Svein Yngvar Willassen

Methods for Enhancement of Timestamp Evidence in Digital Investigations

Doctoral thesis
for the degree philosophiae doctor

Trondheim, January 2008

Norwegian University of Science and Technology
Faculty of Information Technology,
Mathematics and Electrical Engineering
Department of Telematics

NTNU

NTNU
Norwegian University of
Science and Technology

Doctoral thesis
for the degree philosophiae doctor

Faculty of Information Technology,
Mathematics and Electrical Engineering
Department of Telematics

ISBN 978-82-471-6227-9 (printed version)
ISBN 978-82-471-6230-9 (electronic version)
ISSN 1503-8181

Doctoral theses at NTNU, 2008:19

Printed by NTNU-trykk

ACKNOWLEDGMENTS

I would first like to thank my advisor, Stig Frode Mjølsnes. His guidance and helpful feedback has helped shape and direct this work. I would also like to thank the others at Department of Telematics for many discussions providing valuable inspiration. Special thanks to Kristian Gjøsteen for mathematical insights and fruitful discussions about digital forensics and digital security. Thanks to Tore Amble at the Department of Computer and Information Science for introducing me to Shanahan's Event Calculus. Also thanks to Eugene Spafford at Purdue University and Pavel Gladyshev at University College Dublin. The discussions we had during our meetings in Norway as well as during my visits to West Lafayette and Dublin were most inspiring.

Thanks to the many people in the Digital Forensics community in Norway. Special thanks to my previous colleagues in Økokrim, Inger Marie Sunde, Steffen Thorkildsen and Lars Wilberg and to my previous colleagues in Ibas AS. Their work and support has been most inspiring for me. Also great thanks to the anonymous participants in the antedating experiment for supporting this work with their valuable time.

I would also like to thank my family; my wife Ingebjørg and my three children Kristian, Johannes and Sigrid. Their support enabled me to go back to research after working in the public and private sector, and provided me with the necessary time and energy to complete this work.

This work was performed under the project *Timestamps in Digital Forensics*, funded by the Norwegian Research Council under project number 164378/V30. I am deeply grateful for the financial support of the Norwegian government that enabled this work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

This work explores how the evidential value of digital timestamps can be enhanced by taking a hypothesis based approach to the investigation of digital timestamps. It defines the concepts of clock hypotheses, timestamps and causality in digital systems. These concepts are utilized to develop methods that can be used in an investigation to test a clock hypothesis for consistency with timestamps found in an actual investigation, given causality between specific events occurring in the investigated system. Common storage systems are explored for the identification of causality between the events of information storage. By using a logic programming variant of predicate calculus, a formalism for modelling the relationship between events and timestamp updating is defined. This formalism can be used to determine invariants in digital systems.

Invariants and causality relations can be used to check a clock hypothesis for consistency with timestamp evidence. These methods can be utilized in software for digital investigation. By checking the large number of timestamps typically occurring on a digital medium, the methods can assist with the justification of a clock hypothesis, and thereby increase the confidence in specific timestamps found during the investigation. Previously, the checking of timestamps has relied upon the existence of timestamps from other evidence sources. With the methods defined in this work, justification of timestamp interpretation can be achieved without having to rely on timestamps from other sources of evidence.

The methods developed in this work were implemented in a clock hypothesis consistency checker. This checker was tested in an experiment where subjects were asked to antedate a document. The checker was found to be able to produce evidence supporting a hypothesis that the document was antedated.

1 INTRODUCTION

This chapter provides an introduction to timestamps, their use in digital investigation and challenges associated with such use. Section 1.1 describes timestamps and how they are created. Section 1.2 introduces the digital investigation process. Section 1.3 provides an introduction to the use of timestamps in digital investigation. Section 1.4 and 1.5 discuss possible sources for error and uncertainty in timestamps and how they affect digital investigations. Section 1.6 presents the thesis statement of this work. Section 1.7 presents existing related work. The discussion in this chapter serves as background material for the remainder of this work.

1.1 Timestamps and their origin

A timestamp is a recorded representation of a specific moment in time. In digital computing, a timestamp is a recorded representation of a specific moment in time in a digital format. This representation is either stored on a medium storing digital data, or transmitted on a network designed to convey digital data. Timestamps are generated in computers as the result of executed code in the processes running on a processor. The running program creates a timestamp by obtaining the value currently assigned to the local clock of the processor and storing it in memory. The value may subsequently be stored on non-volatile memory (such as a hard drive) or it may be included in data transmitted on a network. When a timestamp is generated, it is possible to associate execution of the code immediately before and after the timestamp was generated with the point in time the timestamp represents. Identification of the time when the enclosing code was executed is usually the purpose of the timestamp itself. The generation of the timestamp makes it possible to identify when the code ran, and therefore when the result of that code took effect. If the timestamp is stored on a non-volatile medium or transmitted and stored on another computer together with information that identifies the executed code, the executed code can be associated with the timestamp. Execution of specific code can be said to constitute an *event*. The timestamp associates the event with a specific point in time. The event can be said to have *generated the timestamp*. Although the code that constitutes the event itself is not the code that copies the

processor clock, the term reflects the cause of the generation of the timestamp; to assign a time to the event. The event that generated the timestamp is the *generating event*.

When stored on a non-volatile medium, timestamps are typically stored in such a way that the generating event is clearly identified. Current computer systems store large numbers of timestamps on their storage. Some of the most common sources of timestamps are:

- File systems: Typically each file on a file system has several associated timestamps, related to user actions on the files. For example, timestamps are commonly stored that indicates when each file was last read, written or otherwise changed by a user process.
- Logs: System logging facilities usually log events from system processes in system logs. Each event has a timestamp. Network servers such as http, smtp, pop, imap and dhcp servers typically log each user transaction in a system log.
- Email: SMTP mandates each smtp server to add its identity and a timestamp to transmitted email. Email messages therefore contain many timestamps. Other messaging protocols, such as SMS in GSM, also add server generated timestamps to each transmitted message. [1]
- Application specific files: Many mainstream applications such as word processors, spreadsheets and web browsers generate timestamps and store them as part of their specific file format. If the format is known, the timestamps and events that generated them can be identified.

Timestamps can be stored in different formats. *Scope* and *resolution* are inherent properties of timestamp formats. With scope is meant the timescale coverage of the format, in other words which specific time periods the format is able to represent. For example, the 32-bit UNIX `time_t` format (see Appendix A.4) has the scope of [1901, 2038], meaning that it is only able to represent specific moments in time in the period between year 1901 and 2038 AD. The resolution of a timestamp format refers to the ability of representing two different moments close in time. Timestamps representing moments closer in time than the resolution of the format are not distinguishable. For example, the resolution of the `time_t` format is one second. It is therefore not possible to reliably distinguish two different timestamps in the `time_t` format if they represent moments in time closer together than one second. In addition to the representation of a

specific moment in time, a timestamp may or may not represent additional information. Some timestamps include information about the time zone in which the timestamp is valid. Such information is necessary in order to be able to compare timestamps generated in different time zones.

Another important notion when dealing with timestamps is accuracy. The accuracy of a timestamp reflects how close in time the timestamp represent the actual time when the event occurred. The accuracy may be limited by a number of factors. Resolution is an important limiting factor of accuracy. The resolution of the computer clock, the timestamp format in computer memory and the timestamp format in non-volatile storage may differ. Therefore the accuracy may not be equal to the resolution of the stored timestamp. For example, the timestamp format used in the NTFS file system has a resolution of 100 ns (see Appendix A.1), but the resolution of the Windows internal clock is 1 ms, therefore the accuracy of timestamps in NTFS on Windows systems cannot be better than 1 ms. Accuracy in timestamps also reflect the time difference that may occur between the occurrence of events and the generation of the timestamps they are stamped with.

1.2 The digital investigation process

Investigations are inquiries into past events. The purpose of an investigation is to find evidence that can establish an understanding of previous events. During an investigation, evidence is examined in order to produce information about past events. Possible sources of evidence include witness statements, documents, physical evidence (i.e. fingerprints or biological evidence) and data stored on digital media. From examination of the evidence, information about the past events can be reconstructed. Event reconstruction is the final outcome of an investigation, and forms the basis for a decision. The final event reconstruction relies on interpretation of the evidence and is usually performed by a person or group of persons separate from those performing the investigation. This person or group is called the *finder of fact*, and could be a judge, magistrate, jury or other depending on the case and jurisdiction. During the investigation, the investigator formulates theories about possible events in order to find further sources of evidence, prepare the case for the correct jurisdiction and present the evidence to the fact finder in an appropriate manner.

Investigation of digital media with the purpose of finding evidence is commonly referred to as *digital investigation*. The purpose of digital investigation is to find evidence related to the events under investigation and present them to the fact finder. The process of investigating digital media has in previous works been divided into different phases. The Electronic Crime Scene Guide divides the digital investigation process into the following phases: Preparation, Collection, Examination, Analysis and Reporting. [2] Here, the collection phase involves identifying digital media potentially containing evidence and collecting them physically or by making a digital copy of their contents. Examination involves searching the collected data for evidence and analysis involves reviewing the examination results for their value in the case. Reporting involves presenting evidence in a form acceptable to the fact finder. Carrier and Spafford have proposed using a model similar to the model used in physical crime scene investigation. [3] In this model, the physical crime scene investigation is divided into Preservation, Survey, Documentation, Search, Reconstruction and Presentation of Theory. The digital crime scene investigation takes place in the Search and Reconstruction phase of the physical process. During these steps, any digital media are found and analyzed. The model proposes similar steps in the digital investigation process: Preservation, Survey, Documentation, Search, Reconstruction and Presentation of Theory. Preservation involves isolating the digital medium and making a forensic image backup of it. Survey, Documentation, Search and Collection involves finding the relevant evidence on the digital medium and preserving it. Reconstruction and Presentation involves performing a preliminary event reconstruction and present the resulting theory to the fact finder.

Both models reflect the steps actually taken by investigators in digital investigations. First, digital media must be found and enumerated. Then, data on it must be preserved in order to secure the evidential integrity. This usually involves copying the data on the medium to another medium in such a way that no data is changed on the original. The data on the copy can then be analyzed for contents relevant as evidence. Due to the large amounts of data stored on modern data storage devices, the search is usually performed by a combination of manual and automatic search. There exist a number of helpful techniques employed by investigators in this phase such as keyword search, hashing and signature search. When any relevant data has been found, it must be documented, usually in the form of a report. Finally, the data is presented to the fact finder, either as printouts, or by having the investigator appear before the fact finder to report the

findings. Although final event reconstruction is up to the fact finder, one should bear in mind that the fact finder usually has little expertise in digital computing. The investigator is often asked to present his theory on how the presented evidence can be used to reconstruct the investigated events.

A formalization of the digital investigation process has been proposed by Carrier, by introducing the concept of an object history. [4] For a computer, the history includes the complete set of configurations, states and events that has occurred during the lifetime of the computer. A state is the sum of all variables that may occur in the computer, whereas an event is any action that may change the state. In this model, the digital investigation process is defined as formulating hypotheses about the history of the computer, and testing them against known values such as known user-input, data from other evidence sources and the final state of the system. With this model, the assumptions on which the event reconstruction is based are more explicit. This makes it possible for the investigator and the fact finder to assess the assumptions and decide if they are justified or not.

1.3 The role of timestamps in digital investigations

Timestamps play an important role in digital investigations. Traditionally, they are used to place the generating event at a specific moment in time, thereby facilitating event reconstruction. The identification that a certain event on a computer took place at a specific time makes it possible to correlate the event with events occurring outside the computer system. A witness statement that a certain computer was used at a particular time might for example be strengthened if the timestamps on the computer shows evidence of usage by the particular user at that time. It is often necessary to use timestamps to identify events occurring on different digital computers within the same timeframe. If a computer intrusion was committed from one computer to another, it would be necessary to examine the timestamps of the victim computer to find the exact time of the intrusion. This information can then be used to examine the activity on the computer where the attack originated.

A particularly important application of timestamps in digital investigation is attribution. Attribution is the ability to attribute events to a specific person. This is important,

because most investigations aim at placing the responsibility for occurred events on one or more individuals. If evidence of the investigated events is digital, it may be necessary to place the event at a specific point in time in order to be able to attribute it to the correct person. If the time of the event inferred from the evidence is incorrect, it may not be possible to attribute it to anyone, or the event may be attributed to the wrong person. The prevalence of dynamic network addresses on the Internet makes timing important in all types of investigations of events that occurred on the Internet. In many such investigations, attribution relies on the identification of which computer were using an IP-address at a particular time. If the IP-address is dynamically assigned, the originating computer can only be identified if a log of the usage of the address exists, and the time of the event can be established with sufficient certainty and accuracy. Only in this case can the originating computer be identified from the usage log by selecting the correct entry from the IP-address and time. When the originating computer has been identified, it can be examined for corroborating evidence.

In many investigations placing the investigated events at a specific moment in time is also important in attributing usage of a computer to the correct person. For example, finding contraband images on an office workstation does not necessarily imply that the current user of the workstation placed them there, or even knew of their existence. In order to attribute storing of the images to the current user, it is necessary to find evidence that links the user and the contraband. The time of the storage event as correlated with the time the user has been known to use the workstation is one possible link.

1.4 Sources of error and uncertainty in timestamps

For a number of reasons, stored timestamps may not accurately reflect the time of the generating event. As already mentioned, a timestamp may have an inherent inaccuracy caused by the resolution of the timestamp format, and the time between the actual event occurring and the generation of a timestamp. This inaccuracy may sometimes be large enough to give the timestamp limited value in an investigation. For example, the resolution of the Last Read timestamp in the FAT file system has a resolution of one day, meaning that it stores only the date and not the time associated with the event.

A timestamp always refers to the clock from which it is generated. Since the timestamp is a function of the clock, it is always relative to the adjustment of the clock. Unfortunately, clocks are not fully reliable. Clocks may drift, thereby generating timestamps gradually more different from those generated from other clocks. Clocks may also fail, and produce completely incorrect timestamps. Further, clocks on most systems may be adjusted at any time by the user of the system to show a different date and time than civil time (standard time in the jurisdiction in question). Such adjustment may be committed intentionally or unintentionally by the user. In the case of intentional adjustment, it may be done with the purpose of producing deviating timestamps (malicious adjustment), or it may be done for other reasons, such as sustaining the function of software programmed to work only a limited time. The consequence is that a timestamp is relative not only to the clock it was generated from in general, but also to the particular adjustment of the clock at the time the timestamp was generated. Therefore, even timestamps generated from the same clock cannot be reliably compared unless it can be justified that the adjustment of the clock is unchanged between creations of timestamps. In order to reliably compare timestamps from different clocks, the difference between the clocks must be found, and it must be justified for all clocks that their adjustment has not changed.

Another consideration when assessing the correctness of a timestamp is events that took place after the timestamp was stored. A timestamp is just another piece of data stored on the storage of the computer. This means that unless the timestamp has been stored on a read-only medium, it can be changed by subsequent events on the computer. The timestamp may for example be deleted and overwritten by other data. It may also be changed to represent a moment in time earlier or later than it originally represented. Such changes may be intentional or unintentional on the user's part and in the case of intentional changes, they may be committed with malicious intent or not. The ease, with which a timestamp may be changed, depends on the format and the storage location. Setting file timestamps to a different time on a UNIX system can for example be done easily with the `date` and `touch` commands. Changing timestamps in undocumented formats directly on disk may be significantly more difficult and may require specialized knowledge.

1.5 Implications of timestamp error and uncertainty in investigations

The uncertainty associated with digitally stored timestamps implies that timestamps in general should not be relied upon as evidence without justification of the factors that can lead to errors. In particular, it should not be blindly assumed that timestamps are based on a clock that is adjusted to civil time. Further, it cannot be assumed that timestamps generated by different clocks are relative to the same clock. Not even when timestamps are based on the same clock, can one be absolutely certain that the time difference between the two events is equal to the difference between the timestamps. These uncertainties are worrying for investigators. If timestamps cannot be relied upon, then it is in many cases not possible to reconstruct the events in the case reliably. In particular, it will not be possible to attribute the events under investigation to a particular individual. This is a major problem, since in many cases the sole purpose of the investigation is to be able to attribute events to an individual.

The current practice in digital investigations is to check the current time of the computer clock at the time of the investigation, and look for corroboration in the form of timestamps from other clocks. [5] The assumption is that if the current state of the clock is within reasonable synchronization with civil time, then that has been the case also earlier in the computer's history. Any timestamps from external sources confirm the assumption, but the assumption is also made if no timestamps from external sources are found. If the clock on the investigated computer is significantly different from civil time and no external time references are found, there seems to be different approaches. Some investigators merely add the observed offset to the value of the timestamps stored on the computer, assuming that the current offset has also been the offset during the computer history. Others conclude that no inferences about the computer's history can be drawn from observed timestamps when the clock differs from civil time at the time of the investigation.

A common investigator attitude is to take evidence at face value unless contested by the opposing party. In the context of timestamps, this attitude means that timestamps in most cases are interpreted as correct and equal to moments in civil time, without further consideration of possible sources of error and uncertainty. As long as the opposing party does not have the knowledge required to challenge this position, the result of this approach will be that the fact finder takes the investigator's interpretation of the

timestamps as fact. This is the result in many current investigations where timestamps are presented as evidence.

The practice of investigating timestamps and present them without any consideration of possible error sources is questionable. Most important, such a practice may lead to a flawed event reconstruction, which may in turn lead to incorrect results such as incorrect convictions or acquittals. This is most certainly the case in an area where the investigation is performed by specialists and where neither counsel (of any party), accused nor fact finders have the necessary expertise to identify the problem. As long as this practice continues, there will be no development of methods that can define hypotheses of potential uncertainty in timestamp evidence and test them. The assumption that timestamps can be trusted as evidence is left unjustified and open to a simple attack. Suppose that the opposing party actually has the knowledge to understand the uncertainties and potential errors in timestamps. The timestamp evidence can in such a situation be contested, for example with an allegation that the clock previously has been erroneous or maladjusted. If the investigator has no method for testing errors and uncertainty in this situation, it is possible that the timestamp evidence will be refuted.

From an engineering standpoint, a possible solution for the mentioned challenges would be to redesign computers to reduce problems with errors and uncertainty in timestamps as evidence. A possible approach would be to use clock synchronization to ensure that clocks on computers are synchronized with a universally recognized clock. Clock synchronization across computer networks has been studied extensively and can be said to be a well understood problem. There exist a number of recognized methods for clock synchronization as well as protocols for synchronizations of clocks across networks. [6, 7] It has also been proposed to change existing file systems, to allow systems to keep trails of previous timestamps for specific files, and not just overwrite timestamps when new timestamps are produced. This scheme was proposed in systems with synchronized clocks, to prevent malicious users from modifying timestamps directly without producing evidence of it. [8]

There are however several problems with this approach. The most fundamental problem is that in most investigations, the computer containing the evidence has been controlled by the individual under investigation during the period of the investigated events. It cannot be assumed that this person would be interested in keeping the computer clock

synchronized in case the computer should be investigated. And even if the computer clock was actually synchronized with an external clock, it could not be assumed that the computer had been synchronized during its full history. It would even be possible for the computer owner to falsify the synchronization by using an external computer acting as a synchronization proxy, thereby making it look like the synchronization was towards a universal clock, when this was in fact not the case.

Another possible solution from en engineering point of view would be to use digitally signed timestamps. In such a system, data would be sent to a trusted third party for timestamping using a trusted clock. The data would then, along with the timestamp be signed with the trusted thirds party's private key. Such a solution has been proposed for use in digital investigations to enhance the trust in correct handling of digital evidence. [9] The usefulness for the investigation itself is however doubtful. As with clock-synchronization, it is not very likely that a secure timestamp technology would be very widely adopted among computer owners in the case they should be investigated.

1.6 Timestamps, causality and invariants, thesis statement

New methods are required for digital investigation of timestamps and use of digitally stored timestamps as evidence. This work takes the approach that timestamp evidence can be tested in the hypothesis based approach suggested by Carrier. In this approach, the history of the medium under investigation is the complete set of configurations, states and events that has occurred during the lifetime of the medium. The data direct observable by the investigator is the final state of the medium. This includes observations of all timestamps stored on the medium. These timestamps are all functions of the computer clock at some previous state in the history, and any subsequent events that affect them. With this definition, it is possible to formulate hypotheses about the adjustment of the clock in previous states and events that affect timestamps. For example, the default investigator assumption in section 1.5 can be formulated as the following hypothesis: "The computer clock has been adjusted to civil time during the entire history of the computer. No subsequent events have changed the values of the timestamps." Having formulated a hypothesis, it can now be tested for consistency with the observed data. This work provides methods for such tests.

In this work, the concept of causality is used to test hypotheses about clocks and timestamps. Causality is a concept used to describe cause and effect. It can be defined as the relationship between two entities A and B, in which A is necessary for B. A and B could be objects, events, states or properties. In the context of digital investigation of timestamps it is useful to define causality as the relationship between events. Causality is then the relationship between two events, in which the first must have occurred in order for the second to occur. In digital computers, events and causality relations between them are defined by the hardware and software constituting the computer. Events change the state of the computer medium in the computer history model. Some events are timestamped and others are not. If the working of the hardware and software that constitutes the computer is known, then causality relationships between events can be inferred. These causality relationships can be used to test hypotheses about the computer clock and events affecting timestamps in the computer history model. For example, if two timestamped events are causally connected, the timestamps have not been changed by subsequent events, and the clock has not been adjusted between the two events, then the timestamp of the second event must be later than the timestamp of the first event. [10] By testing for this and other properties of causal connections between timestamped events in a computer system, the formulated hypothesis can be accepted or refuted.

Further, we explore how invariants can be found in systems containing timestamps. Condider for example a file system in which each file has three different timestamps. These timestamps can be updated by different system actions, where each action may update one or more time stamp. In such a system, there may be invariants. For example if every action updating the Modified timestamp also updates the Accessed timestamp, then the latter timestamp will always be set at the same or a later time than the Modified timestamp. As with causality, such invariants can be used to test a clock hypothesis and accept or refute it.

The thesis statement for this work is then:

The use of timestamps as digital evidence can be enhanced by testing if the observed timestamps are consistent with causality of the events they represent, and invariants in the systems in which they exist.

1.7 Related work

Being recognized as a research challenge, the problem of timestamp interpretation in digital investigation has been studied by a few researchers during recent years. Schatz et al demonstrated the problem of clock drift by observing clock synchronization on a network of computers in a small business. [11] The problem of clock drift and lack of synchronization was confirmed by Buchholz et al in a larger scale study of web server clocks. [12] Schatz suggests mitigating the problem by correlating the timestamps in web cache stored on the computer with records obtained from the web servers. Weil and Boyd et al suggest similar correlation methods, by using timestamps stored on the investigated computer coming from other clocks, such as timestamps in dynamically generated web pages. [5, 13] Such methods would provide correlation for the period for which cached data exist on the investigated computer only. These methods may be able to confirm or refute hypotheses about the clock in the period for which correlation material exists. They may not be able to provide reasonable evidence to refute a hypothesis that timestamps have been changed or the clock has been adjusted during the period for which no correlation material exist. Correlation with server records is only possible when such records actually exist, and the investigator has legal access to them.

Stevens studied clocks and described a clock model where each clock is described as the clock it was originally derived from plus the sum of all adjustments, errors and drift. [14] The clock model described by Stevens was refined by Buchholz, in the formalization of a clock model as the sum of clock drift (skew) and adjustments. [15] These models are versatile and provide good tools for event reconstruction in cases where clock adjustments, error and drift are known or measurable. They can also be used for the modelling of hypothesized clocks. They do not however by themselves assist in the identification of adjustment, error or drift from an observed final state.

Gladyshev studied the use of causality properties for establishing boundaries on period of time in which an event may have occurred. [16] In his approach, time bounding can be established when an event that occurred at an unknown or uncertain time is causally preceded and succeeded by events with known time occurrence. In order to perform time bounding, it is then required to know events of known time causally connected to the investigated events. When used to investigate a computer system, these events of known time must come from external sources. This approach differs from the approach taken in

this work, where no time references from external sources is assumed. The concept of causality is used in this work as well as in Gladyshev's. Although the happened-before relation is defined differently, its use to correlate timestamps bears resemblance.

Gladyshev's event time bounding is based on his finite state machine approach to Event Reconstruction. [17-19] In this work, an investigated system is modelled as a finite state machine, and event reconstruction is performed based on the possible transitions from state to state in the modelled state machine. More recently, the idea of representing a system as a state machine has been taken further by Gladyshev and Enbacka into the development of consistency criteria for a system specified as a state machine and the verification of those criteria using the B-method. [20] This idea bears resemblance with this work, in that it aims to model aspects of a system formally, derive invariants for a system, and test the available evidence for consistency with the invariants. The approach is however different. In Gladyshev and Enbacka's paper, the evidence is examined for internal inconsistencies. In this work, a hypothesis is formulated by the investigator, and the hypothesis is tested for consistency with the available evidence. Further, this work does not attempt to model a system as a finite state machine, but rather determines events in the system and the causal relations between them, as well as formulating parts of a system in predicate logic.

1.8 Outline

This dissertation is organized as follows. Chapter 2 presents a study of four real cases where timestamps have played a role. Chapter 3 describes the formal background for this work and the development of a theory for testing of clock hypothesis consistency with observed events and causal relations between them. In Chapter 4, this theory is applied by studying common types of storage systems and enumerating causal relations in them that can be applied for clock hypothesis consistency testing. Chapter 5 describes how a system can be modelled in predicate logic, and how such a model can be applied to derive invariants for timestamp evidence in the system. These invariants can be used to test clock hypotheses for consistency with timestamp evidence from a system. Chapter 6 describes a simplification of the formalism from Chapter 5, in which the actions in a system is described in an action table, and invariants are derived by simpler methods. In Chapter 7, a software implementation of the methods derived in Chapter 3-6 is

presented. The implementation was used to detect antedating in an experiment where subjects were asked to antedate a digital document in such a way that the antedating was not detectable by an investigator. Chapter 8 evaluates the methods and discusses several possible criticisms. Finally, Chapter 9 summarizes the findings. Appendices A and B gives an overview of common timestamp and clock formats. Appendix C summarizes results from timestamp updating experiments conducted in Windows XP, used as the basis for the implementation discussed in Chapter 7. Appendix D provides information about how to obtain and run the implementation. Appendix E provides copies of published papers with results of this work.

2 CASE STUDIES

Chapter 1 introduced timestamps and the challenges associated with them in digital investigations. In this chapter, a few examples from real investigations will be provided. These examples will be used in the following chapters in the discussion of problems that may be solved by the model proposed in this work. Section 2.1 presents a case where documents where antedated in order to keep assets from being seized in a bankruptcy. Section 2.2 presents a case where timestamps were faked in order to produce false alibi for a murder. Section 2.3 presents a case where a computer clock was manipulated in order to produce evidence in a contract dispute. Section 2.4 presents a set of cases where discussion timestamp consistency played an important role in judicial procedure.

2.1 Antedated documents

Dated documents play an important role in legal processes. Antedating means producing a document with a date that earlier than the date the document was actually produced. Digital investigation can play an important role in determining that antedating has occurred. An example of antedating can be found in the investigations of the Finance Credit case in Norway. [21]

The investigation of Finance Credit started in the fall of 2002. The business of Finance Credit was based on purchasing claims from customers at a value slightly lower than the claim value. To finance this business, the managers in Finance Credit had obtained loans from a syndicate of Norwegian banks for values around NOK 1 billion. (The amount is as of 2007 equivalent to more than 100 million Euro) These loans were secured by the total value of the claims Finance Credit had in its current portfolio. The banks had based their loans on an estimate of the value of the portfolio, derived from information given in the accounts of Finance Credit.

During the fall of 2002, the banks discovered that the total value of the current claim portfolio in Finance Credit was much lower than the values documented in the accounts. The difference was so high that it was likely that the banks had been purposefully misled

by false account figures. Most likely, the company had been run entirely on the loans given from the bank, and not from the revenues from its business.

Finance Credit and the managers Trond Kristoffersen and Torgeir Stensrud were reported to he Norwegian Police Economic Crime Unit, who promptly started an investigation of the matters. In November 2002, Kristoffersen and Stensrud were arrested and searches were performed at the premises of Finance Credit as well as the homes of Kristoffersen and Stensrud. Shortly after, bankruptcy was declared in Finance Credit. The investigation revealed that not only had the claim portfolio been vastly overvalued, but significant funds had also found its way from Finance Credit to bank accounts and property owned by Kristoffersen and Stensrud personally. On this basis, Kristoffersen and Stensrud were also declared bankrupt in early December 2002.

The part of the Finance Credit case of interest here is the transfer of shares in a UK private company Yaar Investment Ltd. Yaar owned a property in Norway, in which the Kristoffersen familiy lived. The value of the property was around NOK 14 million. (Equivalent to approximately 2 million Euro) During the registration of Kristoffersen's assets following his bankruptcy, Kristoffersen declared that the home was not owned by him, since he had transferred the shares in Yaar to his daughter in the year 2000. Yaar Investments and the property it owned was therefore not a part of his estate, and could not be used by his creditors to cover his debt. At this time, Kristoffersen was still imprisoned. He was however released on conditions on January 19th, 2003. In another search at Kristoffersen's home in May 2003, a computer was seized. In the computer, investigators found two documents "Transfer of Shares" and "Share Certificate", dated January 3rd, 2000. These documents detailed the transfer of all shares in Yaar to Kristoffersen's daughter. The content of the documents were in accordance with Kristoffersen's statement to the bankruptcy Receiver that the shares had been transferred in 2000, and therefore was not part of the estate. Investigators believed however that the document had been produced on the computer during early 2003 and not in 2000. Since the shares of Yaar had previously been owned by Kristoffersen, this would mean that the property would be a part of the estate after all, and could be seized by the Receiver. Further, it would mean that Kristoffersen had given false information to the Receiver during the registration, a felony under Norwegian law.

Based on the theory that the documents were antedated, investigators therefore put considerable effort in a digital investigation of the timestamps and events on the computer in order to find evidence of the antedating. No conclusive evidence was found. However, during the investigation other evidence was discovered that indicated that the documents were antedated. Several emails with the company handling the registration of the company indicated that the share transfer had occurred in 2003 and not in 2000. Further, other documents were recovered that indicated that the shares of Yaar had actually been owned by Kristoffersen in late 2000 and early 2001 and not by his daughter. Based on these documents, Kristoffersen was imprisoned again and was kept in custody until the trial in 2004. In the verdict, Kristoffersen was convicted on counts of giving false information to the Receiver of his estate, among many other counts in the charges. Kristoffersen received a sentence of nine years imprisonment and must pay more than NOK 1 billion in damages.

2.2 Timestamps as bogus alibi

Timestamped events are sometimes presented as alibi. An example of a situation where such an alibi was judged as bogus can be found in the investigation following the murder of Jennifer Myers on October 20th, 1997. [22-24]

Myers was found murdered in her art gallery in York, Pennsylvania. She had been shot three times in the chest, shoulder and the left eye. At the time, Kevin Brian Dowling, a citizen of York, was already charged with robbery and attempted rape of Myers that occurred on Aug 5th, 1996. The robbery charge was based on the description of Myers, and the fact that she had subsequently recognized him. Trial on the charges of robbery and attempted rape was set to begin two days after Myers was found dead. Based on the theory that Dowling had murdered Myers to prevent her from appearing as witness in the trial against him, police focused the investigation on Dowling.

When Dowling was first interviewed the day after the body was found, he explained that he had been spending that day fishing at a lake in the area. He had videotaped himself several times during the day, and also explained that he had stopped at a convenience store where he could probably be seen on the videotape of the surveillance camera. Further, Dowling supplied receipts from a boat rental dealer and a bait shop as evidence

that he had actually been fishing that day. The receipts and surveillance camera records confirmed that Dowling had been at the lake on the morning of October 20th. The murder had however occurred at 2 pm, and it would not be impossible for Dowling to drive to the lake first, and then to Myers' art gallery to kill her. After analyzing the videotape, police found that the tape showed Dowling fishing at the lake. The video contained timestamps incompatible with Dowling being the perpetrator. If the timestamps on the video were to be trusted, the video placed Dowling at the lake at the time of the murder and he could not be the killer.

Police immediately suspected that the timestamps on the videotape had been manipulated by Dowling, by making adjustments to the clock on the camera. To test this theory, investigators asked Robert Boyle, an associate professor of physics at Dickinson College to analyze the videotape. Boyle analyzed the tape by building a digital model of the scenery depicted in the video, based on compass measurements in the area around the lake. In the digital model, Boyle could estimate the length of the shadows the sun would cast in the picture at various times of the day, and compare them with the shadows actually seen on the video. The result was that the timestamps on the video differed from civil time with up to three hours. The analysis showed that the timestamps of the video had been manipulated, and that the video did not rule out that Dowling could be the murderer after all. After the analysis evidence had been filed with court, Dowling changed his statement regarding the videotape. He now claimed that he had gone tired of fishing and instead went to a strip club in the afternoon. He had changed the time on the tape because he didn't want his wife to find out.

Based on the videotape analysis, as well as other evidence such as findings of gunshot residue on clothes Dowling had been wearing that day, Dowling was convicted in York County trial court on December 14, 1998 to a verdict of death. The death sentence was upheld by the Supreme Court of Pennsylvania.

2.3 Clock manipulation in a contract investigation

Dates and times of decisions sometimes play an important role in determining if a contract has been upheld or breached. The helicopter contract investigation described in the following is a good example. The helicopter contract investigation was performed by

a digital investigation services provider in Norway in January 2003. It started as a dispute over a purchase contract of a helicopter that was signed between two parties. The seller was a private business operated by an individual, hereafter called Mr A. The buyer was a contractor business operating several helicopters offering air services primarily within the oil industry, hereafter called company B. A had a purchase option agreement on the helicopter from a foreign business, that allowed him to buy the helicopter should he be able to sell it. The contract between A and B was signed and went into force in July 2002. It mandated the delivery of the helicopter in January 2003. The contract also stated in a special clause that both parties could withdraw from the agreement without any consequences other then the cancellation of the agreement until the 15th of October 2002. After this date, the contract went into force and the parties could not withdraw. The contract mandated an initial transfer of funds shortly after the 15th of October. The remainder of the purchase amount would be paid when the helicopter was delivered.

In early October 2002, the managers of B went through the terms of the agreement in order to decide if it should be terminated before the 15th or not. After consideration of the terms, the management of B decided to withdraw from the contract. They phoned A several times and left messages on his voice mail about the termination. Further, they wrote two emails on October 7th and October 13th to A, stating that the contract was terminated. They did not receive any reply on these emails. On October 17th, a formal letter about the termination of the contract was written and sent through postal service. At this point the dispute arose. After receiving the letter, A claimed that the termination was too late, and that the contract therefore was in force. Should B terminate the contract, they would have to pay him compensation, amounting to the gain that he would have earned from the contract. A claimed that he had never received any phone calls or voice mail messages from B. Further, A claimed that he had never received any email messages from B concerning the termination of the contract. According to A, the first time he had heard about termination, was the letter dated October 17th, which was too late for termination. B printed copies of the termination emails they had sent to A before October 15th, serving as evidence of termination, but A still claimed that he had not received any such message. The managers of B then decided to disregard A's claims and act according to their own view that the contract was terminated in due time. Thus, no funds were transferred and A cancelled the purchase option agreement.

During late 2002 and early 2003, A considered whether he should go to the courts to claim compensation from B for what he saw as a breach of contract. In A's view, B's only evidence for terminating the contract before October 15th, was the copies of the emails they had sent. A realized that he needed evidence supporting the claim that he had never received these emails. A therefore decided to have his computer investigated by an independent third party to confirm that he had not received any email regarding contract termination in the period before October 15th. In January 2003, A asked a digital investigation service provider, hereinafter called company C, to do this job.

After receiving A's computer, the investigators at C performed the normal procedure for digital investigations at the time. The computer was dismantled and the hard drive removed. The computer clock was checked and found to be short of Norwegian civil time with only a few seconds. A digital image copy of the hard drive was made in another computer. The copy was mounted in an investigation computer and investigated with standard software for digital investigation. The investigation of existing files showed that the current operating system and application software on the computer had been installed in June 2002. The existing email on the computer was investigated and no evidence of email from company B in early October 2002 was found. Only a few emails were found to have been sent and received in October 2002. In fact, very small amounts of email were found at all. The computer did not contain the amount of user files and data that one would expect from a computer that had been used by a business for half a year. This prompted the investigators to check the unallocated areas of the disk for signs of deleted files. No signs of deleted files were found in this investigation. Instead, it was determined that the complete hard drive had been wiped by a disk wipe utility sold by C prior to the installation of the operating system. This disk wipe utility was programmed to write the current time, the version number and the serial id of the wiper to each sector on the hard drive. The date written to each sector of the hard drive was in early June 2002, on the same day as the date the operating system had been installed, as determined from timestamps on system logs and files in the operating system.

At this point, A's claim that no email had been received regarding termination of the contract would seem to be supported. However, the fact that so little user data was found on the computer prompted the investigators at C to perform further investigation steps. The version number of the disk wipe utility was compared with the development log of the utility at the development department of C. It was found that the particular

version of the wipe utility had been produced in the fall of 2002 and released in December 2002. Further, the serial number of the wipe utility was examined. The serial number showed that this version of the disk wipe utility had been sold in a computer store located near A's office in January 2003, one week before A had sent the computer to C for investigation.

From these results, the following hypothesis about the events on the investigated computer was formulated: In January 2003, A had bought the disk wipe utility at the local computer store. He had then adjusted the computer clock to June 2002, wiped the disk and installed the operating system. He had then used the computer for some time and gradually adjusted the computer clock forward in order to make it look like the activity had occurred during the fall of 2002. Finally, he had adjusted the clock back to the current civil time. The observed state of the system as investigated in January 2003 was consistent with this theory. C concluded that the original question of whether emails had been received in October 2002 could not be answered, since the user of the computer had wiped it in January 2003. These results were written in a report which was sent to A. After receiving the report from C, A decided to not pursue the matter in the courts.

2.4 Timestamps in procedure

Timestamps can be of importance not only in matters of substance but sometimes also in procedural matters. This is most clearly seen in matters of procedure where party representatives must meet a deadline. In these matters, an investigation of timestamps is sometimes performed in order to decide the question of whether the party met the deadline or not. The decisions of the UK Employment Appeal Tribunal in the cases of Midland Packaging v Clark, Woodward v Abbey National and JP Garrett v Cotton are illustrative. [25, 26]

In the mentioned cases, the Employment Appeal Tribunal received appeals against the Registrar's ruling an appeal out of time. Parties in cases before UK Employment Tribunals have 42 days to prepare a Notice of Appeal and submit it to the Employment Appeal Tribunal (hereafter EAT). The notice must have been received by the registrar of the EAT on the 42nd day after the decision of the Employment Tribunal in order to be considered. Notices received later then the 42nd day will not be considered by the EAT.

Thus, if the appeal is too late, the decision of the Employment Tribunal will be final. Further, UK Civil Procedure Rules (CPR) state that deliveries of documents and faxed documents in particular after 4 pm will be treated as filed on the next business day of the court office. The three cases discussed here all involve faxing of Notice of Appeal on the 42nd day, where the sender started the faxing before 4 pm according to their fax machine clock, but the complete notice had not been printed out on the receiving end before after 4 pm according to the clock on the receiving fax machine.

In the Midland case, the appellant faxed the notice just before 4 pm on the 42nd day. After the fax was completed, he received a document from his own fax machine that confirmed the transmission of 21 pages and the time when all the pages had been transmitted, which was 16:09. In the receiving end, the fax machine printed a timestamp on each printed page. The first page of the documents printed from the fax machine at the EAT showed 16:06, the second and third pages 16:07, the fourth and fifth 16:08 and so on. At this rate of approximately two pages per minute, the complete transmission of the 21 pages document would have taken around ten minutes.

In the decision, the court discusses various possibilities that may have caused the difference in the timestamps, for example different clock settings in the different fax machines, delay between scanning and faxing in the sending end or delay between receiving and printing in the receiving end. The court did not have available expert testimony regarding the functioning of the fax machines and could not as such make any assumptions regarding how the fax machines worked. The determining factor for the court was when the actual transmittal of information had occurred from the fax machine in the sending end to the fax machine in the receiving end. Based on the timestamp from the sending end, placing the end of the transmission at 16:09, and the calculation from the receiving end that the transmission took around ten minues, the court found that the transmission must have begun before 4 pm. The court found it probable that the two fax machines had communicated with each other before 4 pm and at least some part of the document had been electronically transferred before 4 pm, even if it had not been printed at the time. On this basis, the court decided that the appeal had been delivered in time. Note that in the basis for this decision, the court relied on the correctness of both the timestamp on the receipt from the sending machine and the accurateness of the transmission time calculation in the receiving end.

The two other cases happened after the decision in the Midland case. In the Woodward case, a 61 pages notice of appeal was transmitted by fax just before 4 pm on the 42nd day, the last day in the allowed appeal period. According to the appellant, the fax machine was ready at 15:35 and the documents started feeding through at 15:40. In the receiving end, a log showed that the faxing started at 15:48 and took 18 minutes and 46 seconds for the 61 pages to be faxed. The print out had timestamps 15:49 on the first page and 16:07 on the last page. In the J P Garrett case, a receipt from the appellant's end showed that faxing had started at 15:59 and took 3 minutes and 35 seconds to be completed. In the receiving end, the fax log and timestamps on the printed fax showed that the faxing started at 16:01 and took 3 minutes and 35 seconds, finished at 16:04. Thus, the situation in the J P Garrett case was exactly as in the Midland case, whereas the situation in the Woodward case was somewhat different, in that also timestamps in the receiving end showed receiving of documents before 4 pm.

In the latter cases, the court recognized the impracticability of having to rely on timestamps from both ends. Having determined the possibility of relying on a fax log in the receiving end, the court departed from the decision in the Midland case, and decided that the timestamps in the receiving end should determine the receiving time of the notice. Further, the court decided, again departing from the Midland decision, that the complete document must have been received before 4 pm. Even if faxing had started before 4 pm in both the Woodward and J P Garrett cases, the complete document had not been printed out at the receiving end before 4 pm in any of them. The decision therefore rendered the appeals in both cases as too late. Recognizing the fact that at least one of the appellants knew about the decision in the Midland case and acted accordingly, the court however granted the appellant the necessary extra time. Therefore, the appeals were accepted even if they had been received too late.

3 REASONING ON SETS OF CAUSAL EVENTS WITH TIMESTAMPS

This chapter presents the formal foundations of this work. Section 3.1 - 3.4 defines causality, time and clocks. Section 3.5 introduces ideal clocks and their properties. Section 3.6 introduces the specification of clock hypotheses. Section 3.7 examines ways of determining the correctness of a clock hypothesis. Section 3.8 defines clock hypothesis consistency. Section 3.9 examines the properties of consistency of subsets, supersets, unions and intersections of observed sets. Finally, Section 3.10 discusses whether the clock hypothesis can be viewed as a scientific hypothesis.

3.1 Causality

In this work, causal properties of systems are used to reason on timestamps and test hypotheses about the events occurring on a computer system. In order to be able to reason about events and causality, it is necessary to start from a definition of causality.

Informally, causality is the relationship between cause and effect. This relationship can be expressed as a relation between events. If two events e_1 and e_2 are related by the causality relation, then whenever e_2 occurs, e_1 must have occurred first. In previous works, causality has been defined by means of the *happened-before* relation $\rightarrow$. The happened-before relation was first used by Lamport, who defined the relation by ordering events happening in a process and sending and receiving messages between processes. [27] This definition was generalized by Fidge to encompass process creation and termination as well as both synchronous and asynchronous message passing. [28] These concepts have been utilized in distributed computing to achieve clock synchronization, and to achieve ordering of events in systems without synchronized clocks.

For use in digital investigation, Gladyshev proposed an extended definition of happened-before. In Gladyshev's version it is defined that $e_1 \rightarrow e_2$ if e_2 uses the result of e_1 or e_1 precedes e_2 in the usual course of business of some organisation or during the normal operation of a machine. [16] In this definition, the meaning of happened-before is

extended beyond computers. This extension is useful, since digital investigation requires the reconstruction of events, both within computers and outside them. Gladyshev's definition might however create doubt about exactly what happened-before means, since it is debatable what exactly constitutes the *normal operation* of a machine and the *usual course* of a business.

In order to be able to utilize the happened-before relation to reason on the causality between timestamp generating events, it is necessary to define happened-before in sufficient general terms to allow its application to causality in different types of systems. Yet, the definition should be sufficiently specific to allow it to represent the intuitive understanding of causality coherently. It is however difficult to define causality in very exact terms without entering into a deep philosophical discourse, something that is not intended here.

Definition 3.1. Let $\rightarrow$ be the *happened-before* relation. If $e_1 \rightarrow e_2$, then the occurrence of e_1 is necessary for e_2 to occur because e_2 depends on the effects of e_1. [1]

Important examples of causality per this definition of the happened-before relation include:

- e_1 produces an item that is necessary input for e_2
  This is equivalent to Gladyshev's definition "e_2 uses the result of e_1". The definition of happened-before in terms of message sending and reception used by Lamport and Fidge also fall within this example.

- e_1 and e_2 are events in a computer program, where e_2 uses data produced by e_1.
  Since events in computer programs use items produced by other events in the same program, such as variables, data stored in memory, registers and stack

[1] In order to be consistent with previous works, the symbol $\rightarrow$ is used to denote the happened-before relation in this work. It should not be confused with implication, which in this work is denoted by $\Rightarrow$.

pointers, many events occurring in computer programs will be related by happened-before. This is a special case of "e_1 produces an item that is necessary input for e_2". The definition of happened-before in terms of events occurring in a process used by Lamport and Fidge falls within this example, with the exception of events that do not use the result of each other. This exception makes the definition suitable for modern computer systems, in which the execution order of a computer program can be rearranged by compilers and processors when the instructions do not depend on the results of each other.

The happened-before relation has the following properties:

$\rightarrow$ is transitive. If e_i happened-before e_j and e_j happened-before e_k, then e_i happened before e_k.

$$(e_i \rightarrow e_j) \wedge (e_j \rightarrow e_k) \Rightarrow e_i \rightarrow e_k$$

The transitivity property implies that events can be causally connected through intermediaries. This follows directly from the definition of happened-before in Definition 3.1. If an event e_j depends on the effect of other events, then any event depending on the effects of e_j will also depend on those events.

$\rightarrow$ is irreflexive. An event e_i cannot happen-before itself.

$$e_i \not\rightarrow e_i$$

The irreflexivity property follows from the relationship implied by $\rightarrow$. An event dependant on the effects of itself cannot occur since such an event would require another event, itself, to occur first. That event would also require another event, itself, to occur first and so on ad infinitum.

$\rightarrow$ is antisymmetric. If e_i happened-before e_j, then e_j cannot happen-before e_i.

$$e_i \rightarrow e_j \Rightarrow e_j \not\rightarrow e_i$$

The antisymmetry property follows from the relationship implied by $\rightarrow$. If one event has occurred on which a second depend, the first cannot depend on the second. Note that any relation that is transitive and irreflexive must also be antisymmetric, since otherwise the self will be reachable transitively and thereby violate irreflexivity.

For any event e, there may be other events that happened-before it. There may also be other events that e happened-before. Any system with events and happened-before relations can be represented as a graph, where events are vertices, and instances of the happened-before relation are edges. Such a graph is shown in Figure 3.1.

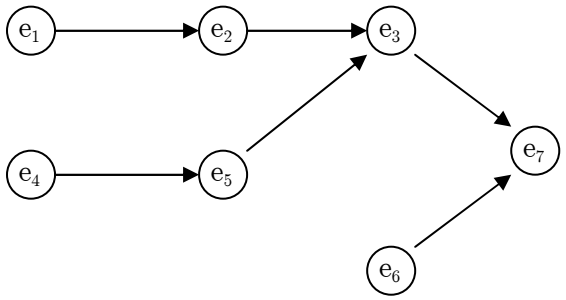

Figure 3.1 Graph of events related by happened-before

Since it is transitive, irreflexive and antisymmetric, the happened-before relation is a strict partial order relation, imposing a strict partial order on a set of events. A system with events and happened-before relationships shown as a graph is therefore a Directed Acyclic Graph (DAG), such as the graph shown in Figure 3.1.

3.2 Time and time values

In order to be able to reason about timestamps and clocks, it is necessary to start from a definition of time. Time as a concept is viewed quite differently in different disciplines such as physics and philosophy. In modern physics, time is viewed as an additional dimension to the three dimensions in space. The four dimensions constitute the space-time-continuum. In philosophy on the other hand, time is often viewed as a container for events. In this view, time is merely a concept that allows us to sort and compare the occurrence of events.

In this work, time is considered to be a fundamental quantity. As a fundamental quantity, time is not itself defined in terms of other quantities, but it is measurable by means of comparison with periodic events, such as the periodic events occurring in clocks. Such periodic events may for example be the swinging of a pendulum (a pendulum clock), the movement of earth (a sundial) or microwave emission from certain materials (an atomic clock). We consider events to have a moment in time associated with them, and assume that these moments in time can be ordered in time by relations $<$ and $=$.

