Chapter 8

Evaluation

As defined in Chapter 4, the aims of this research are *(1) to formalise event reconstruction in a general setting, that is, assuming nothing specific about the digital system under investigation or about the purpose of event reconstruction, and (2) to show that this formalisation can be used to describe and automate selected examples of digital forensic analysis.* The first aim has been achieved in Chapter 6. A formalisation of event reconstruction problem has been developed. It can be used to automate event reconstruction as follows.

1. Formalise the system under investigation as a finite state machine, and formalise the evidence from the incident as an evidential statement;

2. use the event reconstruction algorithm from Chapter 7 to compute the meaning of the evidential statement with respect to the finite state machine.

This chapter evaluates the usefulness of this method as a forensic analysis technique, and demonstrates that it can be used to automate selected examples of digital forensic analysis.

This chapter consists of three sections. Section 8.1 defines criteria for a useful forensic analysis technique and applies them to the above-described method of event reconstruction. The issues of effectiveness, efficiency, and legal admissibility are discussed.

After that, Section 8.3 uses the developed formalisation of event reconstruction to automate two examples of digital forensic analysis. First, the simple example of networked printer anlysis from Chapter 6 is given rigorous treatment in Section 8.3.1. A more complex example is then described in Section 8.3.2, which formalises and automates forensic analysis from a published case study.

Finally, Section 8.4 reviews the problems encountered in the examples and draws conclusions about the usefulness of the developed formalisation of event reconstruction.

8.1 Evaluation criteria

What constitutes a useful forensic analysis technique? As observed in Chapter 3, forensic analysis technique is expected to be both effective and efficient in the type of analysis it performs. In addition, Chapter 2 argued that event reconstruction in digital investigations should satisfy legal requirements to expert evidence, such as Daubert criteria [30]. This gives us three criteria against which to evaluate the event reconstruction method described in the beginning of this chapter:

- *Effectiveness.* How complete and how accurate is the result of event reconstruction?

- *Efficiency.* How much manual effort does event reconstruction involve, and how long does it take to perform?

- *Conformance to admissibility requirements.* Does the event reconstruction method satisfy admissibility requirements?

The following sections discuss these criteria in more detail and evaluate how the proposed method of event reconstruction satisfies these criteria.

8.1.1 Effectiveness of event reconstruction

From effectiveness point of view, formalised and automated event reconstruction offers several advantages over semi-formal event reconstruction techniques described in Chapter 4.

Perhaps the major advantage is that formalisation and automation of event reconstruction reduces the possibility of incorrect event reconstruction. If event reconstruction is automatic, the analyst's involvement into the reconstruction process is limited. The analyst develops a model of the system and formalises the evidence. The rest is automatic process. Although this does not guarantee the absence of errors (errors, for example, can be introduced at the formalisation stage), it does remove the possibility of manual error during event reconstruction. In addition, the experience of formal methods suggests that "...Formal methods enhance existing review processes by encouraging rigorous arguments of why and in what ways the specification is correct. ..." [66]. That is, the need to formalise the incident would encourage the analyst to better understand the incident, which would reduce the possibility of errors. The examples described in Section 8.3 confirm this hypothesis.

Completeness of event reconstruction is another advantage of formalised event reconstruction. With the semi-formal event reconstruction techniques, every sequence of events has to be constructed manually. As a result, all possible sequences of events are rarely constructed. However, by computing the meaning of a single evidential statement, the analyst obtains all possible sequences of events that agree with the available evidence.

8.1.2 Efficiency of event reconstruction

To be useful in investigations, event reconstruction process should be sufficiently quick. With the developed formalisation of event reconstruction, the time required for event reconstruction is divided into the time spent on formalisation of the incident and the time spent on running the event reconstruction algorithm.

The time required for formalisation of an incident is hard to estimate, because it depends on the circumstances of the incident. Note, however, that the developed formalisation of event reconstruction is based on the same mathematical principles as the existing methods for formal specification and verification of computing systems. As a result, formalisation of incidents is likely to require the same kind of effort as formal specification of computing systems.

The running time of the event reconstruction algorithm has been estimated in Chapter 7. An upper bound on the algorithm's running time is given by the inequality 7.28. It is exponential in the size of the evidential statement[1] and polynomial in the size of the finite state machine[2]. The exponential complexity means that the algorithm may not be able to handle large evidential statements with many observation sequences. This is a major problem limiting its application in practical investigations. Development of a more efficient event reconstruction algorithms is an important direction for future research. On the other hand, the examples descried in Section 8.3 show that the event reconstruction algorithm from Chapter 7 may still find practical applications despite its exponential complexity.

8.1.3 Legal admissibility of event reconstruction

Reconstruction of past events in a computer system requires special knowledge and, as such, falls into the category of expert evidence. The following paragraphs give reasons why formalised and automated event reconstruction based on the results of this research can pass such a test. To make the discussion

[1] More precisely, it is exponential in the length of the evidential statement, maximal length of observation sequences, the value of *infinitum*, and the maximal length of prefixes used for representing observed properties.

[2] More precisely, it is polynomial in the size of the state space and the number of possible events

more realistic, each of the following paragraphs addresses one of the Daubert criteria from Chapter 2. Note that Daubert criteria are *non-mandatory* and *non-exclusive*, that is, expert evidence may still be admissible, even if it does not conform to some of the Daubert criteria[3].

Can the technique be (and has it been) tested? The formalisation of event reconstruction developed in this dissertation relies on a well known mathematical apparatus to describe the incident and to perform event reconstruction. An algorithm for performing event reconstruction is provided. This makes the results of event reconstruction repeatable and amenable to independent verification by third-party experts. The formalisation of event reconstruction developed in this dissertation has been tested on example problems using the method described in the beginning of this chapter. The results of testing are described in Section 8.3.

What is the technique's known (or possible) error rate? The error rate associated with the developed method has not been measured. However, as discussed in Section 8.1.1, its error rate is likely to be lower than the error rate of existing semi-formal event reconstruction techniques described in Chapter 4.

If there are standards governing the application of the analysis technique, are they maintained? At the time of writing, there are no such standards.

Has the technique been subjected to peer-review and publication? The results of this research have been published in a peer-reviewed journal. The paper [39] is given in Appendix D.

[3] Ultimately, expert evidence is admissible if the judge is *convinced* that the underlying methodology is scientifically valid.

If the technique is known in the relevant scientific community, is it widely accepted? The results of this research are not widely known at the time of writing, because the paper describing them has been published very recently.

In summary, the formalisation of event reconstruction developed in this dissertation provides sufficient basis for passing the admissibility test, because it uses representation and analysis methods of a well known branch of science, it has been published in a peer-reviewed journal, and it has been tested on example problems described in the next chapter.

