Chapter 6

Formalisation of event reconstruction problem

This chapter uses state machine model of computation to formalise event reconstruction problem in digital investigations. The chapter is organised into two parts. First, the key concepts are introduced informally on a fictional example of networked printer analysis in Section 6.1. Second, the introduced concepts are more rigorously formalised in Section 6.2.

6.1 Informal example of state machine analysis

This section illustrates the possibility of using state machines for event reconstruction in digital investigations. It considers a fictional example of networked printer analysis. First, an informal analysis is given, then it is illustrated using a finite state model of the printer.

6.1.1 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 6.1. The

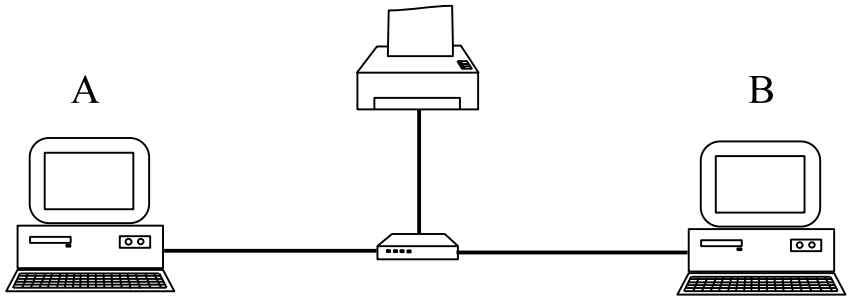

Figure 6.1: ACME Manufacturing LAN topology

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.

In the next subsection, it is shown how the same conclusion can be derived from the finite state model of the print job directory.

6.1.2 Informal analysis illustrated with a state machine

Please look at Figure 6.2. It shows a finite state model of the print job directory. Ellipses correspond to possible states of the directory. Arrows correspond to addition (or deletion) of print jobs. Each ellipse in Figure 6.2 shows the content of the print job directory in the corresponding state. For the sake of simplicity, only the first two directory entries are modeled. For example, the ellipse (A,B) represents the state in which directory contains an active job from Alice, and an active job from Bob:

job from A
job from B
empty
empty
empty
...