Definition 3.2. Let E be the domain of events. Let e be an event. Events are considered to be instantaneous. Let T be the domain of time. Let $t(e)$ be a function $E \mapsto T$, representing the moment in time at which event e occurred.

Further, we assume that *causality is preserved in time*. With the preservation of causality in time, we mean that no event can causally depend on an event occurring at the same time or a later time than itself. This can be expressed explicitly with the happened-before relation as:

$$t(e_i) \leq t(e_j) \Rightarrow e_j \not\rightarrow e_i \tag{3.1}$$

This assumption corresponds to the intuitive understanding of the relationship between causality and time. If such causal relations were allowed, then events in the future would be allowed to affect events in the past, something that has not been shown to occur in the real world.

For two events related by the happened-before relation, Equation (3.1) implies that:

$$e_i \rightarrow e_j \Rightarrow t(e_i) < t(e_j) \tag{3.2}$$

From Equation (3.2), the progression of time is linked to causal sequences of events because time must increase for each event that is causally connected to a previous event. This corresponds to the philosopher's view of time as a container for events. At the occurrence of two causally connected events, an observer would perceive the event happening-before as the event coming first in time, and can infer the progression of time from the occurrences. Consider for example a philosopher watching Galileo dropping

objects from a tower through a telescope. The events “Galileo dropped the stone” and “the stone falls” are clearly causally connected and the first must therefore have occurred at an earlier point in time than the second. Thus, the philosopher can infer the progression of time from his observations. On the other hand, if Galileo did not drop the object and made no movements, the philosopher could not infer the progression of time from what he had seen through the telescope. Instead he would have to rely on other chains of events, such as the movement of the sun or his own heartbeats.

The above imposes an ordering in time on events ordered by the happened-before relation $\rightarrow$. It does not however imply any ordering in time for events not ordered by $\rightarrow$. Also, $t(e_i) < t(e_j)$ does not imply that $e_i \rightarrow e_j$. Events may happen at different moments in time without being related by $\rightarrow$. On the other hand, if two times $t(e_1)$ and $t(e_2)$ are ordered such that $t(e_1) < t(e_2)$, events occurring at those moments in time cannot be causally connected in reverse, such that the $e_2 \rightarrow e_1$.

3.3 Clocks

A clock is a device designed to give the user an approximation of time that is sufficiently coherent to allow him to measure and compare time periods and sufficiently consistent with other clocks so as to allow the owner to perform actions concurrent with other clock users without continuous coordination. Clocks are in other words designed to give an approximation of time. The definition of a clock should be able to reflect the possibility of clock drift and adjustment mentioned in section 1.4.

Definition 3.3. Let V be the domain of time values produced by a clock. $c(t)$ is a clock function $T \mapsto V$

The definition of a clock function does not impose any restrictions on the clock values as a function of time. For example, even if $t_1 < t_2$ it may be the case that $c(t_1) > c(t_2)$. And even if $t_1 < t_2 < t_3$, it may be the case that $c(t_1) = c(t_2) = c(t_3)$. The latter situation may for example occur if the events occurring at t_1, t_2, t_3 are so close together in time that the clock is unable to differentiate between them.

3.4 Timestamped events

A timestamped event is an event for which there exists a timestamp value in domain V. The timestamp value can be represented as a function on the event. Timestamps are created when an event makes a copy of the value provided by a clock. All timestamps in a set of timestamped events are not necessarily related to the same clock.

Definition 3.4. Let E be a set of timestamped events and V a domain of time values. $\tau_c(e)$ is a function $E \mapsto V$ such that $\tau_c(e_i) = c(t(e_i))$. $\tau_c(e_i)$ represents the timestamp associated with the event e_i relative to clock c.

In this definition, a timestamp is the value of the producing clock at the time of the event. The timestamp reflects the clock's representation of time at that particular moment. The definition of timestamps as a function of events and clocks provides a possibility to reason over timestamps and clocks.

3.5 Ideal and non-ideal clocks and their properties

It is useful to introduce the concept of *ideal clocks* and *non-ideal clocks*. An ideal clock is a clock which can only go forward.

Definition 3.5. Let I be the set of ideal clocks. An ideal clock $c(t) \in I$ is a clock which satisfies

$$\forall i \forall j (t(e_i) < t(e_j) \Rightarrow c(t(e_i)) \leq c(t(e_j)))$$
$$\forall i \forall j (t(e_i) = t(e_j) \Rightarrow c(t(e_i)) = c(t(e_j)))$$

Let NI be the set of non-ideal clocks. Non-ideal clocks are clocks that are not ideal clocks $c(t) \notin I$.

An ideal clock is a clock function on time which has the property that the value provided in the function on time is monotonically increasing. While having a monotonically increasing value, values $c(t(e_i))$, $c(t(e_j))$ produced from two different moments in time $t(e_i)$ and $t(e_j)$ where $t(e_i) < t(e_j)$ may be equal. Many clocks represent moments in time

as discrete values. In a discrete clock with limited resolution, two moments close in time will be represented by the same clock value.

An always increasing clock has the property that all timestamps produced from it will always be equal or higher for events occurring at increasing moments in time. In particular, events causally connected to each other would have timestamps with an equal or higher value on the latter event. This can be formulated as follows.

Theorem 3.6. For all ideal clocks $c \in I$, produced timestamps satisfies

$$e_i \rightarrow e_j \Rightarrow \tau_c(e_i) \leq \tau_c(e_j)$$

Proof. From Definition 3.5 an ideal clock satisfies:

$$\forall i \forall j (t(e_i) < t(e_j) \Rightarrow c(t(e_i)) \leq c(t(e_j)))$$

That is, for events e_i and e_j occurring at times $t(e_i)$ and $t(e_j)$ we have:

$$t(e_i) < t(e_j) \Rightarrow c(t(e_i)) \leq c(t(e_j))$$

By replacing the right hand side of Equation (3.2) we now obtain:

$$e_i \rightarrow e_j \Rightarrow c(t(e_i)) \leq c(t(e_j))$$

And from Definition 3.4, $\tau_c(e_i) = c(t(e_i))$, which gives:

$$e_i \rightarrow e_j \Rightarrow \tau_c(e_i) \leq \tau_c(e_j)$$

□

It is also interesting to determine if the opposite is the case, that a non-ideal clock does not necessarily produce such timestamps.

Theorem 3.7. There is at least one possible non-ideal clock $c \in NI$ that does not satisfy

$$e_i \rightarrow e_j \Rightarrow \tau_c(e_i) \leq \tau_c(e_j)$$

Proof. The proof is simply by providing a counterexample. Let $e_i \rightarrow e_j$. It now follows from Equation (3.2) that $t(e_i) < t(e_j)$. Now, let

$$c(t(e_i)) = 1$$
$$c(t(e_j)) = 0$$

The clock c is non-ideal ($c \in NI$) because

$$t(e_i) < t(e_j) \not\Rightarrow c(t(e_i)) \leq c(t(e_j))$$

And in this case,

$$c(t(e_i)) > c(t(e_j)) \Leftrightarrow \tau_c(e_i) > \tau_c(e_j)$$

Thus,

$$\tau_c(e_i) \not< \tau_c(e_j)$$

□

As shown, the monotonic property of ideal clocks guarantee that two causally connected events timestamped by the same ideal clock have timestamps where the timestamp of the latter event is always equal or higher than the timestamp of the first.

It would be desirable if public recognized clocks in the real world were ideal clocks. But even if a clock is publicly recognized, it is not necessarily an ideal clock. Civil time in a country may for example be adjusted for Daylight Savings Time twice a year. If this adjustment breaks the monotonicity requirement of an ideal clock in Definition 3.5, then the civil time in that country is not an ideal clock. UTC (see Appendix B.1) is adjusted regularly to keep it synchronized with the rotation of the Earth. Because of the way it is defined, UTC still maintains monotonicity, and is therefore an ideal clock according to the definition.

3.6 Clock hypotheses, adjustment and drift

In order to be able to test if a certain theory about the clock holds, one must be able to formulate a hypothesis about the clock function. A hypothesis about the clock function is a possible theory about the clock function during the computer history. That hypothesis

can then be tested against the set of observed timestamps. In the following, a clock hypothesis will be denoted $c_h(t)$.

In order to be able to reason on timestamps from a certain clock, it is useful to divide the clock into a base component being an ideal clock and a deviation component representing all adjustment, error and drift in the clock. The only requirement for the base is that it is an ideal clock, but selecting a base clock that allows correlation with other clocks is a good idea. UTC is an example of a universal time source that would allow such correlation. The UTC is moving forward at a known rate, and is adjusted in a predictable manner. There are many clocks in the world that have a known relationship to UTC. Therefore, if events can be timestamped at UTC, it would be easy to correlate their timing with those of events timestamped by other clocks.

Definition 3.8. A clock function $c(t)$ can be divided into two components, one function $b(t)$ which is an ideal clock and one function $d(t)$ representing the deviation from the ideal clock.

$$c(t) = b(t) + d(t)$$

In this scheme, the clock ($c(t)$) is divided into components: $b(t)$ is a base clock which must be an ideal clock. $d(t)$ is the difference between the base clock and the investigated clock. By selecting a common base, two or more clocks can be compared by comparing the deviation only. It is sometimes useful to express the time of an event in terms of the base clock. This can be done by subtracting $d(t)$ in Definition 3.8.

$$b(t) = c(t) - d(t) \tag{3.1}$$

Example 3.9. The clock of the investigated computer is equal to civil time at the time of the investigation. The investigator assumes that this has been the case in the entire history of the computer. Selecting civil time as the base, the hypothesis becomes

$$d(t) = 0$$

In other words

$$c(t) = b(t)$$

Example 3.9 describes the *default hypothesis* - the assumption that the clock has been maintained at the current offset during the computer history. In the default hypothesis, the assumption is that the clock has not been adjusted at any time in its previous history. This assumption is not justified only by the fact that the clock is equal to civil time when the investigation is performed. It could also be the case that the clock had been adjusted to an earlier time and then adjusted forward again, similar to the example in Section 2.3. A clock hypothesis model must be able to model such changes to the clock.

A clock adjustment is an event. If the event of an adjustment to a clock is e_{adj}, then the time of that adjustment is $t(e_{\text{adj}})$. The time value of that adjustment in the base clock is $b(t(e_{\text{adj}}))$. Adjustments on a clock can now be represented by representing $d(t)$ as a discontinuous function on t. By letting the clock adjustment be represented by an event, it is possible to perform reasoning on any other events causally dependant on that event. Records of these events may for example exist in computer systems which log clock changes.

Example 3.10. The clock of the investigated computer is equal to civil time at the time of the investigation. The investigator assumes that the clock was adjusted back one year (e_{adj1}) approximately two years before the investigation (e_{inv}) and then adjusted forward one year (e_{adj2}) approximately one year before the investigation.

$$d(t) = 0,\ t < t(e_{\text{adj1}})$$
$$d(t) = -1\,\text{year},\ t > t(e_{\text{adj1}}) \wedge t < t(e_{\text{adj2}})$$
$$d(t) = 0,\ t > t(e_{\text{adj2}})$$
$$b(t(e_{\text{adj1}})) = b(t(e_{\text{inv}})) - 2\,\text{years}$$
$$b(t(e_{\text{adj2}})) = b(t(e_{\text{inv}})) - 1\,\text{year}$$

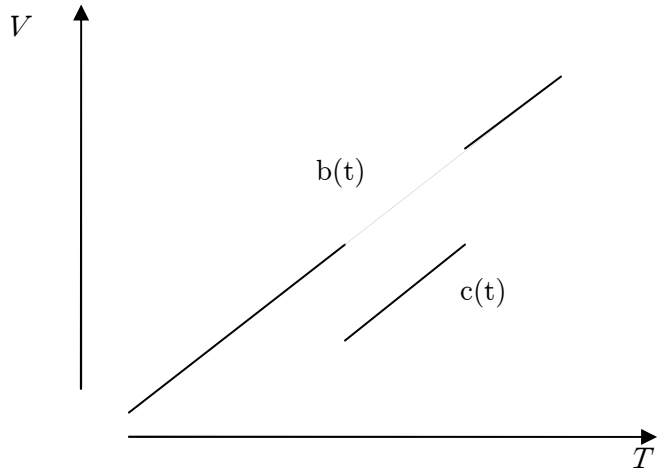

Figure 3.2 Plot of $b(t)$ and $c(t)$ in Example 3.10.

A clock hypothesis must also be able to account for errors and drift in clocks. The occurrence of random errors in clock can be modelled the same way as adjustments, by letting an event represent the occurrence of the error and determining its effects in $d(t)$. Clock drift is a little different. Drift is in most cases a factor on the time since the clock drifts a certain amount per time unit, relative to the time unit. [12]

Example 3.11. A clock was adjusted to UTC (e_{adj1}) and is believed to drift 5 seconds per day in the forward direction.

$$c(t) = b(t) + d(t)$$

$$d(t) = 5\,\text{sec} \cdot \frac{b(t) - b(t(e_{\text{adj1}}))}{1\,\text{day}}$$

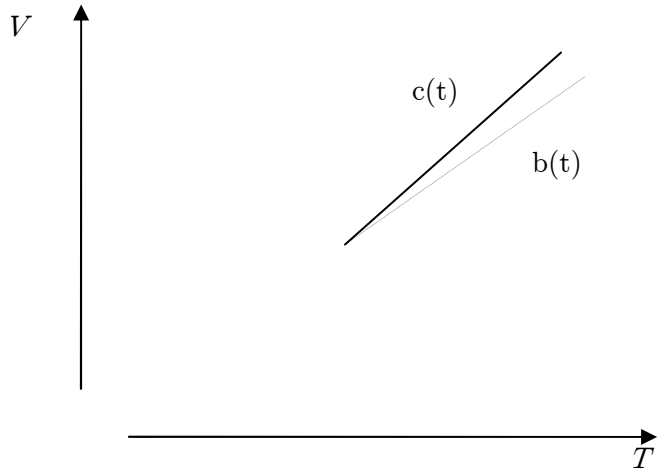

Figure 3.3 Plot of $b(t)$ and $c(t)$ in Example 3.11

This provides ways of defining a clock hypothesis. There may be other occurrences that should be modelled in a clock hypothesis as well. The discussion in the following does not impose any constraints on the clock hypothesis itself, as long as it can be composed of an ideal base clock component and a deviation component.

The clock function as it is represented here is similar to the clock models described by Stevens and Buchholz. [14, 15] These models describe a clock in terms of a base clock by letting it be the sum of clock drift, adjustments and error. These models can be used to represent clock hypotheses to be tested using the methods in this work. It should however be noted that Definition 3.8 requires the base clock to be an ideal clock. This requirement is necessary in the development of consistency tests developed in the following. The requirement is satisfied in Stevens' model, since it uses UTC as base. Buchholz does not however state any such requirement for the reference time used in his model. Thus, if Buchholz' clock model is used for hypothesis formulation and testing with the methods in this work, an ideal clock must be selected as reference time.

3.7 Observed event sets and correctness

During a digital investigation of a computer, the investigator will observe a number of timestamped events that all come from the same clock. Some of these events will be causally connected. This set of observed timestamped events is called the observation set.

Definition 3.12. An *observation set* O, is a set of timestamped events, in which all timestamps are related to one clock $c_o(t)$.

In an observation set, there will typically be a large amount of timestamped events. The number of causal connections may also be large. The data in an observation set can be used to determine if a clock hypothesis holds or not.

Definition 3.13. A clock hypothesis $c_h(t)$ for an observation set O is *correct* if and only if the value of $c_o(t)$ is equal to the value of $c_h(t)$ for all events $e \in O$.

$$c_o(t) = c_h(t)$$
$$\Downarrow$$
$$\forall e_i(\tau_{c_o}(e_i) = c_h(t(e_i)))$$

If a clock hypothesis is correct, then all occurrences of timestamps must match the value predicted by the hypothesis. The correctness property can therefore be utilized to find techniques for testing if a clock hypothesis is correct or not.

Theorem 3.14. In a correct clock hypothesis $c_h(t)$, the timestamps of all causally connected events $e_i \rightarrow e_j$ in an observation set O must be such that the timestamp of the first event minus the deviation from a common base has value less than or equal to the timestamp of the latter event minus the deviation from a common base.

$$e_i \rightarrow e_j \Rightarrow \tau_{c_o}(e_i) - d_h(t(e_i)) \leq \tau_{c_o}(e_j) - d_h(t(e_j))$$

Proof. Let $c_h(t)$ be a correct clock hypothesis. Let $b(t)$ be a common base for $c_h(t)$ and $c_o(t)$. Then

$$b(t) = c_h(t) - d_h(t)$$
$$b(t) = c_o(t) - d_o(t)$$

Thus,

$$c_h(t) - d_h(t) = c_o(t) - d_o(t)$$

And since $c_h(t)$ is correct we have $c_h(t) = c_o(t)$. Therefore

$$d_h(t) = d_o(t)$$
$$b(t) = c_o(t) - d_h(t)$$

And inserting Definition 3.4 yields

$$b(t(e)) = \tau_{c_o}(e) - d_h(t(e))$$

Now, from Definition 3.8 $b(t)$ shall be an ideal clock. From Theorem 3.6 we know that ideal clocks satisfy

$$e_i \rightarrow e_j \Rightarrow c(t(e_i)) \leq c(t(e_j))$$

And then, inserting $b(t)$ gives

$$e_i \rightarrow e_j \Rightarrow b(t(e_i)) \leq b(t(e_j))$$
$$e_i \rightarrow e_j \Rightarrow \tau_{c_o}(e_i) - d_h(t(e_i)) \leq \tau_{c_o}(e_j) - d_h(t(e_j))$$

□

And conversely, if the property examined in Theorem 3.14 does not hold, then the hypothesis is not correct.

Theorem 3.15 (Test-A theorem). If a pair of causally connected events $e_i \rightarrow e_j$ exist in an observation set O, for which the timestamp of e_i minus the hypothesis deviation from a common base has a higher value than the timestamp of e_j minus the hypothesis deviation from a common base, then the clock hypothesis is incorrect. This is called Test-A.

$$\exists e_i \exists e_j ((e_i \rightarrow e_j) \wedge (\tau_{c_o}(e_i) - d_h(t(e_i)) > \tau_{c_o}(e_j) - d_h(t(e_j)))) \Rightarrow c_o(t) \neq c_h(t)$$

Proof. The proof is by contradiction. Let $c_h(t)$ be a clock hypothesis and O an observation set with clock $c_o(t)$. Let (e_a, e_b) be a pair of events in O such that $e_a \rightarrow e_b$ and

$$\tau_{c_o}(e_a) - d_h(t(e_a)) > \tau_{c_o}(e_b) - d_h(t(e_b))$$

Assume that $c_h(t)$ is correct, $c_h(t) = c_o(t)$. If $c_h(t)$ is correct we have from Theorem 3.14 that

$$e_i \rightarrow e_j \Rightarrow \tau_{c_o}(e_i) - d_h(t(e_i)) \leq \tau_{c_o}(e_j) - d_h(t(e_j))$$

But for $i = a$ and $j = b$, we have assumed that $e_a \rightarrow e_b$ and so,

$$(e_a \rightarrow e_b) \wedge (\tau_{c_o}(e_a) - d_h(t(e_a)) > \tau_{c_o}(e_b) - d_h(t(e_b))) \qquad (3.2)$$

This contradicts the result from Theorem 3.14. Therefore, if (3.2) holds, then $c_h(t)$ cannot be correct. There has been no assumption or restriction on the events e_a and e_b. e_a and e_b could therefore have been any event in the observation set O. The result is that for any event e_i and e_j, if (3.2) holds, $c_h(t)$ cannot be correct.

$$\exists e_i \exists e_j ((e_i \rightarrow e_j) \wedge (\tau_{c_o}(e_i) - d_h(t(e_i)) > \tau_{c_o}(e_j) - d_h(t(e_j)))) \Rightarrow c_o(t) \neq c_h(t)$$

□

Test-A can be illustrated by an example.

Example 3.16. Consider the *default clock hypothesis*, where it is assumed that the clock of the investigated computer has always been equal to civil time, say UTC. Then $c_h(t) = b(t)$ and $d_h(t) = 0$. Now, let the observed set consist of timestamps for four events e_1 - e_4, where:

$$\tau_{c_o}(e_1) = \text{Jan 12, 2003, 12:46:34}$$
$$\tau_{c_o}(e_2) = \text{Apr 21, 2004, 10:22:38}$$
$$\tau_{c_o}(e_3) = \text{Feb 9, 2003, 22:16:04}$$
$$\tau_{c_o}(e_4) = \text{Dec 12, 2002, 02:46:32}$$

And where $e_1 \rightarrow e_2$ and $e_3 \rightarrow e_4$. If we now apply Test-A for $i = 3$ and $j = 4$, we see that

$$(e_3 \rightarrow e_4) \wedge (\tau_{c_o}(e_3) > \tau_{c_o}(e_4))$$

And since $d_h(t) = 0$, the test fails. Thus, the default hypothesis is not correct for this observation set.

The result can be explained informally as follows: Since e_4 must have happened after e_3 and the timestamp of e_4 is at an earlier time than the timestamp of e_3, it cannot be the case that the clock has not been adjusted between these two events.

Theorem 3.17 (Test-B theorem). In a clock hypothesis $c_h(t)$, for values c' of $c_h(t)$ for which $c_h(t) = c'$ has no solution, the existence of any timestamps in the observation set O with value

$$\tau_{c_o}(e_i) = c'$$

implies that $c_h(t)$ is incorrect. This is called Test-B.

Proof. The proof is by contradiction. Let $c_h(t)$ be a clock hypothesis and O an observation set with clock $c_o(t)$. Let e_a be an event in O and

$$\tau_{c_o}(e_a) = c'$$

the timestamp of e_a. Let c' have a value such that $c_h(t) = c'$ has no solution. Assume that $c_h(t)$ is correct, $c_h(t) = c_o(t)$. If $c_h(t)$ is correct we have from Definition 3.13

$$\forall e_i(\tau_{c_o}(e_i) = c_h(t(e_i)))$$

Which means that for $i = a$

$$\tau_{c_o}(e_a) = c_h(t(e_a))$$

This is a contradiction since $\tau_{c_o}(e_a) = c'$ and $c_h(t) = c'$ has no solution.

Therefore if $\tau_{c_o}(e_a) = c'$ and $c_h(t) = c'$ has no solution, then $c_h(t)$ cannot be correct.

□

An example will illustrate the use of Test-B.

Example 3.18. Consider the clock hypothesis in Example 3.10. This hypothesis is a discontinuous function for which there exist values c' where $c(t)$ has no solution. If for example a timestamp τ_{odd} from one and a half years before the investigation was found, this would imply the incorrectness of the hypothesis. A plot of this timestamp value and the clock hypothesis is shown in Figure 3.4.

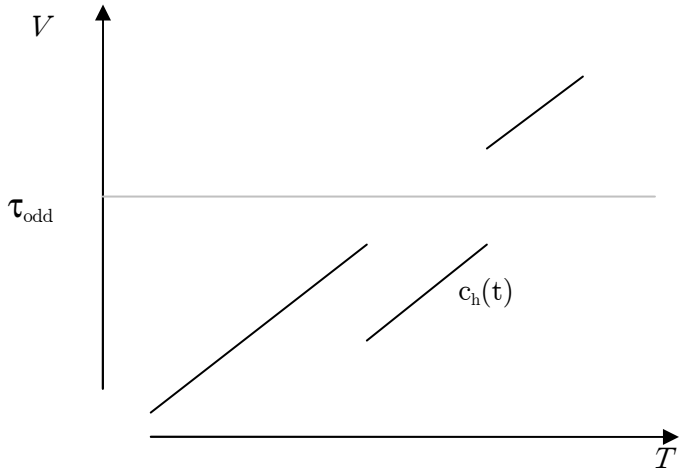

Figure 3.4 An observation of timestamp τ_{odd} for which $c_h(t) = \tau_{odd}$ has no solution

3.8 Clock hypothesis consistency

The results in Theorem 3.15 and Theorem 3.17 are useful, because they can be used to refute a clock hypothesis for observation set O, from observations of the timestamps on events in O. In Test-A, a clock hypothesis is incorrect when observations of timestamps for two causally connected events are not ordered correctly by the clock hypothesis. In Test-B, a clock hypothesis is incorrect if observations of timestamps exist that cannot be produced by the clock hypothesis, because it is a discontinuous function. These theorems provide methods for testing if a clock hypothesis is incorrect. By iterating over all events and pair of events, each timestamp can be checked for consistency with Test-A and Test-B. There are also other tests that can determine if a clock hypothesis is incorrect, as will be more closely examined in Chapter 5-7.

The result of testing all timestamps in the observation set will be either that the clock hypothesis is incorrect or that it is not incorrect. The tests can refute the clock hypothesis, but they can not prove it correct. This leads to the following definition of a consistent clock hypothesis.

Definition 3.19 Given a set of tests Z, a clock hypothesis is *consistent under* Z with an observation set O if no test $z \in Z$ shows that the hypothesis is incorrect for O. A clock hypothesis is *inconsistent under* Z with an observation set O if it is not consistent under Z with O.

Since a correct hypothesis by definition cannot be proven incorrect it follows that

Corollary 3.20 A correct hypothesis $c_h(t)$ for observation set O is always consistent with O.

The distinction that follows from the definitions of correct and consistent is useful in the context of digital investigations. In a correct clock hypothesis all possible time values are always equal to the investigated clock. A correct clock hypothesis can only be derived if the investigated clock has been observed at every moment in its history. Establishing a correct hypothesis about the investigated clock is inconceivable in a real investigation. All the investigator can hope to do is to establish a consistent clock hypothesis. In such a hypothesis there is no evidence available that refutes the hypothesis. Specifically, none of the timestamps of events in the observation set O as applied in tests in the test set Z show that the hypothesis is incorrect. If there is a large number of timestamps and causally connected events present in the observation set O, these requirements impose strict constraints on a consistent hypothesis. This can lead to the justification of the hypothesis. The more data available in O to be fed into the tests in Z, the more justified the clock hypothesis can be. As long as the clock hypothesis is consistent, the data in O is evidence supporting the hypothesis.

3.9 Subsets, supersets, intersections and unions

It is useful to be able to reason on subsets and supersets of the observed set. It is therefore interesting to look at certain properties of such sets in relation to Test-A and Test-B.

Theorem 3.21. If a clock hypothesis $c_h(t)$ is consistent under Test-A with an observation set O, then $c_h(t)$ is also consistent under Test-A with every subset of O.

Proof. Since $c_h(t)$ is consistent under Test-A with O, we have from Theorem 3.15 for O

$$\neg\exists e_i \exists e_j ((e_i \rightarrow e_j) \wedge (\tau_{c_o}(e_i) - d_h(t(e_i)) > \tau_{c_o}(e_j) - d_h(t(e_j))))$$

Which means that for any pair, e_i, $e_j \in O$:

$$\neg((e_i \rightarrow e_j) \wedge (\tau_{c_o}(e_i) - d_h(t(e_i)) > \tau_{c_o}(e_j) - d_h(t(e_j))))$$

Let P be any subset of O, $P \subseteq O$. For every pair of events e_i, $e_j \in P$, since $P \subseteq O$, it is also the case that e_i, $e_j \in O$. Which means that for also for any pair, e_i, $e_j \in P$:

$$\neg((e_i \rightarrow e_j) \wedge (\tau_{c_o}(e_i) - d_h(t(e_i)) > \tau_{c_o}(e_j) - d_h(t(e_j))))$$

Thus for P,

$$\neg\exists e_i \exists e_j ((e_i \rightarrow e_j) \wedge (\tau_{c_o}(e_i) - d_h(t(e_i)) > \tau_{c_o}(e_j) - d_h(t(e_j))))$$

Showing that $c_h(t)$ is not refuted by timestamps in set P using Test-A. From Definition 3.19, $c_h(t)$ is then consistent with P using Test-A.

□

Theorem 3.22. If a clock hypothesis $c_h(t)$ is consistent under Test-B with an observation set O, then $c_h(t)$ is also consistent under Test-B with every subset of O.

Proof. By Theorem 3.17, if $c_h(t)$ is consistent under Test-B with O, then there is no timestamp

$$\tau_{c_o}(e_a) = c'$$

for an event e_a in O, where $c_h(t) = c'$ has no solution.

Let P be any subset of O, $P \subseteq O$. For every event $e_i \in P$, since $P \subseteq O$, it is also the case that $e_i \in O$. Since there is no event e_a in O for which $c_h(t) = c'$ has no solution, there cannot be any such event in P. Consequently, there are no events in P for which $c_h(t) = c'$ has no solution. Therefore Test-B holds also for P, and by Definition 3.19 $c_h(t)$ is then also consistent under Test-B with P.

□

Theorem 3.23. If a clock hypothesis $c_{\mathrm{h}}(t)$ is inconsistent under Test-A with an observation set O, then $c_{\mathrm{h}}(t)$ is also inconsistent under Test-A with every superset of O.

Proof. Let $c_{\mathrm{h}}(t)$ be inconsistent with O using Test-A. Then (from Theorem 3.15):

$$\exists e_a \exists e_b((e_a \rightarrow e_b) \wedge (\tau_{c_{\mathrm{o}}}(e_a) - d_{\mathrm{h}}(t(e_a)) > \tau_{c_{\mathrm{o}}}(e_b) - d_{\mathrm{h}}(t(e_b))))$$

Let Q be any superset of O, $Q \supseteq O$. For every pair of events $e_i, e_j \in O$, since $Q \supseteq O$, it is also the case that $e_i, e_j \in Q$. This means that if $e_a, e_b \in O$ then consequently also $e_a, e_b \in Q$. Therefore, any pair of events e_a, e_b that makes $c_{\mathrm{h}}(t)$ inconsistent with O must also make $c_{\mathrm{h}}(t)$ inconsistent with Os superset Q. We have then for Q:

$$\exists e_a \exists e_b((e_a \rightarrow e_b) \wedge (\tau_{c_{\mathrm{o}}}(e_a) - d_{\mathrm{h}}(t(e_a)) > \tau_{c_{\mathrm{o}}}(e_b) - d_{\mathrm{h}}(t(e_b))))$$

□

Theorem 3.24. If a clock hypothesis $c_{\mathrm{h}}(t)$ is inconsistent under Test-B with an observation set O, then $c_{\mathrm{h}}(t)$ is also inconsistent under Test-B with every superset of O.

Proof. Let $c_{\mathrm{h}}(t)$ be inconsistent with an observation set O. Then by Theorem 3.17 there exists a timestamp $\tau_{c_{\mathrm{o}}}(e_a) = c'$ such that $c_{\mathrm{h}}(t) = c'$ has no solution.

Let Q be any superset of O, $Q \supseteq O$. For any event $e_i \in O$, since $Q \supseteq O$, it is also the case that $e_i \in Q$. Therefore, if the timestamp $\tau_{c_{\mathrm{o}}}(e_a) = c'$ is in O, it is also in Q. Consequently we have also in Q a timestamp $\tau_{c_{\mathrm{o}}}(e_a) = c'$ such that $c_{\mathrm{h}}(t) = c'$ has no solution.

□

These theorems states that with the test methods A and B, consistency will be preserved when treating subsets of the observed set as observed set and inconsistency will be preserved when treating supersets of the observed set as observed set. But is the opposite the case? Is for example consistency preserved when treating a superset of the observed set as observed set? It is easy to see that this is not the case. In the superset, there are events that do not exist in the observed set. These events may have values that cannot

be generated by the clock hypothesis, and therefore violate Test-B. Such events may also be causally connected to each other or to events in the observation set and may have values that are inconsistent with the clock hypothesis in Test-A. Consequently, consistency is not preserved when observing supersets. The opposite, that inconsistency is not necessarily preserved when observing subsets follows from similar reasoning. The events that constitute the inconsistency are not necessarily present in the subset, and a subset of the observed set may therefore be consistent, even if the observed set is not.

In the following, properties of intersections and unions of observation sets in relation to Test-A and Test-B will be examined.

Theorem 3.25. If a clock hypothesis $c_h(t)$ is consistent under Test-A with observation sets O_1 and O_2, then $c_h(t)$ is also consistent under Test-A with the intersection $O_1 \bigcap O_2$. If a clock hypothesis $c_h(t)$ is consistent under Test-B with observation sets O_1 and O_2, then $c_h(t)$ is also consistent under Test-B with the intersection $O_1 \bigcap O_2$.

Proof. Let O be the intersection of O_1 and O_2, $O = O_1 \bigcap O_2$. By the definition of intersection, O must then be a subset of O_1 and a subset of O_2. The preserving of consistency for subsets in Test-A and Test-B has already been proven in Theorem 3.21 and Theorem 3.22.

□

Theorem 3.26. If a clock hypothesis $c_h(t)$ is consistent under Test-B with observation sets O_1 and O_2 using Test-B, then $c_h(t)$ is also consistent under Test-B with the union $O_1 \bigcup O_2$.

Proof. Let O be the union of O_1 and O_2, $O_1 \bigcup O_2$. For every event $e_i \in O$, since $O = O_1 \bigcup O_2$, then e_i must be either in O_1 or O_2, $e_i \in O_1 \vee e_i \in O_2$.

By Theorem 3.17, if $c_h(t)$ is consistent under Test-B with O_1 and O_2, then there is no timestamp $\tau_{c_0}(e_a) = c'$ for an event e_a in O_1 or O_2, where $c_h(t) = c'$ has no solution. Therefore, since $e_i \in O_1 \vee e_i \in O_2$ there is no timestamp $\tau_{c_0}(e_a) = c'$ for an event e_a in O, where $c_h(t) = c'$ has no solution.

□

When examining Test-A in relation to unions of sets, it is necessary to define a *connection set*. A connection set is the set of bordering elements in each of the two different sets, elements that is member in one of the sets and have causal connections to elements in the other set. An example connection set is shown in Figure 3.5.

Definition 3.27. A *connection set* $O_{A\text{-}B}$ is a set consisting of elements from two sets O_A and O_B, where each element e in $O_{A\text{-}B}$ is element of exactly one of the sets O_A and O_B, and is causally connected to another element e_k in the other set. Formally,

$$\forall e_a \forall e_b (e_a \in O_A \wedge e_b \in O_B \wedge e_a \notin O_B \wedge e_b \notin O_A \wedge (e_a \rightarrow e_b \vee e_b \rightarrow e_a) \Rightarrow e_a \in O_{A-B} \wedge e_b \in O_{A-B})$$

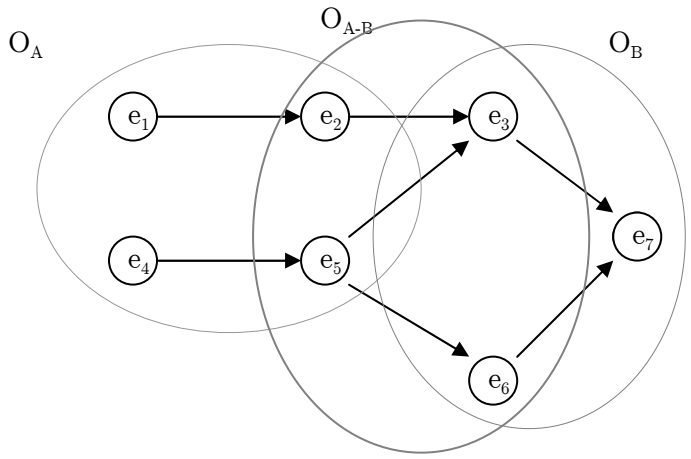

Figure 3.5 Graphical representation of a connection set

Theorem 3.28. If a clock hypothesis $c_h(t)$ is consistent under Test-A with observation sets O_1 and O_2, and consistent with the connection set $O_{1\text{-}2}$, then $c_h(t)$ is also consistent under Test-A with the union $O_1 \bigcup O_2$.

Proof. Let O be the union of two sets O_1 and O_2, $O = O_1 \bigcup O_2$. Let $c_h(t)$ be consistent with O_1 and O_2 and their connection set $O_{1\text{-}2}$. For $c_h(t)$ to be consistent, Theorem 3.14 must be satisfied for all $e_i, e_j \in O$:

$$e_i \rightarrow e_j \Rightarrow \tau_{c_o}(e_i) - d_h(t(e_i)) \leq \tau_{c_o}(e_j) - d_h(t(e_j))$$

Since $O = O_1 \bigcup O_2$, all elements in O must be an element of either O_1, O_2 or both. For every pair (e_i, e_j) in O, it could then be the case that either both e_i and e_j is element in the same set or that e_i is element in one set and e_j is element in the other. For the cases where e_i and e_j are element of the same set, we already know that Theorem 3.14 is satisfied, since $c_h(t)$ is consistent with both O_1 and O_2.

For the cases where e_i and e_j are elements of different sets, we have two cases: Either e_i is element in O_1 and not O_2, with e_j element in O_2 and not O_1, or e_i is element in O_2 and not O_1, while e_j is element in O_1 and not O_2. We do not consider cases where an event is element in both O_1 and O_2, since in that case Theorem 3.14 would be satisfied by the reasoning above. The two cases can be written:

$$e_i \in O_1 \wedge e_j \in O_2 \wedge e_i \notin O_2 \wedge e_j \notin O_1$$
$$e_j \in O_1 \wedge e_i \in O_2 \wedge e_j \notin O_2 \wedge e_i \notin O_1$$

In order to prove that Theorem 3.14 is satisfied, we need only consider cases where $e_i \rightarrow e_j$ or $e_j \rightarrow e_i$. The cases that need to be considered are then reduced to:

$$e_i \in O_1 \wedge e_j \in O_2 \wedge e_i \notin O_2 \wedge e_j \notin O_1 \wedge (e_i \rightarrow e_j \vee e_j \rightarrow e_i)$$
$$e_j \in O_1 \wedge e_i \in O_2 \wedge e_j \notin O_2 \wedge e_i \notin O_1 \wedge (e_i \rightarrow e_j \vee e_j \rightarrow e_i)$$

Since $c_h(t)$ is consistent with the connection set $O_{1\text{-}2}$ it is now sufficient to show that these cases are included in the connection set $O_{1\text{-}2}$. This follows directly from Definition 3.27. Let $a = i$, $b = j$, $A = 1$ and $B = 2$, and it becomes clear that the first cases are included in $O_{1\text{-}2}$. Further, let $a = j$, $b = i$, $A = 1$ and $B = 2$. It then becomes clear that also the second cases are included in $O_{1\text{-}2}$. Since $c_h(t)$ is consistent with $O_{1\text{-}2}$, all pairs of elements in O therefore satisfy Theorem 3.14 and consequently $c_h(t)$ is also consistent with O using Test-A.

□

Finally, it is interesting to determine the properties of an empty observation set. Since Test-A and Test-B only put requirements on $c_h(t)$ in relation to events in an observation

set, it is reasonable that a clock hypothesis is always consistent with an empty observation set.

Theorem 3.29. A clock hypothesis $c_{\mathrm{h}}(t)$ is always consistent under Test-A and Test-B with the empty observation set $O = \varnothing$.

Proof. Let the observation set O be the empty set, $O = \varnothing$. Test-A is given in Theorem 3.15:

$$\exists e_i \exists e_j ((e_i \rightarrow e_j) \wedge (\tau_{c_{\mathrm{o}}}(e_i) - d_{\mathrm{h}}(t(e_i)) > \tau_{c_{\mathrm{o}}}(e_j) - d_{\mathrm{h}}(t(e_j)))) \Rightarrow c_{\mathrm{o}}(t) \neq c_{\mathrm{h}}(t)$$

But if $O = \varnothing$, there can by definition not be any elements $e \in O$. Therefore, Test-A will never fail and by Definition 3.19, $c_{\mathrm{h}}(t)$ is always consistent under Test-A with O.

The same reasoning can be applied for Test-B, given in Theorem 3.17, whereby the test fails at an observation of $\tau_{c_{\mathrm{o}}}(e_i) = c'$ for which $c_{\mathrm{h}}(t) = c'$ has no solution. But if $O = \varnothing$, there can by definition not be any elements $e \in O$, for which there can exist observations of $\tau_{c_{\mathrm{o}}}(e_i) = c'$. Therefore this test will never fail and by Definition 3.19 $c_{\mathrm{h}}(t)$ is always consistent with O using Test-B.

□

Theorem 3.29 reflects the situation in an investigation where no evidence is available. In such an investigation most investigators would conclude that one would not be in the position to infer any conclusions about the history of the investigated object. The fact that any clock hypothesis would be consistent in this case reflects the nature of a hypothesis. A hypothesis cannot be proven correct, it can only be refuted. An empty observation set implies that there is no evidence available. When there is no evidence available the hypothesis cannot be refuted.

3.10 The clock hypothesis as a scientific hypothesis

In the hypothesis based investigation model proposed by Carrier, a digital investigation is a process that formulates and tests hypotheses to answer questions about digital events or the state of digital data. [4] Carrier proposes that the investigation process is scientific

if the hypothesis is scientific and then tested through performing experiments. Carrier cites Popper in that the "criterion of the scientific status of a theory is its falsifiability, or refutability, or testability". Popper elaborates further on the subject: [29]

- "Every 'good' scientific theory is a prohibition: it forbids certain things to happen. The more a theory forbids, the better it is."
- "Every genuine test of a theory is an attempt to falsify it, or to refute it. Testability is falsifiability; but there are degrees of testability; some theories are more testable, more exposed to refutation than other; they take, as it were, greater risks."
- "Confirming evidence should not count except when it is the result of a genuine test of the theory; and this means that it can be presented as a serious but unsuccessful attempt to falsify the theory."

The question here is then if the method for clock hypothesis formulation and testing the set of observed timestamps adhere to these criteria. From the previous discussion, a clock hypothesis is a theory that is falsifiable and therefore testable. After formulating a clock hypothesis, the investigator can use the methods explored in sections 3.7 - 3.9 to test the hypothesis. The outcome of such a test can only be a refutation of the hypothesis. If the hypothesis is not refuted, it is consistent according to the definition in section 3.8, but not necessarily correct according to the definition in section 3.7. The hypothesis cannot be proven correct, but as the amount of evidence supporting it increases, it becomes more and more justified. With the test methods that have been developed, the observed set of timestamps can put a clock hypothesis under scrutiny, especially where there are tens of thousands of timestamps in the observed set, such as on a typical hard drive. It is enough with one mismatched timestamp in an observed set to refute the clock hypothesis.

The clock hypothesis theory described in the previous sections adheres to the requirements of a scientific theory. The hypothesis forbids certain things to happen; the occurrence of timestamp configurations as described in Test-A and Test-B. The described tests examine the evidence for refutation of the theory. They do not look for confirmation, but examine the available evidence for consistency with the theory. When the tests have been applied, and found not to refute the hypothesis, the tests count as serious but unsuccessful attempts to falsify the theory and therefore as confirming evidence. Thus, the methods described in this chapter fall within the hypothesis based

investigation model proposed by Carrier and should be considered an application of this model within the subject of digital investigation of timestamps.

4 CAUSALITY IN STORAGE SYSTEMS

Chapter 3 introduced how causality reasoning can be applied to develop a consistent clock hypothesis. In order to be able to test a clock hypothesis for consistency, it is necessary to enumerate events in the investigated systems with causal connections between them. The study of events and causality in storage systems is of particular interest, since these systems provide the foundations of the way data is stored on the media investigated in digital investigations. In this chapter, several properties of storage systems will be investigated, with the goal of finding causal connections that can be utilized for clock hypothesis consistency determination.

Section 4.1 analyzes causality of append-only storages. Section 4.2 analyzes causality of first-fit storage systems, with or without generation markers. Section 4.3 generalizes these results for other allocation algorithms. Section 4.4 discusses the use of sequence numbers in storage systems. Section 4.5 applies the discussion in the other sections to the analysis of causality in file system events.

4.1 Append only allocation

A common type of storage system is a system in which each new storage location is allocated after the previously allocated storage location. In such a system, previous storage locations are never reused, since there is no support for deleting the contents of previous storage locations and free them for reuse. Without any influence from the outside, the contents of an append-only storage will grow towards infinity.

The use of append only allocation strategies is prevalent in log files. System logs on UNIX-systems are good examples. In these logs, the storage allocation is usually line-based. Each line represents a separate event in the system logging software, and is usually timestamped by the system logging software at the time it is stored. Other examples include line-based logs from ftp-servers, web servers and databases as well as binary logs such as UNIX wtmp-files and Windows event logs. In the latter, each storage

allocation represents timestamped system events. These are stored in an append-only fashion, although the format makes that less obvious than for text based logs.

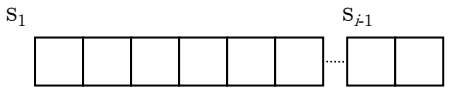

Figure 4.1 Graphical representation of append only storage

Let s_i be the i-th storage location where $i > 0$. Let e_{s_i} be the event of storing data in the i-th storage location. Then, for all i, $e_{s_{i-1}} \rightarrow e_{s_i}$.

On this basis, Test-A and Test-B can be used to test a clock hypothesis for consistency based on the timestamps in an append-only storage. If each storage location s_i is timestamped, Test-A can be applied directly on each pair $e_{s_{i-1}} \rightarrow e_{s_i}$, whereas Test-B can be applied to each event individually.

Example 4.1. Consider a default clock hypothesis, where $c_h(t)$ is considered to be equal to UTC and therefore $d(t) = 0$. Let the observation set O be the timestamps in the following excerpt from a Linux system log, where the creation of the first log line is e_1, the second e_2 and so on. If the log is append-only, then $e_1 \rightarrow e_2 \rightarrow \ldots \rightarrow e_{12}$.

```
Mar 18 20:17:56 liqin sshd[30513]: pam_unix(sshd:session): session closed for user bakksjo
Mar 18 21:43:25 liqin sshd[30680]: Accepted password for sventy from 80.203.37.109 port 63053 ssh2
Mar 18 21:43:25 liqin sshd[30682]: pam_unix(sshd:session): session opened for user sventy by (uid=0)
Mar 18 22:19:31 liqin sshd[30846]: Failed password for sventy from 213.179.57.64 port 38718 ssh2
Mar 18 22:19:34 liqin sshd[30846]: Accepted password for sventy from 213.179.57.64 port 38718 ssh2
Mar 18 22:19:34 liqin sshd[30848]: pam_unix(sshd:session): session opened for user sventy by (uid=0)
Mar 18 22:19:35 liqin sshd[30848]: pam_unix(sshd:session): session closed for user sventy
Mar 18 22:19:41 liqin sshd[30878]: Accepted password for sventy from 213.179.57.64 port 40882 ssh2
Mar 18 22:19:41 liqin sshd[30880]: pam_unix(sshd:session): session opened for user sventy by (uid=0)
Mar 18 22:20:40 liqin sshd[30880]: pam_unix(sshd:session): session closed for user sventy
Mar 18 22:22:46 liqin su: pam_unix(su-l:session): session opened for user root by sventy(uid=500)
Mar 18 22:28:44 liqin su: pam_unix(su-l:session): session closed for user root
```

There is no timestamps in the observation set where $(e_i \rightarrow e_j) \wedge (\tau_{c_o}(e_i) > \tau_{c_o}(e_j))$. Therefore the clock hypothesis is consistent using Test-A. Further, if $c_h(t)$ is equal to UTC, then there is no c' for which $c_h(t) = c'$ has no solution. Therefore the observation set is consistent using Test-B. Thus, the default hypothesis is consistent with O using tests A and B.

It could also be the case that not every storage location is timestamped. In this case the transitivity property of the happened-before relation can be used to extract those events that can be used in Test-A, and the happened-before relationship between them. Only the events of the timestamped store location creations are used in the observation set.

Example 4.2. e_1 - e_6 are the events of creation of six storage locations in an append-only storage system. Only e_1, e_3 and e_6 are timestamped. Since $e_1 \rightarrow e_2 \rightarrow e_3 \rightarrow e_4 \rightarrow e_5 \rightarrow e_6$, by transitivity, $e_1 \rightarrow e_3 \rightarrow e_6$. See Figure 4.2.

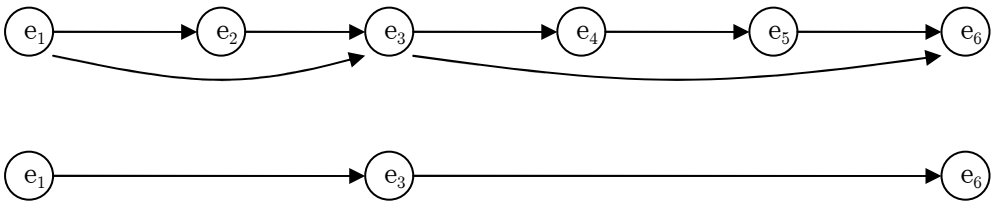

Figure 4.2 Timestamped events related transitively

Generally; due to the transitivity property, in an append-only system every creation of a storage location is causally dependant on the creation of every other storage location preceding it.