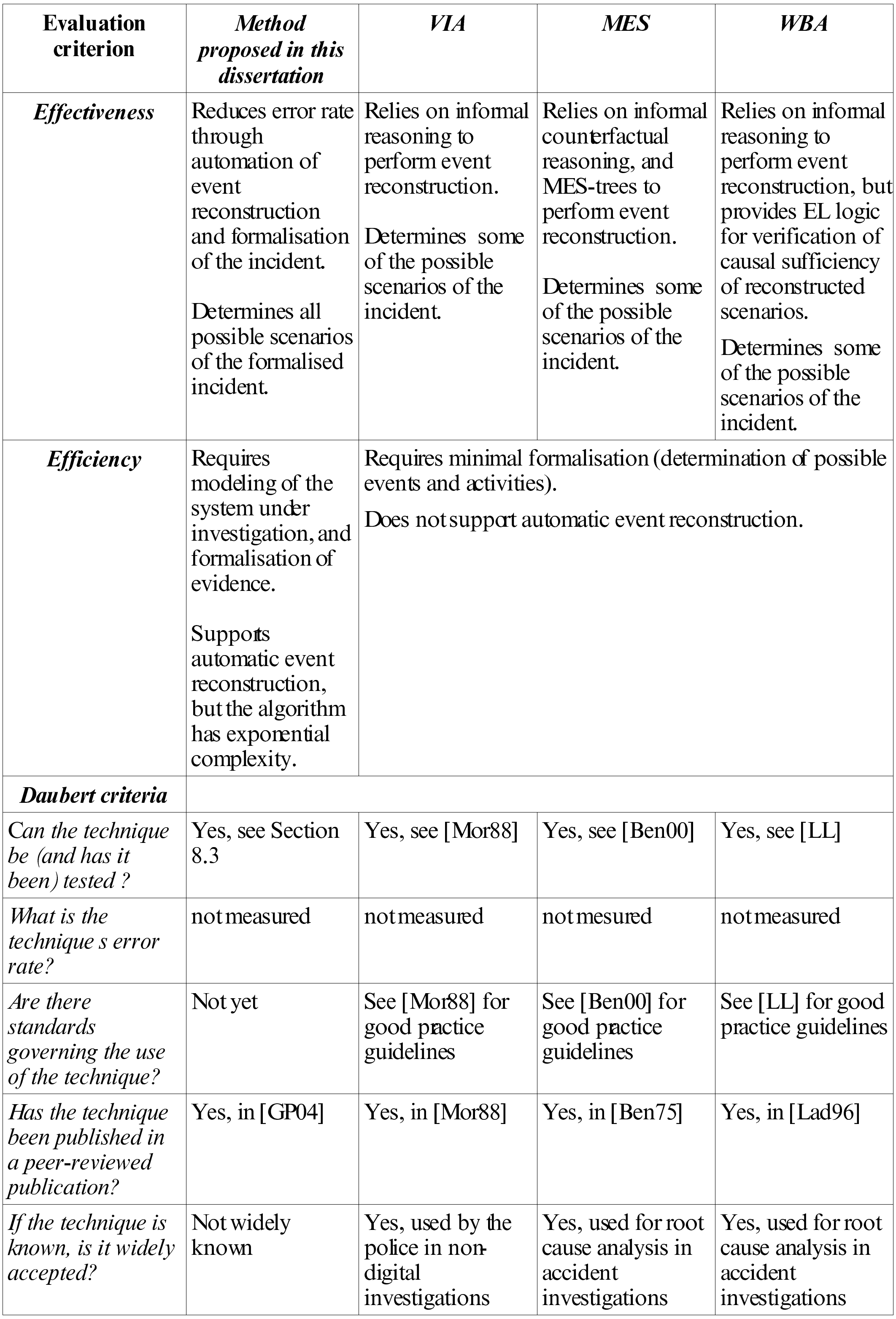
8.2 Comparison with other event reconstruction techniques

To conclude the first part of this chapter, Figure 8.1 compares the event reconstruction method described in this dissertation with existing semi-formal event reconstruction techniques described in Chapter 4. As the basis for comparison, it uses the three criteria defined above. The attack trees are not shown in the table, because a similar technique is already incorporated into MES technique.

As discussed in the previous sections, the method proposed in this dissertation is more effective than semi-formal event reconstruction techniques, but it has potentially lower efficiency, due to higher formalisation effort and exponential complexity of the event reconstruction algorithm.

8.3 Examples of formalised and automated event reconstruction

This section gives two examples of event reconstruction using the method described in the beginning of this chapter. Section 8.3.1 formalises and au-



Evaluation criterion	***Method proposed in this dissertation***	***VIA***	***MES***	***WBA***
Effectiveness	Reduces error rate through automation of event reconstruction and formalisation of the incident. Determines all possible scenarios of the formalised incident.	Relies on informal reasoning to perform event reconstruction. Determines some of the possible scenarios of the incident.	Relies on informal counterfactual reasoning, and MES-trees to perform event reconstruction. Determines some of the possible scenarios of the incident.	Relies on informal reasoning to perform event reconstruction, but provides EL logic for verification of causal sufficiency of reconstructed scenarios. Determines some of the possible scenarios of the incident.
Efficiency	Requires modeling of the system under investigation, and formalisation of evidence. Supports automatic event reconstruction, but the algorithm has exponential complexity.	Requires minimal formalisation (determination of possible events and activities). Does not support automatic event reconstruction.		
Daubert criteria				
Can the technique be (and has it been) tested ?	Yes, see Section 8.3	Yes, see [Mor88]	Yes, see [Ben00]	Yes, see [LL]
What is the technique s error rate?	not measured	not measured	not mesured	not measured
Are there standards governing the use of the technique?	Not yet	See [Mor88] for good practice guidelines	See [Ben00] for good practice guidelines	See [LL] for good practice guidelines
Has the technique been published in a peer-reviewed publication?	Yes, in [GP04]	Yes, in [Mor88]	Yes, in [Ben75]	Yes, in [Lad96]
If the technique is known, is it widely accepted?	Not widely known	Yes, used by the police in non-digital investigations	Yes, used for root cause analysis in accident investigations	Yes, used for root cause analysis in accident investigations

Figure 8.1: Comparison with other event reconstruction techniques

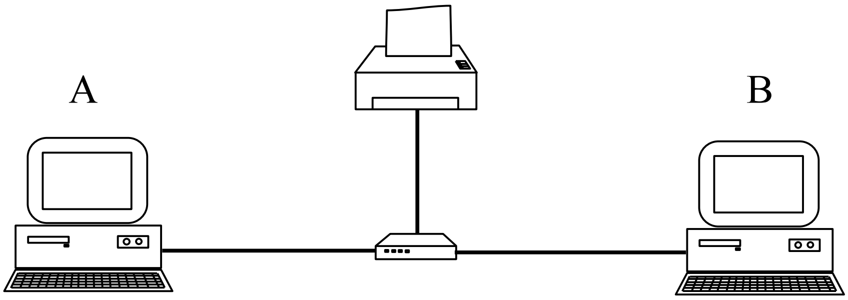


Figure 8.2: ACME Manufacturing LAN topology

tomates the simple example of networked printer anlysis from Chapter 6. A more complex example is then described in Section 8.3.2, which formalises and automates forensic analysis from a published case study.

8.3.1 Example 1. Networked printer analysis

This section illustrates the proposed formalisation of event reconstruction by applying it to the fictional example of networked printer analysis from Section 6. First, for the reader's convenience, the description of ACME investigation is repeated. Then the analysis is completely formalised and solved using the event reconstruction algorithm from Chapter 7.

Investigation at ACME Manufacturing

The dispute. The local area network at ACME Manufacturing consists of two personal computers and a networked printer as shown in Figure 8.2. The cost of running the network is shared by its two users Alice (A) and Bob (B). Alice, however, claims that she never prints anything 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.

The investigation. 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

4. The printer can accept only one print job from each user at a time.

5. Initially, all directory entries are empty.

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

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

The analysis. Carl reasons as follows. If Alice never printed anything, only one directory entry must have been used, because 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 the Alice's print job was overwritten by Bob's subsequent print jobs.

Automated analysis of ACME investigation

This subsection describes automated analysis of the print job directory using formalisation of event reconstruction developed in Chapter 6 and event reconstruction algorithm from Chapter 7.

Formalisation of system functionality The first step is to describe system functionality as a finite state machine. A suitable state machine was shown in Figure 6.2. For the reader's convenience it is reproduced again in Figure 8.3. Given below is a justification for the states and events chosen.

The informal analysis of the incident given in the previous section makes the following implicit assumptions about the incident

1. Alice and Bob have been the only users of the ACME Manufacturing LAN, and that security of the LAN has not been compromised.
2. Printer has always worked according to the manufacturer's description.
3. A directory entry that contains active or deleted print job is not "empty".
4. The state of the print job directory is modified only by addition of new print jobs, and by the printing mechanism fetching the print jobs from the directory.