The initial state of the directory corresponds to the ellipse (e,e). The state

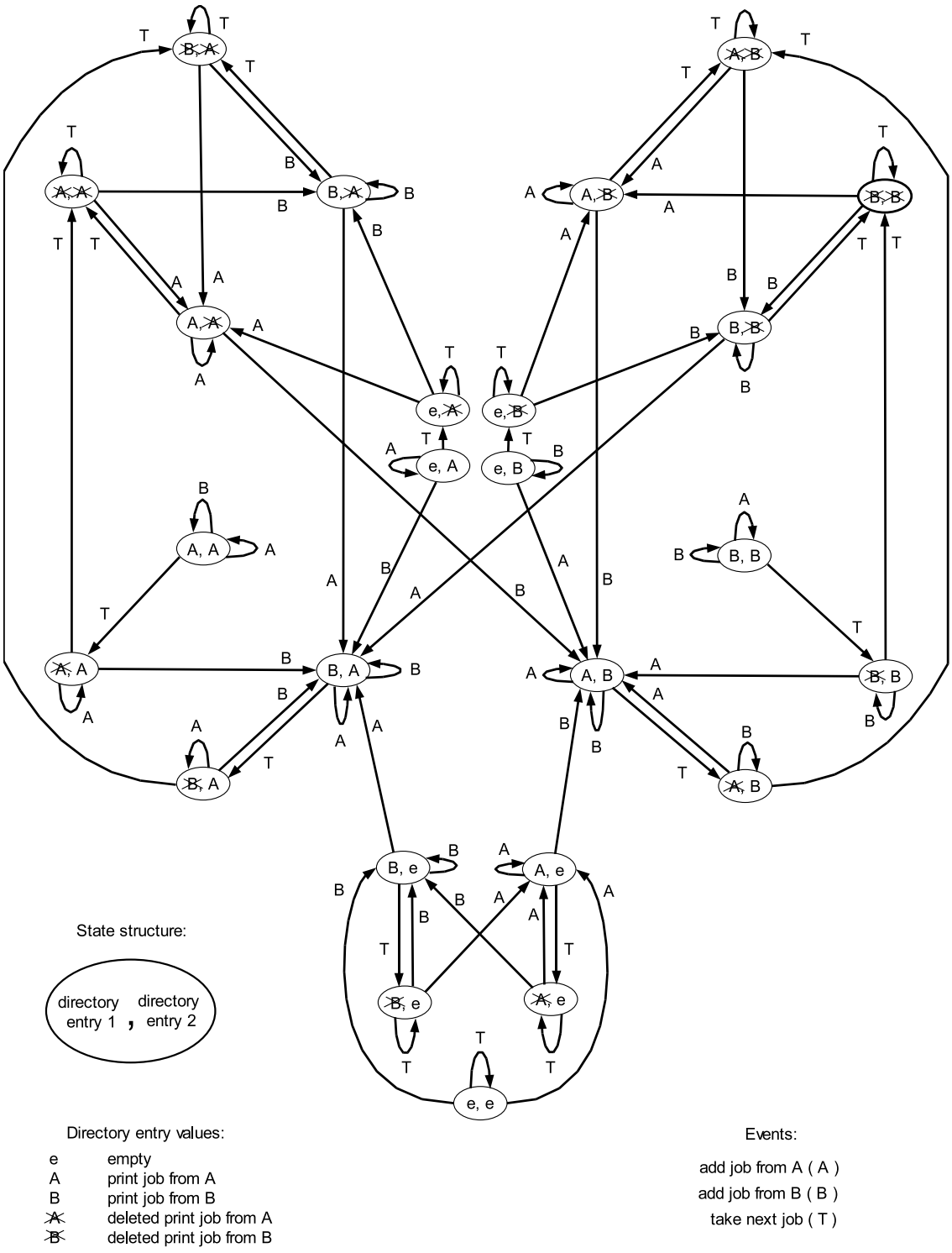

Figure 6.2: Transition graph of the print job directory model

discovered by Carl corresponds to the ellipse (⊠,⊠). Any possible scenario of the incident corresponds to a path from (e,e) to (⊠,⊠). All such scenarios can be found by backtracing transitions leading into (⊠,⊠), or equivalently, by forward-tracing transitions from (e,e).

The Alice's claim that she never printed anything corresponds to a path from (e,e) to (⊠,⊠) that does not have states with "A" in them. By forward-tracing transition from (e,e), one can ensure that any path from (e,e) to (⊠,⊠) has to go through the (A,B) state, which means that Alice is lying.

6.1.3 Evidential statements

As illustrated by the foregoing example, finite state machines can be used as a basis for automatic event reconstruction. However, finite state machines alone are insufficient to completely automate the event reconstruction process. It is also necessary to formalise the available evidence, such as Carl's observation of the final state of the print job directory. So in addition to using state machines to model system functionality, this dissertation defines the *evidential statement* notation for describing the evidence about an incident.

The idea of evidential statements is to formalise pieces of evidence as statements about the properties and change of system state in the past.

Consider, for example, Carl's observation of the print job directory. The knowledge of the incident that he obtained *directly* from the examination of the printer can be described as follows:

1. the state of the print job directory at the moment of examination was (⊠, ⊠)

2. before the print job directory reached that state, it visited zero or more states about which nothing is evident from the examination (it could have been any states).

Similarly, the manufacturer's knowledge of the initial state of the print job directory can be described as follows:

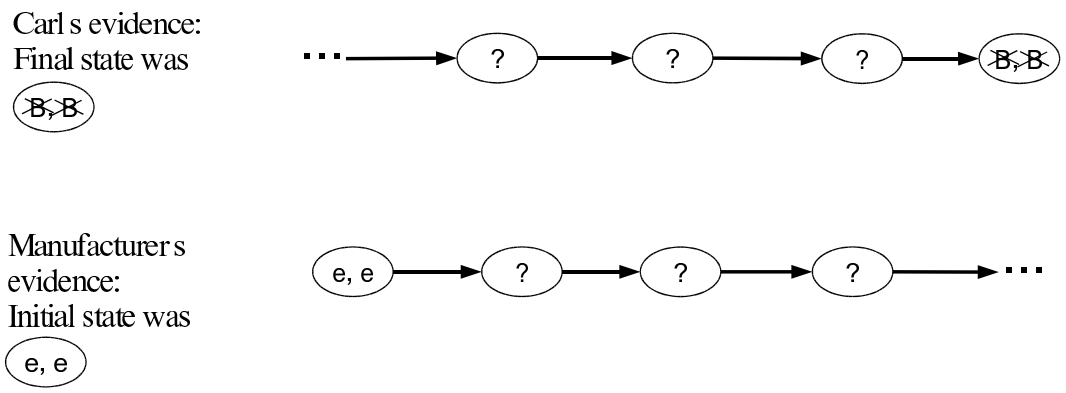

Figure 6.3: Evidence in ACME investigation

1. the initial state of the print job directory was (e, e),

2. after that, the print job directory could have visited zero or more states, but the manufacturer knows nothing about those states.

Both descriptions are illustrated graphically in Figure 6.3. Observe that, essentially, both pieces of evidence restrict possible sequences of transitions that could have occurred during the incident. As a result, event reconstruction can be viewed as the process of finding all sequences of transitions that satisfy these restrictions.

Motivation for the development of new formal notation

Checking that computations of a given state machine satisfy given set of restrictions is the basic problem of model checking. Since both digital forensics and model checking are concerned with the analysis of discrete digital systems, it may seem feasible to use existing formal verification methods in digital investigations. Additional argument for using these methods is that the goals of *some* investigations can be formulated as verification problems (in ACME investigation, for example, the goal is to verify that Alice never printed anything). There are however, considerable problems with the use of existing formal verification methods in forensic context:

1. *Insufficiency of explanations.* The output of formal verification is basically a "true" or "false" answer with respect to the given logical formula. Although model checking tools do provide a counterexample if the formula is false, little other information is given.

   At the same time, forensic analysis of evidence is expected to produce more than a simple "yes" or "no" answer. When preparing for the court hearing, for example, attorneys may want to know about alternative explanations of the available evidence. At the trial, the expert may be challenged to give a comprehensive explanation of how the particular piece of evidence fits with the facts in issue.

   As a result, formal methods in digital forensics should provide more informative explanation of how possible scenarios are linked to the evidence.

2. *The absence of the notion of evidence from formal verification.* The concept of evidence is fundamental in the legal context. Apart from restricting possible sequences of transitions, a piece of evidence can have its own properties related to its discovery. For example, an eyewitness observation may have real-world time associated with it, which may be used to perform event time bounding analysis described in Section 3.2.3. The existing formal verification methods do not provide explicit ways to represent evidence and to reason about its properties.

As a result of the aforementioned problems, it has been decided to create a new formal notation rather than use an existing one. The new notation provides explicit representation for evidence, and defines the basic analytical problem as finding the set of all possible explanations of the given evidence.

6.1.4 Assumption about reliability of evidence

The importance of evidence reliability has been highlighted in the previous section and in Chapter 2. However, due to limited timeframe of this research

work, it has been decided not to address the issue of evidence reliability in this research. Throughout the rest of this dissertation, it is implicitly assumed that all evidence is absolutely reliable. This assumption simplifies the problem of event reconstruction by avoiding reasoning with uncertainty. *This simplification is justified, because the evidence may be incorporated into analysis gradually. First the investigator can perform formal analysis with only the most reliable evidence. If results are unsatisfactory, the analysis can be repeated with less reliable evidence included.*

6.2 Formalisation of event reconstruction problem

This section formally defines the event reconstruction problem. The definition is based on the idea that the knowledge used by forensic expert to reconstruct past events in a digital system can be divided into two categories:

- Knowledge of the system functionality — the expert knowledge
- Evidence — description of the system's final state and clues to the system's behaviour in the past, such as witness statements, printouts, etc.

The proposed definition represents the knowledge of the system functionality as a finite state machine and uses *evidential statement* notation for describing the evidence and investigative assumptions. The event reconstruction is defined as finding all possible explanations for the given evidential statement with respect to the given finite state machine. Appendix C.1 contains formalisation of the evidential statement notations and related notions in ACL2 logic.

6.2.1 Finite state machine

Finite state machine is a tuple of four elements $T = (Q, I, \phi)$, where

- I is a finite set of all possible events,
- Q is a finite set of all possible states,

- $\phi : I \times Q \to Q$ is a transition function that determines the next state for every possible combination of event and state.