Let s_i be the i-th storage location in an append-only storage. Let e_{s_i} be the event of storing data in the i-th storage location. Then, for all i,

$$\forall j < i(e_{s_j} \rightarrow e_{s_i}). \tag{4.1}$$

4.2 First-fit allocation

A first-fit allocation storage is a system in which each new data item is stored in the first available storage location. Deleting data items is allowed and can be done at any time after the data item has been stored in a storage location. After a data item has been

deleted, it may be overwritten by new data at any time. It may be possible to recover deleted data, but it is not possible to recover data that has been overwritten, and it is not otherwise possible to determine if stored data was stored by using a deleted storage location or a previously unused storage location. A first-fit storage without deletion possibility is an append-only storage. Figure 4.3 shows a possible allocation sequence in a first-fit storage.

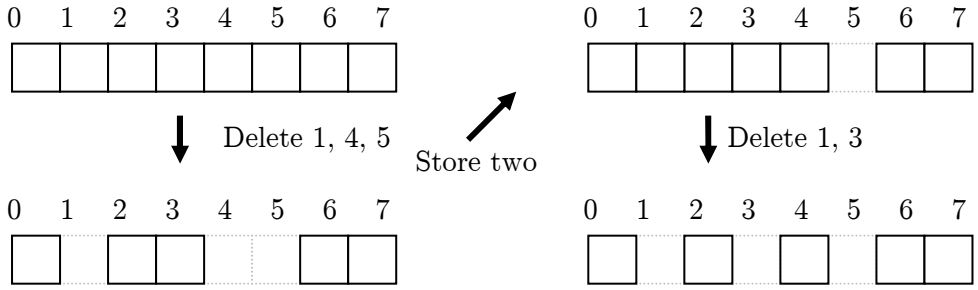

Figure 4.3 A possible allocation sequence in a first-fit storage

With a first-fit storage, one can no longer deduce a causal connection between two neighbours. For any two neighbours s_i and s_{i+1} in an append-only storage, $s_i \rightarrow s_{i+1}$, because new storage locations are only allowed to be allocated after previously allocated storage locations. In a first-fit storage on the other hand, storage locations are allowed to be allocated before other existing storage locations if data was deleted there, freeing storage locations for new allocation. Therefore, in such a storage it is not possible to deduce any causality between the storage of the elements based on the position of the elements themselves.

It is interesting to examine a modified form of the first available storage, the first fit storage *with generation-markers.* In this form, the first available storage is augmented with the possibility of identifying which generation the data in each storage location belongs to. The generation of a storage location is an identification of how many times that data in that storage location has been overwritten. On some systems, this number may be deduced from a stratigraphic analysis of the medium. On other systems, explicit

generation markers exist. Figure 4.4 shows the allocation sequence from Figure 4.3 with generation markers.

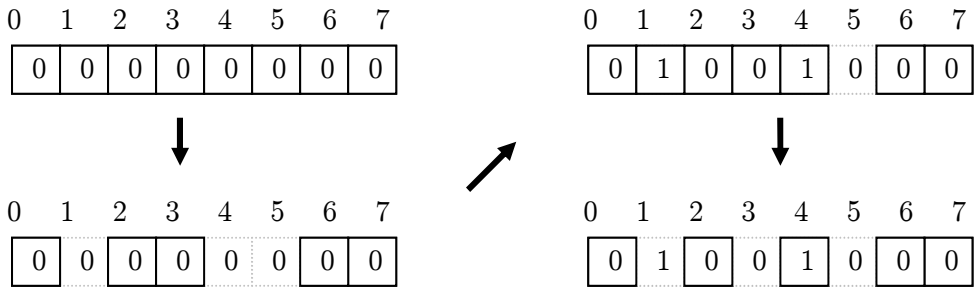

Figure 4.4 The sequence in Figure 4.3 with generation markers added

In a system with generation markers, there is a causal connection between every pair of consecutive generations at the same storage location. The storing of data in the i-th storage location generation g can only commence if the data present in the i-th storage location generation (g-1) has already been stored and deleted. Therefore, for every storage location i, the storage of data in generation (g-1) happened-before the storage of data in generation g. Generally; due to the transitivity property, the event of storing data in a storage location is causally dependant on the storing of all previous generations in that storage location.

Let $s_{i,g}$ be the i-th storage location generation g. Let $e_{s_{i,g}}$ be the event of storing data in the i-th storage location at the g-th generation. Then for all i and g; $e_{s_{i,g-1}} \rightarrow e_{s_{i,g}}$. Due to the transitivity of $\rightarrow$, for all generations g,

$$\forall h < g(e_{s_{i,h}} \rightarrow e_{s_{i,g}}) \tag{4.2}$$

Next, consider the storing of data in storage locations with generation $g = 0$. When $g = 0$, there cannot exist any storage location which has been deleted and then overwritten with another data item, because this would have increased the generation number above 0. The only place where new storage locations can be allocated with generation number 0 is at the end of the storage. The subset of storage locations with $g = 0$ is therefore an append-only storage.

Let $s_{i,0}$ be the i-th storage location in a first-fit storage, generation 0. Let $e_{s_{i,0}}$ be the event of storing data at generation 0 in the i-th storage location. Then, for all i,

$$\forall j < i(e_{s_{j,0}} \rightarrow e_{s_{i,0}}) \tag{4.3}$$

Two different types of causal event sets have now been defined from the first available storage with generation markers; the causality between storage of storage locations with $g = 0$, and causality between storage of increasing generations at one storage location. These sets intersect. Each causality set for increasing generations start at $g = 0$. Each such element is also part of the $g = 0$ causal set. With these two types of causal connections in the first available storage with generation markers, a causal connection on all storage locations in the set is imposed. The ordering imposed by a causal connection is best illustrated by an example.

Example 4.3. Consider the storage location set in Figure 4.5. In the figure, the storage locations are shown horizontally, and generations vertically. Deleted data are shown in the lighter colour. For each storage location, the topmost item is always the current data stored in the location. Figure 4.6 now shows the causal connections in the diagram, where each generation within a storage location happened-before the next generation, and each storage location at generation 0 happened-before the next location at generation 0. The resulting Direct Acyclic Graph of the creation events of the existing storage locations is shown in Figure 4.7.

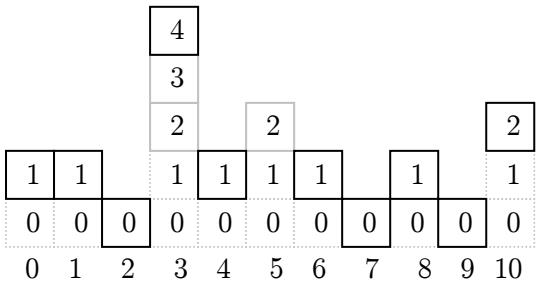

Figure 4.5 Storage locations in a system with generation markers

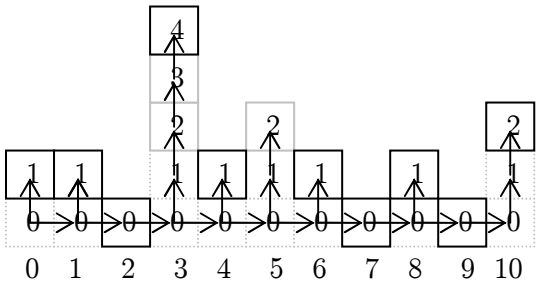

Figure 4.6 Causality between generations and between all locations at $g = 0$

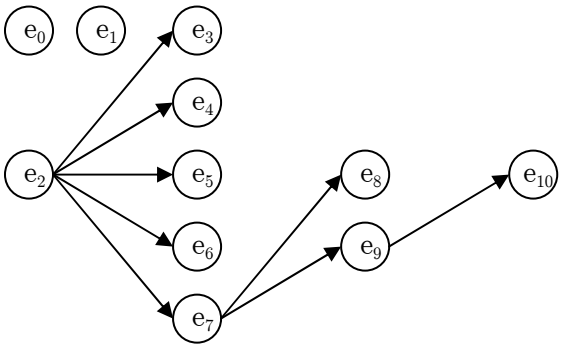

Figure 4.7 Graph of the storing events in Figure 4.6

Example 4.3 shows how causality in a first-fit system with generation markers imposes an ordering on the events in the system. Such an ordering implies causal connections between most storing events in the storage system, providing strict requirements on a clock hypothesis in such a system when tested with the tests developed in Chapter 3.

4.3 Other allocation strategies

There exists many possible allocation strategies in addition to those already discussed. Among these are: [30]

- Next-fit: Similar to first-fit, but the search for the next available location start at the location of the previous allocation, not the start of the storage as in first-fit.
- Best-fit: The available area that best match the data size is always allocated.
- Worst-fit: Opposite from best-fit: All allocation takes place from the largest contiguous storage chunk.
- Optimal-fit: Similar to best-fit, but other criteria than data size match are taken into account.

These strategies all have in common that deletion and reallocation may occur for any location in the storage. In a storage system utilizing any of these strategies without any possibility to determine the storage location generation, it will not be possible to deduce any causal connections between observed elements. The result that no causality can be derived in any such plain system is similar to the situation for first-fit allocation storage with no generation markers, as discussed in section 4.2.

How do these strategies compare to first-fit if augmented with generation markers? With such markers, each storage location has a generation id associated with it, determining the number of times that storage location has been deleted and reused by another data block. This does not change from a first-fit allocation system to any other allocation strategy, so Equation (4.2) still applies. It is still possible to deduce causal connections between each generation of a storage location.

For the first-fit algorithm, the storage of the first data generation is an append-only system in which the storage of each data block depends on all previous data blocks. With this property and the causality between generations, all data blocks in a first-fit system with generation markers can be linked together causally. An interesting question is therefore if similar causality exists in systems with other allocation strategies. From the discussion above, we know that any such system will have many causal connections if the storage of the first generation of storage locations is an append-only system. The question is therefore whether these allocation strategies result in an append-only system for the first generation. The answer to this question will depend on the specific implementation of the allocation algorithm, but some general observations can be made for each of the mentioned allocation algorithms:

- Next-fit: In a next-fit system, each new block in the first generation must have been allocated at a position in the storage, where no other blocks were allocated before. This occurs when there is no empty block where the allocated data would fit between the allocation pointer and the end of the storage. Whenever a new block in the first generation is allocated, all previous data blocks in the first generation must have been allocated before it. The first generation of this system is therefore an append-only system. See Figure 4.8.

- Best-fit: In a best-fit system, whenever a new block in the first generation is allocated, it is because there is no previous available block that would fit the data size better, so the data is stored at the end of the storage. Whenever a new block in the first generation is allocated, all previous data blocks in the first generation must have been allocated before it. Therefore the first generation of this system is also an append-only storage.

- Worst-fit/Optimal-fit: The same type of argument applies to worst-fit and optimal-fit: If a new block in the first generation is allocated, it is because there is no previously block that fit worse or more optimal in the case of optimal-fit. The first generation of such systems would also be append-only.

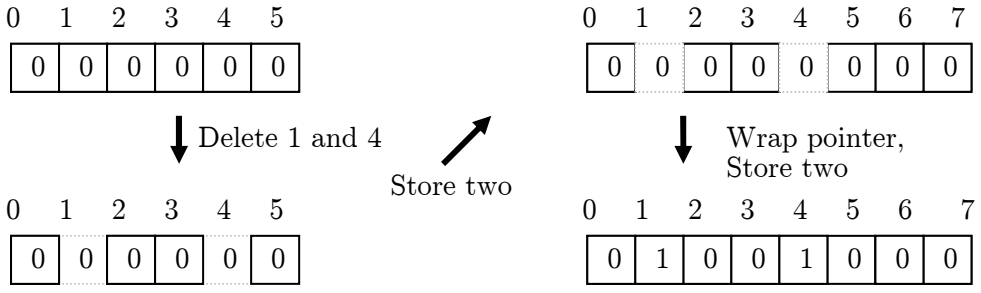

Figure 4.8 A possible allocation sequence in a next-fit system with generation markers

Although the discussion of any specific allocation implementation necessarily must depend on the details of that implementation, it is worth noting that adding generation markers to any allocation system in principle impose causal connections on the complete storage structure. This implies that a forensic investigator can simplify the search for

causality patterns in existing allocation systems by looking for the existence of generation markers. A systems designer would simplify subsequent forensic investigations of the designed system by adding explicit generation markers in the allocation systems used in the designed system.

4.4 Sequence numbers

Sequence numbers is a feature occurring in many digital systems, such as file systems and networks. By using a sequence number, the systems designer ensures that sequence numbered entities can be ordered in the correct order or be distinguished from each other. Sequence numbers are usually implemented by using a counter that increases whenever a new sequence numbered entity is produced and associating a copy of the value of the counter (the sequence number) with that entity. The implementation of sequence numbers are similar to the implementation of clocks, with the exception that the increment of the counter does not occur at regular intervals, so the sequence number cannot be used as an approximation of time. It is useful to distinguish between *wrapping sequence numbers* and *strictly increasing sequence numbers*. In wrapping sequence numbers, the counter has a limited span of values. When the highest value is reached, the counter wraps and starts at the lowest value. A strictly increasing sequence number on the other hand is a sequence number that does not wrap in this fashion. In theory a strictly increasing sequence number would have to be able to represent infinite numbers. In practice however, a sequence number can be viewed as strictly increasing as long as the number of values that can be represented in the sequence number is large enough to produce strictly increasing numbers over a significant time span, for example the lifetime of a computer.

When observing a system of sequence numbered entities, the distinction between wrapping sequence numbers and strictly increasing sequence numbers is important. With a wrapping sequence number, one would not be able to know how many times the counter had wrapped at the time of the generation of the sequence number. When correlating two entities with sequence numbers, one would therefore not be able to determine if one of the entities was produced before the other. In a system with strictly increasing sequence numbers on the other hand, one would be sure that the entity with

the highest sequence number had been produced after the entity with the lowest sequence number.

In a system with strictly increasing sequence numbers, causality can be inferred between the events of production of sequence numbered entities. Each generation of a sequence numbered entity is causally dependant on the generation of every other sequence numbered entity with lower sequence number. This causality property is very useful, as will be seen in section 4.5.5.

4.5 Allocation causality in file systems

Because most file systems record timestamps on a non-volatile medium for events in the file system, the causality properties of file systems is of particular interest to the forensic investigator. If causal connections between events that cause timestamps to be updated can be found, these relations can be utilized to impose strict restrictions on the clock hypothesis, by using the methods described in Chapter 3.

Carrier divides the structures of a file system into the following categories: [31]

File system category: structures containing general file system information
Content category: structures containing the content of files
Metadata category: structures with data describing files such as file size, where content is stored, timestamps and other metadata
File name category: structures storing data that contain the names of the files, typically as directories with contents
Application category: structures containing data pertaining to special features of the file system

Each of these structures may have their own allocation strategy, and should therefore be investigated separately for possibilities of inferring causal connections. Allocation strategies used in file system are not codified in the file system itself, but are rather features of the file system driver in the operating system. Allocation strategies may therefore differ between different operating systems, even if the same file system has been used.

4.5.1 File system category

The file system category structures contains general data about the file system and where important data is located on the disk, such as the Master File Table in NTFS and the superblock and block group description tables in UNIX file systems. These structures are in most cases allocated at the creation of the file systems. Causal relations between their creations are therefore usually not especially relevant to the actions of the user. In the context of forensic investigations, it is more interesting to investigate the contents of the structures pointed to by the structures in the file system category.

4.5.2 Content category

The content category structures contain the data of the files stored in the file system. The ability of performing causality reasoning on the allocation of these structures depends on the ability to identify timestamp information for the creation of a file on the file system, as well as the allocation strategy used and the existence of generation markers.

Allocation strategies used in different operating systems may vary, and different strategies such as next-fit (in Windows 98 FAT), best-fit (in Windows XP NTFS) and first-fit (in Linux Ext2) have been observed in practice. [31] With such allocation strategies, allocation causality could be inferred if it could be identified which generation each allocation unit belongs to. Explicit generation markers are however usually not implemented in the content category. Identification of which generation a content data storage unit belongs to is not required in most file systems, since the data contained in the other categories keep track of which data pertain to which files, and whether data storage units are currently free or not. Identification of data generation in the content category must therefore be inferred from implicit data available based on knowledge of how the file system works. One possible such generation identification is the analysis of data known as file slack. File slack is data stored in a content category allocation unit that does not pertain to the current file stored in it. Such data may exist because a file of greater length than the current file may previously have been stored in the allocation

unit and deleted. From the existence of file slack data, one may infer that the contents of the slack belong to a previous generation than the contents of the current file stored in the allocation unit. It is therefore a causal relation between the storage of the file slack data and the storage of the current file. The impact of this causal connection is however limited. Identification of file metadata and timestamps pertaining to the file in slack space may be difficult or impossible. Further, such analysis may have to be done manually, something that would require a very large amount of work if such analysis were to be done for all files on a file system. This limits the value of file slack analysis for clock hypothesis testing, since many causal connections should be identified to scrutinize the clock hypothesis.

Another theoretical possibility for generation identification of content data is the identification of the data generation by low level analysis of the signal on the medium. It has been postulated that such analysis is possible by using special equipment to read the analog signal stored on magnetic platters in hard drives. [32] It is however to date unknown whether this technique is possible in practice. The technique has not been demonstrated on recent or current hard drives. The complexity and density of current hard drives suggest that such analysis would be very difficult, if possible at all. It's worth noting that if such a technique was indeed possible, it would have to be done on the original evidence medium. It would therefore challenge the current paradigm for digital investigations, in which the original medium is always copied to another medium before it is analyzed.

In summary, identification of generation-markers for data structures in the content category seems to be impractical for current file systems.

4.5.3 Metadata category

Data structures in the metadata category contain metadata about the files stored in the content category structures. These structures are stored in their own data areas and have their own allocation algorithms. Each allocation unit usually stores metadata information pertaining to one file. The NTFS Master File Table (MFT) and file inodes in UNIX file systems are examples of data structures in the metadata category. Like other storage systems, the data units in the metadata category may be deleted and reused. Deletion of

a metadata allocation unit usually occurs when the file to which the metadata pertain is deleted.

The possibility for finding causal connections depends on, like the other categories, the existence of the allocation strategy and the existence of a generation marker. Unlike the content category, for the metadata category, explicit generation markers do exist in some systems. The file entries the NTFS Master File Table is a good example.

Each file stored in the NTFS file system has its own entry in the Master File Table. Each entry occupies two 512-byte blocks on the disk. Data stored in the MFT entry include the file name, the list of data runs where the file data is stored, timestamps and other data such as information on whether the data is compressed or encrypted using the compression and encryption features in the file system. The file entry in the MFT is the central reference point for each file. It is created when a file is stored on the file system, and deleted (but not wiped) when a file is deleted. Allocation of file entries within the MFT occurs on a first-fit basis. [31] Each file entry contains a generation marker, termed *sequence number*, which identifies the generation of the usage of that entry. This number is increased whenever the file entry is reused. With a first-fit allocation algorithm, this system is exactly the system described in section 4.2. In this system, the storage of each new generation in a specific location always depends causally on the storage and deletion of previous generations. All elements in the first generation depend on all previously stored elements of that generation. Thus, causal connections exist between file entries in the Master File Table of the NTFS file system, under the reasoning described in section 4.2.

Example 4.4. Consider the following set of allocated file entries from an NTFS Master File Table. Let e_i be the storage of the current data in entry i.

```
Entry 45 sequence number 0
Entry 46 sequence number 2
Entry 47 sequence number 0
Entry 48 sequence number 1
Entry 49 sequence number 5
Entry 50 sequence number 0
Entry 51 sequence number 3
```

Since $\forall j < i(e_{s_{j,0}} \rightarrow e_{s_{i,0}})$ it can be inferred that $e_{45} \rightarrow e_{47} \rightarrow e_{50}$. Further, since also $e_{s_{j,0}} \rightarrow e_{s_{i,g}}$, we also have $e_{45} \rightarrow e_{46}$, $e_{47} \rightarrow e_{48}$, $e_{47} \rightarrow e_{49}$ and $e_{50} \rightarrow e_{51}$. The resulting causality graph is:

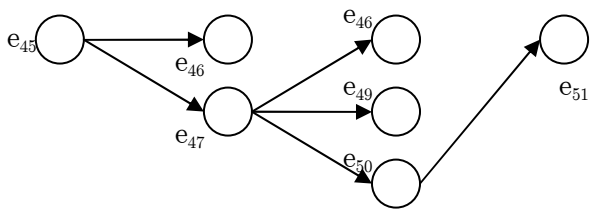

Figure 4.9 Event and happened-before graph in Example 4.4

4.5.4 File name category

Data structures in the file name category contain indices over file names stored in the file system, such as directories. The determination of causal connections between their allocation depends on the allocation algorithm and, unless append-only is used, the possibility of identifying the generation. Some file systems use linear allocation for directory indices, while some use more advanced data structures such as B-trees. In either case, the possibility of deletion of file names and reuse of the storage area in question means that it would be necessary to identify the data generation in order to find causal connections. Since explicit generation markers are not common in file system directory indices, finding causal connections in the allocation in directories may not be possible.

4.5.5 Application category

Some file systems contains special data pertaining to specific features of the file system or operating system. One such specific feature of special interest is File System Journals. When using a file system with journaling, the operating system logs all changes to file system metadata before and after the change is written to disk. This prevents file system inconsistencies resulting from system crashes. When booting after a crash, the system can remove any inconsistencies by checking the journal for actions that were started but not finished.

The possibility to find causal connections in the use of a journal depends on the allocation strategy used in the journal file. A common strategy in journal files is to use an append-only strategy (used in for example Ext3 and NTFS). Each file system action descriptor is appended to the end of the previous descriptor already stored in the file system journal. Since the file system journal has limited size, there must be identification of where the current journal file starts and to which extent old data has been overwritten. This is commonly implemented by using a strictly increasing sequence number. For example, in NTFS, journal file transactions are labelled with a 64-bit number (so called Logical Sequence Number - LSN) that increases throughout the lifetime of the file system. The proper functioning of the journaling feature in NTFS depends on this number being strictly increasing. [33]

On this basis, it is possible to find causal connections by analyzing journal files. The amount of information that can be derived from the journal file itself is however limited. Since every write to a file produces a journal file entry and the journal file has limited size, old entries will quickly be overwritten. It is common for operating systems to restart the log file at every boot of the operating system, thereby overwriting the data written in previous sessions. Due to the limited size of the journal file, the most interesting use of journaling to find causal connections comes from the use of the sequence number in the journal file. Some file systems, such as NTFS, store the journal file sequence number (the LSN in NTFS) in the file metadata entry. If the journal file sequence number is strictly increasing, then it means that the events that generate it are causally connected, as discussed in section 4.4. On such systems, it is possible to impose causal connections on the events of the last change of the file entry on all files stored on the file system.

Example 4.5. Consider the following set of allocated file entries from an NTFS Master File Table. Let e_i be the last update of the current data in entry i.

```
Entry 45 log file sequence number 432627
Entry 46 log file sequence number 186345
Entry 47 log file sequence number 735294
Entry 48 log file sequence number 165093
Entry 49 log file sequence number 878121
Entry 50 log file sequence number 782427
Entry 51 log file sequence number 561987
```

Since logical sequence numbers in the journal file (log file) are allocated sequentially, we can obtain the causal ordering of the last update events by sorting the file entries by their log file sequence number: $e_{48} \rightarrow e_{46} \rightarrow e_{45} \rightarrow e_{51} \rightarrow e_{50} \rightarrow e_{47} \rightarrow e_{49}$.

4.5.6 File system causality

The above discussion has shown that causal connections can be found in the analysis of allocation in several of the storage categories in file systems. Such inferences would provide the necessary basis for the formulation and testing of clock hypotheses. File systems provide the foundations for data storage in digital systems investigated in digital investigations. On a normal medium investigated in a digital investigation, there are tens of thousands or even hundreds of thousands of files. It has been found that, depending on the implementation of the file system, there may exist several ways in which all files on a file system are causally connected. When all these files are causally connected, and there exists timestamps for the events, a strict boundary on a clock hypothesis is produced by the tests investigated in Chapter 3. Such usage of file system causal connections to provide clock hypothesis bounds will be further investigated in Chapter 7.

5 TIMESTAMP REASONING WITH EVENT CALCULUS

Chapter 3 - 4 introduced clock hypothesis testing and how causality relations can be found in storage systems. This Chapter extends this system by introducing a model for systems containing timestamps. The model uses fluents to represent timestamps, and actions to represent events changing them. Since actions may change more than one timestamp at the same time, the creation of timestamps is correlated. This property can be utilized to put a clock hypothesis under additional scrutiny. This Chapter uses a simple theoretical file system to illustrate the process.

Section 5.1 introduces the Simplified Event Calculus. Section 5.2 shows how timestamps can be formulated in this calculus. Sections 5.3 - 5.4 discusses how action hypotheses can be derived from observation sets. Section 5.5 - 5.8 then shows how the system model can be used to test a clock hypothesis. In Section 5.9 - 5.10, invariants is derived to be used for clock hypothesis testing from a more advanced theoretical file system, bearing closer resemblance to a real file system.

5.1 Introduction to Simplified Event Calculus

In this chapter, a form of reasoning will be developed for finding properties that can be used for clock hypothesis testing with the methods developed in Chapter 3. The reasoning is based on modelling these systems in Simplified Event Calculus (SEC), a calculus for reasoning on change, defined by Shanahan. [34]

Simplified Event Calculus is a many-sorted predicate calculus, in which reasoning on change is performed by means of predicates describing properties at particular moments in time and events that can change these properties. In Simplified Event Calculus, the world is modelled with *time*, *fluents* and *actions*.

Definition 5.1. A language of the Simplified Event Calculus is a many-sorted first-order predicate calculus with equality, which includes:

- A sort for fluents, with variables f, f_1, f_2, etc.
- A sort for actions, with variables a, a_1, a_2, etc
- A sort for time, with variables t, t_1, t_2, etc
- the following predicates, whose arguments are fluents, actions and times:
  HoldsAt(f, t) - fluent f holds at time t
  Happens(a, t) - action a occurs at time t
  Initiates(a, f, t) - action a causes fluent f to hold if it happens at time t
  Terminates(a, f, t) - action a causes fluent f not to hold if it happens at time t
  Clipped(t_1, f, t_2) - fluent f ceased to hold in the period between t_1 and t_2
  Initially(f) - fluent f held at $t = 0$

A fluent in the Simplified Event Calculus represents any property in the modelled world. The existence of a fluent at a specific moment in time is denoted with the HoldsAt predicate. An action in the Simplified Event Calculus represents an occurrence that can cause a property to change. The occurrence of an action at a specific moment in time is denoted with the Happens predicate. The change introduced by an action is modelled with the predicates Initiates and Terminates. These predicates represent the change introduced by the occurrence of an action at a specific moment in time; causing a fluent to hold (Initiates) or causing a fluent not to hold (Terminates). The predicate Clipped represents that a fluent ceased to hold in a specific time period. The predicate Initially allows the representation of fluents existing before the events represented in a particular model.

Shanahan defines effect axioms for the Simplified Event Calculus as follows:

$$\text{HoldsAt}(f,t_2) \Leftarrow \text{Happens}(a,t_1) \wedge \text{Initiates}(a,f,t_1) \wedge t_1 < t_2 \wedge \text{not Clipped}(t_1,f,t_2) \quad (5.1)$$

$$\text{Clipped}(t_1,f,t_2) \Leftarrow \text{Happens}(a,t) \wedge \text{Terminates}(a,f,t) \wedge t_1 < t < t_2 \quad (5.2)$$

$$\text{HoldsAt}(f,t) \Leftarrow \text{Initially}(f) \wedge \text{not Clipped}(0,f,t) \quad (5.3)$$

Informally, the effect axioms can be interpreted as:

- If an action happening at time t_1 initiated a fluent f, than the fluent f still holds at time t_2, if it has not been Clipped between t_1 and t_2 (5.1),

- If a fluent held Initially, then it will still hold at time t, if it has not been Clipped between 0 and t (5.3)
- A fluent is Clipped between t_1 and t_2 if an action occurring between t_1 and t_2 terminates that fluent. (5.2)
- Whenever a fluent holds at time t, it must be either because it held Initially (5.3), or because some action Initiated it. (5.1)

Simplified Event Calculus is used in logic programs for reasoning on change. Shanahan defines an event calculus program as follows.

Definition 5.2. An *event calculus program* is the conjunction of,

- A finite set S of Initially clauses of the form,

$$\text{Initially}(f)$$

- A finite set A of Happens clauses of the form,

$$\text{Happens}(a,t)$$

- A finite set E_{I} of Initiates clauses and a finite set E_{T} of Terminates clauses of the form,

$$\text{Initiates}(a,f_1,t) \Leftarrow \Pi$$
$$\text{Terminates}(a,f_1,t) \Leftarrow \Pi$$

where Π does not mention the predicates Initially, Happens, Initiates or Terminates and every occurrence of the HoldsAt predicate is of the form

$$\text{HoldsAt}(f_2,t)$$

- The effect axioms of simplified event calculus, eq. (5.1) - (5.3)

- A finite set of general clauses not mentioning the predicates Initially, Happens, Initiates, Terminates or $<$.

The representation of a model in a logic program allows for the use of resolutions to prove or disprove claims expressed with a predicate. This involves substituting each clause in the claim with all possible substitutions according to the clauses and axioms of the program. The resolution of a claim yields a search space, in the form of a resolution tree such as the tree found in Figure 5.1. From a resolution, it can be determined if the claim is true, and if so, under which conditions. In the next sections, resolutions will be used to explore properties of file systems represented with the Simplified Event Calculus.

A specific feature of the Simplified Event Calculus that should be noted is the use negation-as-failure, a form of default reasoning utilized to avoid having to write explicit propositions for everything that does not change whenever an action happens.[2] Negation-as-failure is represented by using the predicate operator "not", instead of the predicate operator $\neg$. "Not" implies using negation-as-failure instead of explicit negation. Whereas $\neg\text{Clipped}(t_1, f, t_2)$ would require a proposition explicitly stating that there was no terminating action occurring in the time period, $\text{not Clipped}(t_1, f, t_2)$ will only be false if there is actually such a terminating action, and true otherwise. Thus, explicit propositions defining conditions for the persistence of fluents not affected by actions do not have to be written.

For further reading on the background and semantics of Simplified Event Calculus, the reader is referred to Shanahan's original text. Note that this work uses the symbol $\Leftarrow$ for implication in the Simplified Event Calculus, to avoid confusion with the happened-before relation. This distinction from Shanahan's text is purely symbolic and is not intended to imply a semantic difference.

5.2 Representation of timestamps in Simplified Event Calculus

[2] The problem of having to represent non-change is called *the frame problem* in Shanahan's and other works.

With the definitions in Definition 5.2, changes in timestamps on a computer system can now be represented as an event calculus program, where the fluents are timestamps with an associated clock value, and actions are the operations on the computer system that might change the timestamps. In accordance with Shanahan, we consider that the interpretation of the time sort is by the reals, and that $<$ and $=$ is interpreted accordingly. This interpretation is consistent with the assumptions in Section 3.2.

When representing a system with timestamps in Simplified Event Calculus, we must start with defining sets of fluents and actions for our event calculus program. For example, in a file system, there might be several timestamps associated with each file. We may then represent each timestamp with a fluent type, pertaining to a specific file, and holding a specific clock value. In a file system where the last access time and modified time of a file is stored, timestamps of a specific file can be represented with fluents

$$\text{Accessed}(\mathit{file},\tau_a)$$
$$\text{Modified}(\mathit{file},\tau_m)$$

Further, operations in the file system that could potentially change the value of the timestamps can be represented with actions

$$\text{Read}(\mathit{file})$$
$$\text{Write}(\mathit{file})$$

The change introduced by Read and Write, would be represented in the event calculus program as a set of Initiates and Terminates clauses. It could for example be that the Read action sets the Accessed timestamp of a file to the current system clock $c(t)$ and the Write action sets both the Accessed and Modified timestamps to the current system clock $c(t)$:

$$\text{Initiates}(\text{Read}(\mathit{file}),\text{Accessed}(\mathit{file},c(t)),t) \tag{5.4}$$
$$\text{Initiates}(\text{Write}(\mathit{file}),\text{Accessed}(\mathit{file},c(t)),t) \tag{5.5}$$

$$\text{Initiates}(\text{Write}(\mathit{file}),\text{Modified}(\mathit{file},c(t)),t) \tag{5.6}$$

In a real file system, the previous clock values stored as timestamps for a file, would be overwritten when new timestamps were written for that file. This effect must be represented as a set of Terminated clauses:

$$\text{Terminates}(\text{Read}(\textit{file}),\text{Accessed}(\textit{file},c(t_1)),t) \Leftarrow t_1 < t \tag{5.7}$$

$$\text{Terminates}(\text{Write}(\textit{file}),\text{Accessed}(\textit{file},c(t_1)),t) \Leftarrow t_1 < t \tag{5.8}$$

$$\text{Terminates}(\text{Write}(\textit{file}),\text{Modified}(\textit{file},c(t_1)),t) \Leftarrow t_1 < t \tag{5.9}$$

In most real file systems there is always a value assigned to the time stams of a file. It therefore makes sense to define Initially clauses that initiates fluents for the timestamps that holds from the start:

$$\text{Initially}(\text{Accessed}(\textit{file},\tau_0)) \tag{5.10}$$

$$\text{Initially}(\text{Modified}(\textit{file},\tau_0)) \tag{5.11}$$

With the conjunction of the above formulae, a model of a simple file system has been defined. This file system has a fixed set of files. Each file always has two timestamps; the *Accessed* timestamp and the *Modified* timestamp. These timestamps may have an initial value, and may be changed by two actions; the *Read* action and the *Write* action. The Read action sets the Accessed timestamp to the current value of the system clock, whereas the Write action sets both the Accessed timestamp and the Modified timestamp to the current clock value. There is only one timestamp of each type for each file, so whenever a timestamp is changed, the previous value is lost.

In this model, S is the conjunction of formulae (5.10) - (5.11) and E is the conjunction of formulae (5.4) - (5.9). With a definition of a set A of Happens clauses, an event calculus program for this simple file system has been completed. Then, resolutions can be utilized to search the space of possible event histories and test propositions about fluents at particular moments in time.

Example 5.3. Let the file "file1" be Read at $t = t_r$ and subsequently written at $t = t_w$, such that $t_r < t_w$. Let $c(t)$ be an integer, such that $\tau_0 = 0$, $c(t_r) = 5$ and $c(t_w) = 10$.

Let an event calculus program be defined by (5.4) - (5.11) and the following clauses as A:

$$\text{Happens}(\text{Read}(\text{file1}), t_{\text{r}})$$
$$\text{Happens}(\text{Write}(\text{file1}), t_{\text{w}})$$

The value of timestamps at certain moments in time can now be examined by means of resolutions. For example, let us determine if the accessed timestamp has value 10 at time $t = t_{\text{obs}}$, where $t_{\text{r}} < t_{\text{w}} < t_{\text{obs}}$.

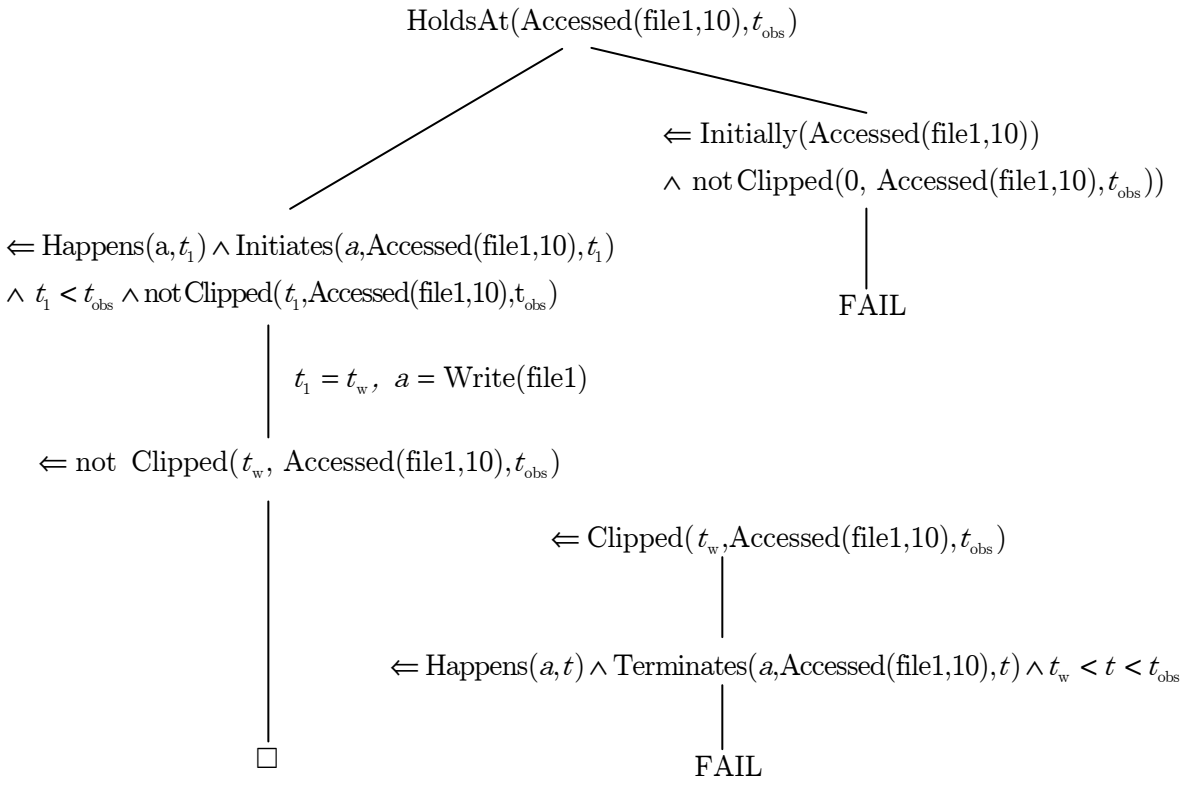

Figure 5.1 Resolution of $\text{HoldsAt}(\text{Accessed}(\text{file1},10), t_{\text{obs}})$ in Example 5.3

Figure 5.1 shows a resolution for the observation of the Accessed timestamp given a specific observation time. The right hand branch of the resolution, representing the case that the timestamp was initially set to the observed value fails due to the fact that there is no Initially clause setting the Accessed timestamp to 10. The left hand branch of the resolution represents the occurrence of an action a at time t_1 initiating the fluent $\text{Accessed}(\text{file1},10)$. The only Happens clause that can satisfy this is Hap-

pens(Write(file1),t_w). Since the evalution of the clause Clipped(t_w,Accessed(file1,10),t_{obs}) fails, the left branch shows that HoldsAt(Accessed(file1,10),t_{obs}) holds.

Example 5.4. Consider the event calculus program from Example 5.3. Determine if it could be the case at $t = t_{obs}$, where $t_r < t_w < t_{obs}$, that the accessed timestamp is set to value 5 and the modified timestamp is set to value 10. The proposition to be tested is then

$$\text{HoldsAt}(\text{Modified}(\text{file1},10),t_{obs}) \wedge \text{HoldsAt}(\text{Accessed}(\text{file1},5),t_{obs}) \qquad (5.12)$$

It is practical to evaluate each of the HoldsAt clauses in the proposition separately. Since both clauses have to hold for the proposition to be true, none of the resolution trees must fail, should the proposition hold. The resolution for the first clause is shown in Figure 5.2 and the second in Figure 5.3.

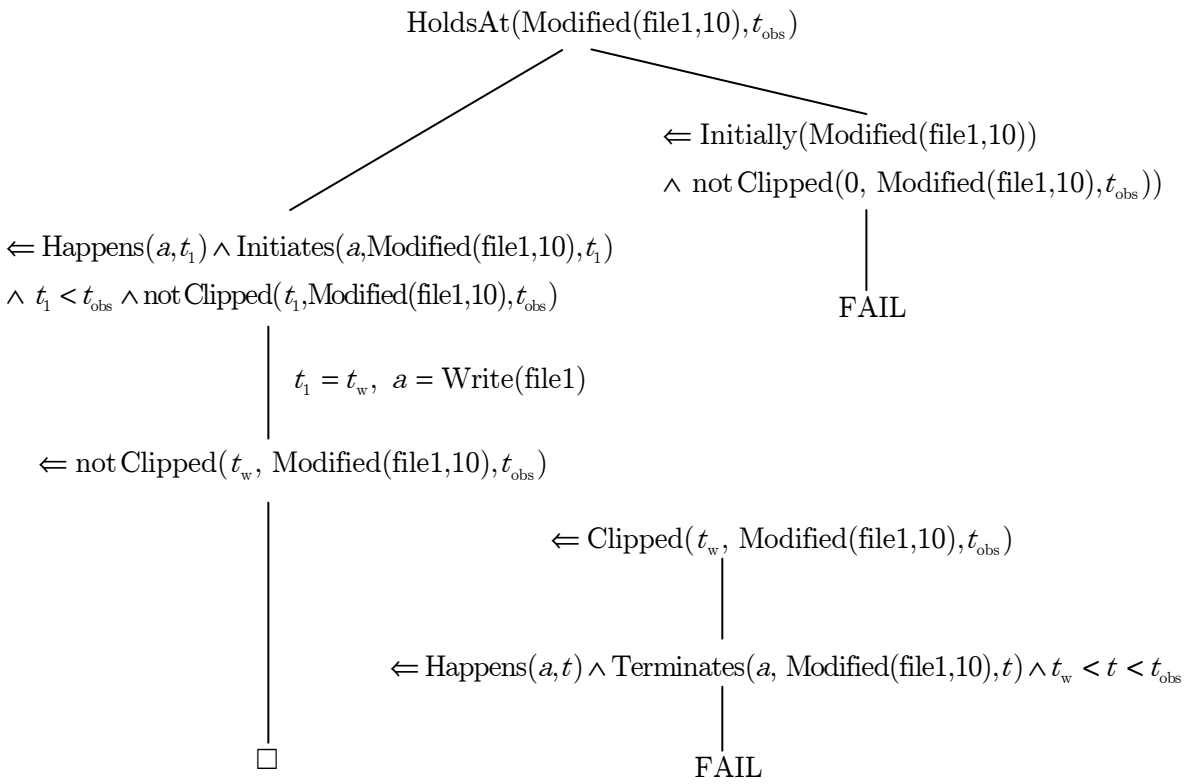

Figure 5.2 Resolution of HoldsAt(Modified(file1,10),t_{obs}) in Example 5.4

As shown on the figures, the term HoldsAt(Modified(file1,10),t_{obs}) holds, but the term HoldsAt(Accessed(file1,5),t_{obs}) fails, since the Accessed(file1,5) fluent initiated by Happens(Read(file1),t_r) is Clipped by Happens(Write(file1),t_w). Consequently, the access timestamp cannot have the value 5 at t_{obs}.

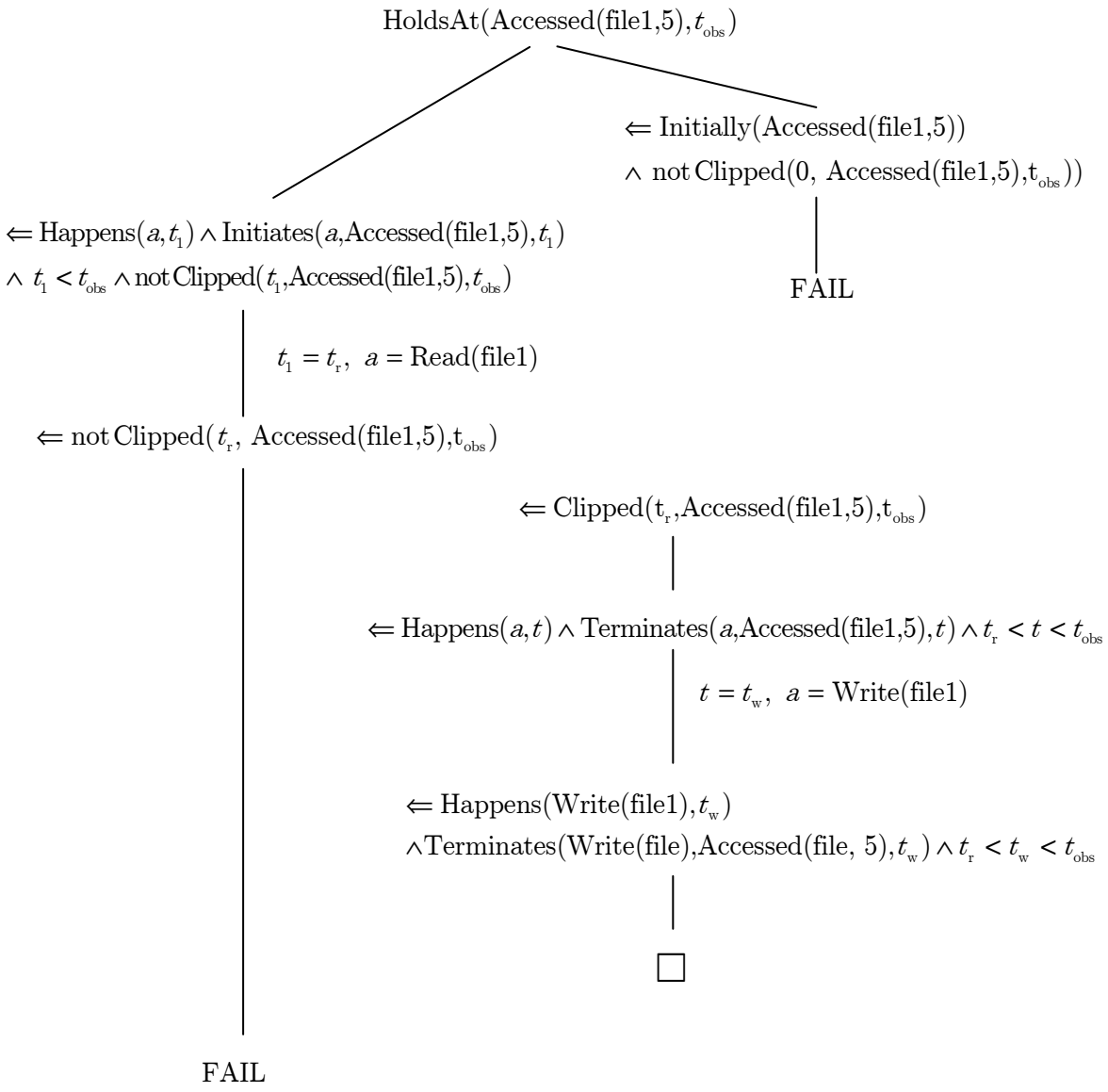

Figure 5.3 Resolution of HoldsAt(Accessed(file1,5),t_{obs}) in Example 5.4

5.3 Observation sets and action hypotheses

The examples in the previous section has shown how Simplified Event Calculus can be utilized to determine if specific timestamp values holds at a specific moment in time, given known occurrence of actions. Definition 3.12 introduced the concept of an observation set. An observation set is the set of timestamps observed by an investigator at the time of the investigation. In Simplified Event Calculus, timestamps are represented as fluents that may or may not hold at a specific moment in time. Formulated in Simplified Event Calculus, an Observation set is therefore a set of fluents that holds at the time of the observation t_{obs}.

Definition 5.5. Formulated in the Simplified Event Calculus, an *Observation Set O* is a finite set of HoldsAt clauses on the form

$$\text{HoldsAt}(f, t_{obs})$$

representing fluents f holding at the time of the observation t_{obs}. The *observation proposition* is the conjunction of the HoldsAt clauses in the observation set. The observation proposition has the form

$$o = \varphi_1 \wedge \varphi_2 \wedge ... \wedge \varphi_n$$

Where each φ is a HoldsAt clause contained in O, and n is the number of elements in O.

With the definition of an observation set, we can start investigating the relationship between an observation set and the sets S, A and E defining an event calculus program. As shown in the previous sections, an event calculus program defines the behaviours occurring in a system in terms of the initial state (S), the effect any actions would have on the states (E) and the actions that actually occurred (A). With known S, E, and A, possible states at a specific moment in time could be tested for consistency with the event calculus program. In Example 5.3 the tested state was consistent, whereas in Example 5.4 the tested state was inconsistent with the event calculus program. When S, E and A are known, resolutions can be used to test observation propositions and therefore confirm or refute possible observation sets O.

From the examples,

$$O = \{\text{HoldsAt}(\text{Accessed}(\text{file1},10),t_{obs})\}$$

was confirmed, whereas

$$O = \{\text{HoldsAt}(\text{Modified}(\text{file1},10),t_{obs}),\ \text{HoldsAt}(\text{Accessed}(\text{file1},5),t_{obs})\}$$

was refuted as a possible observation set for S, A and E. These examples show how possible observation sets can be tested for consistency with an event calculus program.

As we have already seen in Chapter 3, things are different in an investigation situation. In an investigation, the state at the time of the investigation is observable, whereas information about the sets defining the event calculus program is lacking. Under the assumption that the investigator has all information about the initial state S, and also thorough knowledge about the workings of the system, E, the investigator can use the knowledge about the observed state O to derive information about occurred events. In this case A is unknown, whereas S, E and O are known. The investigator can now infer knowledge about A from the observation set O and the detailed knowledge about how the system works, represented by S and E.