It follows from assumptions 1, 2, and 4 that each directory entry has only five possible values: active job from Alice (A), active job from Bob (B), deleted job from Alice (A_deleted), deleted job from Bob (B_deleted), and empty.

$$ENTRY = \{\text{A, B, A\_deleted, B\_deleted, empty}\}$$

It follows from Carl's examination and assumptions 2 and 3 that only two directory entries have ever been used. Thus, the set of states needs to represent only the first two directory entries:

$$Q = ENTRY \times ENTRY$$

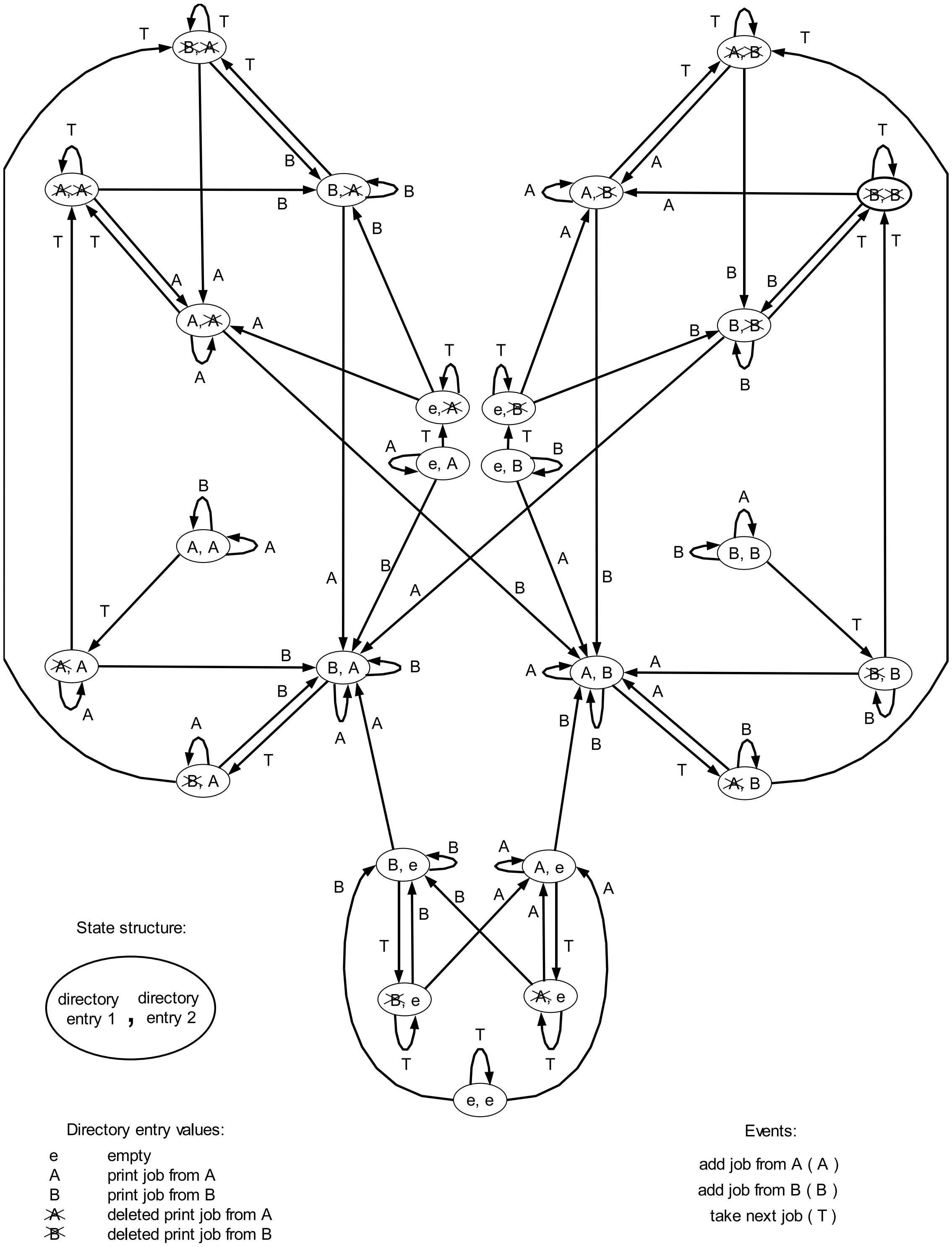


Figure 8.3: Transition graph of the print job directory model

It also follows from assumptions 1, 2, and 4 that the state of the print job directory is modified by only three possible events: submission of a print job by Alice (add_A), submission of a print job by Bob (Add_B), and the fetching of the first active print job by the printing mechanism (take).

$$I = \{\text{add\_A, add\_B, take}\}$$

Appendix C.4 presents formalisation of the state machine in ACL2 / Common Lisp. The set of states Q is defined by the recogniser function `statep`. The set of events I is defined by the recogniser function `eventp`. The transition function, whose graph is shown in Figure 8.3, is implemented by function `st`. The inverse transition function is implemented by function `rev-st`. It can be proved in ACL2 theorem prover that, when `rev-st` is given a proper state y it returns the list of all event-state pairs, whose next state according to `st` is y. The following two ACL2 theorems formalise this statement.

```
(defthm correctness-of-rev-st-wrt-st
  (implies
    (and (statep s)
         (eventp e)
         (statep y)
         (member-equal (list e s) (rev-st y)))
    (equal (st e s) y)))


(defthm completeness-of-rev-st-wrt-st
  (implies
    (and (statep s)
         (eventp e)
         (equal y (st e s)))

    (member-equal (list e s) (rev-st y))))
```

Formalisation of evidence Consider properties observed by the witnesses. The initial state of the print job directory, which was observed by the printer manufacturer, is described by the property

$$P_{\text{empty}} \;=\; \{c \,|\, c \in C_T,\; c_0^q = (\text{empty}, \text{empty})\,\}$$

which says that both directory entries at the moment of observation are empty. The final state of the printer, which was observed by Carl during printer examination, is described by the property

$$P_{\text{B\_deleted}} \;=\; \{c \,|\, c \in C_T,\; c_0^q = (\text{B\_deleted}, \text{B\_deleted})\,\}$$

which says that both directory entries at the moment of observation contain deleted print jobs from Bob.

The complete "stories" told by Carl and the printer manufacturer are captured by two observation sequences. The first observation sequence describes Carl's story:

$$os_{\mathit{Carl}} \;=\; (\,(C_T, 0, \mathit{infinitum}),\; (P_{\text{B\_deleted}}, 1, 0)\,)$$

it says that Carl observed nothing about the state of the print job directory, until he examined the printer and found that the first two directory entries contained deleted print jobs from Bob.

The manufacturer story is that, initially, all directory entries were empty, but then the printer was sold and nothing was observed about its subsequent states:

$$os_{\mathit{manufacturer}} \;=\; (\,(P_{\text{empty}}, 1, 0),\; (C_T, 0, \mathit{infinitum})\,)$$

These observation sequences form the evidential statement

$$es_{\mathit{ACME}} \;=\; (\; os_{\mathit{Carl}},\; os_{\mathit{manufacturer}}\;)$$

The evidential statement combines the knowledge contained in the two observation sequences. The task of event reconstruction is to find all computations that satisfy both observation sequences simultaneously.

Testing investigative hypotheses The purpose of event reconstruction is usually to prove or disprove some claim about the incident. To *disprove* a claim the investigator has to show that there are no explanations of evidence that agree with the claim. To *prove* the claim the investigator has to show that all explanations of evidence agree with the claim[4]. If there are some explanations of evidence that agree with the claim, and some explanations of evidence that disagree with the claim, the claim is neither proven nor disproven. Additional evidence is required to eliminate the explanations that cause the uncertainty.

In the ACME investigation, the claim is that Alice never printed anything. To formally disprove that claim, Carl has to show that all explanations of the evidential statement es_{ACME} involve Alice printing something at one point or another. A straightforward approach would be to compute all possible explanations for es_{ACME} and check them all manually. However, this approach is impractical if the number of explanations is large. An alternative approach is to formulate the claim as an observation sequence, include it into the evidential statement, and try to find explanations that agree with both the evidence and the claim.

For example, Alice's claim can be formalised as observation sequence, which says that Alice did not print anything until Carl examined the printer:

$$P_{Alice} = \{c \,|\, c \in C_T,\ (c_0^q)_0 \neq \mathrm{A} \wedge (c_0^q)_1 \neq \mathrm{A}\,\}$$

$$os_{Alice} = (\,(P_{Alice}, 0, \mathit{infinitum}),\ (P_{\mathrm{B\_deleted}}, 1, 0)\,)$$

The extended evidential statement for the ACME investigation is then

$$es'_{ACME} = (os_{Alice}) \cdot es_{ACME}$$

[4] Note that this is equivalent to disproving the negation of the claim

If there are explanations of es'_{ACME} they must agree with both the evidence and the Alice's claim, which means that the claim may or may not be true. If, however, there are no explanations of es'_{ACME} but there are some explanations of es_{ACME} the claim must be false, because it makes evidential statement inconsistent.