Transition is the process of state change. Transitions are instantaneous.

A *(finite) computation* is a non-empty, finite sequence of steps, where each step is a pair $c_j = (c_j^\iota, c_j^q)$, where $c_j^\iota \in I$ is event, $c_j^q \in Q$ is a state, and any two steps c_k and c_{k-1} are related via transition function:

$$\text{for all } k, \text{ such that } 1 \leq k < |c|, \quad c_k^q = \phi(c_{k-1}^\iota, c_{k-1}^q)$$

The *set of all finite computations* of the finite state machine T is denoted C_T.

Observe that, since there is no upper bound on the possible length of a computation, C_T is infinite.

6.2.2 Run

To formalise transition backtracing, the concept of run is defined. A run is a possibly empty sequence of finite computations, in which the next computation is obtained from the previous computation by discarding its first element. Please look at Figure 6.4, which graphically illustrates this concept.

A *run* is a sequence of computations $r \in (C_T)^{|r|}$, such that if r is non-empty, its first element is a computation $r_0 \in C_T$, and for all integer $1 \leq i < |r|$, $r_i = \psi(r_{i-1})$, where function ψ discards the first element of the given computation.

For two computations $x \in C_T$ and $y \in C_T$, $y = \psi(x)$ if and only if $x = x_0 \cdot y$.

The *set of all runs* of the finite state machine T is denoted R_T.

The *run of computation* c is a run whose first computation is c.

Observe that any run r is completely determined by its length and its first computation.

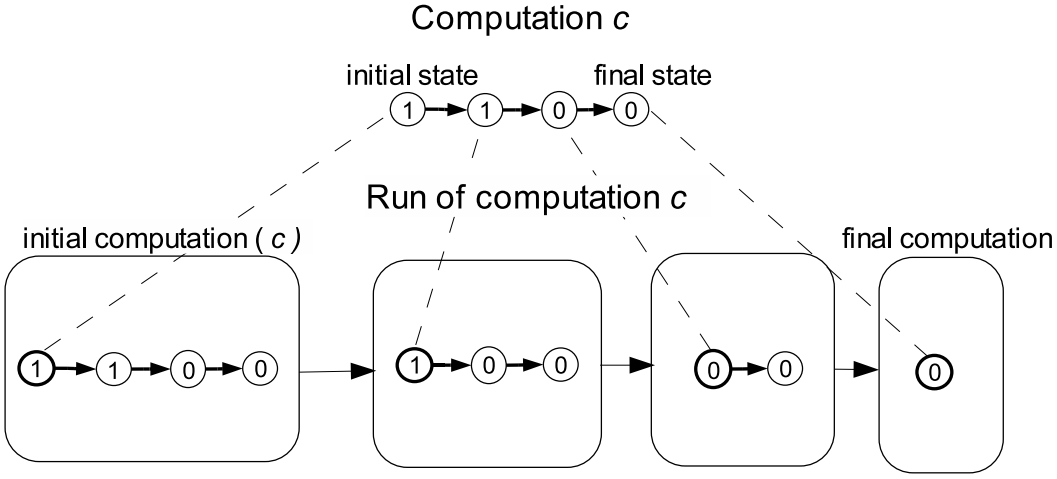

Figure 6.4: Run of computation

6.2.3 Partitioned run

Partitioned run is a finite sequence of runs $pr \in (R_T)^{|pr|}$, such that concatenation of its elements in the order of listing is also a run:

$$(pr_0 \cdot pr_1 \cdot pr_2 \cdot \ldots \cdot pr_{|pr|-1}) \in R_T$$

The *set of all partitioned runs* of the finite state machine T is denoted PR_T.

A *partitioning* of run r is a partitioned run denoted pr_r, such that concatenation of its elements produces r:

$$(pr_{r\,0} \cdot pr_{r\,1} \cdot pr_{r\,2} \cdot \ldots \cdot pr_{r\,|pr|-1}) = r$$

6.2.4 Formalisation of backtracing

The inverse of ψ is function $\psi^{-1} : C_T \to 2^{C_T}$. For any computation $y \in C_T$, it identifies a subset of computations whose tails are y:

$$\text{for all } x \in \psi^{-1}\ ,\ y = \psi(x)$$

In other words, ψ^{-1} backtraces the given computation.