Returning to Example 5.3, if the observed set is $O = \{\text{HoldsAt}(\text{Accessed}(\text{file1},10),t_{obs})\}$ and A is unknown, the investigator can now reason that since (from O) the fluent Accessed(file1, 10) holds at the time of the observation and since (from S) initially Accessed(file1, 0), some action must have occurred that terminated Accessed(file1, 0) and initiated Accessed(file1, 10). From E, the investigator knows that this must have been an action occurring at $t = t_a$, where $c(t_a) = 10$. The investigator also knows that the action must have been either a Read or a Write action, since (again, from E) these are the only actions that can affect the Accessed fluent. The investigator can therefore formulate two hypotheses about occurred actions, H_1 and H_2 where $c(t_a) = 10$.

$$H_1 = \{\text{Happens}(\text{Read}(\text{file1}),t_a)\}$$
$$H_2 = \{\text{Happens}(\text{Write}(\text{file1}),t_a)\}$$

These hypotheses can be tested by resolution of the observation proposition; something that will produce resolutions similar to that shown in Figure 5.2 for both H_1 and H_2, and both hypotheses will be accepted. H_1 and H_2 are hypotheses about actions that actually

took place. If hypotheses about occurred actions are accepted by an event program resolution, it means that they are possible explanations for the observed set O. The hypotheses, even if they are accepted, do not imply full knowledge of the set of actions A. Even if a hypothesis is accepted, it is still in the unknown if there were any actions in A for which there exist no evidence anymore. In Example 5.3, it could for example be the case that the file was Read at some moment prior to t_a. The timestamp fluent resulting from this Read would be Terminated by the Read occurring at t_a, and therefore not be observable at t_{obs}.

The above discussion leads to the following definition:

Definition 5.6. An action hypothesis H is a finite set of Happens clauses on the form

$$\text{Happens}(a, t)$$

derived from an observation set O, given finite sets S and E in an event calculus program.

5.4 Formulating action hypotheses

The acceptance of an action hypothesis means that it is a possible set of actions that can explain the observation set O. It does not however guarantee that there are no other possible explanations. In order to be able to deduct possible courses of events from an observation set, we would like to find all possible hypotheses H, given an observation set O and knowledge about the system, represented by S and E.

From Definition 5.5 the elements of an observation set O are HoldsAt clauses representing the fluents that holds at the time of the observation. The observation proposition to be tested in the event calculus program is the conjunction of these HoldsAt clauses and takes the form

$$o = \varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n \tag{5.13}$$

where each φ is a HoldsAt clause.

As per the effect axioms of the Simplified Event Calculus, these HoldsAt clauses may exist either because they held initially (eq. (5.3)) or because an action occurred that initiated them (eq (5.1)). There is no other way a HoldsAt clause can come to existence than through eq. (5.1) or (5.3). It is therefore possible to find all action hypotheses explaining the observation set O by reasoning on the observation proposition, the Initiates clauses in E and the Initially clauses in S. This reasoning does not have to consider termination of fluents as per the Terminates clause in E, since this will be done by means of resolution when each hypothesis is tested for acceptance. The proposition that all fluents in an observation set has been initiated is the conjunction of the initiation of each fluent and takes the form:

$$q = \kappa_1 \wedge \kappa_2 \wedge ... \wedge \kappa_n \tag{5.14}$$

where each κ is the initiation of the corresponding φ in the observation proposition o. In the following, this proposition will be called the *initiation proposition*.

A fluent may exist because it held initially or because it was Initiated by a clause in E. There may be more than one Initiates clause in E initiating one particular fluent, and these must all be considered. Written in propositional logic, the initiation of a HoldsAt clause takes the form of a disjunction:

$$\kappa_i = \alpha_{i1} \vee \alpha_{i2} \vee ... \vee \alpha_{iu_i} \vee \eta_i \tag{5.15}$$

Where κ_i is the i-th HoldsAt(f, t_2) clause in q, η_i is an Initially(f) clause, u_i is the number of Initiates(a, f, t_1) clauses affecting that fluent and each α_i is a clause on the form

$$\text{Happens}(a, t_1)$$

where there exists a clause Initiates(a, f, t_1) in E.

The initiation of the fluents in the observation proposition can now be found by inserting (5.15) into (5.14), yielding

$$
\begin{aligned}
q \;=\; & (\alpha_{11} \vee \alpha_{12} \vee \ldots \vee \alpha_{1u_1} \vee \eta_1) \\
\wedge\; & (\alpha_{21} \vee \alpha_{22} \vee \ldots \vee \alpha_{2u_2} \vee \eta_2) \\
\wedge\; & \ldots \\
\wedge\; & (\alpha_{n1} \vee \alpha_{n2} \vee \ldots \vee \alpha_{nu_n} \vee \eta_n)
\end{aligned}
$$

q is a conjunction of disjunctive clauses. By reordering it into a disjunction of conjunctive clauses, a set of action hypotheses can be found, where each of the conjunctive clauses in the disjunction is an action hypothesis H.

Example 5.7. Consider an event calculus program with S and E as defined in Example 5.3 and O as defined by the observation proposition in Equation (5.12). Then a set of action hypotheses can be found by the following reasoning:

The observation proposition is given in (5.12)

$$p = \text{HoldsAt}(\text{Modified}(\text{file1},10), t_{\text{obs}}) \wedge \text{HoldsAt}(\text{Accessed}(\text{file1},5), t_{\text{obs}})$$

The initiation of these fluents can then be expressed as a conjunction of disjunctive clauses as follows, where $c(t_{\text{a}}) = 5$ and $c(t_{\text{m}}) = 10$:

$$
\begin{aligned}
q \;=\; & (\text{Happens}(\text{Write}(\text{file1}), t_{\text{m}}) \\
& \quad \vee\; \text{Initially}(\text{Modified}(\text{file1},10))) \\
\wedge\; & (\text{Happens}(\text{Read}(\text{file1}), t_{\text{a}}) \\
& \quad \vee\; \text{Happens}(\text{Write}(\text{file1}), t_{\text{a}}) \\
& \quad \vee\; \text{Initially}(\text{Accessed}(\text{file1},5)))
\end{aligned}
$$

Since there is no Initially(Modified(file1,10)) or Initially(Accessed(file1,5)) in S, we know that these clauses are false. q then becomes:

$$
\begin{aligned}
q \;=\; & \text{Happens}(\text{Write}(\text{file1}), t_{\text{m}}) \\
\wedge\; & (\text{Happens}(\text{Read}(\text{file1}), t_{\text{a}}) \\
& \quad \vee\; \text{Happens}(\text{Write}(\text{file1}), t_{\text{a}}))
\end{aligned}
$$

Rewritten as a disjunction of conjunctive clauses:

$$q = \text{Happens}(\text{Write}(\text{file1}), t_{\text{m}}) \wedge \text{Happens}(\text{Read}(\text{file1}), t_{\text{a}}) \\ \vee \text{Happens}(\text{Write}(\text{file1}), t_{\text{m}}) \wedge \text{Happens}(\text{Write}(\text{file1}), t_{\text{a}})$$

So here we obtain two different hypotheses from the fluent initiation:

$$\text{H}_1 = \{\text{Happens}(\text{Write}(\text{file1}), t_{\text{m}}),\ \text{Happens}(\text{Read}(\text{file1}), t_{\text{a}})\}$$
$$\text{H}_2 = \{\text{Happens}(\text{Write}(\text{file1}), t_{\text{m}}),\ \text{Happens}(\text{Write}(\text{file1}), t_{\text{a}})\}$$

5.5 Using event calculus to test a clock hypothesis

The defined system can be used to put additional constraints on the clock hypothesis that does not follow directly from the reasoning in chapter 3. The investigator can test clock hypotheses for consistency with the observed set O, given knowledge of the initial state S and how the system works, E. Such testing is performed by deriving hypotheses about occurred events from the observation set using the method in section 5.4, and then test them with the clock hypothesis $c_{\text{h}}(t)$ by using resolutions. If no action hypothesis is accepted in the event calculus program for a clock hypothesis, then that clock hypothesis is refuted. If one or more action hypotheses are accepted for a clock hypothesis, then that clock hypothesis is accepted, and the action hypotheses describe the possible sequence of events that could bring the system into the observed state.

Example 5.8. Consider the *default clock hypothesis*, where it is assumed that the clock of the investigated computer has always been equal to civil time, say UTC. Then $c_{\text{h}}(t) = b(t)$ and $d(t) = 0$. Now, assume the event calculus program and observation set of Example 5.7. Then the hypotheses to be tested are:

$$\text{H}_1 = \{\text{Happens}(\text{Write}(\text{file1}), t_{\text{m}}),\ \text{Happens}(\text{Read}(\text{file1}),\ t_{\text{a}})\}$$
$$\text{H}_2 = \{\text{Happens}(\text{Write}(\text{file1}), t_{\text{m}}),\ \text{Happens}(\text{Write}(\text{file1}), t_{\text{a}})\}$$

Now, since $c_{\text{h}}(t_{\text{a}}) = 5$ and $c_{\text{h}}(t_{\text{m}}) = 10$ and $c_{\text{h}}(t)$ is an ideal clock, $t_{\text{a}} < t_{\text{m}}$. H_1 was tested in Example 5.4, where it was found to be refuted.

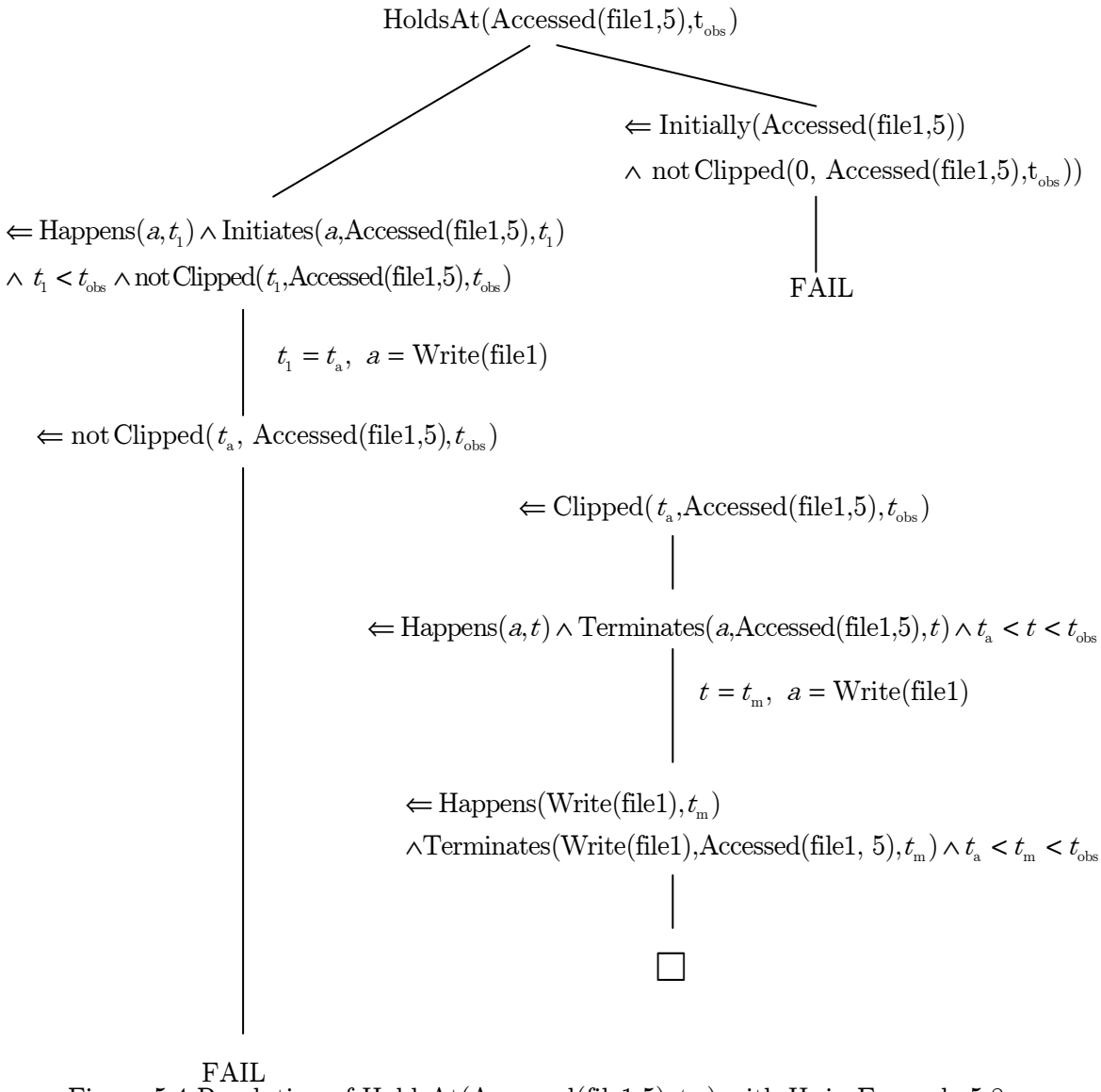

Figure 5.4 Resolution of HoldsAt(Accessed(file1,5),t_{obs}) with H_2 in Example 5.8

The resolution of the second clause in the observation proposition, HoldsAt(Accessed(file,5),t_{obs}) with hypothesis H_2 is shown in Figure 5.4. The resolution fails, which means that H_2 is also not a valid hypothesis for $c_h(t)$. In other words, in the defined simple file system, the observed set cannot come to existence with the default clock hypothesis. The default clock hypothesis is therefore refuted. This would happen for any file where $t_a < t_m$.

5.6 Invariants in the simple file system

The methods in the previous sections can be used to test the observation proposition in the general case, and thereby summarize properties of the simple file system defined in formulae (5.4) - (5.11). For any given file in the file system, the observation proposition is:

$$\text{HoldsAt}(\text{Modified}(\mathit{file}, c(t_m)), t_{obs}) \wedge \text{HoldsAt}(\text{Accessed}(\mathit{file}, c(t_a)), t_{obs})$$

Now, if $c(t_m) \neq \tau_0$ and $c(t_a) \neq \tau_0$, there must have occurred actions initiating these fluents. As seen in Example 5.7, these actions can only have been

$$\begin{aligned} H_1 &= \{\text{Happens}(\text{Write}(\mathit{file}), t_m), \text{Happens}(\text{Read}(\mathit{file}), t_a)\} \\ H_2 &= \{\text{Happens}(\text{Write}(\mathit{file}), t_m), \text{Happens}(\text{Write}(\mathit{file}), t_a)\} \end{aligned} \tag{5.16}$$

Now, three different timestamping orders can be investigated; $t_m < t_a$, $t_m = t_a$ and $t_m > t_a$. The case of $t_m > t_a$ was examined in Example 5.8, and found to always be refuted.

In the case of $t_m = t_a$, (5.16) is reduced to

$$\begin{aligned} H_1 &= \{\text{Happens}(\text{Write}(\mathit{file}), t_m), \text{Happens}(\text{Read}(\mathit{file}), t_m)\} \\ H_2 &= \{\text{Happens}(\text{Write}(\mathit{file}), t_m)\} \end{aligned} \tag{5.17}$$

Written as a disjunction

$$(\text{Happens}(\text{Write}(\textit{file}), t_{\text{m}}) \wedge \text{Happens}(\text{Read}(\textit{file}),\ t_{\text{m}}))$$
$$\vee\ \text{Happens}(\text{Write}(\textit{file}), t_{\text{m}})$$

Which is equivalent to

$$\text{Happens}(\text{Write}(\textit{file}), t_{\text{m}})$$

The effect of Happens(Read(*file*),t_{m}) in H_1 in (5.17) will always be consumed by the effect of Happens(Write(*file*),t_{m}), (5.17) can then be reduced to

$$\text{H}_1 = \{\text{Happens}(\text{Write}(\textit{file}), t_{\text{m}})\}$$

The timestamping order $t_{\text{m}} < t_{\text{a}}$ must be investigated further. The resolution in Figure 5.5 shows that H_2 is refuted if $t_{\text{m}} < t_{\text{a}}$. The resolutions in Figure 5.6 and Figure 5.7 show that H_1 is accepted for $t_{\text{m}} < t_{\text{a}}$.

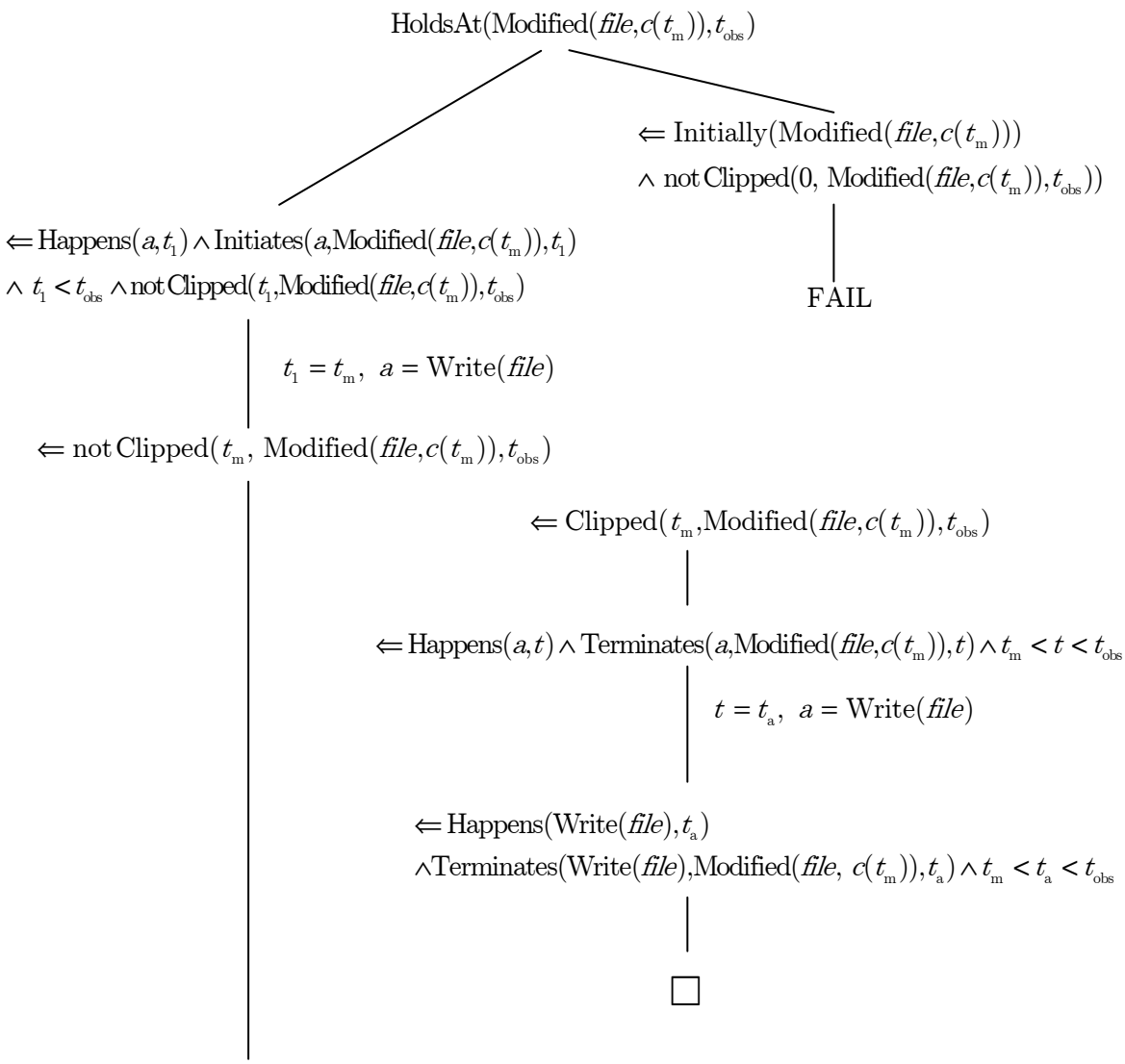

Figure 5.5 The observation proposition fails for H_2 when $t_m < t_a$

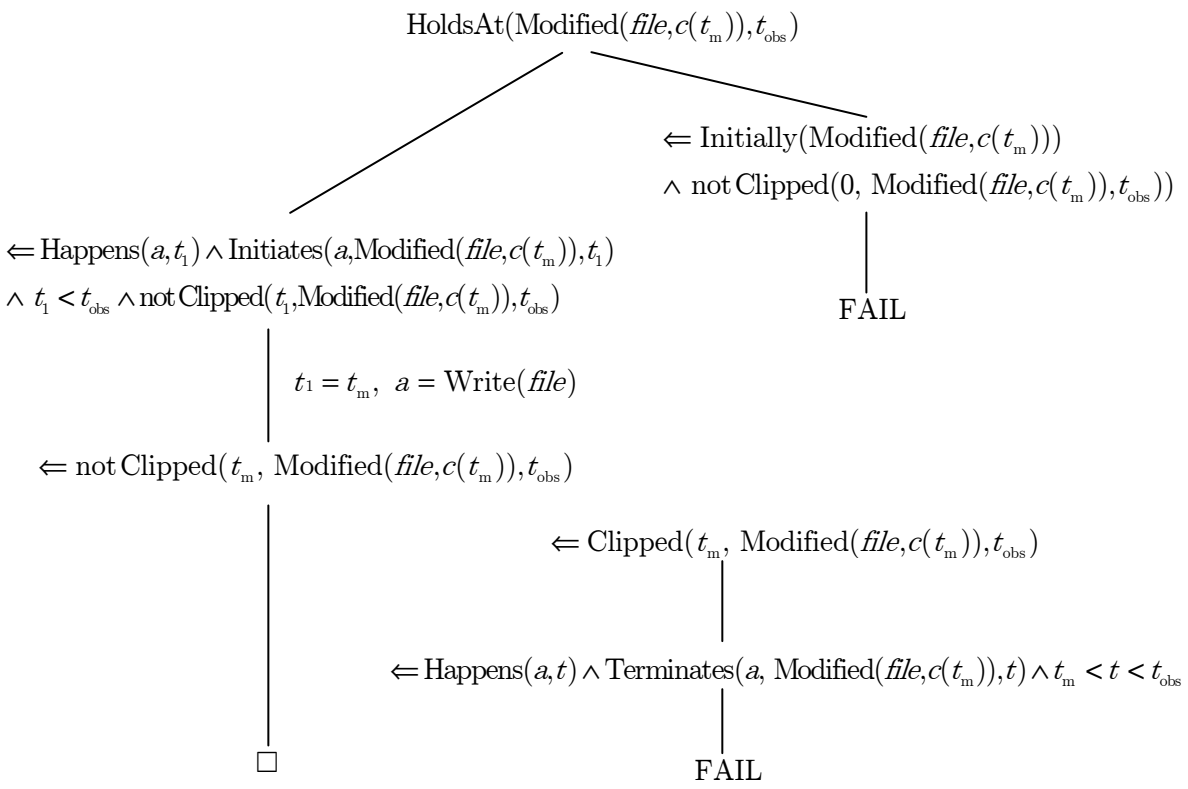

Figure 5.6 HoldsAt(Modified(*file*, $c(t_m)$), t_{obs}) does not fail for H_1 when $t_m < t_a$

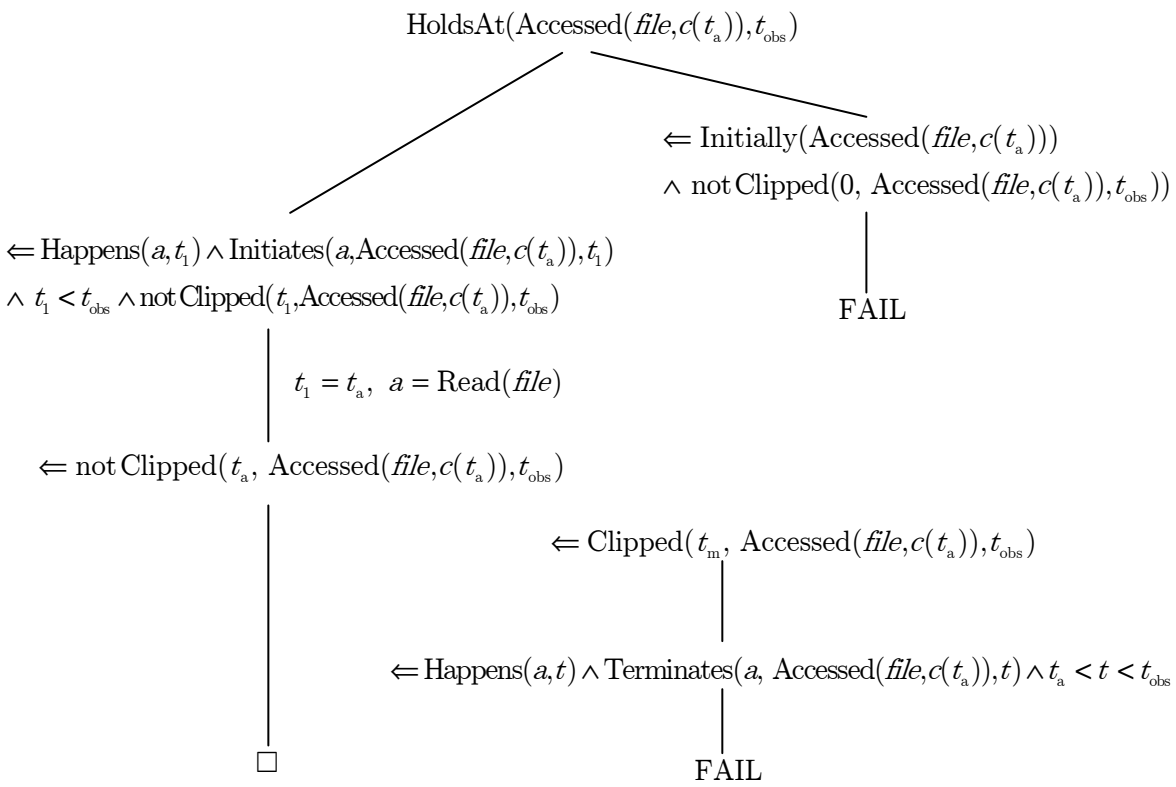

Figure 5.7 HoldsAt(Accessed(*file*, $c(t_a)$), t_{obs}) does not fail for H_1 when $t_m < t_a$

In summary, the above discussion has shown that

$$t_m \not> t_a$$
$$t_m = t_a \Rightarrow \text{Happens}(\text{Write}(file), t_m)$$
$$t_m < t_a \Rightarrow \text{Happens}(\text{Read}(file), t_a) \wedge \text{Happens}(\text{Write}(file), t_m)$$

And therefore, for any observations of timestamps in this simple file system, it is required that:

$$t_m \leq t_a$$

5.7 Using invariants to test a clock hypothesis

In the simple file system, it is now known that $t_m \leq t_a$. But these times are not directly observable in the observation set *O*. Instead, the investigator observes the Modified

timestamp $c(t_m)$ and the Accessed timestamp $c(t_a)$. The clock hypothesis must then match with the observed timestamp values and $t_m \leq t_a$. This requirement can be expressed in the same way as Theorem 3.14:

Theorem 5.9. In a correct clock hypothesis $c_h(t)$, the timestamps of all events e_i, e_j where $t(e_i) \leq t(e_j)$ in an observation set O must be such that the timestamp of the first event minus the deviation from a common base has value less than or equal to the timestamp of the latter event minus the deviation from a common base.

$$t(e_i) \leq t(e_j) \Rightarrow \tau(e_i) - d_h(t(e_i)) \leq \tau(e_j) - d_h(t(e_j))$$

Proof. Let $c_h(t)$ be a correct clock hypothesis. Let $b(t)$ be a common base for $c_h(t)$ and $c_o(t)$. Then

$$b(t) = c_h(t) - d_h(t)$$
$$b(t) = c_o(t) - d_o(t)$$

Thus,

$$c_h(t) - d_h(t) = c_o(t) - d_o(t)$$

And since $c_h(t)$ is correct we have $c_h(t) = c_o(t)$. Therefore

$$d_h(t) = d_o(t)$$
$$b(t) = c_o(t) - d_h(t)$$

And inserting Definition 3.4 yields

$$b(t(e)) = \tau(e) - d_h(t(e))$$

Now, from Definition 3.8 $b(t)$ shall be an ideal clock. From Definition 3.5 ideal clocks satisfy

$$t(e_i) < t(e_j) \Rightarrow c(t(e_i)) \leq c(t(e_j))$$

And then, inserting $b(t)$ gives

$$t(e_i) < t(e_j) \Rightarrow b(t(e_i)) \leq b(t(e_j))$$
$$t(e_i) < t(e_j) \Rightarrow \tau_{c_o}(e_i) - d_h(t(e_i)) \leq \tau_{c_o}(e_j) - d_h(t(e_j))$$

Further, from Definition 3.5 ideal clocks satisfy

$$t(e_i) = t(e_j) \Rightarrow c(t(e_i)) = c(t(e_j))$$

And then, inserting $b(t)$ gives

$$t(e_i) = t(e_j) \Rightarrow b(t(e_i)) = b(t(e_j))$$
$$t(e_i) = t(e_j) \Rightarrow \tau_{c_o}(e_i) - d_h(t(e_i)) = \tau_{c_o}(e_j) - d_h(t(e_j))$$

Thus,

$$t(e_i) \leq t(e_j) \Rightarrow \tau_{c_o}(e_i) - d_h(t(e_i)) \leq \tau_{c_o}(e_j) - d_h(t(e_j))$$

□

And so, the requirement of Theorem 5.9 can be tested as follows:

Theorem 5.10. If, in an known system, timestamps generated by e_i and e_j must be such that $t(e_i) \leq t(e_j)$, then if there exist timestamps in an observation set O, for which the timestamp of e_i minus the hypothesis deviation from a common base has a higher value than the timestamp of e_j minus the hypothesis deviation from a common base, then the clock hypothesis is incorrect.

$$\exists e_i \exists e_j ((t(e_i) \leq t(e_j)) \wedge (\tau_{c_o}(e_i) - d_h(t(e_i)) > \tau_{c_o}(e_j) - d_h(t(e_j)))) \Rightarrow c_o(t) \neq c_h(t)$$

The proof follows the reasoning of the proof for Theorem 3.15 and is omitted here.

This test complements Test-A and Test-B developed in chapter 3, for systems that can be described in Simplified Event Calculus and where invariants can be found by using the methods described in this chapter.

Example 5.11. In Example 5.8 a file in the simple file system was observed with Accessed timestamp $c(t_a) = 5$ and Modified timestamp $c(t_m) = 10$. The default clock hypothesis was tested against this evidence. Now, we know that in the simple file system, $t_m \leq t_a$, and for the default clock hypothesis, $d_h(t) = 0$. Then, applying the test in Theorem 5.10 and inserting t_m, t_a, $c(t_a)$ and $c(t_m)$ yields:

$$(t(e_i) \leq t(e_j)) \wedge (\tau_{c_o}(e_i) - d_h(t(e_i)) > \tau_{c_o}(e_j) - d_h(t(e_j)))$$
$$(t_m \leq t_a) \wedge (c(t_m) > c(t_a))$$
$$(t_m \leq t_a) \wedge (10 > 5)$$

Which is clearly true, hence the default clock hypothesis is refuted.

5.8 Using Simplified Event Calculus to develop tests for a system

In this chapter, a method for developing clock hypothesis tests from the description of a system has been developed. The necessary steps to develop such test are described as follows, hereafter called the *SEC-algorithm*:

1. Determine a set E of the effect of actions in the system by inspecting how the system works or otherwise justify assumptions about the effect of actions in the system.
2. Determine a set S of the initial state of the system by determining the production state of the system or otherwise justify assumptions about the initial state.
3. Find action hypotheses for the system by using the initiation proposition
4. Test the hypotheses against the observation proposition by means of resolution for all timestamping orders
5. Derive invariants for the system from the testing of the observation proposition.
6. Derive tests from the invariants, by using Theorem 5.9.
7. Test the clock hypothesis.

In order to determine how practical this procedure is, it is interesting to evaluate each step in practical terms. Step 1 involves an investigation of how the system reacts to actions. Ideally, such knowledge can be derived directly from the specification or implementation of the system. This is however not always possible. Then, it is necessary to do an active investigation by testing how variables of the system changes as a result of actions. Step 2 involves obtaining knowledge about the start state. In computer systems, this can often be accomplished by determine how the system was configured when it was installed. In other cases, witness statements may be enough to determine how the start state was. Step 3-6 is an exercise of reasoning, by using the tools given in this chapter. It is interesting to note that for any given system type, steps 1-6 may not have to be done more than once. The results, tests to be used in step 7, can be applied over and over again in different investigations involving that system type. For example, in investigation of digital media, investigation of file systems is common. For a specific file system, steps 1-6 could be performed once, and the developed tests could then be implemented in a system for testing clock hypotheses in that file system. Such a system could for example be a part of a software package for digital investigation.

5.9 Extending the simple file system with creation and deletion

In real file systems, files do not exist indefinitely. Instead, files are created by means of special operations, and can also be deleted. Therefore, in order to more closely represent an actual file system, it is necessary to extend the simple file system from chapter 5 to also include creation and deletion of files. The representation should be modelled in such a way that the fluents and actions pertain to file instances. A file instance is the storage of a specific file on a storage medium. A file instance does not represent the file's contents or name as such, since there may be other files with the same content or the same name. The file instance represents the storage of file data at a specific location on the storage medium. A file instance can be created or deleted. It can only be created if it has not been created before. It can only be deleted if it has been created before and has not already been deleted. This definition gives the life cycle of a file instance, shown in Figure 5.8.

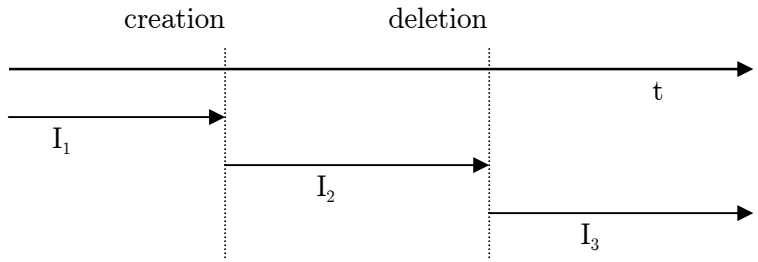

Figure 5.8 Graphical representation of the life cycle of a file instance

A file can only be created if it has not already been created (interval I_1). It can only be deleted if it has already been created, but is not deleted (interval I_2). Other operations, such as Read or Write can only occur in interval I_2. Then, a file instance can only be created exactly once, and it can only be deleted exactly once. Recreation of a deleted file instance cannot occur. Timestamps that has been assigned to a file in the delete operation or before, is retained after the delete operation and cannot be changed thereafter (interval I_3). This reflects a property of real file systems; when a file has been deleted, it is no longer reachable in the file system, so the timestamps can no longer be

set by normal operating system procedures. By using special forensic software, it is however possible to find the values for the timestamps also after the file was deleted.

Based on this model, we obtain the following fluents for representing the state of a file instance:

$$\text{Exists}(\textit{file})$$
$$\text{Deleted}(\textit{file})$$

And the following actions that can change the fluents:

$$\text{Create}(\textit{file})$$
$$\text{Delete}(\textit{file})$$

Clauses for the relationship between the state fluents and the state actions can be defined as follows:

$$\text{Initiates}(\text{Create}(\textit{file}),\text{Exists}(\textit{file}),t) \tag{5.18}$$
$$\text{Initiates}(\text{Delete}(\textit{file}),\text{Deleted}(\textit{file}),t) \Leftarrow \text{HoldsAt}(\text{Exists}(\textit{file}),t) \tag{5.19}$$

Note that further requirements for the clauses are not needed. It is for example not necessary to require that a file does not exist for the Create(*file*) action to initiate the Exists(*file*) fluent. If the file does exist, the Create(*file*) action would not change anything anyway.

The life cycle of a file instance has now been defined, and we can go further to introduce a new fluent to represent the Created timestamp that exist in most real file systems:

$$\text{Created}(\textit{file},\tau_c)$$

It is necessary to change the Initiates and Terminates clauses from the simple file system to reflect the requirement that timestamp fluents can only change while the file exists and is not deleted, and to introduce the Created timestamp fluent:

$$\begin{aligned}&\text{Initiates}(\text{Read}(\textit{file}),\text{Accessed}(\textit{file},c(t)),t)\\ &\qquad\Leftarrow \text{HoldsAt}(\text{Exists}(\textit{file}),t) \wedge \text{not HoldsAt}(\text{Deleted}(\textit{file}),t)\end{aligned} \tag{5.20}$$

$$\begin{aligned}&\text{Initiates}(\text{Write}(\mathit{file}),\text{Accessed}(\mathit{file},c(t)),t)\\&\qquad\Leftarrow \text{HoldsAt}(\text{Exists}(\mathit{file}),t)\wedge \text{not HoldsAt}(\text{Deleted}(\mathit{file}),t)\end{aligned}\tag{5.21}$$

$$\begin{aligned}&\text{Initiates}(\text{Write}(\mathit{file}),\text{Modified}(\mathit{file},c(t)),t)\\&\qquad\Leftarrow \text{HoldsAt}(\text{Exists}(\mathit{file}),t)\wedge \text{not HoldsAt}(\text{Deleted}(\mathit{file}),t)\end{aligned}\tag{5.22}$$

$$\text{Initiates}(\text{Create}(\mathit{file}),\text{Created}(\mathit{file},c(t)),t)\Leftarrow \text{not HoldsAt}(\text{Exists}(\mathit{file}),t)\tag{5.23}$$

$$\text{Initiates}(\text{Create}(\mathit{file}),\text{Modified}(\mathit{file},c(t)),t)\Leftarrow \text{not HoldsAt}(\text{Exists}(\mathit{file}),t)\tag{5.24}$$

$$\text{Initiates}(\text{Create}(\mathit{file}),\text{Accessed}(\mathit{file},t),c(t))\Leftarrow \text{not HoldsAt}(\text{Exists}(\mathit{file}),t)\tag{5.25}$$

$$\begin{aligned}&\text{Terminates}(\text{Read}(\mathit{file}),\text{Accessed}(\mathit{file},c(t_1)),t)\\&\qquad\Leftarrow t_1<t\wedge\text{HoldsAt}(\text{Exists}(\mathit{file}),t)\wedge \text{not HoldsAt}(\text{Deleted}(\mathit{file}),t)\end{aligned}\tag{5.26}$$

$$\begin{aligned}&\text{Terminates}(\text{Write}(\mathit{file}),\text{Accessed}(\mathit{file},c(t_1)),t)\\&\qquad\Leftarrow t_1<t\wedge\text{HoldsAt}(\text{Exists}(\mathit{file}),t)\wedge \text{not HoldsAt}(\text{Deleted}(\mathit{file}),t)\end{aligned}\tag{5.27}$$

$$\begin{aligned}&\text{Terminates}(\text{Write}(\mathit{file}),\text{Modified}(\mathit{file},c(t_1)),t)\\&\qquad\Leftarrow t_1<t\wedge\text{HoldsAt}(\text{Exists}(\mathit{file}),t)\wedge \text{not HoldsAt}(\text{Deleted}(\mathit{file}),t)\end{aligned}\tag{5.28}$$

Terminates clauses for the Created fluent are not introduced. Consequently this fluent will hold indefinitely once Initiated. Similarly, the state of the Modified and Accessed fluents will hold indefinitely when the file has been Deleted. This is what we require; in real file system it is possible to recover the timestamps of previously deleted files. Further, we do not introduce Terminates clauses for the Create action and the Accessed and Modified fluents. This is not necessary, since the Create action can only occur once on each file instance - the Create action cannot subsequently change the Accessed and Modified fluents, and it is therefore not necessary to have Terminated clauses for them.

With the introduction of the Create action, the Initially clauses present in the simple file system are no longer necessary, since a file must exist in order to be affected by other actions. For a file to exist, it must first be created. In this model, files do not exist from the start, but have to be created by the Create action, something that will initiate timestamp fluents. No timestamp fluents are initiated before the file is created, achieving a representation closer to a real file system than the simple file system model.

5.10 Invariants in the simple file system with creation

Let the simple file system with creation be the file system model defined by the conjunction of formulae (5.18) - (5.28) in the Simplified Event Calculus. In this system, E is the conjunction of formulae (5.18) - (5.28), and S is the empty set.

The observation proposition for a file in this system takes the form:

$$\begin{aligned} o \;=\; & \text{HoldsAt}(\text{Created}(\mathit{file}, c(t_{\text{c}}))\, t_{\text{obs}}) \\ \wedge\; & \text{HoldsAt}(\text{Modified}(\mathit{file}, c(t_{\text{m}})), t_{\text{obs}}) \\ \wedge\; & \text{HoldsAt}(\text{Accessed}(\mathit{file}, c(t_{\text{a}})), t_{\text{obs}}) \end{aligned}$$

There are no Initially clauses, so the initiation proposition can be derived from formulae (5.20) - (5.25):

$$\begin{aligned} q \;=\; & (\text{Happens}(\text{Create}(\mathit{file}), t_{\text{c}})) \\ \wedge\; & (\text{Happens}(\text{Create}(\mathit{file}), t_{\text{m}}) \vee \text{Happens}(\text{Write}(\mathit{file}), t_{\text{m}})) \\ \wedge\; & (\text{Happens}(\text{Create}(\mathit{file}) t_{\text{a}}) \vee \text{Happens}(\text{Write}(\mathit{file}), t_{\text{a}}) \vee \text{Happens}(\text{Read}(\mathit{file}), t_{\text{a}})) \end{aligned}$$

Rewritten:

$$\begin{aligned} q \;=& ((\text{Happens}(\text{Create}(\mathit{file}), t_{\text{m}}) \wedge \text{Happens}(\text{Create}(\mathit{file}), t_{\text{c}})) \\ & \quad \vee (\text{Happens}(\text{Write}(\mathit{file}), t_{\text{m}}) \wedge \text{Happens}(\text{Create}(\mathit{file}), t_{\text{c}}))) \\ \wedge\; & ((\text{Happens}(\text{Create}(\mathit{file}), t_{\text{a}}) \wedge \text{Happens}(\text{Create}(\mathit{file}), t_{\text{c}})) \\ & \quad \vee (\text{Happens}(\text{Write}(\mathit{file}), t_{\text{a}}) \wedge \text{Happens}(\text{Create}(\mathit{file}), t_{\text{c}})) \\ & \quad \vee (\text{Happens}(\text{Read}(\mathit{file}), t_{\text{a}}) \wedge \text{Happens}(\text{Create}(\mathit{file}), t_{\text{c}}))) \end{aligned}$$

And expanded:

$$\begin{aligned} q \;=\; & (\text{Happens}(\text{Create}(\mathit{file}), t_{\text{a}}) \wedge \text{Happens}(\text{Create}(\mathit{file}), t_{\text{c}}) \wedge \text{Happens}(\text{Create}(\mathit{file}), t_{\text{m}})) \\ & \vee (\text{Happens}(\text{Write}(\mathit{file}), t_{\text{a}}) \wedge \text{Happens}(\text{Create}(\mathit{file}), t_{\text{c}}) \wedge \text{Happens}(\text{Create}(\mathit{file}), t_{\text{m}})) \\ & \vee (\text{Happens}(\text{Read}(\mathit{file}), t_{\text{a}}) \wedge \text{Happens}(\text{Create}(\mathit{file}), t_{\text{c}}) \wedge \text{Happens}(\text{Create}(\mathit{file}), t_{\text{m}})) \\ & \vee (\text{Happens}(\text{Create}(\mathit{file}), t_{\text{a}}) \wedge \text{Happens}(\text{Create}(\mathit{file}), t_{\text{c}}) \wedge \text{Happens}(\text{Write}(\mathit{file}), t_{\text{m}})) \\ & \vee (\text{Happens}(\text{Write}(\mathit{file}), t_{\text{a}}) \wedge \text{Happens}(\text{Create}(\mathit{file}), t_{\text{c}}) \wedge \text{Happens}(\text{Write}(\mathit{file}), t_{\text{m}})) \\ & \vee (\text{Happens}(\text{Read}(\mathit{file}), t_{\text{a}}) \wedge \text{Happens}(\text{Create}(\mathit{file}), t_{\text{c}}) \wedge \text{Happens}(\text{Write}(\mathit{file}), t_{\text{m}})) \end{aligned}$$

So now we have six different hypotheses, all of which hypothesize a Create action occurring at t_c:

$$H_1 = \{\text{Happens}(\text{Create}(\textit{file}), t_a), \text{Happens}(\text{Create}(\textit{file}), t_c), \text{Happens}(\text{Create}(\textit{file}), t_m)\}$$
$$H_2 = \{\text{Happens}(\text{Write}(\textit{file}), t_a), \text{Happens}(\text{Create}(\textit{file}), t_c), \text{Happens}(\text{Create}(\textit{file}), t_m)\}$$
$$H_3 = \{\text{Happens}(\text{Read}(\textit{file}), t_a), \text{Happens}(\text{Create}(\textit{file}), t_c), \text{Happens}(\text{Create}(\textit{file}), t_m)\}$$
$$H_4 = \{\text{Happens}(\text{Create}(\textit{file}), t_a), \text{Happens}(\text{Create}(\textit{file}), t_c), \text{Happens}(\text{Write}(\textit{file}), t_m)\}$$
$$H_5 = \{\text{Happens}(\text{Write}(\textit{file}), t_a), \text{Happens}(\text{Create}(\textit{file}), t_c), \text{Happens}(\text{Write}(\textit{file}), t_m)\}$$
$$H_6 = \{\text{Happens}(\text{Read}(\textit{file}), t_a), \text{Happens}(\text{Create}(\textit{file}), t_c), \text{Happens}(\text{Write}(\textit{file}), t_m)\}$$

Consider first $t_c = t_m = t_a$. In this case, the initiation proposition will collapse to a simple hypothesis:

$$q = \text{Happens}(\text{Create}(\textit{file}), t_c) \tag{5.29}$$

Then consider $t_c = t_m$. In this case the initiation proposition will collapse to:

$$\begin{aligned} q = {} & (\text{Happens}(\text{Create}(\textit{file}), t_a) \wedge \text{Happens}(\text{Create}(\textit{file}), t_c)) \\ & \vee (\text{Happens}(\text{Write}(\textit{file}), t_a) \wedge \text{Happens}(\text{Create}(\textit{file}), t_c)) \\ & \vee (\text{Happens}(\text{Read}(\textit{file}), t_a) \wedge \text{Happens}(\text{Create}(\textit{file}), t_c)) \end{aligned} \tag{5.30}$$

And for $t_c = t_a$ the result is:

$$\begin{aligned} q = {} & (\text{Happens}(\text{Create}(\textit{file}), t_c) \wedge \text{Happens}(\text{Create}(\textit{file}), t_m)) \\ & \vee (\text{Happens}(\text{Create}(\textit{file}), t_c) \wedge \text{Happens}(\text{Write}(\textit{file}), t_m)) \end{aligned} \tag{5.31}$$

Now, let $t_c < t_m < t_a$ and consider H_1. The resolution for this hypothesis is shown in Figure 5.9. As the resolution shows, H_1 fails with this timestamping order. Now consider other timestamping orders with H_1 where $t_c \neq t_m \wedge t_c \neq t_a$. It is easy to see that the kind of tree shown in Figure 5.9 would arise for all configurations of t_c, t_m and t_a, when tested for the part of the observation proposition pertaining to the middle or last timestamp. Therefore, H_1 cannot be the case and is refuted.