Choosing the value of *infinitum* The final step in formalisation of ACME investigation analysis is to choose appropriate value of *infinitum*. Since the running time of the event reconstruction algorithm is exponential in the value of *infinitum*, the smallest possible value of *infinitum* should be chosen.

Recall that a run explaining an evidential statement must satisfy all observation sequences in it. As a result, if the maximal length of explaining run can be determined for *one* observation sequence in the evidential statement, then *infinitum* does not have to be bigger than that length.

Carl's informal analysis suggests that Alice must be lying. It means that the backtracing process started from the state (B_deleted, B_deleted) would not be able to reach the initial state, because all paths to the initial state would have states with Alice's job in them.

Taking into account these two observations, it was decided to pick a small initial value of *infinitum* and gradually increase it until the set of explanations computed by `solve-os` for os_{Alice} stops growing. This approach quickly proved problematic because of loops in the transition graphs.

The problem is illustrated by Figure 8.4, which shows computations satisfying os_{Alice} with $infinitum = 2$. Consider backtracing of computation

$$c \;=\; (\,(\mathrm{take}, (\mathrm{B\_deleted}, \mathrm{B\_deleted})\,),\; (\mathrm{take}, (\mathrm{B\_deleted}, \mathrm{B\_deleted})\,),\, \ldots\,)$$

Each transition in c represents attempt of the printing mechanism to take the next print job from the empty print job directory. It does not change the state of the print job directory, because the directory is empty. However, unless

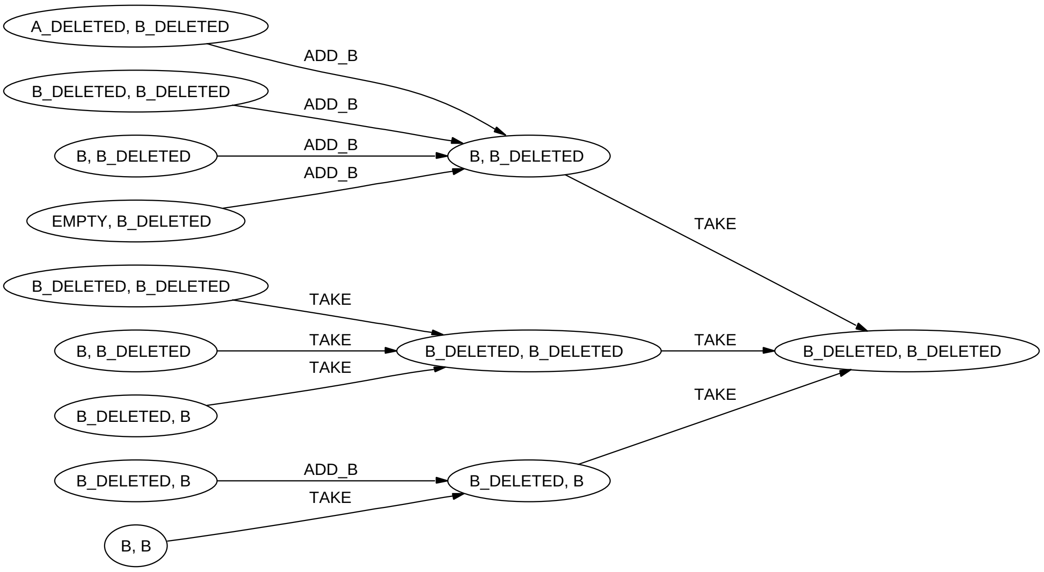


Figure 8.4: Meaning of os_{Alice} with *infinitum* = 2

there is external evidence of presence or absence of such a transition, there is no reason to believe that it never happened, or that it happened once, twice, or any other number of times. In os_{Alice} there is no such evidence. Thus, the event reconstruction algorithm dutifully reconstructs all possible sequences of (take, (B_deleted, B_deleted)) until the current value of *infinitum* is reached.

Another family of computations that cause the same problem are computations of the form

$$c = (\,(\text{Add\_B}, (\text{B\_deleted}, \text{B\_deleted})\,),\ (\text{take}, (\text{B}, \text{B\_deleted})\,), \ldots\,)$$

It represents printing of Bob's documents after the system first entered the state (B_deleted, B_deleted).

The problem was resolved by exploiting the nature of Alice's claim. Recall that the claim is that Alice *never* printed anything.

First observe that, if Alice had printed something, it would have changed the state of the print job directory, because Alice's print job would have been added to the print job directory. Thus, *all* single transitions that do not change

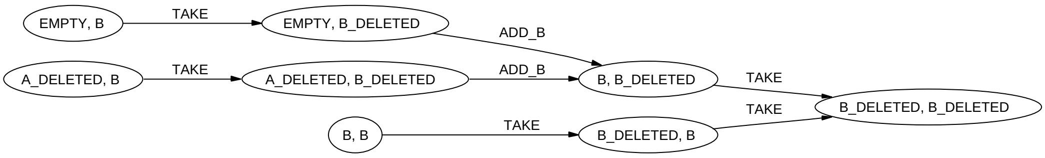


Figure 8.5: Meaning of restricted Alice's claim os'_{Alice} with *infinitum* $= 3$.

state of the print job directory can be excluded from the analysis.

Observe further that the truth or falseness of Alice's claim is not affected by the transition loops, which do not involve Alice printing something. The repetitive printing of Bob's documents represented by the loop $(\,(\text{Add\_B}, (\text{B\_deleted}, \text{B\_deleted})\,),\ (\text{take}, (\text{B}, \text{B\_deleted})\,),\ \ldots\,)$ does not involve Alice printing anything. It means that *that particular* loop can be excluded from the analysis[5].

Reflecting these insights, the property P_{Alice} was extended with two additional restrictions:

$$\begin{aligned} P'_{Alice} = \{c \,|\, c \in C_T,\\ &(c_0^q)_0 \neq \text{A} \wedge (c_0^q)_1 \neq \text{A}\\ &c_0^q \neq c_1^q,\\ &c_0 \neq (\text{Add\_B}, (\text{B\_deleted}, \text{B\_deleted})) \vee c_1 \neq (\text{take}, (\text{B}, \text{B\_deleted}))\} \end{aligned}$$

The first additional restriction $c_0^q \neq c_1^q$ excludes from consideration single transitions that do not change the state of the print job directory. The second additional restriction $c_0 \neq (\text{Add\_B}, (\text{B\_deleted}, \text{B\_deleted})) \vee c_1 \neq (\text{take}, (\text{B}, \text{B\_deleted}))$ excludes from consideration the printing of the Bob's print jobs after the print job directory first entered the state (B_deleted, B_deleted).

[5] This must be formalised in such a way that it does not exclude computations that exit halfway through the loop

The restricted Alice's claim is described by observation sequence

$$os'_{Alice} \;=\; (\,(P'_{Alice}, 0, \mathit{infinitum}),\; (P_{\text{B\_deleted}}, 1, 0)\,)$$

The set of explanations for the restricted Alice's claim stabilises for the values of *infinitum* $>= 3$. The set of computations that satisfy it for *infinitum* $= 3$ is shown in Figure 8.5. Note that none of these computations begin in the state $(\text{empty}, \text{empty})$.

The maximal length of explaining run for observation sequence os'_{Alice} is 4. Thus, *infinitum* $= 4$ is sufficient for the evidential statement extended with the restricted Alice's claim

$$es''_{ACME} \;=\; (os'_{Alice}) \cdot es_{ACME}$$

However, to ensure that some explanations are produced for es_{ACME}, the value of *infinitum* was increased to 6.