Although function ψ^{-1} can be used to formalise backtracing, it is inconvenient, because it takes a single computation and produces a set of computations. As a result, it cannot be applied to its own output. A more convenient alternative is function $\Psi^{-1} : 2^{C_T} \to 2^{C_T}$, which is applied to a set of computations:

$$\text{for } Y \subseteq C_T,\ \Psi^{-1}(Y) = \bigcup_{y \in Y} \psi^{-1}(y)$$

The meaning of functions ψ, ψ^{-1}, and Ψ^{-1} is illustrated in Figure 6.5.

Backtracing of computations is defined as a finite number of compositions Ψ^{-1} applied to a subset of computations:

$$\Psi^{-1}(\Psi^{-1}(\ldots \Psi^{-1}(Y)\ldots))$$

Additional convenience of function Ψ^{-1} is that its software implementation can manipulate implicit symbolic descriptions of computation sets, whereas implementation of ψ^{-1} requires explicit representation of computations.

6.2.5 Formalisation of evidence

In a way, every piece of evidence tells its own "story" of the incident. The aim of event reconstruction can be seen as combining stories told by witnesses and by various pieces of evidence to make the description of the incident as precise as possible. This story-oriented view of event reconstruction is the basis of evidence formalisation presented below.

Observation

Observation is a statement that system behaviour exhibited some property *p continuously for some time*. Syntactically, observation is a triple $o = (P, min, opt)$, where P is the set of all computations of T that possess observed property, *min* and *opt* are non-negative integers that restrict duration of observation. Informally speaking, observation o characterises a set of runs