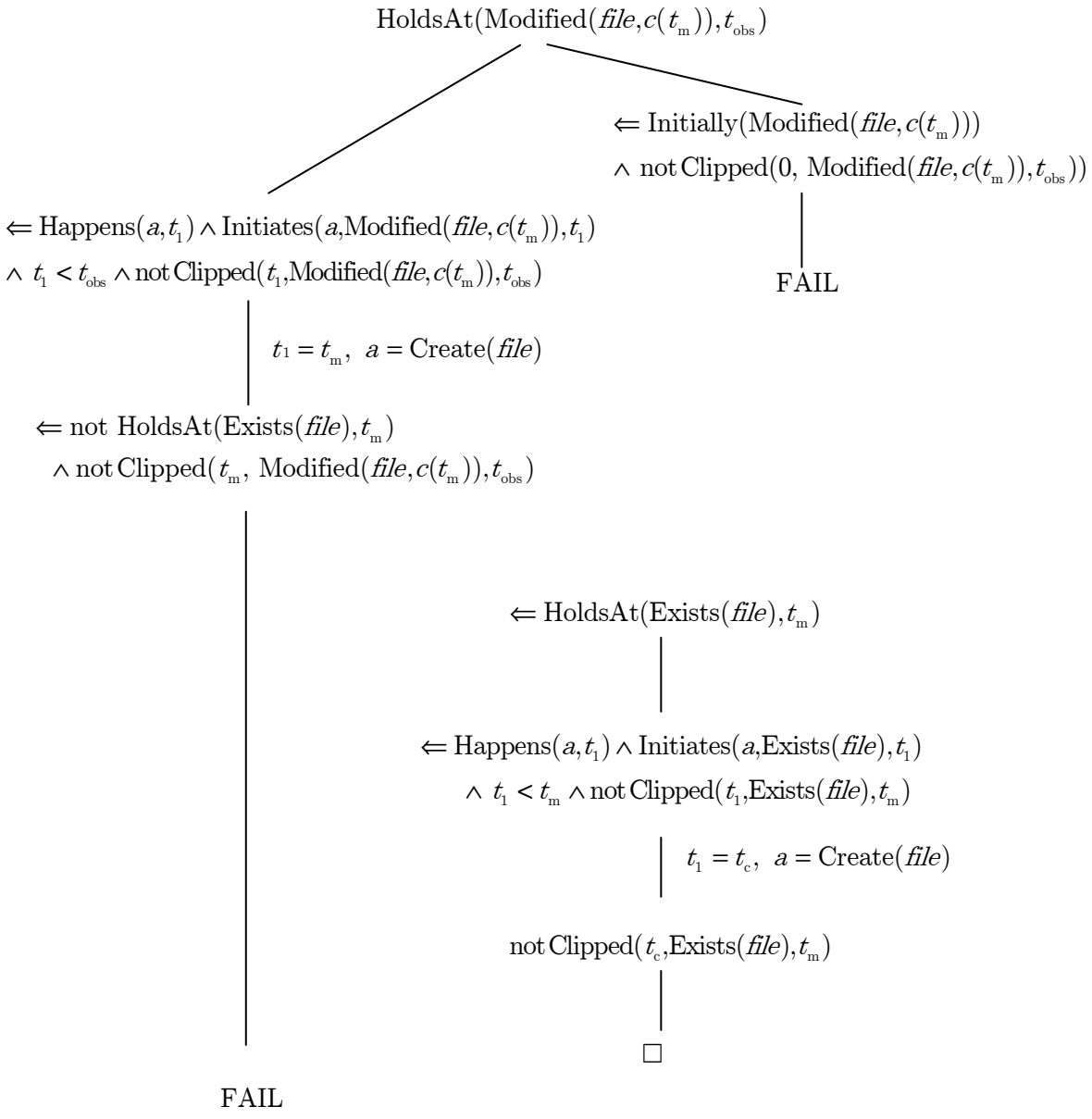

Figure 5.9 Observation of the Modified timestamp with H_1 and $t_c < t_m < t_a$

Now, observe that many of the other hypotheses include multiple Create actions on the same file, namely H_2, H_3 and H_4. These all hypothesize the occurrence of more than one Create action that will set timestamp fluents. As discussed for H_1, all resolutions for the observation proposition for these hypotheses will fail when $t_c \neq t_m \wedge t_c \neq t_a$, since the requirement for the second Create action is that the file does not already exist, but the file exists, because of the first Create action.

Thus, we are left with:

$$H_5 = \{\text{Happens}(\text{Write}(\mathit{file}), t_a),\ \text{Happens}(\text{Create}(\mathit{file}), t_c),\ \text{Happens}(\text{Write}(\mathit{file}), t_m)\}$$
$$H_6 = \{\text{Happens}(\text{Read}(\mathit{file}), t_a),\ \text{Happens}(\text{Create}(\mathit{file}), t_c),\ \text{Happens}(\text{Write}(\mathit{file}), t_m)\}$$

Now, consider H_6 and $t_a < t_c$. Since the initiation of the Accessed timestamp by an action Read(*file*) at t_a requires HoldsAt(Exists(*file*), t_a), we must consider both HoldsAt(Exists(*file*), t_a) and $\neg$ HoldsAt(Exists(*file*), t_a). For $\neg$ HoldsAt(Exists(*file*), t_a), the resolution for the observation of the Accessed timestamp fails, as shown in Figure 5.10. On the other hand, for HoldsAt(Exists(*file*), t_a), the resolution for the observation of the Created timestamp fails, as shown in Figure 5.11, since the initiation of the Created timestamp requires that the file does not exist. The result is then, that for H_6, it cannot be the case that $t_a < t_c$.

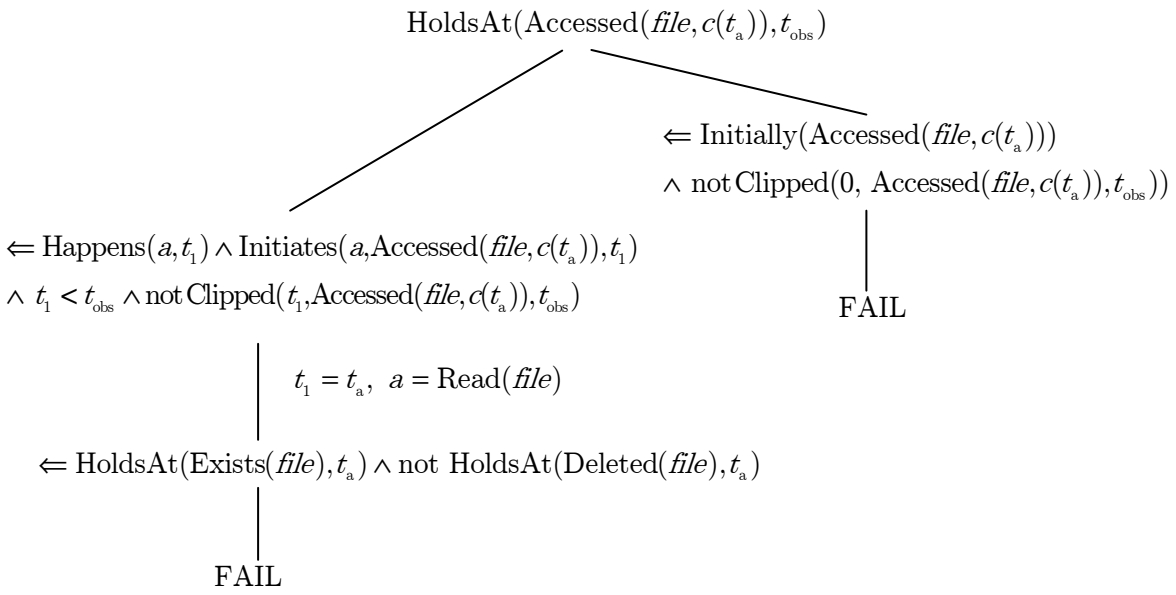

Figure 5.10 The accessed timestamp for H_6 when $t_a < t_c$ and $\neg$ HoldsAt(Exists(*file*), t_a)

Now, consider that $t_m < t_c$ for H_6. One can now apply the same type of reasoning as for $t_a < t_c$, and draw resolutions for the observation of the modified timestamp assuming that $\neg$ HoldsAt(Exists(*file*), t_m) and for the observation of the created timestamp assuming that HoldsAt(Exists(*file*), t_a). These resolutions take the same form as those

shown in Figure 5.10 and Figure 5.11. The result is that $t_m < t_c$ cannot be the case for H_6.

So, how would these results relate to H_5? Since H_5 hypothesizes Write actions at t_a and t_m, the situation for both t_a and t_m is equal to that already discussed for t_m with H_6. Again, resolution trees would be equal to those shown in Figure 5.10 and Figure 5.11, and would show that also for H_5, $t_m < t_c$ and $t_a < t_c$ cannot be the case. The same reasoning would apply for the hypotheses in (5.30) and (5.31), showing that $t_m < t_c$ or $t_a < t_c$ is not possible.

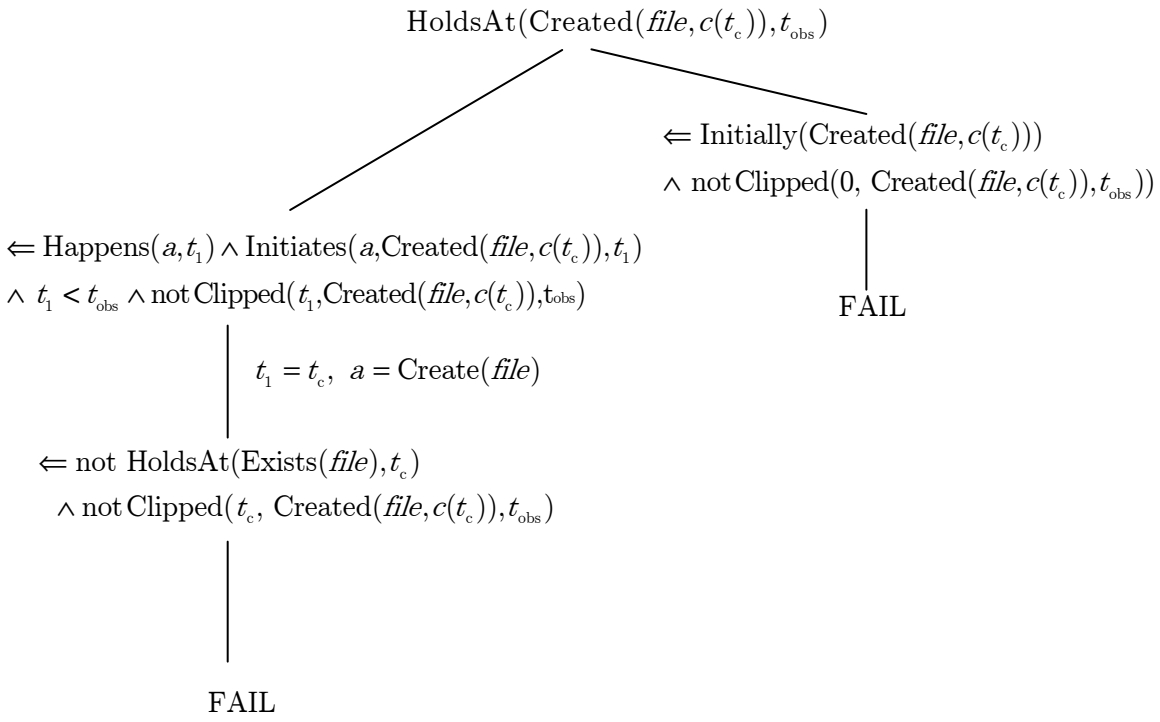

Figure 5.11 The created timestamp for H_6 when $t_a < t_c$ and $\text{HoldsAt}(\text{Exists}(file), t_a)$

From the reasoning above, it can be concluded that for the simple file system with creation, it is always the case that $t_c \le t_m$ and $t_c \le t_a$.

Similar reasoning can be performed concerning the relationship between t_a and t_m. Consider first $t_a = t_m$. Then, since $t_c \le t_m$, either $t_c = t_a = t_m$, in which case the only possible hypothesis is given in (5.29), or $t_c < t_m$ which reduces H_5 and H_6 to:

$$H_5 = \{\text{Happens}(\text{Write}(\mathit{file}), t_m),\ \text{Happens}(\text{Create}(\mathit{file}), t_c),\ \text{Happens}(\text{Write}(\mathit{file}), t_m)\}$$
$$H_6 = \{\text{Happens}(\text{Read}(\mathit{file}), t_m),\ \text{Happens}(\text{Create}(\mathit{file}), t_c),\ \text{Happens}(\text{Write}(\mathit{file}), t_m)\}$$

In other words

$$\text{Happens}(\text{Create}(\mathit{file}), t_c) \wedge ((\text{Happens}(\text{Write}(\mathit{file}), t_m)) \vee (\text{Happens}(\text{Read}(\mathit{file}), t_m) \wedge \text{Happens}(\text{Write}(\mathit{file}), t_m)))$$

Which can be reduced to

$$\text{Happens}(\text{Create}(\mathit{file}), t_c) \wedge \text{Happens}(\text{Write}(\mathit{file}), t_m)$$

This reduction is the equivalent to the reduction in section 5.6 for the simple file system.

Now, consider $t_a < t_m$. Then, if $t_a = t_c$, the hypothesis is given in (5.31):

$$q = (\text{Happens}(\text{Create}(\mathit{file}), t_c) \wedge \text{Happens}(\text{Create}(\mathit{file}), t_m)) \vee (\text{Happens}(\text{Create}(\mathit{file}), t_c) \wedge \text{Happens}(\text{Write}(\mathit{file}), t_m))$$

The first of these hypotheses fail, by the resolution in Figure 5.9. The second also fail, as shown in Figure 5.12.

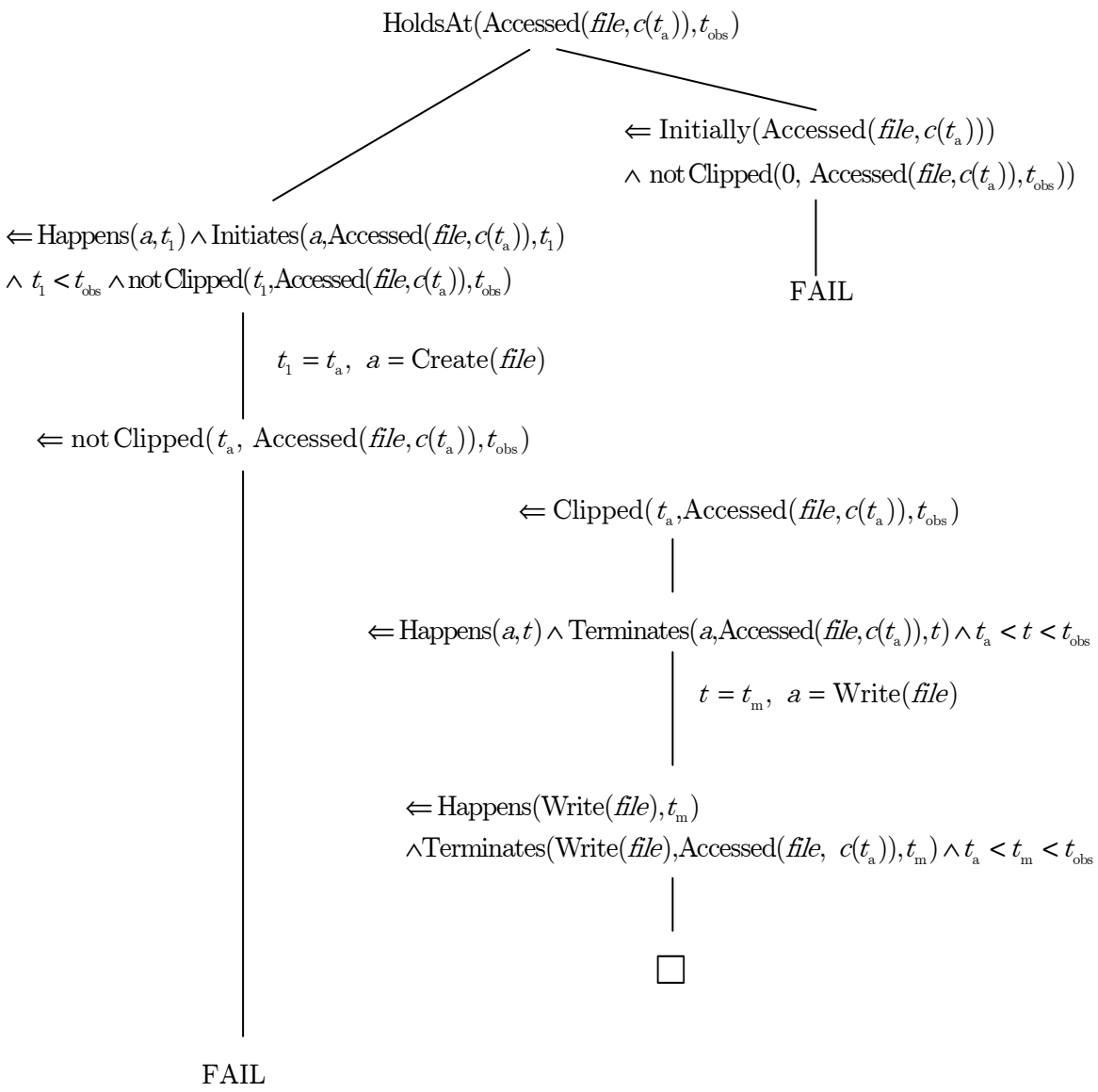

Figure 5.12 Resolution fails for $t_a < t_m$ where $t_a = t_c$

And, if $t_c < t_a < t_m$, H_5 will fail as shown in the resolution in Figure 5.4 and H_6 will fail as shown in the resolution in Figure 5.3.

Thus, it has been shown that, for the simple file system with creation, for every file, the observed Created, Modified and Accessed timestamps t_c, t_m and t_a must obey: $t_c \leq t_m \leq t_a$. This property can be used to test clock hypotheses with observed sets from the simple file system with creation by using the test in Theorem 5.10.

5.11 Complexity

In this chapter, the SEC-algorithm for the determination of invariants from a system model was described. The algorithm has been applied manually, but could just as well be implemented in a computer program, for example in the logic programming language PROLOG. With such a program, invariants could be found with the SEC-algorithm without having to manually draw figures for each resolution. This would save manual work, and allow for more complex systems. It is however not certain that even a computer can handle the amount of computation involved if the system to be tested is too complex. It is therefore interesting to find the computational complexity of the SEC-algorithm, and determine how the amount of necessary computation steps grows when the modelled system gets more complex.

The growth of the SEC-algorithm is determined by two factors; the number of timestamps associated with each file in the system and the number of Initiates clauses in the model. Let n be the number of timestamps associated with each file in the system. Let u_i be the number of Initiates clauses in E_I initiating fluents for timestamp i. The total number of Initiates clauses for the model is then:

$$|E_{\mathrm{I}}| = \sum_{i=0}^{n} u_i$$

Evaluating the initiation proposition q in (5.14), we see that q is a conjunction of n terms, corresponding to the initiation of each fluent in the observation proposition o in (5.13). Each term is a disjunction of u_i Happens clauses corresponding to the initiation of that fluent. After reordering the conjunction of disjunctive clauses into a disjunction of conjunctive clauses, each term is called an action hypothesis. The reordering involves taking the conjunction of each clause in the disjunctions, something that will produce $u_1 u_2 ... u_n$ terms. The number of hypotheses is then:

$$|H| = \prod_{i=0}^{n} u_i \tag{5.32}$$

The number of Happens clauses in each hypothesis is equal to the number of conjunctive clauses in q, which is n.

The number of timestamping orders that must be tested by resolutions is the number of different orderings between n moments in time, ordered with relations < and =. We call this number $p(n)$. $p(n)$ is the number of weak orders on n labelled elements, also called *ordered Bell numbers*. [35, 36] Computing the ordered Bell numbers is not straightforward. For a given n, one must first find all possible partitions $k_1, k_2, ..., k_j$ where

$$\sum_{i=0}^{j} k_i = n$$

The number of orders for each partition is then given by the multinomial coefficients for that ordering:

$$p_{k_1,k_2,...,k_j}(n) = \binom{n}{k_1, k_2, ..., k_j} = \binom{n!}{k_1!k_2!...k_j!}$$

For example, for n = 3, the possible partitions are (1,1,1), (1,2), (2,1) and (3). The multinomial coefficients for each partition then yield:

$$p_{1,1,1}(3) = \binom{3}{1,1,1} = \binom{3!}{1!1!1!} = 6$$

$$p_{1,2}(3) = \binom{3}{1,2} = \binom{3!}{1!2!} = 3$$

$$p_{2,1}(3) = \binom{3}{2,1} = \binom{3!}{2!1!} = 3$$

$$p_3(3) = \binom{3}{3} = \binom{3!}{3!} = 1$$

Thus, p(3) = 6+3+3+1 = 13. The ordered Bell numbers for n up to 9 are given in Table 5.1. [36]

n	$p(n)$
0	1
1	1
2	3
3	13
4	75
5	541
6	4683

7	47293
8	545835
9	7087261

Table 5.1 Ordered Bell numbers $p(n)$ for $n < 10$

Barthelemy showed that the Bell numbers can be approximated as: [36]

$$p(n) = \frac{n!}{2(\log 2)^{(n+1)}} + o((n-1)!)$$

This approximation is appropriate for the purpose of determining the growth of $p(n)$ in order to express the complexity of the SEC-algorithm. It can also be written as:

$$p(n) = \frac{1}{2}\left(\frac{1}{\log 2}\right)^{(n+1)} n! \ + o((n-1)!)$$

From which it is clear that the dominant growth of $p(n)$ comes from the factor $n!$. Thus,

$$p(n) \text{ is } O(n!) \text{ as } n \to \infty . \tag{5.33}$$

In the SEC-algorithm, a resolution must be tested for every timestamping order, action hypothesis and observed timestamp value. The number of observed timestamp values for a file is equal to the number of timestamps, n. The number of resolutions is then

$$O(Hn!) \text{ as } n, |H| \to \infty \tag{5.34}$$

It then remains to determine the number of computation steps involved with a resolution. The number of actions that needs to be tested for each hypothesis is n. Therefore, the computation of a resolution is $O(n)$. The growth of this function when n grows is too small to contribute to the growth compared to the number of resolutions. We therefore obtain the complexity of the SEC-algorithm by inserting $|H|$ into (5.34):

$$O(n!\prod_{i=0}^{n} u_i) \text{ as } n, u_i \to \infty \tag{5.35}$$

It should be recognized that the growth expressed in (5.35) is extraordinarily rapid. The growth is for example comparable to that of an algorithm seeking to solve the well known Traveling Salesman Problem by exhaustive search of all permutations of n cities,

something that would exhibit growth $O(n!)$. The SEC-algorithm can be described as an exhaustive search of all permutations of timestamping to determine which sequences of actions can lead to that permutation, if any. Such an exhaustive search involves a significant amount of computation, with rapid growth as the number of timestamps n grows. For important practical applications, n is however small, and computation therefore feasible. In this chapter, computation has been performed by hand for $n = 2$ and $n = 3$. For larger n, the SEC-algorithm should be implemented as a computer program. With such a program, computation with the SEC-algorithm should be feasible also with larger n. Most real systems have only a few timestamps per file. The given procedure is therefore useful for real applications, even if it grows rapidly with growing n.

6 TIMESTAMP REASONING WITH AFFECTS

Chapter 5 described the Simplified Event Calculus, and its use to create a model of a system for clock hypothesis testing. Finding invariants for a system becomes more complex as the number of actions in the system increases. This chapter simplifies reasoning, by introducing the concept of an affects table, finding all timestamping orders, and testing which action sequence may cause a specific timestamping order.

In Section 6.1 the affects table is described. In Section 6.2 timestamping orders is defined. Section 6.3 then describes how these concepts can be used to derive invariants for a system, which can be used for clock hypothesis testing. In Section 6.4 this algorithm is applied in a model of a real system. In Section 6.7, the use of affects tables is compared with the use of Event Calculus for timestamp reasoning.

6.1 Actions affects timestamps

A feature of the Simplified Event Calculus is the effect axioms whereby actions affect fluents via the Initiates and Terminates predicates. The file systems modelled in Chapter 5 have several timestamps per file. Each timestamp may or may not be updated by a specific action. When such a system is described with Simplified Event Calculus, timestamps are represented with fluents, and updating of timestamps by a specific action is modelled with one Initiates clause for the initiation of the new value and one Terminates clause for the termination of the previous value. Whenever a new timestamp fluent is Initiated, the previous timestamp fluent is always Terminated. This representation reflects the property of real file systems that whenever a timestamp is updated, the timestamp is set to the current value of the clock, and the previous value is lost.

It is possible to represent such a system in a simpler way, under the assumption that every timestamp change sets a new value and removes the previous. Instead of defining the relationship between actions and timestamps by means of Simplified Event Calculus, we can simply list the timestamps and actions, and define which timestamps are affected

by which actions. In this representation, the relationship between timestamps and actions is expressed as *affects*.

Definition 6.1. An action *affects* a timestamp if and only if an occurrence of that action sets a new value for the timestamp and removes the previous value for the timestamp.

Affects then expresses the updating of a timestamp directly, without the need for explicitly stating initiation of a new timestamp value and termination of the old timestamp value. This allows for simplified reasoning. Timestamp affect in the simple file system described in formulae (5.4) - (5.9) can now be summarized as follows:

- Read *affects* Accessed ((5.4) and (5.7))
- Write *affects* Accessed ((5.5) and (5.8))
- Write *affects* Modified ((5.6) and (5.9))

Having defined affects, one can now list all possible combinations of timestamps for a file in a table, and determine which of these combinations correspond to actions.

Definition 6.2. An *affects table* is a table listing all possible combinations of timestamps in a system, and all actions in the system and timestamps they affect. An affects table for a system with n timestamps has 2^n entries.

The affects table for existing non-deleted files in the simple file system with creation is given in the following:

	Created	Modified	Accessed	Actions
0				
1	X			
2		X		
3	X	X		
4			X	Read
5	X		X	
6		X	X	Write

7	X	X	X	Create

Table 6.1 Affects table for the simple file system with creation

The affects table states clearly which timestamps are affected by actions. The affects table also shows which timestamp affects combination does not occur with any action. This information can be utilized to derive invariants on timestamp, by reasoning on sequences of timestamp updating and corresponding sequences of actions.

6.2 Timestamping orders

In an investigation, the investigator observes values of timestamps on each investigated file. Each file has n different timestamps $\theta_1, \theta_2, ..., \theta_n$. The observed values of these timestamps were set at moments in time $t_{\theta_1}, t_{\theta_2}, ..., t_{\theta_n}$, and the values observed by the investigator are $c(t_{\theta_1}), c(t_{\theta_2}), ..., c(t_{\theta_n})$, set by the clock of the investigated system. Since the clock function $c(t)$ of the investigated system is unknown, the investigator cannot map these values directly to the moments in time $t_{\theta_1}, t_{\theta_2}, ..., t_{\theta_n}$ when timestamping occurred. But the investigator can list possible orders of timestamping, and determine if the observed result is consistent with a specific clock hypothesis, given the affects table for the system.

Definition 6.3. In a system with n timestamps, the *timestamp set* Θ is the set of observed timestamps $\theta_1, \theta_2, ..., \theta_n$. The *stamping time* t_{θ_i} for timestamp θ_i is the time at which the observed value of the timestamp was set.

Example 6.4. For the simple file system with creation described in section 5.9, the stamping time set is $\Theta = \{\text{Created, Modified, Accessed}\}$. The stamping times for the simple file system are denoted t_c, t_m and t_a. t_c is the time of production of the observed Created timestamp, t_m is the time of production of the observed Modified timestamp and t_a is the time of production of the observed Accessed timestamp.

To determine which (if any) sequence of actions in the system could have resulted in the observed timestamps, it is interesting to determine the different orders in which timestamping could have occurred. For each pair of timestamps in Θ (θ_i, θ_j), the

corresponding pair of stamping times $(t_{\theta_i}, t_{\theta_j})$ may be related by either $t_{\theta_i} < t_{\theta_j}$, $t_{\theta_i} = t_{\theta_j}$ or $t_{\theta_i} > t_{\theta_j}$.

Definition 6.5. A *timestamping order* is a sequence of the stamping time for every element in the timestamp set Θ, where each stamping time is related to the next stamping time in the sequence with the equals-relation = or the less-than relation <.

The equals relation imply that the stamping times are equal; the two timestamps were set at the same time. The less-than relation imply that the first stamping time is earlier than the second stamping time; the production of the first timestamp occurred at an earlier time than the production of the second timestamp. Each different stamping time in a timestamping order constitutes a *step* in the timestamping order. When two or more stamping times are equal, they constitute a step in the timestamping order together.

Example 6.6. An example timestamping order for the simple file system with creation is $(t_c = t_m < t_a)$. With this timestamping order, the Created and Modified timestamps were set at the same time, and the Accessed timestamp was set at a later time than the Created and Modified timestamps.

A list of all timestamping orders can then be constructed where each stamping of a specific timestamp may have occurred before, after or at the same time as the stamping of the other timestamps. For $n = 2$, all possible timestamping orders are:

Number	Order	
1	$(t_1 < t_2)$	
2	$(t_1 = t_2)$	
3	$(t_2 < t_1)$	

Table 6.2 All timestamping orders, $n = 2$

For $n = 3$, as in the file system models discussed in Chapter 5, the number of timestamping orders is higher:

Number	Order	

1	$(t_1 < t_2 < t_3)$	
2	$(t_1 < t_3 < t_2)$	
3	$(t_2 < t_1 < t_3)$	
4	$(t_2 < t_3 < t_1)$	
5	$(t_3 < t_1 < t_2)$	
6	$(t_3 < t_2 < t_1)$	
7	$(t_1 = t_2 < t_3)$	
8	$(t_3 < t_1 = t_2)$	
9	$(t_2 = t_3 < t_1)$	
10	$(t_1 < t_2 = t_3)$	
11	$(t_1 = t_3 < t_2)$	
12	$(t_2 < t_1 = t_3)$	
13	$(t_1 = t_2 = t_3)$	

Table 6.3 All timestamping orders, $n = 3$

6.3 Action sequences and possible timestamping orders

Timestamps can only be set by actions. The cause of timestamping having occurred in a specific order must have been actions that have occurred in a specific sequence. An action sequence is a sequence of actions of arbitrary length.

Definition 6.7. An *action sequence* is a sequence of one or more actions, where each element is related to the next element in the sequence with the equals-relation = or the less-than relation <. The equals relation imply that the actions occurred at the same time. The less-than relation imply that the first action occurred earlier than the second action.

The relationship between an action sequence and a timestamping order is that every observed timestamping order, must have been created by an action sequence. When considering all timestamping orders, there may be many action sequences that may cause that particular timestamping order. There may however also be timestamping orders,

which cannot be created by any action sequence. These timestamping orders cannot occur in the real system. The relationship between action sequences and timestamping orders can be deduced from the affects table.

Definition 6.8. A timestamping order is *possible in a system* if there is at least one action sequence that may cause the timestamping order. If there is no action sequence that can cause the timestamping order, then the timestamping order is *impossible in the system*.

By using the affects table, it is possible to find all action sequences that may have caused a specific timestamping order by the following algorithm, hereafter called the *AS-algorithm*:

1. Find all actions or combination of actions affecting all timestamps in the first step in the timestamping order.

2. For each following step in the timestamping order, find all actions or combination of actions affecting all timestamps in that step, and not affecting any timestamps listed in previous steps. If there is no such action or combination of actions, then this timestamping order is impossible in the system.

The task of finding all actions or combination of actions can be implemented as follows:

1. For every timestamp θ_i find all actions affecting it, and add them to a set A_i.
2. For every action $a \in A_i$, check if a affects any timestamp θ_j listed in previous steps in the timestamping order. If so, remove it from A_i.
3. Actions $a \in (A_1 \bigcap A_2 \bigcap ... \bigcap A_n)$ affect all timestamps in that step. Remove them from A_i.
4. If all sets A_i are still non-empty, the remaining actions represent combinations of actions affecting all timestamps for that step. The combinations can be found with the Cartesian product $A_1 \times A_2 \times ... \times A_n$

Example 6.9. Find all action sequences for the timestamping order ($t_c < t_m < t_a$) for an existing file in the simple file system with creation.

From the affects table for the simple file system with creation in Table 6.1, the steps in the timestamping order yields:

Step 1 (t_c): Create (t_c is only affected by Create)
Step 2 (t_m): Write (t_m is affected by Create and Write, only Write does not affect t_c)
Step 3 (t_a): Read (t_a is affected by Read/Write/Create, only Read does not affect t_c, t_m)

Thus, the only possible action sequence for timestamping order ($t_c < t_m < t_a$) is (Create < Write < Read).

Example 6.10. Find all action sequences for the timestamping order ($t_m = t_a < t_c$) for an existing file in the simple file system with creation.

From the affects table for the simple file system with creation in Table 6.1, the steps in the sequence yields:

Step 1 ($t_m = t_a$): Create, Write (t_m and t_a are both affected by Create and Write)
Step 2 (t_c): *none* (t_c is only affected by Create, but Create also affects t_m and t_a)

Thus, the timestamping order ($t_m = t_a < t_c$) is not possible in the system.

By using the AS-algorithm for all timestamping orders for a given number of timestamps, one can now complete the reasoning in a system with known actions. The result of this exercise will be a list of timestamping orders impossible in the system and a table of possible action sequences of each timestamping orders possible in the system. These results can be used with Theorem 5.10 to check the consistency of a clock hypothesis.

Example 6.11. Find all action sequences for the simple file system with creation.

This file system has three timestamps for each file ($n = 3$). All timestamping orders for such a system are given in Table 6.3. Assigning $t_1 = t_c$, $t_2 = t_m$ and $t_3 = t_a$ produces all timestamping orders for this system, shown in column "Stamping Order" in Table 6.4.

Following the AS-algorithm for each timestamping order listed in the table by using the affects table for the simple file system with creation given in Table 6.1, gives the possible action sequences for that timestamping order, shown in the column "Action Sequence":

Number	Stamping Order	Action Sequence
1	$(t_c < t_m < t_a)$	(Create < Write < Read)
2	$(t_c < t_a < t_m)$	None
3	$(t_m < t_c < t_a)$	None
4	$(t_m < t_a < t_c)$	None
5	$(t_a < t_c < t_m)$	None
6	$(t_a < t_m < t_c)$	None
7	$(t_c = t_m < t_a)$	(Create < Read)
8	$(t_a < t_c = t_m)$	None
9	$(t_m = t_a < t_c)$	None
10	$(t_c < t_m = t_a)$	(Create < Write)
11	$(t_c = t_a < t_m)$	None
12	$(t_m < t_c = t_a)$	None
13	$(t_c = t_m = t_a)$	(Create)

Table 6.4 Action sequences for the simple file system with creation

The only timestamping orders in Table 6.4 possible in the system are sequences where $t_c \leq t_m \leq t_a$. This is equal to the result achieved for the simple file system with creation in section 5.10.

6.4 Modelling a real file system

The AS-algorithm described in the previous sections can be used to create a model of a real file system, determine which timestamping orders are possible in the system and derive invariants of the file system for use with a clock hypothesis checker. To illustrate this algorithm, this section performs it on the semantics in Windows XP for file timestamps stored in the NTFS $STANDARD_INFORMATION attribute. The basis for

the model described here is the experiments described in Appendix C. The model assumes that the files in question exist, are larger than the file cache size, and that updating of the last accessed timestamp is enabled.

In a system with three timestamps, the affects table contains $2^3 = 8$ entries. The actions are:

Read: reading a file
Create: creating a new file
Write: modifying an existing file
CopySrc: copying a file (source file)
CopyDest: copying a file (destination file)
MoveIntra: moving a file internal to a file system
MoveInterSrc: moving a file across file systems (source file)
MoveInterDest: moving a file across file systems (destination file)

From the experiments, the following affects table can then be constructed:

	Created	Modified	Accessed	Actions
0				
1	X			
2		X		
3	X	X		
4			X	Read, CopySrc, MoveIntra, MoveInterSrc, MoveInterDest (ReadGroup)
5	X		X	CopyDest
6		X	X	Write
7	X	X	X	Create

Table 6.5 Affects table for Windows XP / NTFS

The actions in row 4 of the affects table all have the same effect on timestamps. In the following, they will be group together as ReadGroup, meaning that where this action

occurs, any of the actions Read, CopySrc, MoveIntra, MoveInterSrc or MoveInterDest may have occurred.

With $n = 3$, the timestamping order table in Table 6.3 can be used. Applying the AS-algorithm for each timestamping order yields the table of action sequences listed in Table 6.6.

Number	Stamping Order	Action Sequence
1	$(t_c < t_m < t_a)$	(Create/CopyDest < Write < ReadGroup)
2	$(t_c < t_a < t_m)$	None
3	$(t_m < t_c < t_a)$	(Create/Write < CopyDest < ReadGroup)
4	$(t_m < t_a < t_c)$	None
5	$(t_a < t_c < t_m)$	None
6	$(t_a < t_m < t_c)$	None
7	$(t_c = t_m < t_a)$	(Create/CopyDest=Write < ReadGroup)
8	$(t_a < t_c = t_m)$	None
9	$(t_m = t_a < t_c)$	None
10	$(t_c < t_m = t_a)$	(Create/CopyDest, Write)
11	$(t_c = t_a < t_m)$	None
12	$(t_m < t_c = t_a)$	(Create/Write, CopyDest)
13	$(t_c = t_m = t_a)$	(Create/CopyDest=Write)

Table 6.6 Timestamping orders in Windows XP/NTFS.

From the table, it is evident that there are no possible action sequences where t_a does not occur in the last step. Consequently, in this model, $t_m \leq t_a$ and $t_c \leq t_a$. These invariants can be used to check clock hypotheses for Windows XP systems with NTFS. An implementation of such hypothesis checking will be discussed in Chapter 7.

6.5 Complexity

It is interesting to determine the computational complexity of the AS-algorithm given in Section 6.3. As with the SEC-algorithm given in Chapter 5, this algorithm can be used manually, as well as in a computer program.

Let n be the number of timestamps associated with each file in the system. Let u_i be the number of actions affecting timestamp θ_i. The AS-algorithm involves finding the possible actions sequences, if any, for every timestamping order. All timestamping orders must be found, and the procedure of finding possible action sequences must be repeated for every timestamping order. The number of timestamping orders impacts the complexity of the AS-algorithm directly. The number of timestamping orders is, as already discussed in Section 5.11, equal to $p(n)$, the ordered Bell numbers. A list of these numbers for n up to 9 is given in Table 5.1. The complexity of $p(n)$ is given in Equation (5.33).

A timestamping order has at most n steps. For every step, the AS-algorithm mandates finding all actions affecting every timestamp, adding them to sets, and checking the resulting sets for actions or combination of actions that affect all timestamps in the step. This algorithm is $O(u_i)$ for every timestamp to find actions for, or

$$O(\sum_{i=0}^{n} u_i) \text{ as } n, u_i \to \infty$$

for finding actions for all timestamps. The actions must then be checked if they affect the timestamps in the previous steps in the timestamping order, something that will increase the complexity by a factor n, since the timestamping order has at most n steps. We can then express the computation complexity of the task for every step in the timestamping order as

$$O(n\sum_{i=0}^{n} u_i) \text{ as } n, u_i \to \infty$$

And since the timestamping order has at most n steps, the computation for every timestamping order is

$$O(n^2\sum_{i=0}^{n} u_i) \text{ as } n, u_i \to \infty$$

Inserting Equation (5.33), the term n^2 can be removed, since it grows considerably slower than the factorial growth of the Bell numbers. We then obtain for the AS-algorithm:

$$O(n!\sum_{i=0}^{n} u_i) \text{ as } n, u_i \to \infty$$

It is interesting to note the difference of the growth of the AS-algorithm with the growth of the SEC-algorithm, expressed in Equation (5.35). While the growth when $n \to \infty$ in both cases is proportional to the growth of the Bell numbers, the growth of the AS-algorithm when $u_i \to \infty$ is proportional to the *sum* of the number of affects for each timestamp. In the SEC-algorithm, the growth is proportional to the *product* of the number of Initiates clauses for each timestamp in the event calculus program. This difference can be considerable in the evalution on a model with a large number of actions.

The difference can be explained by the order of checking the actions affection on timestamps. In the SEC-algorithm, we first formulate action hypotheses, and then check if each action hypothesis can occur with the different timestamping orders. When the action hypotheses are formulated, there is no implicit knowledge of which hypotheses will be refuted in the different timestamping orders, so all possible combinations of initiating actions must be hypothesized, yielding the number of hypotheses expressed in Equation (5.32). In the AS-algorithm on the other hand, actions are selected for every step of every timestamping order and tested for consistency with the previous steps. This eliminates the need for computing all possible action combinations and testing every combination with every timestamping order, thereby reducing the growth when $u_i \to \infty$.

6.6 Representing the affects table as a graph

A further simplification of the described derivation of invariants can be achieved by reasoning directly on the elements of the affects table. This simplification is best illustrated by representing the affects table as a graph.

Definition 6.12. An *affects graph* is a representation of the affects table as a bipartite graph, in which timestamps and actions are represented with vertices and affects with edges. Timestamps are represented with vertices of one color and actions with vertices of another color. Affects are represented with edges between timestamp vertices and action vertices.

Example 6.13. The graph in Figure 6.1 shows the affects graph for the simple file system with creation as per the affects table in Table 6.1.

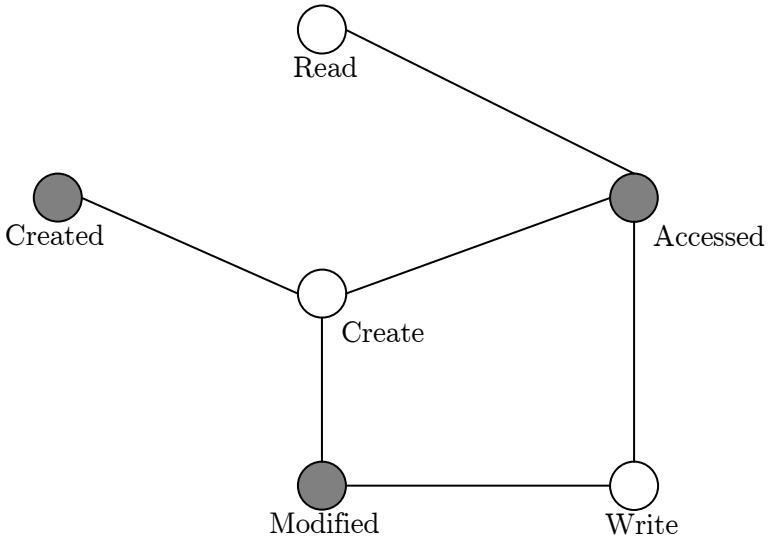

Figure 6.1 Affects graph for the simple file system with creation

The description of the affects table as a graph highlights how the affects table can be seen as a system of interconnected entities, where timestamps and actions are entities and affects are connections. This suggests a type of reasoning directly on the connections between timestamps through intermediary actions, without having to rely on an exhaustive search of different timestamping orders. Consider the Created timestamp in Figure 6.1. This timestamp is affected by the Create action, which in turn affects the Modified timestamp and the Accessed timestamp. Create is also the only action affecting the Created timestamp. Thus, whenever the Created timestamp is updated in this system, the Accessed and Modified timestamps are updated too. Now consider the Modified timestamp. This timestamp is affected by the Create action and the Write action. These actions both in turn affect the Accessed timestamp. Thus, whenever the Modified timestamp is updated, the Accessed timestamp is also updated. These relationships can be expressed as a directed graph of timestamps as shown in Figure 6.2. The arcs in this directed graph represent that whenever the tail timestamp is updated, the head timestamp is updated too. Since any update to the tail timestamp also updates the head timestamp, the arcs is this graph translate to invariants by which the stamping time of the tail timestamp must be less than or equal to the stamping time of the head timestamp.

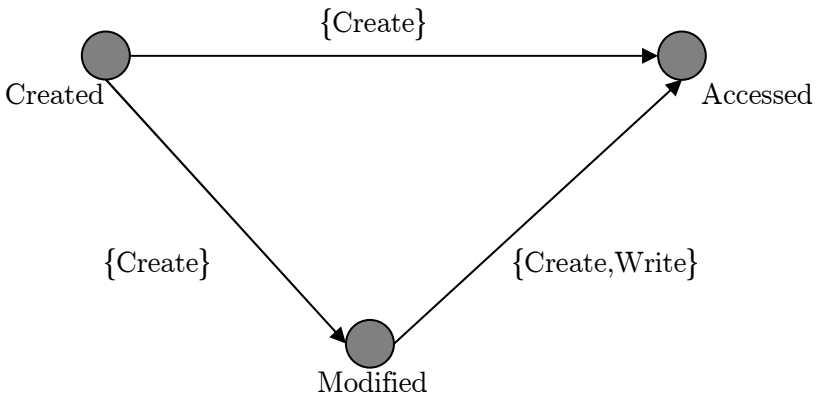

Figure 6.2 Timestamps in the simple file system with creation.

From Figure 6.2, we see that any updates of the Created timestamp also update the Modified timestamp and the Accessed timestamp, and any updates of the Modified timestamp also updates the Accessed timestamp. Thus, $t_c \leq t_m$, $t_c \leq t_a$ and $t_m \leq t_a$ and consequently $t_c \leq t_m \leq t_a$, which equals previous results for this file system.

Definition 6.14. An *invariant graph* is a directed graph in which timestamps are represented with vertices and invariants with arcs. An arc from timestamp θ_i to θ_j represents the invariant $t_{\theta_i} \leq t_{\theta_j}$

It is now possible to devise an algorithm for the derivation of the invariant graph directly from the affects graph, hereafter called the *IG-algorithm*:

- Every timestamp vertex in the affects graph is a vertex in the invariant graph
- For every timestamp vertex θ_i in the affects graph:
    - For every action a_j affecting θ_i:
        - Build a set Ω_{a_j} of timestamps affected by a_j not including θ_i
    - Find $\Omega = \Omega_{a_1} \bigcap \Omega_{a_2} \bigcap ... \bigcap \Omega_{a_m}$
    - For every $\theta_k \in \Omega$, insert an arc from θ_i to θ_k in the invariant graph

Example 6.15. Draw the invariant graph in for Windows XP / NTFS. The affects table for XP / NTFS is given in Table 6.5. From this table, we get the affects graph shown in Figure 6.3. By applying the IG-algorithm, we obtain the invariant graph shown in Figure 6.4.

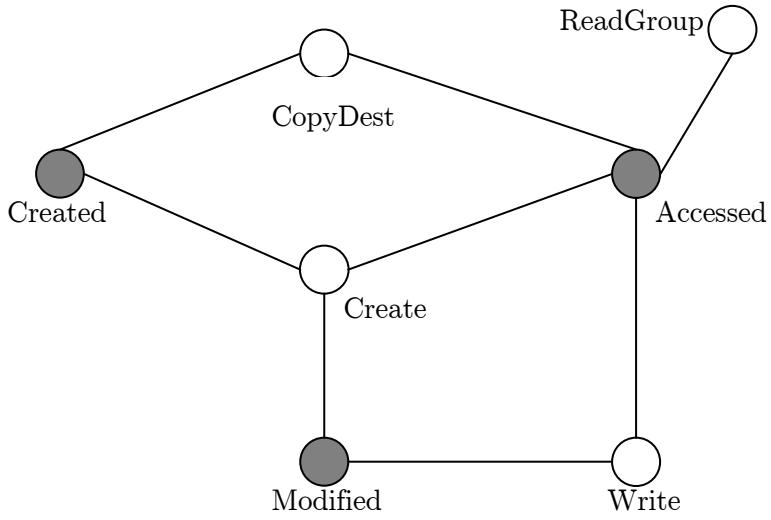

Figure 6.3 Affects graph for Windows XP / NTFS

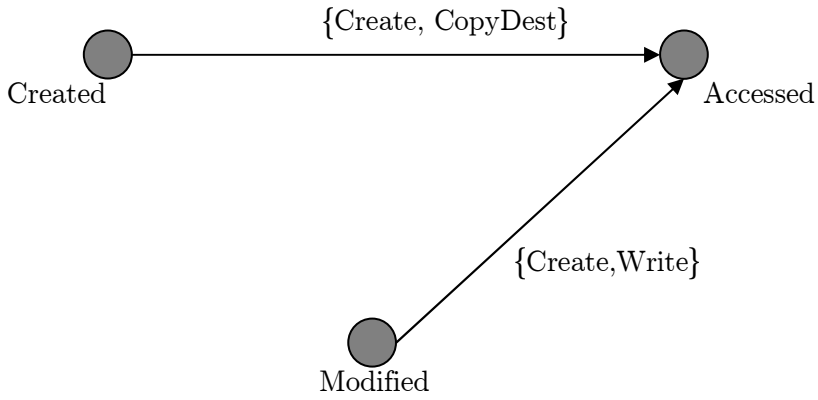

Figure 6.4 Invariant graph for Windows XP / NTFS

To compare the IG-algorithm with the previously described algorithms, its complexity must be determined. The IG-algorithm is characterized by iterating over all timestamps and finding all different connections to other timestamps through the actions connected

to these timestamps with *affects*. Let n be the number of timestamps. We then call k_i the number of ways timestamp i is connected to other timestamps through actions. Since the IG-algorithm iterates over all timestamps and for each timestamp finds other timestamps connected to it through actions, the complexity can be expressed in terms of k_i as:

$$O(\sum_{i=0}^{n} k_i) \text{ as } n, k_i \to \infty$$

It is interesting to determine the growth of k_i in terms of the number of timestamps, actions and affections in the affection table. To do this, one must find the relationship between the number of affects and k_i. In order to visualize this relationship, a graph of connections between timestamps through actions can be derived from the affects graph. Such a graph is shown in Figure 6.5 for the affects graph in Figure 6.3.