Running the automated test The code given in Appendix C.4 was run and the computed meanings of evidential statements es_{ACME} and es''_{ACME} were manually checked. While the meaning of es_{ACME} contained single explanation shown in Figure 8.6, the meaning of es''_{ACME} was empty, which means that Alice's claim contradicts the evidence. *The result of the automated analysis, therefore, agrees with the informal analysis.*

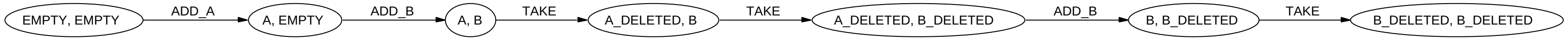


Figure 8.6: Meaning of evidential statement es_{ACME} with *infinitum* $= 6$

8.3.2 Example 2. Example of event time bounding

This section uses the developed formalisation of event reconstruction to analyse correctness of investigative reasoning of a published case study [9]. More specifically, this section analyses the proof that refutes the suspect's alibi. The proof is an example of event time bounding reasoning, which was described in Section 3.2.3.

The example is organised into four parts. First, a description of the case study is given. Second, event time bounding is formalised in terms of concepts developed in Chapter 6. Third, a model of the system is created. Finally, automatic event reconstruction followed by event time bounding are performed. The automated analysis was able to detect several implicit assumptions, whose validity is not justified in [9].

A blackmail investigation

The incident The following description, with some omissions, is taken from [9]. "The police in the UK received a complaint from a Mr. C, alleging that he was being blackmailed. The evidence was in the form of a floppy disk on which was a word processor data file which contained number of allegations, threats and demands. The floppy was known to be sent by a Mr. A, a computer consultant and friend of Mr. C. Police officers immediately went to interview Mr. A and found that he was on holiday abroad. However, his business premises were open and a computer found there was seized for examination.

When Mr. A returned from holidays, he was interviewed, and admitted sending the disk. He also admitted writing the letter found on his own machine but denied making the threats and demands. He suggested that Mr. C had added these himself in order to discredit Mr. A . . . (skipped) . . . Mr. A offered his full co-operation but suggested that care should be taken in the investigation since during his absence on holidays, his computer was available for Mr. C to use. It was therefore possible that Mr. C had used the computer to introduce the threats and demands into the file on the floppy disk and this may have left traces which might be misinterpreted as suggesting that Mr. A had made them."

Forensic examination and analysis The contents of Mr. A computer's hard drive was examined. A total of 17 recognisable fragments of the letter located in various areas of the disk space were identified. One of the fragments was a "clean" letter, without threats, stored in an active file. Other fragments contained threats and were found in unallocated disk space.

It was concluded by the investigators that the fragments found in unallocated space were deleted versions of the letter. The conclusion follows from the fact that, when a file is deleted, FAT-based file systems do not erase the content of clusters previously used by the deleted file.

The textual contents of the fragments was compared and it "enabled the fragments to be placed in a unique sequence indicating precisely how the original document had been created and subsequently edited through a number of revisions [9]." The timestamps available in the file system indicated that all modifications happened before Mr. A went on holiday. The timestamps, however, were considered to be inconclusive. To fix the editing sequence in time, a form of event time bounding was used instead.

The time bounding relied on the properties of so-called *slack space*, which is unused space at the end of the last cluster of an active file. The formation of slack space is illustrated in Figure 8.7. One of the blackmail fragments was found in the slack space of another letter unconnected with the incident. When the police interviewed the person to whom that letter was addressed, he confirmed that he had received the letter on the day that Mr. A had gone abroad on holiday. It was concluded that

> "This fixed the whole sequence in time and showed Mr. A's story to be completely false. The threats and demands had been reintroduced into the letter at least two days before Mr. A went on

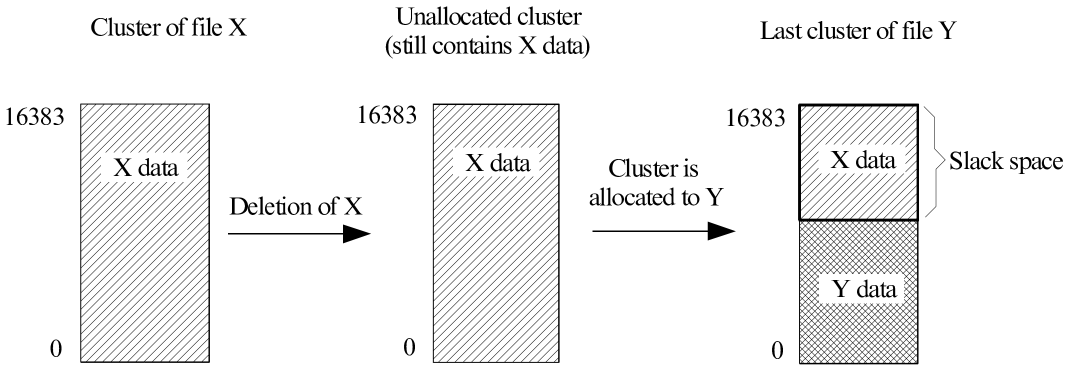


Figure 8.7: Formation of the slack space

> holiday – Mr. C could not have been involved [9].
>
> Mr. A has pleaded guilty to the charge of blackmail but there are many other complicating factors in this case and investigation are continuing."

The final piece of reasoning that stroke the final blow to the integrity of Mr. A's theory must have been that

1. The letter unconnected with the incident must have been written after the letter with threats and demands, because of the way the slack space is formed.

2. Since the letter unconnected with the incident was received on the day the Mr. A had gone on holidays, it must have been written and posted at least two days before (because of the way the postal service works).

3. Based on 1 and 2, the letter with threats and demands must have been written before Mr. A went on holiday.

Note that the first step in this reasoning is event reconstruction, while the second and the third steps are examples of event time bounding. Formalisation of this reasoning is the subject of the next two sections.

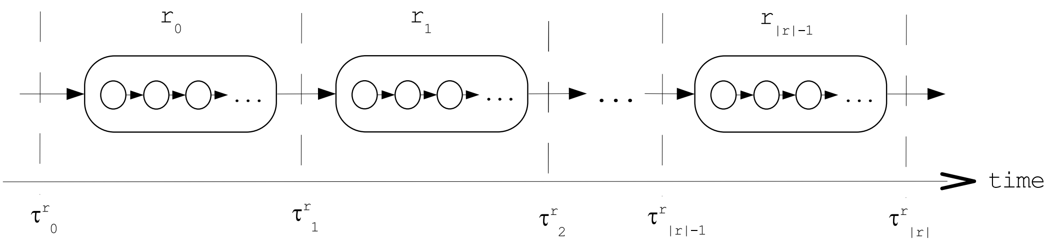


Figure 8.8: Times of transitions

Section 8.3.2 shows how formalisation of event reconstruction can be extended with real times of observations, and how event time bounding can be formalised in its context. Section 8.3.2 applies these results to the analysis of blackmail investigation. The formal analysis identified some of the implicit assumptions present in [9].

Formalisation of event time bounding

Assigning time to transitions and runs Real time can be introduced into state machine model of Chapter 6 by associating real times with transitions in style of [6].

Any run r is associated with $|r|+1$ transitions. There are $|r|$ transitions *into* each computation of r and one transition *out* of the last computation of r.

Transition times of a run. The sequence of transition times of a run $r \in R$ is denoted τ^r. It consists of $|\tau^r|$ elements.

$$|\tau^r| = |r| + 1$$

All elements of τ^r are real valued numbers. The relationship between elements of τ^r and computations of r is shown in Figure 8.8. The first element τ^r_0 represents the time of transition into computation r_0, the first computation of run r. The last element $\tau^r_{|r|}$ represents the time of transition out of computation $r_{|r|-1}$, the last computation of run r. An intermediate element τ^r_i represents the time of transition from computation r_{i-1} to computation r_i. Elements of τ^r are ordered in time. For all integer i, such that $0 \leq i < |r|$,

$$\tau_i < \tau_{i+1} \tag{8.1}$$

If r is empty, then r corresponds to a single moment, whose time is τ^r_0.

The following definition formalises what is meant by a run happening before another run.