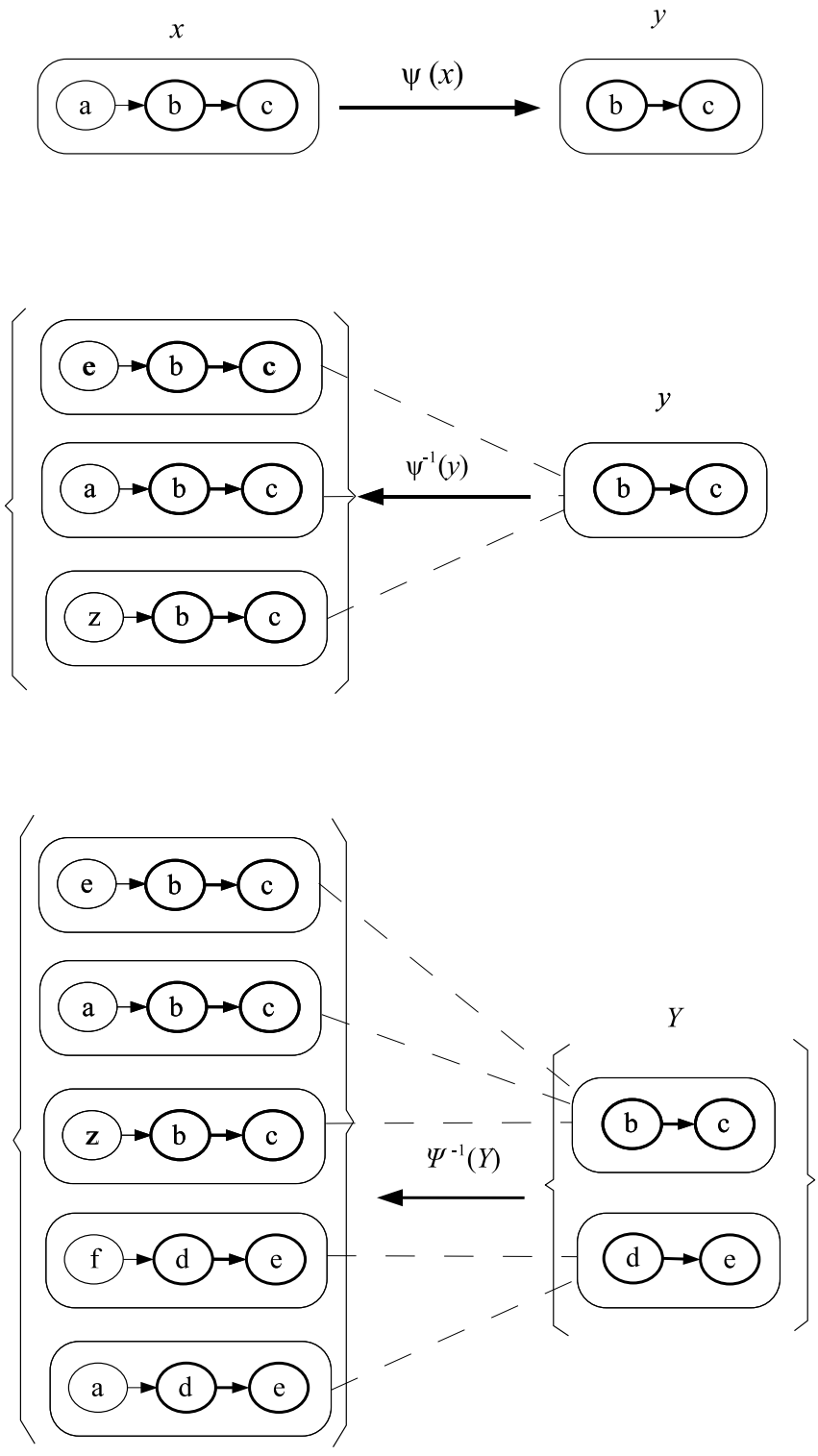

Figure 6.5: Functions ψ, ψ^{-1}, and Ψ^{-1}

R_o, whose lengths are limited by *min* and *opt* and whose computations satisfy P.

Since witness observations are external to the system, transitions that do not change observed property of the state are invisible to the witness. It means that, in general, a sequence of states could have been observed rather than a single state. Elements *min* and *opt* restrict the length of runs comprising R_o. Element *min* specifies the minimal length of runs in $r \in R_o$, and *opt* specifies maximal "excess" of length in addition to *min*:

$$min \leq |r| \leq (min + opt)$$

An explanation of observation o is a run $r \in R$, such that every element of run r possesses observed property: for all $0 \leq i < |r|,\ r_i \in P$, and the length of run r satisfies *min* and *opt*: $min \leq |r| \leq (min + opt)$.

The *meaning* of observation o is the set $R_o \subseteq R_T$ of all runs that explain o.

A note on *min* and *opt*. The introduction of restrictions on the length of observations is motivated by the following reason. Although the witness may not always tell how many transitions have been observed, it is sometimes possible to set a limit. Consider Carl's observation of the print job directory. The length of run is at least one, because the deleted print jobs from Bob were actually observed. Moreover, the length of runs corresponding to the final state observation is exactly one, because there were no more transitions from the final state.

The upper limit on the length of observation is introduced to ensure termination of reconstruction process. The introduction of the upper limit is permissible, because digital forensic analysis reconstructs only final computations. Therefore, introduction of a sufficiently large upper limit can be used to model infinity. The constant *infinitum* is introduced for this purpose:

The *infinitum* is an integer constant that is greater than the length of any computation that may have happened during the incident.

Types of observations. Observations can be divided into several types:

- *Fixed length observation* is observation of the form $(P, x, 0)$. Any run explaining it has length x.
- *Zero-observation* is observation of the form $(P, 0, 0)$. The only run explaining it is the empty sequence ε.
- *No-observation* is observation $(C_T, 0, \textit{infinitum})$ that puts no restrictions on computations that could have happened during the incident.

Observation sequence

An *observation sequence* is a non-empty sequence of observations listed in chronological order:

$$os = (\ observation_A,\ observation_B,\ observation_C,\ \ldots)$$

Informally, an observation sequence represents uninterrupted eyewitness story. The next observation in the sequence begins immediately when the previous observation finishes. Gaps in the story are represented by no-observations.