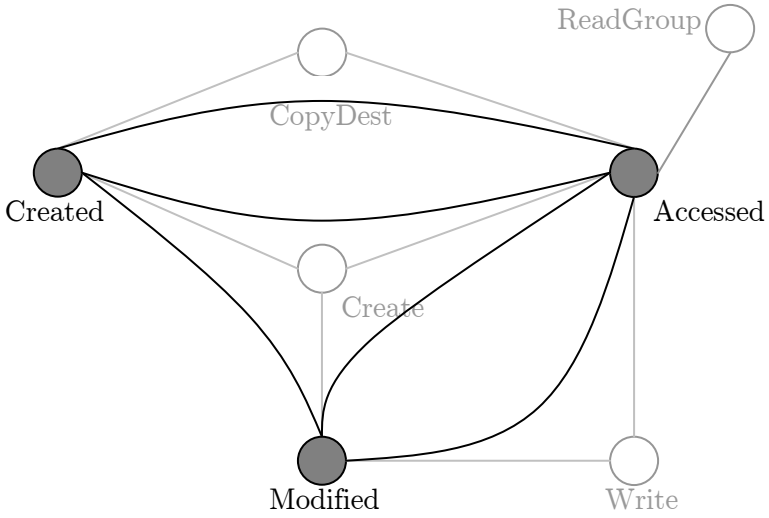

Figure 6.5 Connections between timestamps through actions

In Figure 6.5, k_i is the number of black arcs connecting timestamp i to other timestamps. For example, for the Modified timestamp, $k_{\text{m}} = 3$. Now, let the total number of connections between timestamps through actions be ω. For the graph in Figure 6.5, ω is the number of black arcs, which is 5. Now, since k_i is the number of connections from each timestamp, we have:

$$\omega = \frac{1}{2}\sum_{i=0}^{n} k_i \quad \Leftrightarrow \quad \sum_{i=0}^{n} k_i = 2\omega$$

ω can also be expressed in terms of the number of actions, and the number of affects they have. Let v_i be the number of timestamps affected by action a_i. The number of connections between timestamps through action a_i is then given by the triangular number $T(v_i)$, where,

$$T(v_i) = \sum_{j=0}^{v_i} j = \frac{v_i(v_i + 1)}{2}$$

So the total number of connections is a matter of summing over actions. Let m be the number of actions. Then,

$$\omega = \sum_{i=0}^{m} T(v_i) = \sum_{i=0}^{m} \frac{v_i(v_i + 1)}{2}$$

The complexity of the IG-algorithm can now be found by inserting 2ω as follows:

$$\sum_{i=0}^{n} k_i = 2\omega = 2\sum_{i=0}^{m} \frac{v_i(v_i + 1)}{2}$$

And the complexity is:

$$O(\sum_{i=0}^{m} {v_i}^2) \text{ as } m, v_i \to \infty$$

The complexity of the IG-algorithm depends on the number of actions and the number of affects on each action. This is different from the previous algorithms, whose complexity depends on the number of timestamps n and the number of affects on each timestamp. In order to be able to compare the two approaches, it is interesting to find an upper bound on the complexity of the IG-algorithm. For v_i, we know that the upper bound is n, since no action can affect more than the total number of timestamps in the system. With $v_i = n$ the boundary on the IG-algorithm in terms of n becomes:

$$O(\sum_{i=0}^{m} n^2) = O(mn^2) \text{ as } m, n \to \infty$$

Thus, the growth of the IG-algorithm is quadratic with the growth of n and linear with the growth of m. This compares quite favourably with the $O(n!)$ algorithms given in previous sections.

6.7 Comparison with Simplified Event Calculus

The affects table reasoning described in this chapter provides a simplification compared to the Event Calculus logic described in Chapter 5. The construction of a model is simpler; instead of defining a complete Event Calculus program, the model can now be constructed by defining an affects table. The affects table lists the timestamp fluents as columns and all possible affects combinations as rows. Then, actions in the system are listed in rows according to the timestamp fluents it affects. This way of constructing a model makes the property that the setting of a new timestamp overwrites the previous value of the timestamp implicit, rather than the explicit Initiates and Terminates clauses used in the Simplified Event Calculus. This makes it more straightforward to build a model of a system with many actions, and also easier to understand the model. This was seen in Section 6.4, where an affects table for Windows XP was constructed, whereas a model for Windows XP in Simplified Event Calculus would have to include a significant number of actions and therefore also a significant number of Initiates and Terminates clauses. The grouping of actions in the affection table also allows for a further simplification, since actions with equal timestamp affection can be grouped together and handled as a single action.

The simplification however means that the ability to represent arbitrary fluents and actions is lost. In the construction of an affects table, it is assumed that all actions affect the timestamps by terminating the previous value and initiating a new value. Further, representation of fluents and actions that are not timestamps is not allowed. Thus, the simpler model represented by the affects table is not able to represent all kinds of systems in the same way as is possible in the Simplified Event Calculus. For example, the affects table is not able to represent the fluent Exists in the simple file system with creation described in Section 5.9, and so the affects table shown in Table 6.1 is only valid for existing files.

The most important simplification achieved by the use of affects tables over event calculus is in the determination of invariants in the system. With the AS-algorithm, this is achieved by determining which action sequences, if any, can cause each timestamping order. With the IG-algorithm, invariants are determined directly from the affects table and its representation as a graph. These algorithms find invariants that must hold for the system, such as $t_c \leq t_m \leq t_a$ for the simple file system with creation. In the SEC-algorithm on the other hand, it is necessary to first find action hypotheses from an observation proposition by finding the initiation proposition and then test each of the action

hypotheses by resolution for each timestamping order, as shown for the simple file system with creation in Figure 5.10. In this process, many of the action hypotheses found with the initiation proposition will be refuted, because the fluents initiated by the hypothesized actions will be terminated by subsequent hypothesized actions. In a model with many actions, this process will become overly complex, due to the number of different hypotheses that must be tested.

Another difference that should be noted is the lack of determination of action sequences corresponding to timestamping orders in the IG-algorithm. In the SEC-algorithm and the AS-algorithm, all action sequences are tested for consistency with every timestamping order. This means that for an observed timestamping order, the investigator can not only determine if it is consistent with a given clock hypothesis. He can also determine which action sequences may have caused it. Such an interpretation can not be made when the IG-algorithm has been used, since it only determines invariants and not possible actions sequences for specific timestamping orders. It is however possible to use the IG-algorithm for invariant derivation and the AS-algorithm for determination of possible action sequences for specific observed timestamping orders.

In summary, the use of an affects table for reasoning about possible action sequences provides a simplification of the derivation of invariants for a system, compared with the Simplified Event Calculus Reasoning. The given algorithms can be used to derive invariants and test clock hypotheses in systems with greater number of possible actions. As shown with the model of Windows XP, the simplification makes it possible to represent and test real systems manually. The affects table is however less expressive, so the simplification comes at the cost of possibly not being able to express all states that might occur in a real system.

7 IMPLEMENTATION AND EXPERIMENT

Chapter 3-6 has introduced and refined a theoretical model for clock hypothesis testing. In this chapter, this theoretical model is implemented in software for analysis of a real file system. The implementation is then used to find evidence of antedating in an experiment where four subject were asked to antedate a document.

Section 7.1 - 7.2 introduces the implementation. Section 7.3 details the results of an initial test of the implementation. Section 7.4 - 7.6 then describes the document antedating experiment, and the results when the experiment results were analyzed with the implementation.

7.1 Purpose

The focus of this work has been the development of a theoretical system for the formulation and testing of clock hypotheses in digital investigations. The theoretical system can be used as a base for new methods for digital investigation and digital evidence interpretation in real digital investigations. Such methods may be manual methods employed by investigators or fact finders in specific cases. It may also be automated methods implemented in computer programs specifically made for the purpose of timestamp investigation, or as part of an all-purpose program for digital investigation.

A question of importance is how the theories presented in the previous chapters can be utilized in software, and how the results from such software can be interpreted. Hard drives investigated in typical digital investigations contain tens- or perhaps even hundreds of thousands of timestamps. All these timestamps may take part in the establishment of a consistent clock hypothesis. The large number of timestamps found in real investigations makes it difficult to establish a consistent clock hypothesis by using manual methods. Performing this analysis with automated tools is therefore desirable.

In order to determine if and how the presented theories could be translated into a program, an implementation of a clock hypothesis consistency tester was made in software. The implementation is named TimeStampLogic, and realizes clock hypothesis

consistency tests on the Windows XP operating system. In order to find out if the TimeStampLogic implementation could make a difference in a real investigation, a document antedating experiment was performed. Subjects were asked to antedate documents on a computer, and the results were analyzed using the TimeStampLogic program. The purpose of this experiment was to create the same kind of investigation challenges facing the investigators in the cases presented in Section 2.1 and 2.3 in a controlled environment, and then determine if the TimeStampLogic implementation would help solving them.

7.2 TimeStampLogic implementation

The TimeStampLogic program is implemented in Java J2SE. The program uses the utilities in a modified version of the Sleuthkit [37] to find file instances in an NTFS file system image. TimeStampLogic parses the output of these utilities and produces internal representations of file instances, which can then be analyzed using the reasoning in Chapter 3-6. The TimeStampLogic source code can be obtained by following the instructions in Appendix D.

In the implementation, a clock hypothesis is defined by a class implementing the `ClockHypothesis` interface. The implementation provides one hypothesis; `DefaultHypothesis`, in which the hypothesis is that the clock of the investigated system is an ideal clock.

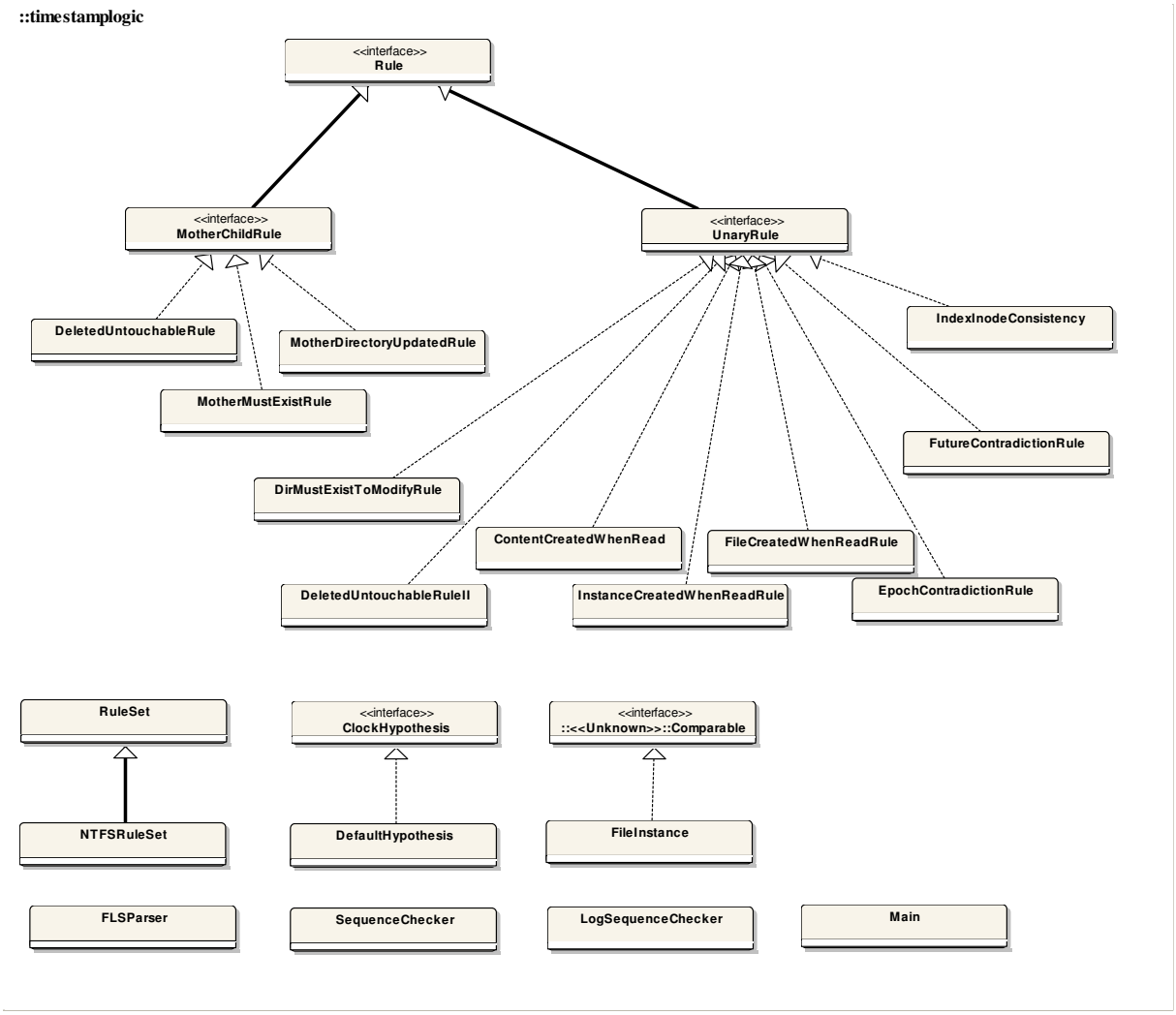

Figure 7.1 UML class diagram of the `TimeStampLogic` implementation

The input for the timestamp logic implementation is provided by a modified version of the Sleuthkit. The class `FLSParser` runs `fls` on a target image and `istat` on each file instance listed. It then parses the output for each file instance. Each file is represented with an instance of the class `FileInstance`. This class contains members representing the timestamps found in the image, as well as members representing the type (file or directory), the status (existing or deleted), the file sequence number and the log file sequence number.

The implementation tests the defined clock hypothesis by means of a set of tests derived from the analysis of a particular system. The tests implement the `Rule` interface, through

subinterfaces `UnaryRule` and `MotherChildRule`. With `UnaryRule`, the tests pertain to timestamps of a single file or directory instance. With `MotherChildRule`, each test is applied to a pair consisting of a file or directory instance (the child), and the directory in which that instance is contained (the mother). A complete set of tests for a particular system is defined by implementing the interface `RuleSet`. Implementations of this interface specify all tests that apply for a particular type of system. The implementation provides the class `NTFSRuleSet` containing a set of consistency tests for NTFS.

The following tests were implemented and used in `NTFSRuleSet`:

- *EpochContradiction:* A timestamp should not have been set at a moment prior to the invention of digital computers.
- *FutureContradiction:* A timestamp should not have been set at a moment posterior to the time of the investigation.
- *FileCreatedWhenRead:* The stamping time of the Created timestamp should be prior to the stamping time of the Accessed timestamp. (See Table 6.6)
- *ContentCreatedWhenRead:* The stamping time of the Modified timestamp should be prior to the stamping time of the Accessed timestamp. (See Table 6.6)
- *MotherDirectoryUpdated*: The stamping time of the Modified timestamp should be prior to the stamping time of the Created timestamps of files and directories contained in it. (See Appendix C.3)
- *DeletedUntouchable*: Deleted files are not timestamped. Therefore, the stamping time of deleted file timestamps should be prior to the Modified timestamp of the mother directory. (See Appendix C.3)
- *DeletedUntouchableII*: When a file entry has been reallocated to another file, the stamping time of the Accessed timestamp of that file should be posterior to the stamping time of all timestamps of the deleted file that previously occupied that file entry. (See Appendix C.3)
- *IndexInodeConsistency:* For existing (non-deleted) files, the timestamps in the directory index should match those in the $STANDARD_INFORMATION attribute of the MFT entry.

The tests in NTFSRuleSet are applied on all files and directories in the investigated image by FLSParser. For every test producing a negative result, the file name and timestamps are printed with an identification of the test that failed. A failed test

indicates that the clock hypothesis is incorrect, or that actions not included in the model generating the tests have occurred in the system.

The class `SequenceChecker` tests the sequence number causality of the file entry sequence numbers, by the reasoning in Section 4.5.3. It reads the sequence of file entries from the Master File Table with `ils`. All entries are compared with the last preceding entry with the base generation sequence number. Since the allocation of Master File Table entries are done in a first-fit fashion, all entries have been stored at a later time than the last preceding entry with the base generation sequence number. The implementation compares both the Created timestamp and the Accessed timestamp of each entry with the last preceding entry with the lowest sequence number.

The class `LogSequenceChecker` tests the causality of the updating of file entries based on the log file sequence numbers stored in the Master File Table file entries, by the reasoning in Section 4.5.5. The `FileInstance` instances read by `SequenceChecker` are inserted into a sorted `TreeSet`. By implementing the `Comparable` interface in `FileInstance`, comparing the log file sequence numbers, this structure sorts all file instances by their log file sequence numbers. The file instances are then printed in sorted order, so that their timestamps can be inspected for evidence of clock hypothesis inconsistency. The MFT Entry Modified timestamp in the $STANDARD_INFORMATION attribute is of special interest in this context, because storing a new Log File Sequence Number is a modification of the MFT Entry.

7.3 Results from initial TimeStampLogic test runs

Several test runs of the TimeStampLogic implementation were performed. The test runs were made on a real Windows XP/NTFS image, in order to make an initial assessment of how this implementation can be utilized in real investigations, and if the results can really be used to put a clock hypothesis under scrutiny. The clock on the investigated computer had been within reasonable synchronization with Norwegian civil time. It may have drifted a few minutes, but errors on the scale of days or years should not have occurred. The image was tested for consistency with `DefaultHypothesis`.

7.3.1 RuleSet

The initial runs produced a large number of inconsistencies for the tests in `NTFSRuleSet`. Most, but not all, of the inconsistencies fell within the following categories:

1. Instances where the compared timestamps were only small fractions of a second from satisfying the test. Most of these inconsistencies were with the *MotherDirectoryUpdated* test.
2. Instances of files that had been unpacked with packers such as WinZIP and WinRAR.

It is reasonable to attribute the inconsistencies in both of these categories to an incomplete model. In the case of category 1, the most likely explanation is that the updating of the Created timestamp of a file or directory and the Modified timestamp of its mother directory does in fact not occur at the same time, but with a small delay. This would occur for example if the file creation implementation in the operating system starts by creating the MFT index for the file, then writes the file to disk, and then finally updates the mother directory. In such a system, the delay between the updating of the two timestamps will vary with the length of the file, something that is consistent with the observed results for Windows XP/NTFS. This possibility can only be confirmed by close inspection of the operating system. (See C.5) In the case of category 2, the most likely explanation is that the packer programs update file timestamps in ways that are not consistent with the handling in the operating system. By performing experiments with these packers, in a similar fashion to the operating system experiments detailed in Appendix C, one could determine how these handle timestamps and update the model to reflect the changes.

Most of the remaining inconsistencies were related to a series of digital images shot with a digital camera and then copied directly to the computer in question. These were inconsistent with ContentCreatedWhenRead, such that the Accessed timestamp were 2-3 months prior to the Modified timestamp. It is known that the Modified timestamp is retained from the source medium, and the Accessed and Created timestamps produced on the destination medium during copy operations. (See Appendix C) A possible cause for the inconsistencies could therefore be a maladjusted clock on the digital camera. The camera was therefore checked, and it was indeed found that its clock was adjusted

approximately 4 months into the future. The Modification time had been set to a point in time in the future when the images were shot, and this timestamp had been retained when the images later had been copied to the computer.

A few inconsistencies that could not be explained by any of the above were also reported. It is unknown if these were due to an incomplete model or if the clock hypothesis was incorrect.

7.3.2 SequenceChecker

The initial run for SequenceChecker also produced a number of inconsistencies, although remarkably fewer than for RuleSet. Approximately 98% of the files on the file system were ordered when compared in this fashion. The remaining files which were reported as inconsistent were scattered out across the image. No specific reason for the inconsistencies could be inferred from knowledge about the files where inconsistencies were reported. It is likely that these consistencies root in the lack of full knowledge of how the operating system works. There may for example be operations occurring within the operating system where MFT entries are allocated without setting the Created timestamp of the file in question. The opposite may also be the case; that the Created timestamp is updated when in fact a new MFT entry was not allocated. As will be seen in Section 7.6, this does not however mean that this method is without value.

7.3.3 LogSequenceChecker

For the initial run of LogSequenceChecker, only a small number of inconsistencies were reported, when the MFT modified timestamp was compared. The list of files and associated timestamps sorted by Log Sequence Number, provided a list of when the computer was used, in which MFT Entry modified timestamps came in succession for more than 99,9% of the file entries. Most of the inconsistencies were reported on file entries pertaining to deleted files, although a few were also reported on existing files. A possible theory regarding deleted files is that new log sequence numbers is assigned when files are deleted, and the Entry Modified timestamp is not updated in this case. The

reported inconsistencies were scattered out across the file list and did not represent many files in sequence.

The results from the initial test indicate that the LogSequenceChecker test could be an adequate tool for use in real digital investigations, for testing clock hypotheses and for finding antedated material.

7.4 Document antedating experiment

The document antedating experiment was designed to produce the same kind of investigation situation as in the case discussed in Section 2.1. Subjects were handed a computer and asked to antedate documents on it. In order to limit the workload associated with preparation and analysis, the number of subjects was kept small. The subjects were chosen so that they represented users with diverse level of experience.

A laptop computer was prepared for the experiment. First, the hard drive of the computer was wiped with the tool Win-Hex [38], writing value 0x00 in every byte of every block on the hard drive. The computer was then started and the system clock was adjusted to approximately two and a half years before the time of the experiment with the BIOS setup program. Then, the Windows XP operating system was installed. After installation, a series of shutdowns, clock adjustments and reboots were performed. The goal of this procedure was to produce data on the hard drive similar to data that would have been produced by real usage of the computer. For each step, the computer was shut down, then started in BIOS setup, where the clock was adjusted forward. The computer was then booted into Windows XP and used for websurfing, downloading files or other typical user activity. Table 7.1 summarizes the clock adjustments that were used and the associated activities.

Date	Activity
01-Mar-2004	Operating system install
02-Mar-2004	Install completed
02-Mar-2004	Websurfing
02-Jun-2004	Websurfing
02-Sep-2004	Websurfing, downloaded videos

02-Sep-2005	Websurfing
02-Jan-2006	Websurfing, downloaded thesis
02-Mar-2006	Websurfing, added user account
02-Jun-2006	Websurfing, downloaded video
02-Jul-2006	Changed password, websurfing
02-Aug-2006	Websurfing
12-Aug-2006	Installed software
24-Aug-2006	Websurfing
01-Sep-2006	Edited documents
12-Sep-2006	Websurfing
October 2006	Time of the experiment

Table 7.1 Timeline of experiment computer

After this procedure, the hard drive was copied to an image file on another hard drive using the disk dump utility `dd`, producing a reference image of the experiment computer. The reference image can be obtained by following the instructions in Appendix D.

The experiment computer was then handed to the participating subjects with the following task: "Store a document on this computer in such a way that a person investigating the computer will conclude that the document was produced on 17-May-2006." When each subject returned the computer, the hard drive was copied to an image file on another hard drive for analysis. Then, the experiment image was copied back to the computer before it was handed to the next subject. The subjects participating in the experiment are listed in Table 7.2.

.

Subject no	Computer experience level
1	Average computer user, using computer every day for office work
2	Law Enforcement Computer Forensic Investigator
3	Inexperienced office user, mostly used to websurfing and light office work
4	Advanced computer user with some programming experience

Table 7.2 Participating subjects

Each image was analyzed using the TimeStampLogic program. Each subject was also interviewed to determine how they had chosen to perform the task. The statements of the subjects and the results of analysis using TimeStampLogic in each case are summarized in Section 7.6.

7.5 Analysis of the reference image

At the start of the experiment, the reference image was analyzed with TimeStampLogic. The reference image contained considerably less data than the image analyzed in the initial test described in 7.3, and in this case it was also known in more detail how the data on the computer had been stored on it.

As with the initial test, a significant number of inconsistencies were reported with RuleSet. These inconsistencies were reported for the same categories of files as described in Section 7.3.1. It was therefore decided to focus on the SequenceChecker and LogSequenceChecker tests on the images in the antedating experiment. The results with these tests were consistent with the results in the initial test. SequenceChecker produced a significant number of inconsistencies, whereas the number of inconsistencies with LogSequenceChecker was fairly small and mainly with deleted files.

7.6 Results

In the following, each of the images resulting from imaging the experiment computer after each subject had completed the task is analyzed. The purpose of the analysis is to determine if the document in question has been antedated or not. This can be formulated as two different hypotheses:

H_0: The document was produced on 17th of May.
H_1: The document was produced later than 17th of May, but has been antedated to 17th of May.

The task for the investigator is then to find evidence supporting or rejecting H_0 and H_1 using TimeStampLogic and other investigative tools and present them to the fact finder.

7.6.1 Subject 1

The subject gave the following information about how the task was completed: *I adjusted the clock on my Mac to May 17th. I then produced the document in Microsoft Word on the Mac. When saved on the Mac, I copied the document to my USB stick and inserted it into the PC. I then copied the document from the USB stick to the PC. I believe producing the document on the Mac may have prevented the creation of timestamps inside the Word document.*

When analyzed with TimeStampLogic, the results of this operation did not produce a result significantly different from the analysis of the reference image. The introduction of new files when the computer was booted and a new document was copied to it, did not produce any new inconsistencies reported by TimeStampLogic. The document has Modified timestamp on the 17th of May, and Created and Accessed timestamps on the date of the experiment. This is consistent with timestamps produced when files are copied to a medium. (See C.3) Other evidence suggesting that the file had been copied to the medium on the date of the experiment was also found, for example a link-file to an external drive, showing that an external drive had been connected to the computer. If the file had been copied from another computer, the Modified timestamp would then be related to the clock of that computer and not the investigated computer. Since no evidence is available to test clock hypotheses for the other computer, there is no evidence to either support or reject a hypothesis that the production of the document actually occurred on 17th of May civil time. The analysis is therefore inconclusive in this case. The reasonable investigative response in cases like this is to try to get hold of the computer on which the document was produced and do the same type of analysis on that.

In response to the subject's claim that timestamps had not been created inside the Word document, it was examined for timestamps in the metadata. Such timestamps were found, identifying that the document had been created and last changed on May 17th. These timestamps would also refer to the clock on the other computer, which will have to be analyzed for evidence.

7.6.2 Subject 2

The subject gave the following information about how the task was completed: *I started the PC and connected it to the Internet. I then downloaded and installed OpenOffice on the PC. I then restarted the computer, went into BIOS and adjusted the date back to May 17th. After booting the computer again, I used OpenOffice to create and store the document. I then booted again and adjusted the clock back to current time. I used OpenOffice because I think it doesn't have the same amount of metadata as Microsoft Word. I also think downloading and installing OpenOffice would prevent a proper investigation, since it probably overwrote a lot of deleted data.*

When analyzed with TimeStampLogic, a significant higher number of inconsistencies were reported with both SequenceChecker and LogSequenceChecker. Listing all files on the medium ordered by both the MFT Entry number (SequenceChecker) and Log Sequence Number (LogSequenceChecker), showed several hundred files with Created, Modified and Accessed timestamps on Oct 11th in the time period 07:28-07:40 AM. After these (in terms of entry number and log sequence number), approximately 50 files with Created, Modified and Accessed timestamps on May 17th time period 07:42-07:48 were listed. All timestamps of the document in question were set to May 17th in the period 07:42-07:48.

The timestamps on the document itself were in this case set to May 17th, in contrast to Subject 1. This fact does not by itself support either H_0 or H_1, since the document may have been antedated by timestamp or clock manipulation.

There is however evidence in this case supporting H_1:

- Storing of a significant number of files causally dependant on storing of files occurring on Oct11th, were timestamped May 17th, something that is not possible unless the clock has been adjusted, or the timestamps changed.
- While the date changes from Oct 11th to May 17th, the time of day only moves approximately 2 minutes forward. This indicates that the subject changed the date but did not bother to change the time of day. If the clock adjustment had

occurred by an error or some other mystery event, it is not very likely that it would have ended up on this exact time of day.

The subject's claim that he made the investigation more difficult by installing OpenOffice, does not seem to be correct in the context of using TimeStampLogic to check clock hypothesis consistency. It may be the case that installing a new program would overwrite previously overwritten material, but this does not help, since TimeStampLogic does not rely on the investigator's ability to recover deleted material. The claim that OpenOffice documents contain less metadata than Word documents was not investigated. Since the document was created with the clock adjusted to May 17th, any timestamps within the document would be on May 17th anyway.

7.6.3 Subject 3

The subject gave the following information about how the task was completed: *I don't know how to manipulate timestamps, so I just went into the control panel and set the date to May 17th. Then I used Microsoft Word to produce the document. Then I set the current date again in the control panel.*

On this image, TimeStampLogic produced the same type of results as on the image from Subject 2. Approximately 10 files were listed with Created, Modified and Accessed timestamps on Oct 12th from 9:17-9:44 PM. After this (in terms of MFT entry sequence and LSN sequence), approximately 10 files were listed with timestamps at May 17th 9:46-9:52 PM. This gives evidence for H_1, for the same reasons as for Subject 2.

In this case, as opposed to the case of Subject 2, the clock change was done in the operating system. Therefore, the event logs of the system were searched to determine if the clock change had logged a system event. No such event was found. Windows XP has a system logging feature that allow logging of clock change events. This particular event is however logged only if Privileged Use logging is enabled, something it is not by default. [39] Since very few users change the default settings of the event logging system, the logging of clock changes in Windows XP is not of much use in digital investigations.

It is interesting to note that both Subject 2 and 3 changed the date without changing the time of day. Both in the BIOS of the experiment computer and in the Windows XP control panel, changing date is done by a separate control than changing time of day, although they are both related to the same underlying clock. A plausible rationale for not changing the time of day could be that it would then be easier to adjust the clock back to the current time, because one would then not have to resynchronize with an external clock. When asked about this, subject 3 said: *I didn't think about that. I just wanted the correct date on the document. The time of day didn't matter to me. I might have thought about it if the time of day were of any importance, for example if it mattered if I were at work at the time or not.*

7.6.4 Subject 4

The subject gave the following information about how the task was completed: *I used my own pc for the antedating. I adjusted its clock back to May 17^{h}, and produced the document using Microsoft Word. I then copied the document over to the experiment PC using my USB-stick.*

The story of Subject 4 matches the story of Subject 1, and the results of TimeStampLogic were similar. No additional inconsistencies were found, and the results were inconclusive on the question of whether the document was antedated or not. Also in this case, link files pointing to an external medium identified another computer as the likely source for the document.

.

7.7 Summary

In the document antedating experiment, four subjects were asked to antedate a document in such a way that it could not be determine that the document file was antedated. Two of the subjects performed the antedating in such a way that the methods described in this work could produce evidence supporting the hypothesis that the document was antedated and not produced on the date it was timestamped to. Two of the subjects did the antedating itself on another computer and copied the resulting document to the investigated computer. In this case, it could not be determined that the document was

antedated, but it could be determined that the document had been copied from another computer, and so another possible item of evidence was found. From the explanation from the subjects, it is known that they produced the antedated document on the other computer by adjusting the clock back to May 17th, which is the same method used by Subject 2 and 3 on the investigated computer. Investigation of the other computer with the methods described in this work would therefore most likely have produced evidence supporting the hypothesis that the document was antedated.

The antedating methods used by the subjects in the experiment are certainly not the only possible methods for document antedating. Other possible methods can be conceived:

1. Produce the document at current time, then changing its timestamps by special software. This can be done without introducing the software in question on the investigated computer by removing the medium and perform the change on another computer.
2. Finding another file matching the desired timestamps, then replacing the contents of that file with specialized software.
3. Using the same method as used by the subjects in the experiment. Then use special software that adjusts all timestamps on the medium to match the default clock hypothesis.

In the case of conceived method 1, TimeStampLogic would probably report the single file as an inconsistency. In the case of conceived method 2 and 3 however, it is not likely that TimeStampLogic would be able to find any inconsistencies. Producing evidence of antedating in these cases would have to rely on other methods, if possible at all. Thus, clock hypothesis testing methods described here are not perfect methods that cannot be avoided by a crafty antedater. This can however be said about any investigation method, as discussed in Sections 8.4 - 8.5.

In summary, the document antedating experiment has shown that TimeStampLogic can provide evidence of antedating of computer files in practical situations, where subjects have put effort into antedating a file. As previously mentioned, the purpose of TimeStampLogic has been to determine if clock hypothesis testing could make a difference in a real investigation. It has not been intended as a program for direct use in

real investigations. The described methods can easily be implemented in existing tools for digital investigation such as the Sleuthkit or EnCase. This would provide investigators already using these tools with the possibility of reason on time and causality and define and test clock hypotheses also in real investigations.

8 EVALUATION AND CRITICISMS

In the previous chapters, a scheme for reasoning on timestamps and clock hypothesis testing was discussed. In this chapter, several possible criticisms against this scheme are discussed.

Section 8.1 discusses the state explosion involved in the reasoning, and how it affects investigations. Sections 8.2 - 8.3 discusses the correctness and completeness of the model representing the investigated systems. Section 8.4 discusses the Arms Race Argument, by which every forensic technique is thought to ultimately be hindered by an anti-forensic technique. Finally, Section 8.5 discusses how falsified evidence can be created to match an invented event history.

8.1 Complexity

In this work, a system is modelled by representing changes in the system as events, and then hypothesizing possible sequences of events that may have resulted in the state of the system observed during the investigation. A conceivable criticism against any scheme in which the working of a digital system is represented in a model consisting of states and actions, is the complexity of real systems. If a real system is to be represented accurately in a model, the result can be *state explosion*; a complex model with a large number of states and actions, making reasoning in the model inconvenient and perhaps even impossible. This can also be said about schemes in which the model consists of possible states and events in an actual investigation. When the number of states and events that has to be represented in a model becomes large, reasoning becomes difficult due to all the possibilities that have to be considered.

The implications of state explosion can be observed in Chapter 5 and 6, where invariants were found in the simple file system and the simple file system with creation by reasoning with Simplified Event Calculus. Where the computation for the simple file system in Section 5.6, had to test hypotheses by resolution in three different orders of two

timestamps, the computation in Example 6.11 in Section 6.3 had to consider thirteen different orders of three timestamps. In the SEC-algorithm, resolutions would have to be tested for every hypothesis derived from the initiation proposition for each timestamping order, yielding considerable work with resolution testing. Further, the number of hypotheses derived from the initiation proposition grew considerably, as each new Action added to E resulted in many new possible hypotheses for the initiation of each observed fluent.

The growth of the number of computations required to find invariants with the methods proposed in Chapter 5 and 6 was quantified in sections 5.11, 6.5 and 6.6. The quantification shows that the algorithm proposed in Chapter 6 is considerably more efficient than the algorithm shown in Chapter 5 with growing number of actions and timestamps in a system. The algorithm in Chapter 6 grows linearly with the number of actions in a system and quadratically with the number of timestamps in a system. It has been shown that it is practically possible to use all proposed algorithms for systems where a small number of timestamps is associated with each file. In Chapter 5 and 6, the necessary computations for $n = 2$ and $n = 3$ was performed manually with the SEC-algorithm and the AS-algorithm. With a software implementation, it would be possible to make these computations also with a larger n, although the factorial growth would render the procedure impossible as n increases further. With a software implementation of the IG-algorithm, invariants can also be found programmatically for larger n.

It should be noted that the factorial growth of the SEC-algorithm and the AS-algorithm does not rule out the analysis of systems with many timestamps for each file with these algorithms. Consider for example a system with 32 timestamps per file (n = 32). The number of computation steps would be on the order of 10^{35} should this system be analyzed in full with these algorithms. This would surely be infeasible. But there is no requirement for the investigator to analyze a system completely. The purpose of the analysis is just to find invariants that can be used to test a clock hypothesis. Such invariants can be found also with partial analysis of the system. The investigator may for example divide the system into 8 separate subsystems, each with 4 timestamps. He can then analyze each of the subsystems separately and find invariants for them. These invariants can then be used to test the clock hypothesis. Should the investigator aim at finding *all* invariants he would have to analyze the complete system, something that would be infeasible with these algorithms. But finding all invariants is not necessary; it is

enough to find some invariants to impose constraints on the clock hypothesis. The number of invariants found will increase as the investigator puts more effort into finding them.

The large number of timestamps present in most real cases could also be seen as a state explosion problem. This would most certainly be true if the scheme requires every timestamp present in the system to be modelled as a state or fluent, that can possibly be changed by events. Such modelling is not required in the schemes presented in this work. Instead, this work provides methods for finding relations between timestamps that must hold true for all timestamp instances. This allows the investigator to test his clock hypothesis by testing each observed timestamp for consistency with the prediction of the model. The observations of the investigator and his hypotheses about the user's actions are not a part of the model. The model only describes the system behaviour with respect to updating timestamps. The observed timestamps are then tested for consistency with the system model and the clock hypothesis. No state explosion is introduced, even if large numbers of timestamps must be tested for inconsistency. Timestamps are tested with the invariants derived from the model, an operation where the complexity is proportional to the number of timestamps. As shown in Chapter 7, such testing can be performed in software, without problems specifically related to complexity. In evidential terms, a large number of timestamps to be tested will increase the confidence in the clock hypothesis. A large number of timestamps is therefore an evidential asset to the investigator, and should not be seen as a burden increasing complexity.

8.2 Completeness of the model

In Chapter 3-7, theory and methods for clock hypothesis consistency testing were developed. The methods are based on creating a model of a system, in which changes to timestamps stored on the digital medium are modelled with *actions*. The system model is constructed by analyzing the behaviour of a system or by inferring the possible actions in a system by detailed analysis of the implementation. Any change possible in a real system that can affect the state of the digital medium must be matched by an action in the model, in order for the model to be complete. This raises an important question: Is it possible to be sure that all actions affecting the state of the medium are included in the model? If one cannot be sure that all actions are included in the model, how can one be

sure that the cause of an inconsistency is an incorrect clock hypothesis and not the existence of some action in the real system that was not included in the model? Say for example that there existed an action *TouchCreate* in Windows XP, updating the Created timestamp, but not the Modified or Accessed timestamps for a specific file. The introduction of such an action in the reasoning in Section 6.4 would change the reasoning and invalidate the result that $t_c \leq t_a$. How can one be sure that such actions do not exist? In fact, one cannot. In the Windows XP operating system specifically, system calls for timestamp updates exist that can be called by any program. (See Appendix A.1 for details.) In any system, even if the full details of the system is known, one cannot rule out the possibility that the state has been changed by some action occurring outside that system. One cannot for example completely rule out the possibility that the storage medium has been removed from its system and accessed and changed in another system. Changes committed outside the modelled system are not included in the system model, so they may produce inconsistencies when testing with a clock hypothesis, even if that hypothesis was in fact a correct representation of the actual clock on the system. Determining with absolute certainty if such inconsistencies is the result of an inconsistent clock hypothesis or the result of missing actions is not possible. At first sight, the above reasoning may seem to imply that the methods presented in this work produces ambiguous results and therefore should not be used in investigations. After all, the purpose of an investigation is to produce evidence a fact finder can rely upon in his reconstruction of the events that took place. There are however several reasons not to draw such a conclusion.

First, in most cases, an investigated digital medium will contain not only timestamps and usage data, but also the programs that have been used to manipulate the data. If the system contains some special program updating timestamps in a specific way, then the investigator can find it during the digital investigation of the system. If for example a Windows XP system contains the *TouchCreate* program mentioned above, the investigator would likely find it during the investigation. When found, the investigator can analyze its properties by detailed analysis of its construction, or by running the program on a separate system to determine its effects on timestamps. The model can then be updated with actions corresponding to the program's effects.

Further, the number of theories in an investigation is usually limited. Consider an investigation where contraband physical objects are found in the home of a suspect. In

principle, there are a large number of possible theories of how those objects ended up there. But since it's the suspect's home, investigators are likely to conclude that the suspect placed them there. Now, if the suspect has other theories, he can present them to the investigators or as part of his defence. The investigators will then investigate if there is any evidence available to support or refute the alternative theories and if so adjust their working theory accordingly. Ultimately, it is up to the fact finder to assess the evidence supporting the alternative theories and decide which of them likely caused the objects to be located in the suspect's home. A large amount of other possible explanations still exist, but these are usually not considered. After all, the objects have been within the domain of the suspect, and one can reasonably expect him to be able to explain how they got there. Since the suspect has offered his explanation, this is the only considered theory in addition to the investigator's theory.

If the number of possible explanations can be limited in an investigation on physical objects, it can in a digital investigation too. Like a home, a computer is likely to be within the domain of one specific person or a small group of persons. It is reasonable to expect computer users to be able to explain what kinds of actions have been taken on the computer, at least at the level visible to the computer user. If they cannot, then at least they can present their own theory about what happened, and the investigator can search for evidence of that theory as well as his own. It will then be up to the fact finder to assess the evidence for the different theories.

One popular alternative theory presented by suspects seems to be the so called Trojan horse defence, the theory that contraband was stored by someone performing computer intrusion on the computer. [40] The analogy in the physical world is that someone broke in to the house and placed the contraband there.

Limiting possible theories also applies in investigations involving clock hypothesis testing. In such cases, it is reasonable to ask the persons having the computer in their domain to explain what actions have been committed on the computer, including changes on the system clock and which programs have been run on it. Their clock hypothesis and action model can then be tested against the available evidence using the tests described in this work. If the test produces inconsistencies, the investigator must find another hypothesis and test it against the evidence.

As an example, consider the actions of Subject 3 in the antedating experiment. If the motivation of these actions was to antedate a document, it is likely that the subject when interviewed would assert the default clock hypothesis. As shown in Section 7.6.3, the default hypothesis is inconsistent when tested against the available evidence. A possible next step for the investigator would then be to hypothesize the antedating of the document by reverse adjustment of the clock by the suspect. This hypothesis, when tested, would show to be consistent with the evidence. Thus, in this case, the fact finder would have two conflicting theories to choose from, the suspect's version which was refuted by the evidence, and the investigator's version which was upheld.

8.3 Correctness of the model

The reasoning on the behaviour of a system in this work is based on constructing a model of the system. A problem with this approach is the feasibility of detailed analysis of real systems. Modern computer systems are complex, and contain many individual pieces of hardware and software that fits together. The study of individual parts of the system may be feasible, but may not necessarily provide sufficient information to construct a correct model of the system. Moreover, in many cases implementation specifications and source code is not available for the investigator. Studying a system implementation in detail without access to specifications and source code can be a very difficult task. In the case where no system documentation is available, a possible approach is to test system functions in various test cases. It is however difficult to know if the test cases cover all functionality included in the system. Since the internal working of the system is unknown, there could be input data not covered by the test cases that would produce a different behaviour in the system. This behaviour would then remain unknown, and would not be included in the system model. In the context of the time stamp reasoning discussed in this work, several system behaviour types are of special interest:

- Determining which events update which timestamps
- Determining which events are causally related, and which are not
- Determining how events updating timestamps correspond with user actions

The difficulty of establishing a correct model of a system is levied by the fact the most investigated systems are standard systems deployed over a long range of computers, and

therefore subject to many investigations. For example, the properties of the Windows XP operating system and the NTFS file system, as discussed in Chapter 6-7, is very interesting since this currently is the most common operating system, and therefore the subject of a long range of investigations. If it can be justified that the function of the operating system itself has not been altered in the case in question, a model of a specific system can be constructed once, and then reused in a long range of investigations. In each investigation, the model can be re-examined by the investigator and any experts appointed by the opposing party. If shortcomings are detected during these examinations, the model can be updated, and the updates can be reused in subsequent investigations. In a scientific context, such a process can be said to be equivalent to the formulation of a hypothesis and its acceptance in the scientific community. It is also possible for system designers and vendors to contribute in this process, without necessarily having to reveal internal system details. The need for correct system models therefore call for the establishment of forums for the exchange of system models between practioneers within digital investigation. The establishment of system models would facilitate event reconstruction for digital systems in general, not only within the subject of digital timestamps. With such a forum, investigators, prosecutors, defence attorneys as well as fact finders, can benefit from the improved understanding of how systems work.

8.4 The Arms Race Argument

The ongoing research within tools and methods for digital investigation has resulted in efforts in researching tools and methods for "anti-forensics". *Anti-Forensics* can be defined as the research into and development of tools to compromise the availability or usefulness of evidence to the forensic process. [41] This can be accomplished in a variety of ways ranging from overwriting deleted data to prevent them from being discovered, through usage of strong encryption to obscure contraband data to the introduction of directory loops in file system that causes forensic software to deadlock or crash when reading file system structures. [42]

The parallel development of forensic and anti-forensic tools can be thought of as an arms race where development in one field stimulates development in the other. The development of techniques for forensic recovery of deleted information for example led to the development of the tool EvidenceEliminator [43], which removes temporary files and

overwrites areas of hard drives containing deleted material with random data. This has in turn led to the development of forensic techniques, by which specific patterns resulting from the use of EvidenceEliminator are recognized, so the investigator can prove that this particular anti-forensic tool was run. [44] Conversely, the existence of steganography techniques for hiding files within other files has inspired the development of forensic techniques that can examine files for superfluous data as well as tools for determining signatures produced by specific steganography tools. [45] A common argument is derived from extrapolating the existence of forensic - anti-forensic technique pairs into a belief that any forensic technique will ultimately be matched by an anti-forensic technique that will block evidence extraction performed with the forensic technique. This argument can be called the *Arms Race Argument*. Supporters of this argument believe that any new developments within forensics can only be effective for a limited amount of time, since ultimately an anti-forensic technique will prevent the forensic technique from being used effectively.

In this work, new methods for investigation of timestamps have been proposed, providing ways to formulate and test hypotheses about clocks, and thereby increase the possibilities for interpretation of timestamps in an investigated system. Would these methods withstand a purposeful attack by an anti-forensics tool? They would not. It is easy to conceive an anti-TimeStampLogic tool that will thwart the efforts of the TimeStampLogic forensic tool described in Section 7. All this tool would have to do is to implement the same model as TimeStampLogic, and then go through all files and directories, sequence numbers and other causal properties and change the timestamps so they all match a predefined clock, for example civil time. If such a tool had been run on the PC after the antedating experiments described in Section 7, a subsequent investigation using TimeStampLogic or other tools based on the same principles would justify the wrong clock hypothesis, namely a hypothesis being equal to the clock of the anti-forensic tool and not the actual clock used on the computer. Thus, the purpose of running TimeStampLogic would not be met. The run of the anti-forensics tool can be seen as a change of the system not included in the model as discussed above, so the missing ability to detect a run of an anti-forensics tool can be seen as a sub-problem of the missing ability to distinguish between inconsistent clock hypotheses and detection of actions not included in the model.

There are several fallacies with the arms race argument that reduces its weight as an argument against the effectiveness of forensic techniques. First, in most cases, it is possible to determine if an anti-forensics program has been used. It is difficult to make an anti-forensics program in such a way that it does not leave any pattern specific to that program. An anti-TimeStampLogic program would for example be very likely to produce specific patterns between timestamps it would have to change in order to perform its function. These patterns could be detected by an anti-anti-TimeStampLogic program. Although hypotheses about the original clock could then no longer be tested properly, the evidence of usage of anti-TimeStampLogic would create an impression that there was something to hide. This reduces the desirability of using an anti-forensics tool significantly.

The most serious fallacy in the arms race argument is however the underlying assumption that anti-forensic techniques will always be available and will be used by everyone possessing potential sources of digital evidence. Consider the adversaries in a digital investigation, the *Investigator* and the *Perpetrator*. The Investigator usually possesses knowledge of digital investigation and tools that can comb a digital medium for evidence, including tools for digital imaging and data recovery. The Perpetrator is on the other hand likely to be an average computer user, and does not know how to protect himself from the scrutiny of a digital investigation or where he can obtain the necessary tools. The Investigator also has time on his side. Once a digital medium has been forensically imaged, he has plenty of time to investigate its contents. The Perpetrator on the other hand never knows when the Investigator will turn up to seize his data, if ever. He therefore has to be prepared at all times and run the anti-forensic tools again and again after every action that would leave incriminating evidence. There is no room for mistakes by the Perpetrator. If he makes a small mistake in his anti-forensic procedures, the evidence may be there waiting to be discovered by the Investigator. The Investigator on the other hand can make a lot of mistakes, as long as he doesn't mess up the original data. He can always start from a fresh image at a later time, should he feel that there is more to find or that current results rely on misinterpretations. All in all, the Investigator has a tremendous advantage over the Perpetrator in digital investigations. This is true in other types of investigations too, hence the saying *"There is no such thing as a perfect crime"*.

The above is not to say that the arms race argument is without value. The assumption of the arms race argument that tools and opportunity to use them exist is still valid on the Investigator's side. The Perpetrator is not likely to have the tools or knowledge required to counter forensic techniques. The Investigator on the other hand, *is* likely to have the tools and knowledge required to counter the use of anti-forensics techniques. This reduces the effectiveness of popular anti-forensics tools such as steganography and strong encryption in hiding evidence from the Investigator. Thus, the arms race argument has weight, but for the most part in favour of the Investigator only.

8.5 Falsified evidence creation

Aside from using an anti-forensics tool as mentioned above, a Perpetrator wishing to deceive the Investigator has another option: creating falsified evidence. In the context of digital investigations, creating falsified evidence involves preparing a digital medium to contain information matching the story the Perpetrator wants to present. This can be accomplished for example by antedating documents or other files such as contraband images. Another example could be the wilful installation of a Trojan horse by the Perpetrator on his own computer, in order to be able to claim the Trojan horse defence. These methods of creating falsified evidence could be exposed by the timestamp investigation methods described in this work. Investigation of the activities on the computer at the time of the Trojan horse installation can together with timestamp investigation methods establish evidence that the Perpetrator installed the Trojan horse himself in an effort to create false evidence.