Temporal precedence of runs A run ra precedes run rb in time, if the ending time of ra is less than or equal to the beginning time of rb:

$$\tau^{ra}_{|ra|} \leq \tau^{rb}_0$$

If ra and rb are sub-runs of some run rc, then positions of ra and rb in rc can be used to determine temporal precedence between ra and rb.

Let i be the index of the first computation of ra in rc, and let j be the index of the first computation of rb in rc, then

$$\tau^{ra}_{|ra|} = \tau^{rc}_{i+|ra|}$$

and

$$\tau^{rb}_0 = \tau^{rc}_j$$

Run ra precedes run rb if

$$\tau^{ra}_{|ra|} \leq \tau^{rb}_0$$

or, equally,

$$\tau^{rc}_{i+|ra|} \leq \tau^{rc}_j$$

which by definition of τ^r is true if and only if

$$i + |ra| \leq j \tag{8.2}$$

Times of observations Witness observations regarding time of events are formalised as known times of observations.

Observation identifier. Observation identifier is a pair $id = (i, j)$. It denotes observation $e_{i,j}$ at the j-th position of the i-th observation sequence of evidential statement e.

Known time of observation. A known time of observation is a pair $t = (id, tim)$, where $id = (i, j)$ is an observation identifier and tim is a real valued number that represents time. The meaning of t is an assertion that for any run r explaining observation $e_{i,j}$, the following inequality holds

$$\tau_0^r \leq tim \leq \tau_{|r|}^r \tag{8.3}$$

A known time of observation corresponds to a witness statement that moment *tim* happened during the witness's observation. Such a statement may result from human looking at a clock during observation, or from an operating system appending clock reading to a log file entry.

Time bounding algorithm Time bounding algorithm uses known times of observations to determine time boundaries for any given observation $e_{i,j}$ within evidential statement. The idea of the algorithm is straightforward. An observation $e_{i,j}$ can happen only after the latest of observations preceding $e_{i,j}$ in time and only before the earliest of observations following $e_{i,j}$ in time.

The meaning of "preceding" and "following" observations is captured by the "happened-before" relation defined as follows. An observation $e_{i,j}$ happened before observation $e_{k,l}$ if and only if in *every* sequence of partitioned runs explaining e the run explaining $e_{i,j}$ precedes the run explaining $e_{k,l}$. The actual algorithm is based on the following two ideas.

1. Consider an sequence of partitioned runs *spr* that explains the evidential statement e

$$spr = (pr_0, pr_1, \ldots, pr_n)$$

By definition of explanation of evidential statement given in Chapter 6, all elements of *spr* are partitionings of the same run r. That is, any element of any pr_i is a sub-run of r. As a result, the precedence between the run that explains observation $e_{i,j}$ and the run that explains observation $e_{k,l}$ can be established by comparing their starting and ending positions within r.

2. The position of a sub-run that explains given observation $e_{i,j}$ can be calculated directly from the corresponding MSPR returned by $\mathit{SolveES}(e)$. Let (C, *lenlist*) be such an MSPR. In any sequence of partitioned runs represented by this MSPR, the indices of the first and last computation of the (non-empty) run that explains observation $e_{i,j}$ are

$$\sum_{l=0}^{j-1}(\mathit{lenlist}_i)j$$

and

$$\Big(\sum_{l=0}^{j}(\mathit{lenlist}_i)j\Big) - 1$$

respectively.

ACL2 code of the time bounding algorithm is given in Appendix C.5. It can be divided into three parts: (1) utility functions, (2) calculation of the earliest possible time for an observation, and (3) calculation of the latest possible time for an observation.

Calculation of the earliest time consists of two parts. First, the set of observations *bef* that happened before given observation is determined. Second, the maximal known time among observations in *bef* is found. Calculation of the latest time is similar. First, the set of observations *aft* that happened after given observation is determined. Second, the minimal known time among observations in *aft* is found.

The following paragraphs describes each part of the code in turn.

Utility functions. The algorithm uses four utility functions. Function allobs returns list of all observation identifiers for the given observation sequence.

```
(defun allobs (obs m n)
  (if (atom obs)
    nil
    (cons (list (nfix m) (nfix n))
          (allobs (cdr obs) (nfix m) (+ (nfix n) 1)))))
```

The caller must specify index m of the given observation sequence obs in the evidential statement. Counter n must be reset to 0.

Function alles returns list of all observation identifiers for the given evidential statement.

```
(defun alles (es n)
  (if (atom es)
    nil
    (append (allobs (car es) (nfix n) 0)
            (alles (cdr es) (+ (nfix n) 1)))))
```

The caller must reset counter n to 0.

Function intersection-equal takes two lists x and y and returns a list whose elements are members of both x and y.

```
(defun intersection-equal (x y)
  (declare (xargs :guard (and (true-listp x) (true-listp y))))
  (cond ((endp x) nil)
((member-equal (car x) y)
 (cons (car x) (intersection-equal (cdr x) y)))
(t (intersection-equal (cdr x) y))))
```

Finally, function sumpref adds first n elements of the given list l.

```
(defun sumpref (n l)
  (if (or (zp n) (atom l))
    0
    (+ (nfix (car l)) (sumpref (1- n) (cdr l)))))
```

If n is greater or equal to the length of l, function sumpref returns the sum of all elements of l.

Calculation of the earliest time. The set *bef* of observations that happened before observation with the given identifier is calculated using three functions shown in Figure 8.9.

```
(defun befpm (pm cnt pos i j)
  (if (atom pm)
    nil
    (if (<= (+ cnt (car pm)) pos)
      (cons (list i j)
            (befpm (cdr pm) (+ cnt (car pm)) pos i (+ j 1)))
      (befpm (cdr pm) (+ cnt (car pm)) pos i (+ j 1)))))

(defun befpml (pos pml i)
  (if (atom pml)
    nil
    (append (befpm (car pml) 0 pos i 0)
            (befpml pos (cdr pml) (+ i 1)))))

(defun findbef (v bef i j)
  (if (atom v)
    bef
    (findbef (cdr v)
             (intersection-equal
                bef
                (befpml (sumpref j (nth i (car (cdr (car v)))))
                        (car (cdr (car v)))
                        0))
             i j)))
```

Figure 8.9: Finding observations that happened before given observation

Function befpm takes an MPR pm and finds all runs whose last computation appears in the partitioned run before or at the position pos. A list of observation identifiers corresponding to each of the runs is returned.

Function `befpml` applies `befpm` to every combination of $(C, \mathit{lenlist}_i$ in the given MSPR `pml`. The lists returned by `befpm` are concatenated.

Function `findbef` processes a list of MSPRs. For each MSPR it determines the set of observations that happened before observation with the identifier (i, j). The determined sets are intersected with each other and with parameter bef. The resulting set consists of observations that happened before observation with the identifier (i, j) in *all* MSPRs. In the initial call to `findbef`, parameter `bef` must contain the list of all observation identifiers in the evidential statement. Function `alles` is used for generating such a list. To process a list of MSPRs, function `findbef` determines the beginning position of the run explaining observation $e_{i,j}$ and uses function `befpml` to find identifiers of observations explained by runs that end before that position.

Once the set *bef* is calculated, the function `maxtime` shown in Figure 8.10 finds the latest known observation time among the elements of bef. It scans the list of known times and pick the latest time whose observation identifier is a member of bef.

Finally, the calculation of the set *bef* and finding the latest known time of its elements is combined in function `lbound`, which calls functions `findbef` and `maxtime`.

Calculation of the latest time. ACL2 code of this part of time bounding algorithm is shown in Figures 8.11 and 8.12. It is very similar to the code for calculating the earliest time. There are three differences with the code for calculating the earliest time.

1. Function `aftpm` returns a list of indices of observations whose starting positions in the given MPR `pm` are greater than or equal to the specified position `pos`;

```
(defun maxtime (max l tim)
  (if (atom tim)
    max
    (let ((time (car (cdr (car tim))))
          (id (car (car tim))))
      (if (not (member-equal id l))
        (maxtime max l (cdr tim))
        (if (null max)
          (maxtime time l (cdr tim))
          (if (< max time)
            (maxtime time l (cdr tim))
            (maxtime max l (cdr tim))))))))

(defun lbound (i j es v tim)
  (maxtime nil
           (findbef v (alles es 0) i j)
           tim))
```

Figure 8.10: Calculating the earliest possible time of given observation

2. Function `findaft` calculates *sumpref*$(j+1)$ rather than *sumpref*(j), because the result of *sumpref*$(j+1)$ is the position *after* the last computation of the j^{th} element of the i^{th} element of *listlen*.