An *explanation of observation sequence os* is a partitioned run *pr* such that the length of *pr* is equal to the length of *os*:

$$|pr| = |os|$$

and each element of *pr* explains the corresponding observation of *os*:

$$\text{for all } 0 \leq i < |os|, \quad pr_i \in R_{os_i}$$

Note that the same run can explain the same observation sequence in a number

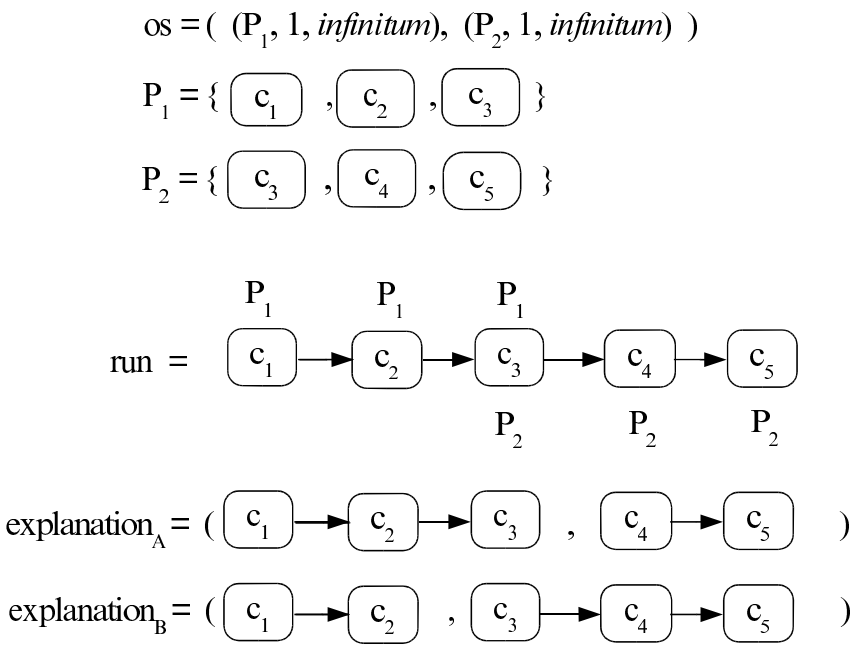

Figure 6.6: A run that gives two explanations to an observation sequence

of ways, each corresponding to a different partitioning of the run. Figure 6.6 illustrates this possibility of multiple explanations.

The *meaning of observation sequence* os is the set $PR_{os} \subseteq (R_T)^{|os|}$ of *all* partitioned runs that explain os.

A run r *satisfies* an observation sequence os if and only if there exists a partitioning of r that explains os. There may be more than one partitioning of r that explains os.

A computation c *satisfies* an observation sequence os if and only if there is a run r that satisfies os and $r_0 = c$.

Evidential statement

Evidential statement is a non-empty sequence of observation sequences

$$es = (os_A, os_B, os_C, \ldots)$$

Evidential statement combines restrictions imposed by all of its observation sequences – a computation satisfying one observation sequence must also satisfy all other observation sequences in the evidential statement.

An *explanation of evidential statement es* is a sequence of partitioned runs *spr*, such that all elements of *spr* are partitionings of the same run:

$$\begin{aligned} & spr_{00} \cdot spr_{01} \cdot \ldots \cdot spr_{0|spr_0|-1} = \\ = & \; spr_{10} \cdot spr_{11} \cdot \ldots \cdot spr_{1|spr_1|-1} = \\ & \vdots \\ = & \; spr_{|es|-1_0} \cdot spr_{|es|-1_1} \cdot \ldots \cdot spr_{|es|-1_{|spr_{|es|-1}|-1}} = r \end{aligned}$$

and the length of *spr* is equal to the length of *es*:

$$|spr| = |es|$$

and each element of *spr* explains corresponding observation sequence of *es*:

$$\text{for all } 0 \leq i < |es|, \quad spr_i \in PR_{es_i}$$

The *meaning of evidential statement es* is the set of all sequences of partitioned runs $SPR_{es} \subseteq (PR_{es_0} \times PR_{es_1} \times \ldots \times PR_{es_{|es|-1}})$ that explain *es*.

Evidential statement is *inconsistent* if it has empty set of explanations: $SPR_{es} = \emptyset$.

Figure 6.7 illustrates the relationship between the evidential statement and other formal notions introduced in this section.

Definition of event reconstruction problem

In terms of the above defined formalisation of evidence, event reconstruction problem is defined as *calculating the meaning* SPR_{es} *of the given evidential statement es with respect to the given finite state machine* T.