But what if the Perpetrator also takes timestamps into account when creating falsified evidence? The complete contents of the digital medium could for example be built up from the bottom by gradually installing content, and by using a series of forward adjustments to the system clock. As long as the clock is only adjusted forward, no timestamps inconsistent with the default clock hypothesis will be introduced. When clock adjustments are not recorded on the medium, it is not possible from the Investigator's point of view to distinguish between reboots and clock adjustments in succession, and actual usage where the computer has been unused for long periods of time. By using such a method, a Perpetrator would be able to create false evidence matching his own timeline. The methods presented in this work would not be able to determine that the

content was actually produced at a different time. In fact, the method just described is the exact method used to prepare the computer used in the antedating experiment described in Chapter 7. This method was also used by one of the parties in the helicopter contract investigation described in Section 2.3. In that investigation, timestamps alone did not reveal the true history of the computer.

The burden on those wishing to create the perfect falsified evidence is however heavy. As discussed above, the advantage in an arms race between the Investigator and the Perpetrator is on the side of the Investigator. Should the Perpetrator wish to create falsified evidence, he must do so with outmost care, to prevent the introduction of inconsistencies. As shown in the results of the document antedating experiment, it is very difficult to manipulate the clock and timestamps of a computer in such a way that no inconsistencies are introduced. The best bet of the Perpetrator falsifying evidence is probably to build the complete evidence up from the bottom by wiping the complete hard drive and continue with forward clock adjustments only. But then the perpetrator is faced with another challenge: How to introduce user data in such a way that the result looks like real computer usage? There is a need to introduce real user data; otherwise the evidence will look like something that has been manufactured for the purpose. If the Perpetrator wants to copy real user data from another device, outmost care must be taken to prevent the introduction of timestamp inconsistencies. When copying documents from a computer to another any timestamps retained from the source medium (Such as the Modified timestamp in Windows XP), must be changed to match the new timeline. Any time evidence existing within the actual data must also be changed. This is where the perpetrator in the homicide investigation discussed in Section 2.2 missed. By reconstructing the real time from the length of shadows shown in the picture, investigators could show that there was a mismatch between the time shown on the video and the real time. Great care must also be taken to make sure that the new timeline matches evidence that may be available from sources outside the Perpetrator's control. This is where the falsifier in the contract investigation missed; he used a version of the wiping program that wasn't available at the time he was supposed to have used it. A falsifier of evidence needs to make sure that every detail in the falsification is consistent. Thus, creating believable falsified evidence turns out to be quite difficult - even when using the procedure of clock forward only adjustments.

Another problem facing the creator of false evidence is analogous to the problem of using anti-forensic tools: he needs to know when it should be done. In most situations, the Perpetrator cannot know for sure when the digital medium will be investigated. But he has to select a time for the production of false evidence. If he does it too early, he runs a greater risk of introducing inconsistencies in the evidence after falsification. If he does it too late, he risks that the Investigator moves first, and seizes the evidence before the falsification has been produced. In most real cases, the Perpetrator has no reason to believe that he is being investigated until the Investigator actually turns up to seize the evidence. This is reflected in experiences from real digital investigations. Cases where suspects have used disk wipe utilities and other anti-forensic tools to remove or alter digital evidence prior to seizure are few and far between, although they do exist. Accounts of cases where suspects have tried to remove evidence when the Investigator came to seize it are however numerous. In many of these cases, suspects have probably previously thought that they should remove the evidence, but postponed it because of the impracticalities involved.

9 SUMMARY

9.1 Accomplishments

This work has shown that it is possible to enhance the understanding and use of digital timestamps as evidence, by formulating hypotheses about clocks and testing them for consistency with the observed timestamps. The consistency tests check evidence by utilizing known properties of a system, such as known causality between events generating timestamps. Testing clock hypotheses can show that a hypothesis is consistent. It cannot prove that a hypothesis is correct, but inconsistent hypotheses will be refuted. For any consistent hypothesis, the tested timestamp evidence is evidence supporting the hypothesis. With many tested timestamps, the clock hypothesis can be justified as the clock to be used in the event reconstruction. This is a significant aid to event reconstruction, as timestamps pertaining to specific events of importance in the investigation can then be interpreted according to the justified clock hypothesis. Previously, there was no system for clock hypothesis formulation and testing. Inferences from timestamps were drawn with ad-hoc methods, often by assuming that the clock of the investigated computer had not changed during the computer history. This could lead to incorrect event reconstruction, or it could create doubt about the reconstructed event line in cases where the assumption was found to be unjustified.

It was shown in this work how clock hypotheses can be tested against timestamps from events known to be causally related. Under the assumption that events cannot causally affect events occurring earlier in time, causally related events must be ordered in time. This allows for the testing of a clock hypothesis against timestamps generated by events known to be causally related, by testing each pair of timestamps from causally related events. This concept was utilized to study properties of storage systems such as log files and file systems. Causal relations occurring in these systems were identified, allowing the testing of a clock hypothesis against the timestamps in these systems. Such clock hypothesis testing was implemented and evaluated.

Further, it was shown how a model of the part of a system affecting timestamps can be created in predicate logic, and how such a model can be developed to derive invariants

concerning the relationship between timestamps. These invariants can then be used for clock hypothesis testing in a similar fashion to the use of causal relations. It was also shown how such a model can be simplified to cater for more timestamps and events, thus making it possible to create a model of a real file system, and derive invariants for it. The derivation of invariants for clock hypothesis testing of a real file system was performed, and the hypothesis tester was implemented and evaluated.

9.2 Implications

This work describes methods that can be utilized by investigators to enhance the confidence in digital timestamps as evidence. The most important implication is the ability to produce more accurate event reconstructions from the analysis of digital evidence containing timestamps. The formulation and testing of clock theories is expected to be conducted by investigators during the investigation phase, and presented to fact finders as part of the presentation of the evidence in the case. During the investigation the investigator tests different clock hypotheses, perhaps conflicting hypotheses presented by the different parties in a case. He can present to the fact finder how these hypotheses match with the available evidence in form of timestamps matching or not matching with the clock hypothesis. It is then up to the fact finder to decide which of the hypotheses he finds is the most justified one.

The testing of a clock hypothesis against timestamps found in the evidence is a task involving the matching of a large number of timestamps against each other. This task is not expected to be done manually by investigators. A better solution would be to implement the process of clock hypothesis testing as part of tools for digital investigation. The implementation of `TimeStampLogic` in this work provides an example of such an implementation. This implementation is only meant as proof of concept. A real implementation should allow for easy definition of a clock hypothesis, perhaps by using a graphical tool. It should also be easy for investigators to define the semantics of the underlying system, in terms of causal relations and invariants. Investigators can then use an agreed set of causal relations and invariants for the specific system under investigation, as well as changing the semantics in response to the specifics of the investigation, such as discovered programs that change timestamps in a non-standard way. With such a tool, the formulation and testing of clock hypotheses can be performed

in a large number of cases, without increasing the workload on the investigator significantly.

The antedating experiment conducted in this work, has shown that the method of clock hypothesis formulation and testing can expose antedating. By testing the available timestamp evidence with the causality and invariants of the timestamps in the file system, the default clock hypothesis was found to be refuted, because the system clock had been adjusted back to a previous date to produce an antedated document. Evidence in the form of timestamps on the date the document was antedated to was found not only on the document, but also on other files being affected by the operating system. This demonstrated that the methods presented in this work can be used to find instances of clock manipulation, for example in connection with antedating. The presented methods are not provide investigation methods that cannot be circumvented by someone who would like to create evidence of an event history differing from the real. The construction of evidence supporting alternative event histories is however more difficult than without these methods, as demonstrated in the antedating experiment.

9.3 Future Directions

This final section discusses some possible directions that can be taken by future research within the area discussed in this work.

9.3.1 An implementation of invariant derivation

In Chapter 5 and 6, procedures for the derivation of invariants from system models were given. These procedures were employed manually for models of imaginary systems and for a real system. The work required for derivation of invariants with the devised methods increases with increasing number of timestamps per file in the modelled system. Finding invariants manually rapidly becomes impractical. For the analysis of larger and more complex systems, the derivation of invariants should therefore be implemented in a computer program. This can be accomplished by implementing the algorithms given in Chapter 5 and Chapter 6.

9.3.2 A database of system functionality

As discussed in Chapter 8, a significant challenge with any method attempting to apply reasoning based on the functionality of a digital system, is to establish an understanding of the functionality of the digital system. The functionality needs to be known; otherwise it will be impossible to build a model for reasoning that corresponds to the real system, and the inferences drawn from the model may not apply to the real system. In this work, system models have been built from descriptions of systems in previous works, as well as by inference from tests with real systems. It has not been attempted to prove or otherwise rigorously demonstrate that the system models corresponds exactly to the real systems. In real cases, there is however a need to show that there is a correspondence between the modelled system and the real system. The need to demonstrate such correspondence is not limited to timestamp investigations. In any investigation where an event history is reconstructed by observing system states and hypothesizing system events, it is necessary to show some correspondence between the modelled relationship between events and system states and how actions on the real system change stored data. This challenge needs to be addressed.

Carrier proposed the establishment of a complex event database, in which a description of which events caused by different programs would be stored. For example, the database could store lists of which files are read or written by a specific program. [4] Although it is probably inconceivable to establish a database listing all possible actions taken by all possible programs, the database could focus on specific types of events, for example if specific programs alter timestamps in ways differing from the standard operating system read and store methods. The database could also focus on the most common investigated systems, thereby providing data that could be used in a long range of investigations, without having to undertake the laborious work associated with determining functionality of less common systems. The existence of a database does not however itself provide the determination of the relationship between actions and their effect on real systems. In order to be able to list action-effect correspondence for a program in a database, it is necessary that someone actually determines this correspondence in the real system.

A problem with the determination of the effect of actions in a real system, also noted by Carrier, is that this determination is non-trivial. The source code and other system specification for the system may not be available, and it may be very difficult to

determine the system functionality without them. In practice, the investigator trying to determine the functionality of a system may have to resort to performing experiments testing user actions in the system and determining which effects they have on the data stored on the system. The outcome of such experiments will be a relationship between actions undertaken by the investigator and effects measured on a specific system. This relationship may however be different under other circumstances than those tested by the investigator, and due to the number of variables existing in complex digital systems it may not be clear exactly which circumstances held during the experiment. Thus, a specific experiment conducted by an investigator during a specific set of circumstances, may not yield results qualifying for the inclusion in the event database. The maintainer of an event database would need to establish requirements for the admissibility of experiment results in the database.

The situation described above is analogous to the situation in science. There is no source code or system specification describing the exact nature of the universe. Therefore, a researcher studying nature must resort to conducting experiments and study the effects of those experiments. But since the circumstances under which the experiments were taken may be difficult to repeat, scientific researchers understand that a hypothesis about nature cannot be accepted unless repeated experiments conducted by different researchers have fail to refute it. Indeed, hypotheses about nature are often debated for many years in scientific communities before they are generally accepted, or before someone devises an experiment refuting them.

Taking a scientific approach to digital investigation, the hypothesis based approach can also be used in the determination of the effect of actions in digital systems. If the system specifications cannot be studied in detail, then the results of experiments must be used to provide hypotheses about the system functionality. These hypotheses can be stored in the event database, detailing the nature of the experiments producing those results, and who conducted them. Any conflicting results can also be stored. Should the constructor of the system should be willing to participate; he can present his hypotheses about how the system works, based on experiments as well as studies of the system specification. He does not however have to reveal the system specification itself. After all, if the constructor's hypothesis about the system is based on reading the source code, then it is very unlikely that anyone else will find results conflicting with that hypothesis. With such a database, the investigator in a specific case can model the system according to the

prevailing hypothesis for the functionality of the system or according to several different hypotheses if there is more than one. If one of the parties involved in the investigation (for example the defence counsel), disagrees with the hypothesis for the system functionality, then he may present alternative hypotheses. The alternative hypotheses may then be tested with experiments, and the results can be used in the case in question, as well as being stored in the event database for future use. In this way, an event database detailing the functionality of digital systems can be built gradually by application of scientific principles.

9.3.3 Forensic friendly systems

It has been an underlying assumption in this work that the cooperation of the owner of the investigated system cannot be assumed. Specifically, it has been assumed that the owner of the computer cannot be assumed to be using digitally signed timestamps from external systems, or keep system clocks completely synchronized with universal clocks, in case the system should be investigated. This does not however imply a rejection of proposals to change existing systems in ways that would facilitate future investigations. As already mentioned, most of the investigated systems are standard systems. There are many things that could be done with these standard systems to enhance the possibilities for future investigations. Some proposals have already been made. For example, Buchholz proposed to bind pervasive labels to data in a system. [46] Barik et al proposed to extend existing file systems in such a way that historical timestamp values are stored in addition to the current. [8] These changes would enhance the possibilities of using the methods for timestamp investigation presented in this work. In the case of pervasive labels, the propagation of labels in a system would create many possibilities of causality inference not existing before, thereby facilitating clock hypothesis testing. In the case of storing historical clock values, these clock values would add to the timestamp evidence and therefore put the clock hypothesis under additional scrutiny. If the historical clock values were stored in an ordered fashion as proposed by Barik et al, the sequence could act as a generation marker, thus making causality inference possible and enabling reasoning similar to that explored in Section 4.2. Another possible change that would facilitate investigations is adding generation markers to storage systems, as discussed in Section 4.3.

There are however two considerations that should be made when considering extending systems with forensic friendly features. The first have already been mentioned; if specific features have been installed in a system just to allow it to be forensic friendly, then people who know about them are likely to turn them off in order to avoid the evidence production associated with them. The second is the fact that features added to a system specifically to be forensic friendly, are likely to create a controversy among users. Many users will feel that the addition of specific features to help investigators investigating them is a violation of their privacy rights. In fact, in some jurisdictions, adding such features would be seen as unjustified in relation to existing privacy regulations. [47] The controversy resulting from systems with specific forensic friendly features may increase the proportion of users who know about these features, and therefore can turn them off to avoid evidence production.

System designers do not primarily design systems so that they can be investigated. The primary focus of system designers is usually to make a robust system, create a good user experience and attract as many as possible to the system. Creating controversies with users does not fit with these goals. System designers will therefore probably be reluctant to add specific forensic friendly features to their systems, unless there are some other specific goals that can be addressed by doing so. The acceptance for forensic friendly features among users and system designers' motivation to introduce them is an interesting area of further study. If it were possible to motivate all system designers and users to accept systems built from the bottom up to be forensic friendly, then research within digital forensics could be focused on how to build systems recording all user activities in forensically sound ways. Until then, research into tools and methods for analyzing data not originally intended as evidence will still be required.

LIST OF REFERENCES

[1] S. Y. Willassen, "Forensics and the GSM mobile telephone system," *International Journal of Digital Evidence,* vol. 2:1, 2003.

[2] United States National Institute of Justice Technical Workgroup for Electronic Crime Scene Investigation, *Electronic Crime Scene Investigation: A Guide for First Responders*, 2001.

[3] B. Carrier and E. H. Spafford, "Getting Physical with the Digital Investigation Process," *International Journal of Digital Evidence,* vol. 2:2, 2003.

[4] B. Carrier, "A hypothesis-based approach to digital forensic investigations," Center for Education and Research in Information Assurance and Security, Purdue University Tech Report 2006-06, 2006.

[5] C. Boyd and P. Forster, "Time and date issues in forensic computing - a case study," *Digital Investigation,* vol. 2004:1, pp. 18-23, 2004.

[6] L. Lamport and P. M. Melliar-Smith, "Synchronizing Clocks in the Presence of Faults," *Journal of the ACM,* vol. 32:1, pp. 52-78, 1985.

[7] D. L. Mills, "Network Time Protocol (NTP)," in *RFC 958*, 1985.

[8] M. S. Barik, G. Gupta, S. Sinha, A. Mishra, and C. Mazumdar, "An efficient technique for enhancing forensic capabilities of Ext2 file system," *Digital Investigation,* vol. 2007:4S, pp. 55-61, 2007.

[9] C. Hosmer, "Proving the Intergrity of Digital Evidence with Time," *International Journal of Digital Evidence,* vol. 1:1, 2002.

[10] S. Y. Willassen and S. F. Mjolsnes, "Digital Forensic Research," *Telektronikk,* vol. 2005:1, 2005.

[11] B. Schatz, G. Mohay, and A. Clark, "A correlation method for establishing provenance of timestamps in digital evidence," *Digital Investigation,* vol. 2006:3S, pp. 98-107, 2006.

[12] F. Buchholz and B. Tjaden, "A brief study of time," *Digital Investigation,* vol. 2007:4S, pp. 31-42, 2007.

[13] M. C. Weil, "Dynamic Time & Date Stamp Analysis," *International Journal of Digital Evidence,* vol. 1:2, 2002.

[14] M. W. Stevens, "Unification of relative time frames for digital forensics," *Digital Investigation,* vol. 2004:1, pp. 225-239, 2004.

[15] F. Buchholz, "An Improved Clock Model for Translating Timestamps," James Madison University, Department of Computer Science JUM-INFOSEC-TR-2007-001, 2007.

[16] P. Gladyshev and A. Patel, "Formalising Event Time Bounding in Digital Investigations," *International Journal of Digital Evidence,* vol. 4:2, 2005.

[17] P. Gladyshev, "Formalising Event Reconstruction in Digital Investigation," 2004.

[18] P. Gladyshev and A. Patel, "Finite State Machine Approach to Digital Event Reconstruction," *Digital Investigation,* vol. 2004:2, pp. 130-149, 2004.

[19] P. Gladyshev, "Finite State Machine Analysis of a Blackmail Investigation," *International Journal of Digital Evidence,* vol. 4:1, 2005.

[20] P. Gladyshev and A. Enbacka, "Rigorous Development of Automated Inconsistency Checks for Digital Evidence Using the B Method," *International Journal of Digital Evidence,* vol. 6:2, 2007.

[21] Oslo Tingrett, "Okokrim Statsadvokatembeter mot Trond Gunnar Kristoffersen." Decision Ref. 03-006955MED-OTIR/05, 2004.

[22] Supreme Court of Pennsylvania, "Commonwealth of Pennsylvania v. Kevin Brian Dowling." Decision Ref. 255 CAP, 2005.

[23] R. Halperin, "Abolish Archives, October 1998," http://venus.soci.niu.edu/~archives/ABOLISH/oct98/0392.html Accessed: 31. Jan 2007

[24] J. Cohan, "Physicist key to unlocking case," in *Dickinson Magazine* Dickinson, Pennsylvania, 2002.

[25] Employment Appeal Tribunal, "Woodvard v. Abbey National, JP Garrett v. Cotton, Appeal against Registrar's Order." Decision Ref. UKEATPA/0534/05/SM, UKEATPA/0030/05/DZM, 2005.

[26] Employment Appeal Tribunal, "Midland Packaging Ltd v. Clark, Appeal From Registrar's Order." Decision Ref. UKEATPA/1146/04/RN, 2005.

[27] L. Lamport, "Time, Clocks and the Ordering of Events in a Distributed System," *Communications of the ACM,* vol. 21:7, pp. 558-565, 1978.

[28] C. Fidge, "Logical Time in Distributed Computing Systems," *Computer,* vol. 24:8, pp. 28-33, 1991.

[29] K. Popper, "Science: Conjectures and Refutations," in *Introductory Reading in the Philosopy of Science*, 3. ed: Prometheus Books, 1998.

[30] P. Wilson, M. Johnstone, M. Neely, and D. Boles, "Dynamic Storage Allocation: A Survey and Critical Review," in *International Workshop on Memory Management*, 1995.

[31] B. Carrier, *File system forensic analysis*. Upper Saddle River, N.J.: Addison-Wesley, 2005.

[32] P. Gutmann, "Secure Deletion of Data from Magnetic and Solid-State Memory," in *Sixth USENIX Security Symposium* San Jose, California, 1996.

[33] M. E. Russinovich and D. A. Solomon, *Microsoft Windows internals : Microsoft Windows Server 2003, Windows XP, and Windows 2000*, 4th ed. Redmond, Washington: Microsoft Press, 2005.

[34] M. Shanahan, *Solving the frame problem : a mathematical investigation of the common sense law of inertia*. Cambridge, Massachusetts: MIT Press, 1997.

[35] On-Line Encyclopedia of Integer Sequences, "Number of preferential arrangements of n labeled elements," AT&T http://www.research.att.com/~njas/sequences/A000670 Accessed: Nov 17 2007

[36] R. Bailey, "The number of weak orderings of a finite set," *Social Choice and Welfare,* vol. 15:4, pp. 559-562, 1998.

[37] B. Carrier, "Sleuthkit," Available at: www.sleuthkit.org.

[38] X-Ways Software, "WinHex," Available at: www.winhex.com.

[39] University of Delaware Police Computer Forensics Lab, "Time Change Captured in Event Log," http://128.175.24.251/forensics/timechange.htm Accessed: Oct 3. 2007

[40] D. Haagman and B. Ghavalas, "Trojan defence: A forensic view," *Digital Investigation,* vol. 2005:1, pp. 23-30, 2005.

[41] R. Harris, "Arriving at an anti-forensics consensus: Examining how to define and control the anti-forensics problem," in *Digital Forensics Research Workshop*, 2006.

[42] T. Newsham, C. Palmer, A. Stamos, and J. Burns, "Breaking Forensics Software: Weaknesses in Critical Evidence Collection," in *BlackHat*, 2007.

[43] Robin Hood Software Ltd., "Evidence Eliminator," Available at: http://www.evidence-eliminator.com.

[44] Radsoft.net, "The Evidence Eliminator Documents," http://www.radsoft.net/resources/software/reviews/ee/05.html Accessed: Nov 30 2007

[45] J. Fridrich, M. Goljan, D. Soukal, and T. Holotyak, "Forensic Steganalysis: Determining the Stego Key in Spatial Domain Steganography," in *SPIE Electronic Imaging*, San Jose, CA, 2005, pp. 631-642.

[46] F. Buchholz, "Pervasive Binding of Labels to System Processes," Purdue University 2005.

[47] M. A. Caloyannides, *Privacy Protection and Computer Forensics*, 2 ed.: Artech House, 2004.

[48] Microsoft, "FILETIME ": Platform SDK, Windows System Information.

[49] Microsoft, "SYSTEMTIME ": Platform SDK, Windows System Information.

[50] Microsoft, "DosDateTimeToFileTime," Platform SDK Windows System Documentation.

[51] G. Le Bodic, *Mobile messaging technologies and services : SMS, EMS, and MMS*. New York: J. Wiley, 2002.

[52] M. Friedman and O. Pentakalos, "File Cache Performance and Tuning," in *Windows 2000 Performance Guide*: O'Reilly, 2002.

APPENDIX

A. TIMESTAMP FORMATS

A.1 FILETIME

FILETIME is the timestamp format used to record timestamps associated with files stored on NTFS file systems. The FILETIME is a 64 bit value representing the number of 100-nanosecond intervals since January 1, 1601 00:00:00 UTC. The value is an unsigned 64-bit integer. The FILETIME format can represent

$$2^{64} \Big/ (10^7 \cdot 60 \cdot 60 \cdot 24 \cdot 365) \approx 58494 \text{ years}$$

The resolution is 0,1 microseconds. Applications may however not be able to set timestamps with a finer resolution than 1 millisecond.

Windows applications can get and set file time values using system calls GetFileTime() and SetFileTime(), supplying FILETIMEs for *CreationTime*, *LastAccessTime* and *LastWriteTime*. This operation will succeed for all file systems, but the result of the operation will depend on the file system. For example, on FAT file systems, the LastAccessTime has a resolution of 1 day, whereas on NTFS file systems it has a resolution of 1 second. [48]

A.2 SYSTEMTIME

SYSTEMTIME is a time representation used in Windows applications. It is a structure containing 16-bit values for (in this order) *Year, Month, DayOfWeek, Day, Hour, Minute, Second, MilliSecond.* Each member of the structure is a 16-bit unsigned integer. This structure can represent $2^{16} = 65536$, but *year* must be between 1601 and 30827 on current operating systems.

Windows applications can obtain the system time as a SYSTEMTIME with a call to GetSystemTime() or GetLocalTime(). The SYSTEMTIME must be converted to a FILETIME with SystemTimeToFileTime() if the value is to be stored as file timestamps. [49]

A.3 FAT date and time

In the FAT file system, timestamps are recorded as a 16-bit value for the date and a 16-bit value for the time. These are comprised of:

DATE

Bit	Meaning
0-4	Day of the Month (1-31)
5-8	Month (1-12)
9-15	Year offset from 1980 (0 = 1980)

TIME

Bit	Meaning
0-4	Second divided by 2
5-10	Minute (0-59)
11-15	Hour (0-23)

This gives the FAT timestamp a resolution of 2 seconds. The date and time are stored together for *CreationTime* and *LastWriteTime*. However, only the Date is stored for *LastAccessTime*, yielding a resolution of 1 day.

Windows NT and newer systems store an additional byte for the *CreationTime* timestamp in FAT adding finer resolution. This byte is an 8-bit unsigned integer and gives the number of 10 millisecond intervals since the change of the 2-second interval. [50]

A.4 UNIX time_t

Unix file systems express timestamps as POSIX time, in which time is expressed as the number of seconds that has elapsed since January 1, 1970 00:00:00 UTC (often referred to as the *Unix Epoch*). The `time_t` is an implementation of Posix Time representing the time as a signed integer. The `time_t` originally was a 32-bit signed integer, and still is on most implementations. Some implementations use a 64-bit signed integer instead and it is expected that most systems gradually will move to using 64-bit values.

The resolution of `time_t` is 1 second. Implemented as a 32-bit signed integer, the timestamp can represent values from December 13, 1901 20:45:52 to January 19, 2038 03:14:07. The end date is sometimes referred to as the *year 2038 problem*, referring to the fact that any system still using this time format on that date will cease to function properly as the counter wraps and revert to December 13, 1901. It is however expected that most systems will have switched to 64-bit `time_t` representation before 2038.

The unix `time_t` format with 32-bits signed integer representation is used in most unix applications and unix file systems.

A.5 Java time

In the Java programming language dates and moments in time can be represented by various classes, all of which are based on an internal representation of time where time is expressed as the number of milliseconds since January 1, 1970 00:00:00 UTC. This number is represented as a long - a 64 bit signed integer. This gives the time representation in Java a resolution of 1 millisecond.

The Java time format is able to represent

$$2^{64} \Big/ (10^3 \cdot 60 \cdot 60 \cdot 24 \cdot 365) \approx 584\ 942\ 417 \text{ years}$$

A.6 GSM / UMTS

The GSM / UMTS mobile phone system specifications specify timestamp formats for timestamp inclusion in text messages. In the absolute timestamp representation, a time value is encoded over 7 octets (bytes) in the following manner: [51]

Octet	Representing
1	Year
2	Month
3	Day
4	Hour
5	Minute
6	Second
7	Time Zone

Octet 1-6 are coded in BCD (Binary Coded Decimal) with the most significant digit in bit 0-3 and the least significant digit in bit 4-7. The Time Zone is encoded as the number of 15 minute intervals difference from UTC encoded as BCD with bit 4-7 as the least significant digit and bit 0-2 as the most significant. Bit 3 expresses the sign, where 0 is positive.

This gives the GSM timestamp a resolution of 1 second. Since there is no defined Epoch, it is not defined which century a timestamp pertains to. As a result, GSM timestamps are inherently ambiguous. In practical terms, the century can however usually be inferred from the context of the timestamp.

B. CLOCKS

B.1 Universal Time Coordinated (UTC)

In UTC, time is divided into days, hours, minutes and seconds. Days are identified using the Gregorian calendar. In UTC, the normal day has 24 hours, each consisting of 60 minutes, each consisting of 60 seconds. The normal day therefore consists of 86400 seconds. In addition, UTC defines irregular days in which the last minute has 59 or 61 seconds. These days have either 86399 or 86401 seconds. The purpose of these irregular days is to keep the year defined in the UTC and the Gregorian calendar in synchronization with the astronomical year. The usage of irregular days is commonly termed "inserting/removing a leap second", although formally the result is a longer or shorter day. Decisions on leap seconds are taken by the International Earth Rotation and Reference Systems Service (IERS), based on astronomical observations.

Coordination of the UTC is performed by Bureu International des Poids et Mesures in Paris. It imports a clock signal from more than 50 highly accurate clocks around the world. Each clock is weighted according to its accuracy, and the final clock is produced as the sum of these clocks. The result is used as the official time source in most countries in the world.

C. FILE SYSTEM TIMESTAMP EXPERIMENTS

During this work, experiments were performed on a computer running Microsoft Windows XP with the NTFS file system. These experiments were performed to understand the effects on file timestamps, and include them in a model. (See Chapter 6)

C.1 Timestamps in NTFS

NTFS (Originally Windows NT file system) is the standard file system in Windows NT, Windows 2000 and Windows XP. The file system is organized as a tree of directories under the root directory containing files. Metadata for files and directories are stored in the Master File Table (MFT), which is stored as a regular file. The MFT contains attributes for each file and directory, such as timestamps, extent tables (where the file is located on the disk) and file size.

The $STANDARD_INFORMATION attribute in each MFT-entry in NTFS contains four different timestamps, associated with the file or directory to which the MFT-entry pertains. These are the creation time, last modified time (also called "last written time"), last access time and MFT Modified time. Each timestamp is stored as a FILETIME record.

Informally, the four different timestamps can be described as follows:

Timestamp	Description
Creation time:	The time of the creation of the file on that file system
Last modified time:	The time of the last change of the file content
Last access time:	The time of the last access to the file content
MFT Modified time:	The time of the last change of file metadata

The $FILE_NAME attribute present in each MFT-entry, and also in the parent directory of each file or directory. These fields also contain the described set of four timestamp values. These values are not updated by Read or Write operations in the same way as those stored in the $STANDARD_INFORMATION attribute.

For further reading on NTFS timestamp locations and file system organisation, the reader is referred to Carriers work on the subject. [31]

C.2 Experiments

A set of test files and directories were generated. Test operations on files and directories were performed using standard operating system methods on a Windows XP computer. The system clock was observed at the time of each operation. The tool Win-Hex [38] was utilized to check the effect of operations on the $STANDARD_INFORMATION file system timestamps. Operations were performed both internally on one file system (intra) and from one file system to another (inter) on one computer. They were also repeated on another Windows XP computer mounting a file system across a network from the first test computer using the Windows standard file system sharing mechanism.

The tested operations are grouped into the categories read, write, delete, move and copy. The following operations were tested:

Read operations:
- Reading a file into the “notepad” application
- Typing a file on the command prompt with “type”
- Listing a directory with “dir” on command prompt
- Showing a directory by double-click on it in Windows Explorer
- Looking at a directory or file’s properties with right-click and select “properties”

Write operations:
- Creating a file in the “notepad” application
- Creating a file with the “echo” command on command prompt
- Changing an existing file with the “notepad” application
- Creating a new directory with “md” command on command prompt
- Creating a new directory with right-click, “new”, “new folder”
- Changing an existing directory by creating a new file in it

Delete operations:

- Deleting a file with right-click, “delete”: will put the file in the recycler
- Deleting a directory with right-click, “delete”: will put the directory in the recycler
- Emptying the recycler (a real delete operation)

Copy operations:
- Copying files and directories with “copy” command on command prompt
- Copying files/directories with right-click, “copy” and then right-click “paste”
- Copying files/directories with drag-and-drop over different file systems

Move operations:
- Moving files and directories with “move” command on command prompt
- Moving files/directories with right-click, “cut” and then right-click “paste”
- Moving files/directories with drag-and-drop on one file system

C.3 Results

In the following, the results from the tests of the different operations are presented and commented. Where the timestamp is unchanged as a result of the operation, a dash “-“ is shown. Where the timestamp is changed to the current time, a plus “+” is shown. Comments for each result can be found below. When referring to directory hierarchies in the comments, “top” means the directory that contains other directories whereas “bottom” means files and directories that is contained within other directories.

Read operations

Operation	Created	Modified	Accessed
Read file in notepad	-	-	+ CACHE
Read file with “type”	-	-	-
Read dir, windows explorer	-	-	+ CACHE (1)
Read dir, with “cd”, “dir”	-	-	+ CACHE (2)
View properties of a file	-	-	+ CACHE
View properties of a dir	-	-	+ CACHE (3)

During the tests, it was discovered that accessed timestamps are subject to Windows file caching. All versions of Windows with NTFS keep a file cache in memory to allow faster

loading of frequently used files. The file cache has a page size of maximum 256 kB. [52] As a result, files larger than 256 kB cannot be cached in full and must be read from disk. From the results, it was evident that where a read operation resulted in a cache hit and the file was read from cache in full, the last accessed timestamp on disk was not updated. On the other hand, if part of the file was read from disk, the timestamp was updated. In the tests, reading of files larger than 256 kB always updated the accessed timestamp. When reading smaller files the accessed timestamp was sometimes updated, sometimes not.

(1) When entering an absolute path in an explorer window or a file dialog box, Windows performs read-ahead of directories for filename completion, resulting in access to directories. Reading a directory with Windows Explorer may also result in updating of the accessed timestamps of the files contained in the directory. This happens if Explorer recognizes the file type and reads the file in order to display information to the user. For example; Explorer automatically reads Word Documents and MP3-files in order to show document name and author to the user while browsing files in Explorer.

(2) At command "cd", the accessed timestamps of all directories above the directory in question are updated. The accessed timestamp of the directory itself is only updated on a subsequent "dir".

(3) When viewing properties of a directory, Explorer calculates the total size of its content. This results in access to all of its subdirectories, updating accessed timestamps on these, but not to any of the files in the directory or any subdirectories.

Write operations

Operation	Created	Modified	Accessed
New file, Notepad	+	+	+
New file, "echo"	+	+	+
Modify file content, Notepad	-	+	+
New directory, windows	+	+	+
New directory, "md"	+	+	+
New file in dir (dir stamps)	-	+	+

Delete operations

Operation	Creation	Modified	Accessed
Delete file (windows menu)	-	-	+ (1)
Empty recycle bin (file)	-	-	-
Delete directory	-	-	+ (1) (2)
Empty recycle bin (dir)	-	+ (3)	+ (3)

Deleting a file/directory is actually a move operation where the file is moved to the recycler. The resulting timestamps are equal to the results of an intra file system move operation.

(2) Deleting a directory moves the whole tree of subdirectories to the recycler. This updates the accessed timestamp on all subdirectories, but not files. Access is top-down giving the top-directory the earliest access time.

(3) A recursive true delete (such as empty recycle bin) traverses the tree and start deleting files at the bottom. File timestamps are not updated, but directories are written when files are deleted thereby updating the directory modified timestamp. The result is a top-down sequential pattern on directory accessed timestamps and a bottom-up sequential pattern on directory modified timestamps.

Copy operations

File Operations	Created	Modified	Accessed
Copy, intra file system, source	-	-	+ CACHE
Copy, intra file system, dest	+	-	+
Copy, inter file system, source	-	-	+ CACHE
Copy, inter file system, dest	+	-	+
Drag-drop, inter file system, source	-	-	+ CACHE
Drag-drop, inter file system, dest	+	-	+
Copy-paste, intra, source	-	-	+ CACHE
Copy-paste, intra, dest	+	-	+
Copy-paste, inter, source	-	-	+ CACHE
Copy-paste, inter, dest	+	-	+

Directory Operations	Created	Modified	Accessed
Copy, intra file system, source	-	-	+ CACHE (1)
Copy, intra file system, dest	+	+ (2)	+
Copy, inter file system, source	-	-	+ CACHE (1)
Copy, inter file system, dest	+	+ (2)	+
Drag-drop, inter, source	-	-	+ CACHE (1)
Drag-drop, inter, dest	+	+ (2)	+
Copy-paste, intra, source	-	-	+ CACHE (1)
Copy-paste, intra, dest	+	+ (2)	+
Copy-paste, inter, source	-	-	+ CACHE (1)
Copy-paste, inter, dest	+	+ (2)	+

(1) A recursive copy operation updates the accessed timestamp on all directories and files in the source tree that is not present in the cache. The copy operation reads the directory tree before it starts reading files, (Explorer displays "Preparing to copy…"). This result in a pattern where all directories have accessed timestamps in sequence and all files have accessed timestamps in sequence. All accessed timestamps on directories are prior to the first accessed timestamp on a file. The operation start can be determined from the accessed timestamp of the top directory. The operation end can be determined from the latest accessed timestamp in the tree. In the tests, accessed timestamps on files below 256 kB were seldom updated on the source system during copy operations, whereas accessed timestamps on all files above 256 kB were updated, indicating that the caching mechanism is active here also. The result is a timestamp pattern in which all directories are accessed in rapid succession and then all files above 256k accessed in succession, somewhat less rapid, depending on file sizes and copy bandwidth.

(2) On the destination of recursive copy operations, all three timestamps are updated on directories, and creation and accessed timestamps updated on all files. The copy operation works in a top-down manner, creating directories, then creating subdirectories and files. For each directory in the tree the creation time indicates when the directory was created and modification and accessed timestamps indicate when the last subdirectory or file of that directory was written. The operation start can be determined from the creation time on the top directory. The operation end can be determined from the latest creation timestamp in the tree.

Move operations

File Operations	Creation	Modified	Accessed
Move, intra file system, source	N/A	N/A	N/A
move, intra file system, dest	-	-	+
Move, inter file system, source	-	-	+
Move, inter file system, dest	+	-	+
Drag-drop, intra file system, source	N/A	N/A	N/A
Drag-drop, intra file system, dest	-	-	+
Cut-paste, intra file system, source	N/A	N/A	N/A
Cut-paste, intra file system, dest	-	-	+
Cut-paste, inter file system, source	-	-	+
Cut-paste, inter file system, dest	-	-	+

Directory Operations	Creation	Modified	Accessed
Move, intra file system, source	N/A	N/A	N/A
Move, intra file system, dest	-	-	+ (1)
Move, inter file system, source	-	+ (2)	+ (2)
Move, inter file system, dest	+	+ (3)	+ (3)
Drag-drop, intra, source	N/A	N/A	N/A
Drag-drop, intra, dest	-	-	+ (1)
Cut-paste, intra source	N/A	N/A	N/A
Cut-paste, intra, dest	-	-	+ (1)
Cut-paste, inter, source	-	+ (2)	+ (2)
Cut-paste, inter, dest	+	+ (3)	+ (3)

(1) All directories below are also accessed but not files. Access is depth first, giving the top directory the latest access time.

(2) A recursive move operation across file systems had the same effect on the source system as a copy operation and then (true) delete. All files had the accessed timestamp updated (if it was not in cache, prefetch of files less then 256 kB applied here as well). All directories had both accessed and modification timestamps updated.

(3) On recursive move operations, directory timestamps on destination systems were updated as on copy operations. This applied to the top directory and all subsequent directories. Files contained in directories, however, only had the accessed timestamps updated, as opposed to a recursive copy operation where the creation timestamps were updated as well.

When interpreting move operations it is important to distinguish between intra- and inter- file system operations. When moving a file or directory to a location on the same file system, only the file entry is changed. When moving across a file system boundary, the effect on the directories are equal to copying and subsequently deleting. Files contained in directories on the destination system do not update the creation timestamp.

C.4 Disabling update of Last Accessed

It should be noted that it is possible to turn off updating of the accessed timestamp on a file system by running the command `fsutil behavior set disablelastaccess 1`. The status of this setting can be checked with `fsutil behavior query disablelastaccess`. Default setting is that updating of the accessed timestamp is on (disablelastaccess = 0). Setting disablelastaccess to 1 will disable the updating of the accessed timestamp on read operations. The accessed timestamp will still be updated on operations where the other timestamps are affected.

C.5 Verification of results

In order to be able to verify the above results by inspecting the file system driver, the author applied for access to the source code of the Windows XP NTFS file system driver through the Microsoft Shared Source Initiative. Such access was denied. Verification of the results must therefore be done with other methods, such as larger scale testing of actions in the operating system, or possibly by reverse engineering the file system driver, thereby determining its semantics. Verification with these methods has been out of the scope of this project.

D. TIMESTAMP LOGIC IMPLEMENTATION

The timestamp logic implementation is available for download at:

`http://www.willassen.no/phd_thesis/implementation/`

The Java source files for the timestamp logic implementation can be found in the file `TimeStampLogic.zip`. The archive file contains a complete project directory for use with NetBeans IDE 5.0 or newer. Alternatively the java sources can be compiled directly with Java compiler version 1.5 or higher.

The implementation uses a modified version of Sleuthkit 2.05 [37] for the parsing of file system images and extraction of file timestamps. Sleuthkit has been modified to extract specific timestamp data that the original version did not extract, log file sequence numbers, as well as printing the timestamps in absolute format so they can be read into the TimeStampLogic implementation without introducing errors from format printing and reparsing. The modified and original Sleuthkit 2.05 can be found at the location mentioned above in the file `sleuthkit-2.05-willassen-modified.tar.gz`. A diff file summarizing the modifications to sleuthkit can also be found in this package. Download, unzip and follow the building instructions in the folder `sleuthkit-2.05-willassen` to build.

The TimeStampLogic implementation must be run on images of NTFS file systems. Such an image may be produced by dumping the contents of an NTFS file system with the disk dump utility `dd`. It is also possible to run the timestamp logic implementation directly on the operating system's partition device (for example `/dev/hdc1`) on a partition with the NTFS file system. The images that were used for the experiments described in Chapter 7 are available at:
`http://www.willassen.no/phd_thesis/images/`

Authoring, building and testing of the timestamp logic implementation as well as the modified version of Sleuthkit 2.05 was performed on a workstation running Fedora Core 5 Linux on the i386 architecture. It has not been tested on other platforms, but should work on any platform supported by Sleuthkit 2.05 and Java 1.5.

E. PAPERS

This appendix contains copies of four papers presenting results from this work. The papers have been accepted for publication in January – March 2008.

1. **Hypothesis based Investigation of Digital Timestamps**
   presented at the 4th IFIP WG 11.9 Workshop on Digital Evidence in Kyoto, Japan, January 2008. To be printed in *Advances in Digital Forensics IV.*

2. **Timestamp Evidence Correlation by model based clock hypothesis testing**
   presented at ACM/ICST E-forensics 2008, Adelaide, Australia, January 2008. To be printed in conference proceedings.

3. **Finding Evidence of Antedating in Digital Investigations**
   presented at ARES 2008, Barcelona, Spain, March 2008. To be printed in conference proceedings.

4. **Using Simplified Event Calculus in Digital Investigation**
   Presented at ACM Symposium on Applied Computing, Fortaleza, Brazil, March 2008. To be printed in conference proceedings.

Chapter 1

HYPOTHESIS BASED INVESTIGATION OF DIGITAL TIMESTAMPS

Svein Yngvar Willassen

Abstract Timestamps stored on digital media play an important role in digital investigations. Unfortunately, timestamps may be manipulated, and also refer to a clock that can be erroneous, failing or maladjusted. This reduces the evidentiary value of timestamps. This paper takes the approach that historical adjustments to a clock can be hypothesized in a clock hypothesis. Clock hypotheses can then be tested for consistency with stored timestamps. A formalism for the definition and testing of a clock hypothesis is developed, and test methods for clock hypothesis consistency are demonstrated. With the number of timestamps found in typical digital investigations, the methods presented in this paper can justify clock hypotheses without having to rely on timestamps from external sources. This increases the evidentiary value of timestamps, even when the originating clock has been erroneous, failing or maladjusted.

Keywords: digital investigation, timestamps, causality, hypothesis based

1. Introduction

A timestamp is a recorded representation of a specific moment in time. In digital computing, a timestamp is a recorded representation of a specific moment in time in a digital format. This representation is either stored on a medium storing digital data, or transmitted on a network designed to convey digital data. Timestamps play an important role in digital investigations. Traditionally, they are used to place the event generating the timestamp at a specific moment in time, thereby facilitating event reconstruction. The identification that a certain event on a computer took place at a specific time makes it possible to correlate the event with events occurring outside the computer system. This can be events that occurred in another digital system, or in the physical world. A hard drive of a Windows system, investigated in typical digital

investigations, usually contains tens or even hundreds of thousands of timestamps.

1.1 Error and uncertainty in timestamps

For a number of reasons, stored timestamps may not accurately reflect the time of the generating event. A timestamp always depends on the adjustment of the clock from which it is generated. Since the timestamp is a function of the clock, it is always relative to the setting of the clock. Unfortunately, clocks are not fully reliable. Clocks may drift, thereby generating timestamps gradually more different from those generated from other clocks. Clocks may also fail, and produce completely incorrect timestamps. Further, clocks on most systems may be adjusted at any time by the user of the system to show a different date and time than civil time. The consequence is that a timestamp is relative not only to the clock it was generated from in general, but also to the particular adjustment of the clock at the time the timestamp was generated. Therefore, even timestamps generated from the same clock cannot be reliably compared unless it can be justified that the adjustment of the clock is unchanged between creations of timestamps. In order to reliably compare timestamps from different clocks, the difference between the clocks must be found, and it must be justified for all clocks that their adjustment has not changed.

The uncertainty associated with digitally stored timestamps implies that timestamps in general cannot be relied upon as evidence without justification of the factors that can lead to errors. In particular, it cannot be blindly assumed that timestamps are based on a clock that is adjusted to civil time. Further, it cannot be assumed that timestamps generated by different clocks are relative to the same clock. Not even when timestamps are based on the same clock, can one be absolutely certain that the time difference between the two events is equal to the difference between the timestamps. These uncertainties are worrying for investigators. If timestamps cannot be relied upon, then it is in many cases not possible to reconstruct the events in the case reliably.

1.2 Timestamps and causality

New methods are required for digital investigation of timestamps and use of digitally stored timestamps as evidence. This work takes the approach that time stamps and their evidence value can be tested in the hypothesis based investigation model suggested by Carrier. [2] In this model, the history of the medium under investigation is the complete set of configurations, states and events that has occurred during the lifetime

of the medium. The data direct observable by the investigator is the final state of the medium. This includes observations of all timestamps stored on the medium and the clock. These timestamps are all functions of the computer clock at some previous state in the history, and any subsequent events that affect them. This paper uses these properties to develop a formalism for clock hypothesis definition and tests that can be used to test it for consistency with observed evidence.

1.3 Related work

Being recognized as a research challenge, the problem of timestamp interpretation in digital investigation has been studied by a few researchers during recent years. Schatz et al demonstrated the problem of clock drift by observing clock synchronization on a network of computers in a small business. [3] Schatz suggests mitigating the problem by correlating the timestamps in web cache stored on the computer with records obtained from the web servers. Weil and Boyd et al suggest similar correlation methods, by using timestamps stored on the investigated computer coming from other clocks, such as timestamps in dynamically generated web pages. [1, 4] Such methods would provide correlation for the period for which cached data exist on the investigated computer only. These methods may be able to confirm or refute hypotheses about the clock in the period for which correlation material exists. They may not be able to provide reasonable evidence to refute a hypothesis that timestamps have been changed or the clock has been adjusted during the period for which no correlation material exist. Correlation with server records is only possible when such records actually exist, and the investigator has legal access to them.

Gladyshev studied the use of causality properties for establishing boundaries on period of time in which an event may have occurred. [5] In his approach, time bounding can be established when an event that occurred at an unknown or uncertain time is causally preceded and succeeded by events with known time occurrence. In order to perform time bounding, it is then required to know events of known time causally connected to the investigated events. When used to investigate a computer system, these events of known time must come from external sources. This approach differs from the approach taken in this work, where no time references from external sources is assumed. The concept of causality is used in this work as well as in Gladyshev's. Although the happened-before relation is defined differently, its use to correlate timestamps bears resemblance.

2. Hypothesis based timestamp investigation

2.1 Causality

Informally, causality is the relationship between cause and effect. This relationship can be expressed as a relation between events. In previous works, causality has been defined by means of the *happened-before* relation $\rightarrow$. The happened-before relation was first used by Lamport, who defined the relation by ordering events happening in a process and sending and receiving messages between processes. [6] This definition was generalized by Fidge to encompass process creation and termination as well as both synchronous and asynchronous message passing. [7]

For use in digital investigation, Gladyshev proposed an extended definition of happened-before. In Gladyshev's version it is defined that $e_1 \rightarrow e_2$ if e_2 uses the result of e_1 or e_1 precedes e_2 in the usual course of business of some organisation or during the normal operation of a machine. [5] In this definition, the meaning of happened-before is extended beyond computers. This extension is useful, since digital investigation requires the reconstruction of events, both within computers and outside them. Gladyshev's definition might however create doubt about exactly what happened-before means, since it is debatable what exactly constitutes the *normal operation* of a machine and the *usual course* of a business.

Definition. Let $\rightarrow$ be the *happened-before* relation. If $e_1 \rightarrow e_2$, then the occurrence of e_1 is necessary for e_2 to occur because e_2 depends on the effects of e_1.

Important examples of causality per this definition of the happened-before relation include:

- e_1 produces an item that is necessary input for e_2

  This is equivalent to Gladyshev's definition "e_2 uses the result of e_1". The definition of happened-before in terms of message sending and reception used by Lamport and Fidge also fall within this example.