3. Function `mintime` picks the minimal known time among observations whose elements are in the set *aft*.

Reliability of known times of observations Time bounding algorithm presented above assumes the truth of all known times of observations. This assumption simplifies reasoning by avoiding reasoning with uncertainty. This assumption is acceptable, because known times can be introduced into analysis gradually. First, time bounding can be performed with only the most reliable known times. If the results of time bounding are unsatisfactory, it can be repeated with less reliable known times included.

Reliability of time bounding results can be improved by checking consistency of known times. All known times must respect happened-before ordering imposed by the evidential statement. This can be checked by calculating the

```
(defun aftpm (pm cnt pos i j)
  (if (atom pm)
    nil
    (if (<= pos cnt)
      (cons (list i j) (aftpm (cdr pm) (+ cnt (car pm)) pos i (+ j 1)))
      (aftpm (cdr pm) (+ cnt (car pm)) pos i (+ j 1)))))

(defun aftpml (pos pml i)
  (if (atom pml) nil
    (append (aftpm (car pml) 0 pos i 0)
            (aftpml pos (cdr pml) (+ i 1)))))

(defun findaft (v aft i j)
  (if (atom v)
    aft
    (findaft (cdr v)
             (intersection-equal
                aft
                (aftpml (sumpref (+ j 1) (nth i (car (cdr (car v)))))
                        (car (cdr (car v)))
                        0))
             i j)))
```

Figure 8.11: Finding observations that happened after given observation

earliest $t^{e_{i,j}}_{min}$ and the latest $t^{e_{i,j}}_{max}$ times for every observation $e_{i,j}$ in the evidential statement. Every known time tim of observation $e_{i,j}$ must fall in between the two calculated times $t^{e_{i,j}}_{min} \leq tim \leq t^{e_{i,j}}_{max}$.

Automated analysis of the blackmail investigation

Formalisation of the system functionality The first step is to define a finite state machine that adequately describes the system under investigation. In the blackmail example, the functionality of the last cluster of a file was used to determine the sequence of events. Thus, the scope of the model can be restricted to the functionality of the last cluster in a file.

The last cluster in a file can be modeled as an array of bits augmented with a length (see Figure 8.13). The array of bits represents cluster data. The length

```
(defun mintime (min l tim)
  (if (atom tim)
    min
    (let ((time (car (cdr (car tim))))
          (id (car (car tim))))
      (if (not (member-equal id l))
        (mintime min l (cdr tim))
        (if (null min)
          (mintime time l (cdr tim))
          (if (< time min)
            (mintime time l (cdr tim))
            (mintime min l (cdr tim))))))))

(defun ubound (i j es v tim)
  (mintime nil
           (findaft v (alles es 0) i j)
           tim))
```

Figure 8.12: Calculation of the latest possible time of given observation

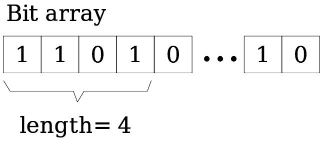


Figure 8.13: State machine model of the last cluster in a file

specifies how many bits from the beginning of the cluster are actually used by the file. Although real clusters do not have any length field, the number of data bits in the file's last cluster can be calculated from the file length and the known size of cluster in the file system. Zero length in the model would represent unallocated cluster.

Unfortunately, the event reconstruction program described in Chapter 7 was unable to work with that cluster model, because of the need to explicitly represent enormous number of possible states. It is noted in [11], that the size of cluster on the hard drive of Mr. A's computer was 16384 bytes. This results in 2^{131072} possible distinct states of the cluster model.

Despite inability to conduct analysis of the full-sized model, it was decided to continue analysis with a simplified model. The hope was that, although properties of simplified model are not the same as the properties of the full-sized model, it may still indicate some flaws in the investigative reasoning.

To make the cluster model tractable, the size of cluster was reduced to two bits – the smallest cluster size in which slack space is possible. The state space of the simplified model is defined by

$$BIT = \{0, 1\}$$

$$LENGTH = \{0, 1, 2\}$$

$$Q = LENGTH \times BIT \times BIT$$

In FAT-based file systems, the state of the last cluster can be changed by three types of events: (a) direct writes into the cluster bypassing the file system, (b) writes into the file to which the cluster is allocated, and (c) deletion of the file. Each of these events is considered separately below.

Direct writes into the cluster. Alarmingly, there is no mentioning in [9] that cluster content can be modified directly, for example by using a low-level disk editor. It seems that an implicit assumption was made in [9] that Mr. C could not have performed low-level changes on Mr. A's computer. Reflecting this assumption, direct writes have also been excluded from the model.

Writes into the file. When cluster is modified as part of the file, the new data is written into consecutive locations starting from the beginning of the cluster. In the simplified cluster model, there are only six possible sequences that can be written into the two-bit cluster:

$$WRITE = \{(0), (1), (0, 0), (0, 1), (1, 0), (1, 1)\}$$

Apart from replacing one or two bits of data, every such event also modifies the length of active data in the cluster.

Deletion of the file. After a file is deleted, the information about the number of bits stored in the last cluster of the file sooner or later becomes unavailable. This happens when the deleted file's directory entry is reused by another file, or when the FAT chain of the deleted file is broken.

To model this eventual loss of length, the deletion event del is introduced. It sets the length of the model to zero. The set of all events in the simplified cluster model is defined by

$$I \; = \; \mathit{WRITE} \cup \{\mathrm{del}\}$$

The ACL2 / Common Lisp implementation of the simplified cluster model is given in the Appendix C.6. The set of states Q is defined by the recogniser function `statep`. The set of events I is defined by the recogniser function `eventp`. The transition function and its inverse are implemented by functions `st` and `rev-st` respectively.

Automated analysis of the simplified model The exact evidential data was not published in [9]. As a result, the specific cluster contents for the blackmail letter and for the unrelated letter had to be chosen arbitrarily. Sequence $(1, 1)$ is chosen to represent the contents of the blackmail letter. Sequence (0) is chosen to represent the contents of the unrelated letter. With these choices, state $(1, 0, 1)$ represent the final state discovered by investigators in the blackmail investigation. The state describes a non-empty cluster whose active content – the letter unrelated to investigation – is sequence (0), and whose slack space contains the end of the blackmail letter – the sequence (1). The observation of this state is captured by the following property:

$$P_{\mathit{final}} \; = \; \{\, c \,|\, c \in C_T,\, c^q_0 = (1, 0, 1)\}$$

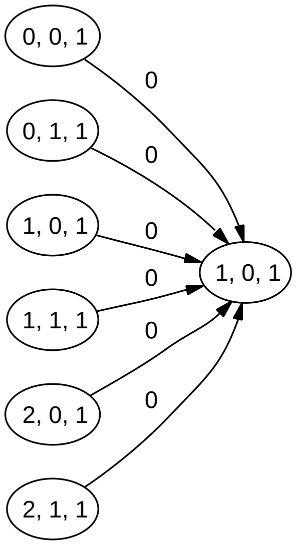


Figure 8.14: One step of event reconstruction of observation sequence os_{final}

The observations about the cluster content made by investigators in the blackmail investigation are formalised by the observation sequence os_{final}:

$$os_{\mathit{final}} \;=\; (\,(C_T, 0, \mathit{infinitum}),\; (P_{\mathit{final}}, 1, 0)\,)$$

It states that nothing was observed about the cluster's content until forensic examination, which found that the cluster was in the state $(1, 0, 1)$.

