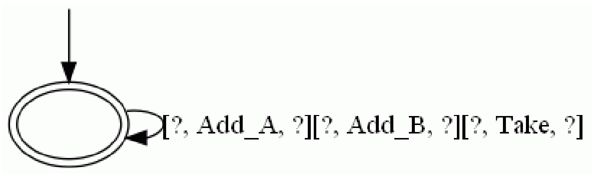

Fig. 12. Model of Bob's Statement "Bob"

Next the witness statement for Alice is created. Her statement is simply that she did not print. The resulting automaton is a single state that accepts triples containing events 'Add_B' and 'Take', but not 'Add_A'.

Because every event in *Printer*$_{Restricted}$ is also present in "Bob", intersecting will cause no reduction in states. This means that the final observed state (Del_B, Del_B) is *possible* with Bob's statement.

Alice's statement, however, does restrict the possible transitions of *Printer*$_{Restricted}$. The result of intersecting *Printer*$_{Restricted}$ and "Alice" is shown in Figure 14. Observe that the resulting automaton does not have any accepting states. This means that the statement given by Alice is not possible in accordance with the system model, or in other words, that she must be lying.

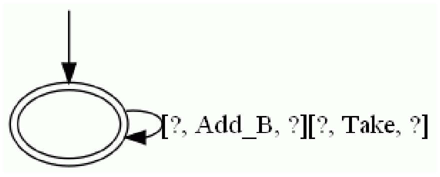

Fig. 13. Model of Alice's Statement "Alice"

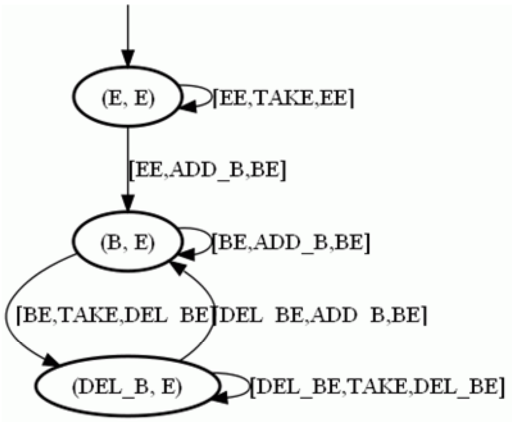

Fig. 14. Result of Intersection between the Printer Model and Alice's Statement (no accepting states present)

5 Conclusions

The advantage of state machine analysis is the fact that each possibility *is* considered. According to the method proposed in this paper, both the system under investigation and the witness statements are modeled as finite automata. This allows for a given witness statement to be tested against the system itself to determine if it is computationally possible. However, the downfall of FSM analysis continues to be the fact that the analysis of real systems remains impractically susceptible to exceedingly large state spaces. This makes the modeling of even the simplest real systems challenging. As such, any practical application would require either a greatly abstracted system, analysis of a very specific sub-set of data, or a combination of both.

6 Applications and Future Work

For the practical application of this method the investigator must focus on simple systems or be able to greatly generalize the system in question. The modeling of some components of a computer system could be automated, however since computers are a collection of sub-systems interacting with each other, some level of abstraction would be needed to derive only the operations unique to a given sub-system. There has yet to be a rigorous method for determining the appropriate level at which to model complex systems as FSMs that would be general enough for practical event reconstruction purposes. This is the focus of future research. Given the complexity of real systems, and even sub-systems, one possible way to proceed appears to be modeling a collection of sub-systems and grouping them to represent processes of the overall system. It is not clear, however, how to reconstruct these systems in a post-mortem analysis. Another approach involves simplification of the system by focusing on events that are known to have definitely happened rather than reconstructing all possible events. These could be user or system actions that can be proved to have definitely happened due to traces in the system.

References

1. Arasteh, A.R., Debbabi, M., Sakha, A., Saleh, M.: Analyzing multiple logs for forensic evidence. Digital Investigation 4, 82–91 (2007)
2. Carrier, B.D.: A Hypothesis-Based Approach to Digital Forensic Investigations. PhD Thesis, Purdue University, CERIAS, West Lafayette (2006)
3. Carrier, B.D., Spafford, E.H.: Categories of digital investigation analysis techniques based on the computer history model. Digital Investigation 3(1), 121–130 (2006)
4. Gladyshev, P.: Finite State Machine Analysis of a Blackmail Investigation. Internationl Journal of Digital Evidence 4(1), 1–13 (2005)
5. Gladyshev, P.: Formalising Event Reconstruction in Digital Investigations. State Machine Theory of Digital Forensic Analysis (August 2004), `http://formalforensics.org/publications/thesis/index.html` (retrieved January 12, 2009)
6. Gladyshev, P., Patel, A.: Finite State Machine Approach to Digital Event Reconstruction. Digital Investigation, 130–149 (2004)
7. Kozen, D.C.: Automata and Computability. In: Gries, D., Schneider, F. (eds.). Springer Science + Business Media, LLC, New York (1997)
8. Rekhis, S.: Theoretical Aspects of Digital Investigation of Security Incidents. The Communication Network and Security (CN&S) research Laboratory. Carthage: CN&S Research Lab (2008)
9. Stallard, T., Levitt, K.: Automated analysis for digital forensic science: Semantic integrity checking. In: 19th Annual Computer Security Applications Conference, Las Vegas (2003)
10. Warren, D.S.: Regular Expressions. Finite State Machines (July 31, 1999), `http://www.cs.sunysb.edu/~warren/xsbbook/node39.html` (retrieved February 17, 2009)
11. Willassen, S.: Hypothesis-Based Investigation of Digital Timestamps. In: Ray, I., Shenoi, S. (eds.) IFIP International Federation for Information Processing. Advances in Digital Forensics IV, vol. 285, pp. 75–86 (2008)