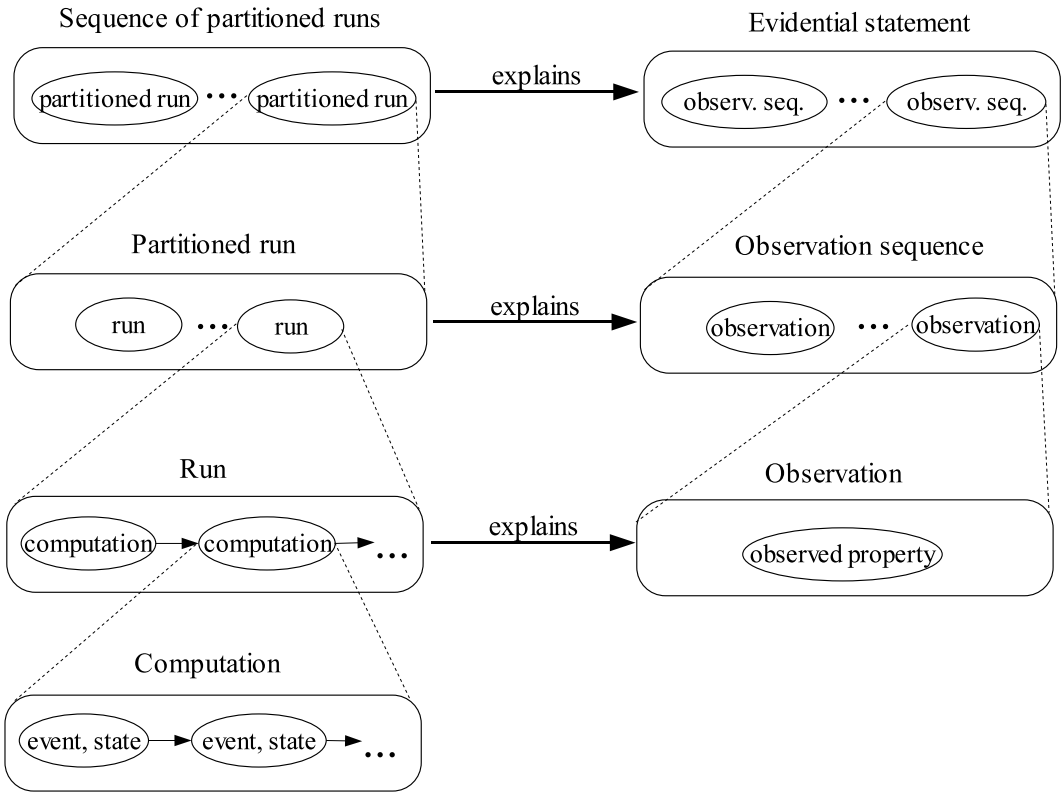

Figure 6.7: Evidential statement and related notions

A note on the use of *infinitum* The *infinitum* constant is designed as a "no-limit" constant for the *opt* parameter of observations. It is possible, therefore, that several observations in an observation sequence will have *infinitum* as their *opt* parameter. Such observation sequence may have explanations whose lengths are several times longer than *infinitum*, because each "no-limit" observation may have explaining run, whose length is *infinitum*.

By definition given in Section 6.2.5, *infinitum* is greater than any computation than might have happened during the incident. As a result, there is no practical reason to calculate the entire SPR_{es}. If *infinitum* is used and is chosen correctly, it should suffice to calculate only a part of SPR_{es} that includes all explanations of es, whose total length is less then *infinitum*.

6.3 Summary

This chapter has demonstrated that event reconstruction in digital investigations can be formalised using state machine model of computation. The following approach to event reconstruction has been proposed.

1. Create a finite state model of the system under investigation and formalise the evidence in terms of that model.

2. Use transition backtracing or any other suitable method to find all sequences of transitions that agree with the formalised evidence.

This chapter was primarily concerned with the step one of this approach. To allow formalisation of evidence, it defined evidential statement notation. The problem of event reconstruction has been defined as finding all possible explanations of the given evidential statement with respect to the given finite state machine.

The next chapter addresses the second step of the proposed approach to event reconstruction. It presents an algorithm that computes the meaning of

the given evidential statement. It also analyses the complexity of the algorithm and describes a "proof-of-concept" implementation of the algorithm in Common Lisp.