Figure 8.14 shows the result of a single step of event reconstruction for the observation sequence os_{final}. The figure shows that, as expected, the current active content of the cluster – (0) – was produced by writing it into the cluster. However, there are two distinct situations, in which that writing could have taken place. First, the cluster could have been unallocated before (0) was written into it. This possibility is represented by transitions

$$(0, 1, 1) \overset{(0)}{\rightarrow} (1, 0, 1)$$

$$(0, 0, 1) \overset{(0)}{\rightarrow} (1, 0, 1)$$

Second possibility is that, the cluster was already allocated to the file, and new value was written into it. That possibility is represented by transitions

$$(1,1,1) \xrightarrow{(0)} (1,0,1)$$

$$(1,0,1) \xrightarrow{(0)} (1,0,1)$$

$$(2,1,1) \xrightarrow{(0)} (1,0,1)$$

$$(2,0,1) \xrightarrow{(0)} (1,0,1)$$

Note the last two transitions. They suggest that the blackmail message in the slack space was *not* caused by overwriting deleted blackmail message, but by truncating the unrelated letter, which already contained the piece of the blackmail letter! This however, does not change the conclusions of the investigators.

To complete formalisation of evidence two more observation sequences need to be created. The first observation sequence $os_{blackmail}$ says that, at some point in time, the piece of the blackmail letter was written into the cluster:

$$P_{blackmail} = \{\, c \,|\, c \in C_T,\, c_0^\iota = (1,1) \,\}$$

$$\begin{aligned} os_{blackmail} = & \\ & (\,(C_T, 0, \mathit{infinitum}), \\ & \;(P_{blackmail}, 1, 0), \\ & \;(C_T, 1, \mathit{infinitum})\,) \end{aligned}$$

The minimal length of the last observation in $os_{blackmail}$ is set to 1 to exclude the possibility that the writing of the blackmail coincided with the observation of the final state.

The second observation sequence $os_{unrelated}$ says that the unrelated letter was created at some time in the past, and that later it was received by the person to whom it was addressed:

$$P_{unrelated} = \{\, c \,|\, c \in C_T,\, c_0^\iota = (0) \,\}$$

$$\begin{aligned}
&os_{\mathit{unrelated}} \;= \\
&\;(\,(C_T, 0, \mathit{infinitum}), \\
&\;\;(P_{\mathit{unrelated}}, 1, 0), \\
&\;\;(C_T, 0, \mathit{infinitum}), \quad (C_T, 0, 0), \\
&\;\;(C_T, 1, \mathit{infinitum})\,)
\end{aligned}$$

The zero-observation $(C_T, 0, 0)$ represents the reception of the letter by the addressee.

The evidential statement for the blackmail example combines os_{final}, $os_{\mathit{blackmail}}$, and $os_{\mathit{unrelated}}$:

$$es_{\mathit{blackmail}} \;=\; (\; os_{\mathit{final}},\; os_{\mathit{unrelated}},\; os_{\mathit{final}})$$

Once $es_{\mathit{blackmail}}$ was defined, *infinitum* was arbitrarily chosen to be 4, and the reconstruction was performed. The code given in the Appendix C.6 saves the result of reconstruction of $es_{\mathit{blackmail}}$ with $\mathit{infinitum} = 4$ to the constant `*SOL-4*`.

The result of event reconstruction was then used to perform time bounding of the blackmail writing. Since the exact time of reception of the unrelated letter is not specified in [9], and since there is no other timed event in $es_{\mathit{blackmail}}$ except the unrelated letter reception, the time of the reception was arbitrarily chosen to be 5. The code that performs event time bounding is shown below:

```
(defconst *tim* ’(((1 3) 5)))
(defconst *l* (lbound 2 1 *ES-BLACKMAIL* *SOL-4* *tim*))
(defconst *u* (ubound 2 1 *ES-BLACKMAIL* *SOL-4* *tim*))
```

The outcome of this computation was that both variables `*u*` and `*l*` were equal to `NIL`, which means that the algorithm was unable to determine nether upper nor lower time bound for the blackmail writing. To investigate this problem, the reconstruction results contained in *SOL-4* were examined. This revealed that one of the possible explanations of $es_{\mathit{blackmail}}$ was the sequence of transitions

$$\ldots \overset{(0)}{\to} (1, 0, 1) \overset{(11)}{\to} (2, 1, 1) \overset{(0)}{\to} (1, 0, 1)$$

This sequence of events suggests that someone could have framed Mr. A by

1. finding an unrelated letter, which was written by Mr. A earlier,
2. writing the last piece of the blackmail letter into the last cluster of the unrelated letter,
3. writing the last piece of the unrelated letter back into the cluster.

This could have been easily accomplished using some low-level disk editor. However, as was noted earlier, the possibility of using such tools seems to be excluded from [9]. If ordinary text editing tools were used instead, this result is unlikely, because text editors tend to save modified document in a new file rather than modify the original[6]. Unfortunately, there is not enough information in [9] about the system software to make any reliable assumptions.

To replicate the investigative reasoning, an assumption had to be forced that the unrelated letter was written into the cluster only once. The modified observation sequence and evidential statement are

$$P_{no\_unrelated} = \{\, c \mid c \in C_T,\ c_0^{\iota} \neq (0) \,\}$$

$$\begin{aligned} os'_{unrelated} = \\ (\,&(P_{no\_unrelated}, 0, \mathit{infinitum}),\\ &(P_{unrelated}, 1, 0),\\ &(P_{no\_unrelated}, 0, \mathit{infinitum}),\\ &(P_{no\_unrelated}, 0, 0),\\ &(P_{no\_unrelated}, 1, \mathit{infinitum})\,) \end{aligned}$$

$$es'_{blackmail} = (\, os_{final},\ os'_{unrelated},\ os_{final})$$

[6] Microsoft Word, for example, does not modify the existing file. The changed document is first written into a new file, then the original document is deleted [1].

The automated analysis of $es'_{blackmail}$ with *infinitum* $= 4$ yields the expected result – that the upper time bound for the writing of the blackmail letter is the time of reception of the unrelated letter.

8.4 Summary

The discussion presented in this chapter has demonstrated that formal approach to event reconstruction can be useful at least in some cases of digital forensic investigations. Perhaps the most important benefit of formality is its ability to focus attention of the analyst on the obscure detail, which in turn, reduces the possibility of analytical error.

In the beginning of the chapter, three criteria for a useful forensic analysis technique have been put forward. They are efficiency, effectiveness, and conformance to legal admissibility requirements. The approach to event reconstruction developed in this dissertation has been evaluated against these criteria. It has been shown that the event reconstruction approach proposed in this dissertation has better effectiveness than existing semi-formal event reconstruction techniques, because it reduces error rate and provides comprehensive reconstruction of possible incident scenarios. At the same time, it has potentially lower efficiency due to higher formalisation effort and exponential complexity of the event reconstruction algorithm. In addition to this analysis, two examples of formal event reconstruction have been performed.

The *ACME investigation* example has shown that devices with well defined and relatively simple functionality can be successfully analysed using developed approach. The most likely examples of such systems are controllers embedded in consumer electronics and appliances. The continuing integration of computing technology into human habitat is making forensic analyses of such devices increasingly likely.

The *blackmail example* has shown that even incomplete model of the system may be useful in the investigation. The need to formalise system functionality combined with probing explorations of the state space forces the analyst to consider many aspects of the system and the evidence, thus facilitating a better understanding of the investigation. This, in turn, may suggest missed assumptions or omissions in expert reasoning. The blackmail example was able to detect two implicit, obscure assumptions in a published case study. It suggests that formal analysis might be particularly useful for analysing expert reports produced by the opposing party in legal proceedings.

In addition to exponential complexity of event reconstruction algorithm, complexity of real world systems is also likely to be a major challenge for formal event reconstruction. The event reconstruction approach presented in this dissertation relies on automatic exploration of a finite state machine's state space to perform event reconstruction. The state machines considered in this chapter are very simple. The ACME investigation, for example, required creation of a finite state machine with only 25 states and 75 possible transitions. Although similar cases are possible in real life, the majority of investigations is likely to encounter systems, whose exact finite state machine models are much more complex.

The relative success of model checking suggests that the complexity problem can be dealt with using model reduction techniques and symbolic representation of state sets. The investigation of the applicability of these techniques in the domain of digital investigations is an important direction for future research.