- e_1 and e_2 are events in a computer program, where e_2 uses data produced by e_1.

  Since events in computer programs use items produced by other events in the same program, such as variables, data stored in memory, registers and stack pointers, many events occurring in computer programs will be related by happened-before. This is a special case of "e_1 produces an item that is necessary input for e_2". The definition of happened-before in terms of events occurring in a

process used by Lamport and Fidge falls within this example, with the exception of events that do not use the result of each other. This exception makes the definition suitable for modern computer systems, in which the execution order of a computer program can be rearranged by compilers and processors when the instructions do not depend on the results of each other.

2.2 Time and time values

In this work, time is considered to be a fundamental quantity. As a fundamental quantity, time is not itself defined in terms of other quantities, but it is measurable by means of comparison with periodic events, such as the periodic events occurring in clocks. Such periodic events may for example be the swinging of a pendulum (a pendulum clock), the movement of earth (a sundial) or microwave emission from certain materials (an atomic clock). We consider events to have a moment in time associated with them, and assume that these moments in time can be ordered in time by relations $<$ and $=$.

Definition. Let E be the domain of events. Let e be an event. Events are considered to be instantaneous. Let T be the domain of time. Let $t(e)$ be a function $E \mapsto T$, representing the moment in time at which event e occurred.

Further, we assume that *causality is preserved in time*. With the preservation of causality in time, we mean that no event can causally depend on an event occurring at the same time or a later time than itself. This can be expressed explicitly with the happened-before relation as:

$$t(e_i) \leq t(e_j) \Rightarrow e_j \not\rightarrow e_i \tag{1}$$

This assumption corresponds to the intuitive understanding of the relationship between causality and time. If such causal relations were allowed, then events in the future would be allowed to affect events in the past, something that has not been shown to occur in the real world.

For two events related by the happened-before relation, Equation (1) implies that:

$$e_i \rightarrow e_j \Rightarrow t(e_i) < t(e_j) \tag{2}$$

The above imposes an ordering in time on events ordered by the happened-before relation $\rightarrow$. It does not however imply any ordering in time for events not ordered by $\rightarrow$. Also, $t(e_i) < t(e_j)$ does not imply that $e_i \rightarrow e_j$. Events may happen at different moments in time without being related by $\rightarrow$. On the other hand, if two moments in time $t(e_1)$ and $t(e_2)$ are ordered such that $t(e_1) < t(e_2)$, events occurring at those

moments in time cannot be causally connected in reverse, such that the $e_2 \rightarrow e_1$.

2.3 Clocks

A clock is a device designed to give the owner an approximation of time that is sufficiently coherent so as to allow the owner to measure and compare time periods and sufficiently consistent with other clocks so as to allow the owner to perform actions concurrent with other clock owners without continuous coordination. Clocks are in other words designed to give an approximation of time. The definition of a clock should be able to reflect the possibility of clock drift and adjustment mentioned in Section 1.2.

Definition. Let V be the domain of time values produced by a clock. $c(t)$ is a clock function $T \mapsto V$

The definition of a clock function does not impose any restrictions on the clock values as a function of time. For example, even if $t_1 < t_2$ it may well be the case that $c(t_1) > c(t_2)$. And even if $t_1 < t_2 < t_3$, it may be the case that $c(t_1) = c(t_2) = c(t_3)$. The latter situation may for example occur if the events occurring at t_1, t_2, t_3 are so close together in time that the clock is unable to differentiate between them.

2.4 Timestamped events

A timestamped event is an event for which there exists a timestamp value in domain V. The timestamp value can be represented as a function on the event. Timestamps are created when an event makes a copy of the value provided by a clock. All timestamps in a set of timestamped events are not necessarily related to the same clock.

Definition. Let E be a set of timestamped events and V a domain of time values. $\tau_c(e)$ is a function $E \mapsto V$ such that $\tau_c(e_i) = c(t(e_i))$. $\tau_c(e_i)$ represents the timestamp associated with the event e_i relative to clock c.

In this definition, a timestamp is the value of the producing clock at the time of the event. The timestamp reflects the clock's representation of time at that particular moment. The definition of timestamps as a function of events and clocks provides a possibility to reason over timestamps and clocks.

2.5 Ideal and non-ideal clocks and their properties

It is useful to introduce the concept of *ideal clocks* and *non-ideal clocks*. An ideal clock is a clock which can only go forward.

Definition. Let I be the set of ideal clocks. An ideal clock $c(t) \in I$ is a clock which satisfies

$$\begin{aligned} \forall i \forall j (t(e_i) < t(e_j) &\Rightarrow c(t(e_i)) \leq c(t(e_j))) \\ \forall i \forall j (t(e_i) = t(e_j) &\Rightarrow c(t(e_i)) = c(t(e_j))) \end{aligned}$$

An ideal clock is a clock function on time which has the property that the value provided in the function from time is monotonically increasing. While having a monotonically increasing value, the values $c(t(e_i))$, $c(t(e_j))$ produced from two different moments in time $t(e_i)$ and $t(e_j)$ where $t(e_i) < t(e_j)$ may be equal. Many clocks represent moments in time as discrete values. In a discrete clock with limited resolution, two moments close in time will be represented by the same clock value.

Theorem 1. *For all ideal clocks $c \in I$, produced timestamps satisfies*

$$e_i \rightarrow e_j \Rightarrow \tau_c(e_i) \leq \tau_c(e_j)$$

Proof for the theorem is given in the Appendix.

The monotonic property of ideal clocks guarantee that two causally connected events timestamped by the same ideal clock have timestamps where the timestamp of the latter event is always equal or higher than the timestamp of the first.

2.6 Clock hypotheses

In order to be able to test if a certain theory about the clock holds, one must be able to formulate a hypothesis about the clock function. A hypothesis about the clock function is a possible theory about the clock function during the computer history. That hypothesis can then be tested against the set of observed timestamps. In the following, a clock hypothesis will be denoted $c_h(t)$.

Definition. A clock function $c(t)$ can be divided into two components, one function $b(t)$ which is an ideal clock and one function $d(t)$ representing the deviation from the ideal clock.

$$c(t) = b(t) + d(t)$$

In this scheme, the clock $(c(t))$ is divided into components: $b(t)$ is a base clock which must be an ideal clock. $d(t)$ is the difference between

the base clock and the investigated clock. By selecting a common base, two or more clocks can be compared by comparing the deviation only. It is sometimes useful to express the time of an event in terms of the base clock. This can be done by subtracting $d(t)$.

$$b(t) = c(t) - d(t) \tag{3}$$

2.7 Observed event sets and correctness

During a digital investigation of a computer, the investigator will observe a number of timestamped events that all come from the same clock. Some of these events will be causally connected. This set of observed timestamped events is called the *observation set*.

Definition. An observation set O, is a set of timestamped events, in which all timestamps are related to one clock $c_o(t)$.

In an observation set, there will typically be a large amount of timestamped events. The number of causal connections may also be large. The data in an observation set can be used to determine if a clock hypothesis holds or not.

Definition. A clock hypothesis $c_h(t)$ for an observation set O is *correct* if the value of $c_o(t)$ is equal to the value of $c_h(t)$ for all t.

$$\begin{gathered} c_o(t) = c_h(t) \\ \Downarrow \\ \forall e_i(\tau_{c_o}(e_i) = c_h(t(e_i))) \end{gathered}$$

If a clock hypothesis is correct, then all occurrences of timestamps must match the value predicted by the hypothesis. The correctness property can therefore be utilized to find techniques for testing if a clock hypothesis is correct or not.

Theorem 2. *In a correct clock hypothesis $c_h(t)$, the timestamps of all causally connected events $e_i \rightarrow e_j$ in an observation set O must be such that the timestamp of the first event minus the deviation from a common base has value less than or equal to the timestamp of the latter event minus the deviation from a common base.*

$$e_i \rightarrow e_j \Rightarrow \tau_{c_o}(e_i) - d_h(t(e_i)) \leq \tau_{c_o}(e_j) - d_h(t(e_j))$$

Proof for the theorem is given in the Appendix.

Conversely, if the property examined in Theorem 2 does not hold, then the hypothesis is not correct.

Theorem 3. (Test-A theorem). *If a pair of causally connected events $e_i \rightarrow e_j$ exist in an observation set O, for which the timestamp of e_i minus the hypothesis deviation from a common base has a higher value*

than the timestamp of e_j minus the hypothesis deviation from a common base, then the clock hypothesis is incorrect. This is called Test-A.

$$\exists e_i \exists e_j ((e_i \rightarrow e_j) \wedge (\tau_{c_o}(e_i) - d_h(t(e_i)) > \tau_{c_o}(e_j) - d_h(t(e_j)))) \Rightarrow c_o(t) \neq c_h(t)$$

Proof for the theorem is given in the Appendix.

Example 1. Consider the *default clock hypothesis*, where it is assumed that the clock of the investigated computer has always been equal to civil time, say UTC. Then $c_h(t) = b_h(t)$ and $d_h(t) = 0$. Now, let the observed set consist of timestamps for four events $e_1 - e_4$, where:

$$\begin{aligned} \tau_{c_o}(e_1) &= \text{Jan 12, 2003, 12:46:34} \\ \tau_{c_o}(e_2) &= \text{Apr 21, 2004, 10:22:38} \\ \tau_{c_o}(e_3) &= \text{Feb 9, 2003, 22:16:04} \\ \tau_{c_o}(e_4) &= \text{Dec 12, 2002, 02:46:32} \end{aligned}$$

And where $e_1 \rightarrow e_2$ and $e_3 \rightarrow e_4$. If we now apply Test-A for $i = 3$ and $j = 4$, we see that

$$(e_3 \rightarrow e_4) \wedge (\tau_{c_o}(e_3) > \tau_{c_o}(e_4))$$

And since $d_h(t) = 0$, the test fails. Thus, the default hypothesis is not correct for this observation set.

The result can be explained informally as follows: Since e_4 must have happened after e_3 and the timestamp of e_4 is at an earlier time than the timestamp of e_3, it cannot be the case that the clock has not been adjusted between these two events.

Theorem 4. (Test-B theorem). *In a clock hypothesis $c_h(t)$, for values c' of $c_h(t)$ for which $c_h(t) = c'$ has no solution, the existence of any timestamps in the observation set O with value $\tau_{c_o}(e_i) = c'$, implies that $c_h(t)$ is incorrect. This is called Test-B.*

Proof for the theorem is given in the Appendix.

2.8 Clock hypothesis consistency

The results in Theorem 3 and Theorem 4 are useful, because they can be used to refute a clock hypothesis for observation set O, from observations of the timestamps on events in O. In Test-A, a clock hypothesis is incorrect when observations of timestamps for two causally connected events are not ordered correctly by the clock hypothesis. In Test-B, a clock hypothesis is incorrect if observations of timestamps exist that

cannot be produced by the clock hypothesis, because it is a discontinuous function. These theorems provide methods for testing if a clock hypothesis is incorrect. By iterating over all events and pair of events, each timestamp can be checked for consistency with Test-A and Test-B.

The result of testing all timestamps in the observation set will be either that the clock hypothesis is incorrect or that it is not incorrect. The tests can refute the clock hypothesis, but they can not prove it correct. This leads to the following definition of a consistent clock hypothesis.

Definition. Given a set of tests Z, a clock hypothesis is *consistent under* Z with an observation set O if no test $z \in Z$ shows that the hypothesis is incorrect for O. A clock hypothesis is *inconsistent under* Z with an observation set O if it is not consistent under Z with O.

The distinction that follows from the definitions of correct and consistent is useful in the context of digital investigations. In a correct clock hypothesis all possible time values are always equal to the investigated clock. A correct clock hypothesis can only be derived if the investigated clock has been observed at every moment in its history. Establishing a correct hypothesis about the investigated clock is inconceivable in a real investigation. All the investigator can hope to do is to establish a consistent clock hypothesis. In such a hypothesis there is no evidence available that refutes the hypothesis. Specifically, none of the timestamps of events in the observation set O as applied in tests in the test set Z show that the hypothesis is incorrect. If there is a large number of timestamps and causally connected events present in the observation set O, these requirements impose strict constraints on a consistent hypothesis. This can lead to the justification of the hypothesis. The more data available in O to be fed into the tests in Z, the more justified the clock hypothesis can be. As long as the clock hypothesis is consistent, the data in O is evidence supporting the hypothesis.

2.9 The clock hypothesis as a scientific hypothesis

In the hypothesis based investigation model proposed by Carrier, a digital investigation is a process that formulates and tests hypotheses to answer questions about digital events or the state of digital data. [2] Carrier proposes that the investigation process is scientific if the hypothesis is scientific and then tested through conducting experiments. Carrier cites Popper in that the "criterion of the scientific status of a theory is its falsifiability, or refutability, or testability".

The question here is then if the method for clock hypothesis formulation and testing the set of observed timestamps adhere to these criteria.

From the previous discussion, a clock hypothesis is a theory that is falsifiable and therefore testable. The clock hypothesis theory described in the previous sections adheres to the requirements of a scientific theory. The hypothesis forbids certain things to happen; the occurrence of timestamp configurations as described in Test-A and Test-B. The described tests examine the evidence for refutation of the theory. They do not look for confirmation, but examine the available evidence for inconsistency with the theory. When the tests have been applied, and found not to refute the hypothesis, the tests count as serious but unsuccessful attempts to falsify the theory and therefore as confirming evidence.

3. Concluding remarks

This paper has presented a formalism for the definition of a clock hypothesis and testing it for consistency with evidence in the form of observed timestamps. When the number of timestamps is high, and many of them are causally related, these tests will put a clock hypothesis under close scrutiny. This is the typical situation when investigating digital media like hard drives. In order to test hypotheses on large number of stored timestamps, the tests can and should be implemented in software. The tests can then be used in digital investigations, typically by testing alternative clock hypotheses, such as alternative hypotheses provided by a plaintiff and a defendant. When a clock hypothesis is justified by these methods, the evidentiary value of the investigated timestamps is increased; the real time when a timestamp was created can now be found by using the clock hypothesis.

References

[1] C. Boyd and P. Forster, "Time and date issues in forensic computing - a case study," *Digital Investigation*, vol. 2004:1, pp. 18-23, 2004.

[2] B. Carrier, "A hypothesis-based approach to digital forensic investigations," Center for Education and Research in Information Assurance and Security, Purdue University Tech Report 2006-06, 2006.

[3] B. Schatz, G. Mohay, and A. Clark, "A correlation method for establishing provenance of timestamps in digital evidence," *Digital Investigation*, vol. 2006:3S, pp. 98-107, 2006.

[4] M. C. Weil, "Dynamic Time & Date Stamp Analysis," *International Journal of Digital Evidence*, vol. 1:2, 2002.

[5] P. Gladyshev and A. Patel, "Formalising Event Time Bounding in Digital Investigations," *International Journal of Digital Evidence*, vol. 4:2, 2005.

[6] L. Lamport, "Time, Clocks and the Ordering of Events in a Distributed System," *Communications of the ACM*, vol. 21:7, pp. 558-565, 1978.

[7] C. Fidge, "Logical Time in Distributed Computing Systems," *Computer*, vol. 24:8, pp. 28-33, 1991.

4. Proofs

Theorem 1.

Claim: *For all ideal clocks $c \in I$, produced timestamps satisfies*

$$e_i \rightarrow e_j \Rightarrow \tau_c(e_i) \leq \tau_c(e_j)$$

Proof: By definition an ideal clock satisfies:

$$\forall i \forall j (t(e_i) < t(e_j) \Rightarrow c(t(e_i)) \leq c(t(e_j)))$$

That is, for events e_i and e_j occurring at times $t(e_i)$ and $t(e_j)$ we have:

$$t(e_i) < t(e_j) \Leftrightarrow c(t(e_i)) \leq c(t(e_j))$$

By replacing we now obtain:

$$e_i \rightarrow e_j \Rightarrow c(t(e_i)) \leq c(t(e_j))$$

And then, $\tau_c(e_i) = c(t(e_i))$, which gives:

$$e_i \rightarrow e_j \Rightarrow \tau_c(e_i) \leq \tau_c(e_j)$$

□

Theorem 2.

Claim: *In a correct clock hypothesis $c_h(t)$, the timestamps of all causally connected events $e_i \rightarrow e_j$ in an observation set O must be such that the timestamp of the first event minus the deviation from a common base has value less than or equal to the timestamp of the latter event minus the deviation from a common base.*

$$e_i \rightarrow e_j \Rightarrow \tau_{c_o}(e_i) - d_h(t(e_i)) \leq \tau_{c_o}(e_j) - d_h(t(e_j))$$

Proof: Let $c_h(t)$ be a correct clock hypothesis. Let $b(t)$ be a common base for $c_h(t)$ and $c_o(t)$. Then

$$b(t) = c_h(t) - d_h(t)$$

$$b(t) = c_o(t) - d_o(t)$$

Thus,

$$c_h(t) - d_h(t) = c_o(t) - d_o(t)$$

And since $c_h(t)$ is correct we have $c_h(t) = c_o(t)$. Therefore

$$\begin{aligned} d_h(t) &= d_o(t) \\ b(t) &= c_o(t) - d_h(t) \end{aligned}$$

And inserting definition yields

$$b(t(e)) = \tau_{c_o}(e) - d_h(t(e))$$

Now, $b(t)$ shall be an ideal clock. From Theorem 1 we know that ideal clocks satisfy

$$e_i \rightarrow e_j \Rightarrow c(t(e_i)) \leq c(t(e_j))$$

And then, inserting $b(t)$ gives

$$\begin{aligned} e_i \rightarrow e_j &\Rightarrow b(t(e_i)) \leq b(t(e_j)) \\ e_i \rightarrow e_j &\Rightarrow \tau_{c_o}(e_i) - d_h(t(e_i)) \leq \tau_{c_o}(e_j) - d_h(t(e_j)) \end{aligned}$$

□

Theorem 3.

Claim: *If a pair of causally connected events $e_i \rightarrow e_j$ exist in an observation set O, for which the timestamp of e_i minus the hypothesis deviation from a common base has a higher value than the timestamp of ej minus the hypothesis deviation from a common base, then the clock hypothesis is incorrect.*

$$\exists e_i \exists e_j ((e_i \rightarrow e_j) \wedge (\tau_{c_o}(e_i) - d_h(t(e_i)) > \tau_{c_o}(e_j) - d_h(t(e_j)))) \Rightarrow c_o(t) \neq c_h(t)$$

Proof: The proof is by contradiction. Let $c_h(t)$ be a clock hypothesis and O an observation set with clock co(t). Let (e_a, e_b) be a pair of events in O such that $e_a \rightarrow e_b$ and $\tau_{c_o}(e_a) - d_h(t(e_a)) > \tau_{c_o}(e_b) - d_h(t(e_b))$. Assume that $c_h(t)$ is correct, $c_h(t) = c_o(t)$. If $c_h(t)$ is correct we have from Theorem 3 that

$$e_i \rightarrow e_j \Rightarrow \tau_{c_o}(e_i) - d_h(t(e_i)) \leq \tau_{c_o}(e_j) - d_h(t(e_j))$$

But for $i = a$ and $j = b$, we have assumed that,

$$(e_a \rightarrow e_b) \wedge (\tau_{c_o}(e_a) - d_h(t(e_a)) > \tau_{c_o}(e_b) - d_h(t(e_b))) \qquad (4)$$

This contradicts the result from Theorem 3. Therefore, if (4) holds, then $c_h(t)$ cannot be correct. There have been no assumption or restriction on the events a and b. a and b could therefore have been any event in the observation set O. The result is that for any event e_i and e_j, if (4) holds, $c_h(t)$ cannot be correct.

$$\exists e_i \exists e_j ((e_i \rightarrow e_j) \wedge (\tau_{c_o}(e_i) - d_h(t(e_i)) > \tau_{c_o}(e_j) - d_h(t(e_j)))) \Rightarrow c_o(t) \neq c_h(t)$$

□

Theorem 4.

Claim: *In a clock hypothesis $c_h(t)$, for values c' of $c_h(t)$ for which $c_h(t) = c'$ has no solution, the existence of any timestamps in the observation set O with value $\tau_{c_o}(e_i) = c'$, implies that $c_h(t)$ is incorrect.*

Proof: The proof is by contradiction. Let $c_h(t)$ be a clock hypothesis and O an observation set with clock $c_o(t)$. Let e_a be an event in O and $\tau_{c_o}(e_a) = c'$ the timestamp of e_a. Let c' have a value such that $c_h(t) = c'$ has no solution. Assume that $c_h(t)$ is correct, $c_h(t) = c_o(t)$. If $c_h(t)$ is correct we have

$$\forall e_i(\tau_{c_o}(e_i) = c_h(t(e_i)))$$

Which means that for $i = a$

$$\tau_{c_o}(e_a) = c_h(t(e_a))$$

This is a contradiction since $\tau_{c_o}(e_a) = c'$ and $c_h(t) = c'$ has no solution. Therefore if $\tau_{c_o}(e_a) = c'$ and $c_h(t) = c'$ has no solution, then $c_h(t)$ cannot be correct.

□

Timestamp evidence correlation by model based clock hypothesis testing

Svein Yngvar Willassen
Department of Telematics, Norwegian University of Science and Technology
O.S. Bragstads plass 2B
7491 Trondheim, Norway
+47 92449678
svein@willassen.no

ABSTRACT
Timestamps play an important role in digital investigations, since they are necessary for the correlation of evidence from different sources, including network tracing. Use of timestamps as evidence can be questionable due to the reference to a clock with unknown adjustment. This work addresses this problem by taking a hypothesis based approach to timestamp investigation. Historical clock values can be formulated as a clock hypothesis. This hypothesis can be tested for consistency with timestamp evidence by constructing a model of actions affecting timestamps in the investigated system. Acceptance of a clock hypothesis with timestamp evidence can justify the hypothesis, and thereby establish when events occurred in civil time. The results can be used to correlate timestamp evidence from different sources, including identifying correct originators during network trace.

Categories and Subject Descriptors
F.4.1 [**Mathematical Logic and formal languages**]: Mathematical Logic – *model theory, temporal logic*.

General Terms
Theory, Legal Aspects, Verification.

Keywords
Digital investigation, event logic, clock hypothesis

1. INTRODUCTION
Investigations are inquiries into past events. The purpose of an investigation is to find evidence of previous events. Investigation of digital media with the purpose of finding evidence is commonly referred to as *digital investigation*. In recent works, efforts have been made to make the digital investigation based on scientific principles, by using a hypothesis-based approach. [1] In this approach, the investigator formulates his hypothesis about the occurred events, and tests them using the available evidence.

A timestamp is a recorded representation of a specific moment in time Timestamps play an important role in digital investigations. Traditionally, they are used to place the event generating the timestamp at a specific moment in time. The identification that a certain event on a computer took place at a specific time makes it possible to correlate the event with events occurring outside the computer system. These may be events occurring in another digital system, or in the physical world. A particularly important application of timestamps in digital investigation is attribution; the ability to attribute events to a specific person. This is important, because most investigations aim at placing the responsibility for occurred events on one or more individuals. If evidence of the investigated events is digital, it may be necessary to place the event at a specific point in time in order to be able to attribute it to the correct person. If the time of the event inferred from the evidence is incorrect, it may not be possible to attribute it to anyone, or the event may be attributed to the wrong person. The prevalence of dynamic network addresses on the Internet makes timing important in all types of investigations of events that occurred on the Internet. In many such investigations, attribution relies on the identification of which computer were using an IP-address at a particular time. If the IP-address is dynamically assigned, the originating computer can only be identified if a log of the usage of the address exists, and the time of the event can be established with sufficient certainty and accuracy. Only in this case can the originating computer be identified from the usage log by selecting the correct IP-address and time entry.

A timestamp always refer to the clock from which it is generated. Since the timestamp is a function of the clock, it is always relative to the adjustment of the clock. Unfortunately, clocks are not fully reliable. Clocks may drift, thereby generating timestamps gradually more different from those generated from other clocks. Clocks may also fail, and produce completely incorrect timestamps. [2, 3] Further, clocks on most systems may be adjusted at any time by the user of the system to show a different date and time than civil time. The uncertainty associated with digitally stored timestamps implies that timestamps should not be relied upon as evidence without justification of these factors. In particular, it should not be blindly assumed that timestamps are based on a clock that is adjusted to civil time. These uncertainties are worrying for investigators. If timestamps cannot be relied upon, then it is in many cases not possible to trace the use of an IP-address, since identification of the time of the event is necessary to find the correct originator.

This work takes the approach that time stamps can be tested in the hypothesis based investigation model. The investigator can

e-Forensics 2008, Jan, 2008, Adelaide, Australia.

formulate a hypothesis about historical values of the clock. By defining a model of the investigated system and observing the timestamp values on the investigated computer, the clock hypothesis can be tested for consistency with the available evidence. Such testing can provide justification for a particular clock hypothesis. When a clock hypothesis is justified, the time of the events on the computer can be interpreted accordingly, and can then be used for correlation with other sources. Previously, a formalism for clock hypotheses and consistency testing with causality between timestamped events has been defined. [4] In this work, a system with actions and timestamps will be defined. This can be used to develop additional consistency tests for clock hypotheses.

2. CLOCKS

A clock is a device designed to give the user an approximation of time that is sufficiently coherent to allow him to measure and compare time periods and sufficiently consistent with other clocks to allow him to perform actions concurrent with other clock users without continuous coordination. Clocks are in other words designed to give an approximation of time.

Definition 1. Let T be the domain of time. Let V be the domain of time values produced by a clock. $c(t)$ is a clock function $T \mapsto V$.

The definition of a clock function does not impose any restrictions on the clock values as a function of time. For example, if $t_1 < t_2$ it may well be the case that $c(t_1) > c(t_2)$. And even if $t_1 < t_2 < t_3$, it may be the case that $c(t_1) = c(t_2) = c(t_3)$. The latter situation may for example occur if the events occurring at t_1, t_2, t_3 are so close together in time that the clock is unable to differentiate between them.

A clock hypothesis is a hypothesis about historical values of a clock. In an investigation, the investigator can formulate clock hypotheses and test them for consistency with the available evidence. [4]

Definition 2. A clock hypothesis $c_h(t)$ is a clock function $T \mapsto V$ that is hypothesized to represent the real clock on $c(t)$ on a system.

In an investigation, the investigator formulates a clock hypothesis, which is then the working theory about the historical values of the clock on the investigated computer. The investigator may for example hypothesize that the clock has always been adjusted to UTC+10.

The clock hypothesis may also include previous adjustments to the clock. It is for example possible for the investigator to formulate a hypothesis in which the owner of the computer adjusted the clock one year back, then created some (antedated) documents, and then adjusted the clock forward again. In such a hypothesis, the clock function $c_h(t)$ will be a discontinuous function.

3. ACTIONS AFFECT TIMESTAMPS

In order to test if a clock hypothesis is consistent with the timestamps in a system; we can build a model of the investigated system, by representing the operations in the system that can possibly change the timestamps as *actions*. A model of a system with timestamps can then be described as a table listing the timestamps and the actions that affect them. We call this an affects table.

Definition 3. An action *affects* a timestamp if and only if an occurrence of that action sets a new value for the timestamp and removes the previous value for the timestamp. An *affects table* is a table listing all possible combinations of timestamps in a system, and all actions in the system and timestamps they affect. An affects table for a system with n timestamps has 2^n entries.

Example 1. Create an affects table for the following simple file system: A file system contains files, and each file has a Created timestamp, an Accessed timestamp and a Modified timestamp. Files can be Created, Read or Written. Reading a file causes the Accessed timestamp to be updated. Writing a file causes both the Accessed timestamp and the Modified timestamp to be updated, and Creating a file causes all three timestamps to be updated. There is only one timestamp of each type for each file, so whenever a timestamp is changed, the previous value is lost.

The affects table for this file system is given in the following:

Table 1. Affects table for the file system in Example 1

	Created	Modified	Accessed	Actions
0				
1	X			
2		X		
3	X	X		
4			X	Read
5	X		X	
6		X	X	Write
7	X	X	X	Create

The affects table states clearly how timestamps are affected by actions. The affects table also shows which timestamp affect combination does not occur with any action. This information can be utilized to derive invariants on timestamp, by reasoning on sequences of timestamp updating and corresponding sequences of actions.

4. TIMESTAMPING ORDERS

In an investigation, the investigator observes values of timestamps on each investigated file. Each file has n different timestamps $\theta_1, \theta_2, ..., \theta_n$. The observed values of these timestamps were set at moments in time $t_{\theta_1}, t_{\theta_2}, ..., t_{\theta_n}$, where the values observed by the investigator are $c(t_{\theta_1}), c(t_{\theta_2}), ..., c(t_{\theta_n})$, set by the clock of the investigated system. Since the clock function $c(t)$ of the investigated system is unknown, the investigator cannot map these values directly to the moments in time $t_{\theta_1}, t_{\theta_2}, ..., t_{\theta_n}$ when timestamping occurred. But the investigator can list possible sequences of timestamping, and determine if the observed result is consistent with a specific clock hypothesis, given the affects table for the system.

Definition 4. In a system with n timestamps, the *stamping time set* Θ is the set of moments in time $t_{\theta_1}, t_{\theta_2}, ..., t_{\theta_n}$ when each

observed timestamp value $c(t_{\theta_1}), c(t_{\theta_2}), ..., c(t_{\theta_n})$ for the observed timestamps $\theta_1, \theta_2, ..., \theta_n$ was set.

Example 2. For the file system described in Example 1, the stamping time set is $\Theta = \{t_c, t_m, t_a\}$, where t_c is the time of production of the observed Created timestamp, t_m is the time of production of the observed Modified timestamp and t_a is the time of production of the observed Accessed timestamp.

To determine which (if any) sequence of actions in the system could have resulted in the observed timestamps, it is interesting to determine the different sequences in which timestamping could have occurred. Each pair of values in Θ, (t_i, t_j), may be related by either $t_i < t_j$, $t_i = t_j$ or $t_i > t_j$.

Definition 5. A *timestamping order* is a sequence of all elements in the stamping time set Θ, where each element is related to the next element in the sequence with the equals-relation = or the less-than relation <. The equals relation imply that the stamping times are equal; the two timestamps were set at the same time. The less-than relation imply that the first stamping time is earlier than the second stamping time; the production of the first timestamp occurred at an earlier time than the production of the second timestamp. Each different stamping time in a timestamping order constitutes a *step* in the timestamping order. When two or more stamping times are equal, they constitute a step in the timestamping order together.

An example timestamping order for the simple file system is ($t_c = t_m < t_a$). With this timestamping order, the Created and Modified timestamps were set at the same time, and the Accessed timestamp was set at a later time than the Created and Modified timestamps.

A list of all timestamping orders can be constructed where each stamping of a specific timestamp may have occurred before, after or at the same time as the stamping of the other timestamps. The list of possible sequences for $n = 3$ can be found in Table 2.

Table 2. All timestamping orders, n = 3

Number	Sequence	
1	$(t_1 < t_2 < t_3)$	
2	$(t_1 < t_3 < t_2)$	
3	$(t_2 < t_1 < t_3)$	
4	$(t_2 < t_3 < t_1)$	
5	$(t_3 < t_1 < t_2)$	
6	$(t_3 < t_2 < t_1)$	
7	$(t_1 = t_2 < t_3)$	
8	$(t_3 < t_1 = t_2)$	
9	$(t_2 = t_3 < t_1)$	
10	$(t_1 < t_2 = t_3)$	
11	$(t_1 = t_3 < t_2)$	
12	$(t_2 < t_1 = t_3)$	
13	$(t_1 = t_2 = t_3)$	

5. POSSIBLE ACTION SEQUENCES

When all timestamp updating is represented by actions, the cause of timestamping having occurred in a specific sequence must have been actions that have occurred in a specific sequence. An action sequence is a sequence of actions of arbitrary length.

Definition 6. An *action sequence* is a sequence of one or more actions, where each element is related to the next element in the sequence with the equals-relation = or the less-than relation <. The equals relation imply that the actions occurred at the same time. The less-than relation imply that the first action occurred earlier than the second action.

The relationship between an action sequence and a timestamping order is that every observed timestamping order must have been created by an action sequence. When considering all timestamping orders, there may be many action sequences that may cause a particular timestamping order. There may however also be timestamping orders, which cannot be created by any action sequence. These timestamping orders cannot occur in the system. The relationship between action sequences and timestamping orders can be deducted from the affects table.

Definition 7. A timestamping order is *possible in a system* if there is at least one action sequence in the system that may cause the timestamping order. If there is no action sequence that can cause the timestamping order, then the timestamping order is *impossible in the system.*

By using the affects table, it is possible to find all action sequences that may have caused a specific timestamping order, by the following procedure:

1. Find all actions or combination of actions affecting all timestamps in the first step in the timestamping order.
2. For each following step in the timestamping order, find all actions or combination of actions affecting all timestamps in that step, and not affecting any timestamps listed in previous steps. If there is no such action, then this timestamping order is not possible in the system.

The task of finding all actions or combination of actions can be implemented as follows:

1. For every timestamp θ_i find all actions affecting it, and add them to a set A_i.
2. For every action $a \in A_i$, check if a affects any timestamp θ_j listed in previous steps in the timestamping order. If so, remove it from A_i.
3. Actions $a \in (A_1 \cap A_2 \cap ... \cap A_n)$ affect all timestamps in that step. Remove them from A_i.
4. If all sets A_i are still non-empty, the remaining actions represent combinations of actions affecting all timestamps for that step. The combinations can be found with the Cartesian product $A_1 \times A_2 \times ... \times A_n$

Example 3. Find all action sequences for the timestamping order ($t_c < t_m < t_a$) for a file in the file system in Example 1.

From the affects table for the simple file system in Table 1, the steps in the sequence yields:

Step 1 (t_c): Create (t_c is only affected by Create)

Step 2 (t_m): Write (t_m is affected by Create and Write, only Write does not affect t_c)

Step 3 (t_a): Read (t_a is affected by Read/Write/Create, only Read does not affect t_c, t_m)

Thus, the only possible action sequence for timestamping order ($t_c < t_m < t_a$) is (Create < Write < Read).

Example 4. Find all action sequences for the timestamping order ($t_m = t_a < t_c$) for a file in the file system in Example 1.

From the affects table for the simple file system in Table 1, the steps in the sequence yields:

Step 1 (t_m=t_a): Create, Write (t_m and t_a are both affected by Create and Write)

Step 2 (t_c): *none* (t_c is only affected by Create, but Create also affects t_m and t_a)

Thus, the timestamping order ($t_m = t_a < t_c$) is not possible in the system.

By using this procedure for all timestamping orders for a given number of timestamps, one can now complete the reasoning in a system with known actions. The result of this exercise will be a list of timestamping orders impossible in the system and a table of possible action sequences of each timestamping order possible in the system.

Example 5. Find all action sequences for the simple file system.

This file system has three timestamps for each file (n = 3). All timestamping orders for such a system are given in Table 2. Assigning $t_1 = t_c$, $t_2 = t_m$ and $t_3 = t_a$ produces all timestamping orders for this system, shown in column "Timestamping order" in Table 3. Following the action sequence procedure for each timestamping order listed in the table by using the affects table for the simple file system given in Table 1, gives the possible action sequences for that timestamping order, shown in the column "Action Sequence":

Table 3. Action sequences for the simple file system

Number	Timestamping order	Action Sequence
1	($t_c < t_m < t_a$)	(Create, Write, Read)
2	($t_c < t_a < t_m$)	None
3	($t_m < t_c < t_a$)	None
4	($t_m < t_a < t_c$)	None
5	($t_a < t_c < t_m$)	None
6	($t_a < t_m < t_c$)	None
7	($t_c = t_m < t_a$)	(Create, Read)
8	($t_a < t_c = t_m$)	None
9	($t_m = t_a < t_c$)	None
10	($t_c < t_m = t_a$)	(Create, Write)
11	($t_c = t_a < t_m$)	None
12	($t_m < t_c = t_a$)	None
13	($t_c = t_m = t_a$)	(Create)

The only timestamping orders in Table 3 possible in the system are sequences where $t_c \le t_m \le t_a$. Thus, $t_c \le t_m \le t_a$ is a property that always holds for this system, an invariant.

Invariants for a system that has been found using the reasoning above can be used to test a clock hypothesis. In the example file system, it is now known that $t_c \le t_m \le t_a$. If for example $c(t_c) > c(t_a)$, a hypothesis that the clock has always been adjusted to UTC+10 would be rejected, since UTC is never adjusted backwards.

6. MODELLING A REAL FILE SYSTEM

The procedure described in the previous sections can be used to create a model of a real file system, determine which timestamping orders are possible in the system and derive invariants of the file system for use with a clock hypothesis checker. To illustrate this procedure, this section performs it on the semantics in Windows XP for file timestamps stored in the NTFS $STANDARD_INFORMATION attribute. The basis for the model described here is the semantics determined by Carrier. [5] The model assumes that the files in question exist, are larger than the file cache size, and that updating of the last accessed timestamp is enabled.

In a system with three timestamps, the affects table contains $2^3 = 8$ entries. The actions are:

Read: reading a file

Create: creating a new file

Write: modifying an existing file

CopySrc: copying a file (source file)

CopyDest: copying a file (destination file)

MoveIntra: moving a file internal to a file system

MoveInterSrc: moving a file across file systems (source file)

MoveInterDest: moving a file across file systems (destination file)

The following affects table can then be constructed:

Table 4. Affects table for Windows XP / NTFS

	Created	Modified	Accessed	Actions
0				
1	X			
2		X		
3	X	X		
4			X	Read,

				CopySrc, MoveIntra, MoveInterSrc, MoveInterDest (ReadGroup)
5	X		X	CopyDest
6		X	X	Write
7	X	X	X	Create

The actions in row 4 of the affects table all have the same effect on timestamps. In the following, they will be grouped together as ReadGroup, meaning that where this action occurs, any of the actions Read, CopySrc, MoveIntra, MoveInterSrc or MoveInter-Dest may have occurred.

With $n = 3$, the timestamping order table in Table 2 can be used. Applying the action sequence procedure for each timestamping order yields the table of action sequences listed in Table 5.

Table 5. Timestamping orders in Windows XP/NTFS.

No	Timestamping order	Action Sequence
1	($t_c < t_m < t_a$)	(Create / CopyDest < Write < ReadGroup)
2	($t_c < t_a < t_m$)	None
3	($t_m < t_c < t_a$)	(Create / Write < CopyDest < ReadGroup)
4	($t_m < t_a < t_c$)	None
5	($t_a < t_c < t_m$)	None
6	($t_a < t_m < t_c$)	None
7	($t_c = t_m < t_a$)	(Create / CopyDest=Write < ReadGroup)
8	($t_a < t_c = t_m$)	None
9	($t_m = t_a < t_c$)	None
10	($t_c < t_m = t_a$)	(Create / CopyDest, Write)
11	($t_c = t_a < t_m$)	None
12	($t_m < t_c = t_a$)	(Create / Write, CopyDest)
13	($t_c = t_m = t_a$)	(Create / CopyDest=Write)

From the table, it is evident that there are no possible action sequences where t_a does not occur in the last step. Consequently, in this system, $t_m \leq t_a$ and $t_c \leq t_a$. These invariants can be used to check clock hypotheses for Windows XP systems with NTFS.

7. RESULTS

This work studied how a system model can be created used to test a clock hypothesis for consistency with timestamp evidence. A system model can be created by listing the actions in the system and their effect on timestamps in an affects table. By listing all possible timestamping orders, it can be determined which timestamping orders are possible in the system and which action sequences that may cause them. A procedure for deriving possible action sequences from the list of possible timestamping orders is given. From the list of possible and impossible timestamping orders, invariants for a system can be derived. These invariants can be used to test a clock hypothesis for consistency with evidence in the form of timestamps stored on an investigated system.

On the systems examined in real digital investigations, there will exist tens- or even hundreds of thousands of timestamps. By modelling the system using the techniques described in this paper, it is then possible to test a clock hypothesis against a large number of timestamps. This will put a clock hypothesis under close scrutiny, and will lead to its justification if there is no evidence to refute it. Justification of a clock hypothesis is important in digital investigations, because it will provide a possibility to translate the timestamps observed on a system to an independent clock. Thus, the real time of stamping can be established, which can be used to correlate the time of the events on a digital system with events occurring elsewhere.

8. CONCLUDING REMARKS

The testing of clock hypotheses provided in this work requires a model of the investigated system to be constructed. In order to provide a complete model of a real system one must clearly understand the system completely, something that can probably only be accomplished by studying the implementation details of the system. It might however be reasonable to construct a partial model only by studying the effects of operations on the real system, if it can be justified that the only actions taken on the system were those that were included in the model. In a real operating system, this could for example be accomplished by testing the different operations in the system and how they affect timestamps. If one could not be sure that all possible operations in the operating system had been included, one would not know for certain if the rejection of a clock hypothesis was caused by a wrong clock hypothesis or by missing actions in the model. This does not have to be a serious problem in digital investigations, where timestamp operations must be manifested in software, which can be found during the investigation.

The method provided in this work can be applied during investigations of digital media, such as seized computers. Since most systems use common operating systems, the construction of a model does not have to be repeated in every investigation. It is enough that the model has been built for a specific system type once, it can thereafter be used in all digital investigations concerning that system type. The method presented here are therefore well suited for implementation in integrated software packages for digital investigation.

9. REFERENCES

[1] B. Carrier, "A hypothesis-based approach to digital forensic investigations," Center for Education and Research in Information Assurance and Security, Purdue University Tech Report 2006-06, 2006.

[2] B. Schatz, G. Mohay, and A. Clark, "A correlation method for establishing provenance of timestamps in digital

evidence," *Digital Investigation,* vol. 2006:3S, pp. 98-107, 2006.

[3] F. Buchholz and B. Tjaden, "A brief study of time," *Digital Investigation,* vol. 2007:4S, pp. 31-42, 2007.

[4] S. Y. Willassen, "Hypothesis based investigation of digital timestamps," in *IFIP WG 11.9 Workshop*, Kyoto, Japan, 2008.

[5] B. Carrier, *File system forensic analysis*. Upper Saddle River, N.J.: Addison-Wesley, 2005.

Finding Evidence of Antedating in Digital Investigations

Svein Yngvar Willassen

Abstract— **Finding evidence of antedating is an important goal in many digital investigations. This paper explores how causality can expose antedating by investigating storage systems for causality and correlate causality with stored timestamps. Causality is determined in two different system types; storage systems using sequence numbers and storage systems using the first-fit allocation strategy. Causality found in these systems was used to implement a timestamp consistency checker for the NTFS file system. The implementation was then tested in an experiment, in which four subjects were asked to antedate a document on a given computer in such a way that the antedating could not be determined by an investigator. The results from this experiment show that the implemented consistency checker can be used to expose antedating. Investigators can use this method to find evidence of antedating to be presented to fact-finders in real cases.**

Index Terms— **digital investigation, antedating, timestamps, causality**

I. Introduction

Antedating is the creation of files, documents and other material with date- or timestamps set to another date than the date the material was created. Exposing antedating is a common goal in digital investigation, either because the matters under investigation involves documents produced with digital computers, or because the timing of the production of information stored on a computer is otherwise important. The typical digital investigation involving antedating is investigations of financial crimes or other matters where the date of production of a document has legal implications. In these matters, the goal of the digital investigation is often to find out if the document was really produced at the date printed on the document, or if it could have been produced at a later date.

When typing a document in a word processor, it is easy to change the written date. This would not change the timestamps stored on the file system when the document is stored. It is however possible to antedate these timestamps too, by altering the computer system clock to represent a different date than the current. If the system clock is altered before the document is produced, the timestamps associated with the produced document will be set to the date the system clock was adjusted to. With this procedure, it will not be possible to determine that the document was antedated with previous digital investigation methods.

In this work, causality in digital systems will be used to determine if particular timestamps have been antedated or not. Causality defines which events are necessary for others to occur. In a digital investigation, the medium to be investigated is a storage system storing digital data. In such a system, the events of storage of specific items can be causally related to the events of storage of other items. Since the stored data may contain timestamps, the causal relation can be used to test the consistency of the timestamps, and thereby expose antedating.

II. Related work

Being recognized as a research challenge, the problem of timestamp interpretation in digital investigation has been studied by a few researchers during recent years. Schatz et al suggests mitigating the problem by correlating the timestamps in web cache stored on the computer with records obtained from the web servers. [1] Weil and Boyd et al suggest similar correlation methods, by using timestamps stored on the investigated computer coming from other clocks, such as timestamps in dynamically generated web pages. [2, 3] Such methods would provide correlation for the period for which cached data exist on the investigated computer only. Correlation with server records is only possible when such records actually exist, and the investigator has legal access to them.

Stevens studied clocks and described a clock model where each clock is described as the clock it was originally derived from plus the sum of all adjustments, errors and drift. [4] The clock model described by Stevens was refined by Buchholz, in the formalization of a clock model as the sum of clock drift and adjustments. [5] These models are versatile and provide good tools for event reconstruction in cases where clock adjustments, error and drift are known or measurable. They do not however by themselves assist in the identification of clock adjustment, error or drift.

Manuscript received October 11, 2007.

S. Y. Willassen is with the Department of Telematics, Norwegian University of Science and Technology, O.S. Bragstads plass 2B, 7491 Trondheim, Norway (phone: +47 92449678; email: svein@willassen.no).

In previous works, we have defined a formalism for clock hypotheses and consistency testing with causality between timestamped events. [6] This formalism has been utilized to develop models for the updating of timestamp values in digital systems. [7] In this work, we use the happened-before relation defined in [6] to analyze specific properties of file systems. These properties are then used to find evidence of antedating.

III. SEQUENCE NUMBER CAUSALITY

Sequence numbers is a feature occurring in many digital systems, such as file systems and networks. By using a sequence number, the systems designer ensures that sequence numbered entities can be ordered in the correct order and be distinguished from each other. Sequence numbers are usually implemented by a counter increasing whenever a new sequence numbered entity is produced and associating a copy of the value of the counter (the sequence number) with that entity. It is useful to distinguish between *wrapping sequence numbers* and *strictly increasing sequence numbers*. Wrapping sequence numbers have a limited span of values. When the highest value is reached, the counter wraps and starts at the lowest value. A strictly increasing sequence number on the other hand is a sequence number that does not wrap. In theory a strictly increasing sequence number would have to be able to represent infinite numbers. In practice however, a sequence number can be viewed as strictly increasing as long as the number of values that can be represented is large enough to produce strictly increasing numbers over a significant time span, for example the lifetime of a computer.

When investigating a system with sequence numbered entities, the distinction between wrapping sequence numbers and strictly increasing sequence numbers is important. With a wrapping sequence number, one would not be able to know how many times the counter had wrapped when the sequence number was generated. When correlating two entities with sequence numbers, one would therefore not be able to determine if one of the entities was produced before the other. In a system with strictly increasing sequence numbers on the other hand, one can be sure that the entity with the highest sequence number has been produced after the entity with the lowest sequence number. In such a system, each production of a sequence numbered entity is causally dependant on the production of every other sequence numbered entity with lower sequence number.

Many file systems contain File System Journals with sequence numbered entities. For example, in NTFS, journal file transactions are labelled with a 64-bit number (so called Logical Sequence Number - LSN) that increases throughout the lifetime of the file system. The proper functioning of the journaling feature in NTFS depends on this number being strictly increasing. [8]

In these systems, it is possible to find causal connections by analyzing journal files. The amount of information that can be derived from the journal file itself is however limited. Since every write to a file produces a journal file entry and the journal file has limited size, old entries will quickly be overwritten. Some file systems, such as NTFS, store the journal file sequence number (the LSN in NTFS) in the file metadata entry. If the journal file sequence number is strictly increasing, the generating events are causally connected. Causal connections then exist between the events of the last change of the file entry on all files stored on the file system.

Example 1. Consider the following set of allocated file entries from an NTFS Master File Table. Let e_i be the last update of the current data in entry i.

Entry 45 log file sequence number 432627
Entry 46 log file sequence number 186345
Entry 47 log file sequence number 735294
Entry 48 log file sequence number 165093
Entry 49 log file sequence number 878121
Entry 50 log file sequence number 782427
Entry 51 log file sequence number 561987

Since logical sequence numbers in the journal file (log file) are allocated sequentially, we can obtain the causal ordering of the last update events by sorting the file entries by their log file sequence number: $e_{48} \rightarrow e_{46} \rightarrow e_{45} \rightarrow e_{51} \rightarrow e_{50} \rightarrow e_{47} \rightarrow e_{49}$.

IV. ALLOCATION SEQUENCE CAUSALITY

A first-fit allocation storage is a system in which each new data item is stored in the first available storage location. Deleting data items is allowed and can be done at any time after the data item has been stored in a storage location. After a data item has been deleted, it may be overwritten by new data at any time. It may be possible to recover deleted data, but it is not possible to recover data that has been overwritten.

A special form of first-fit storage is first-fit storage *with generation-markers.* In this storage, it is possible to identify which generation the data in each storage location belongs to. The generation of a storage location is an identification of how many times that data in that storage location has been overwritten. Fig. 1 shows a possible allocation sequence with generation markers.

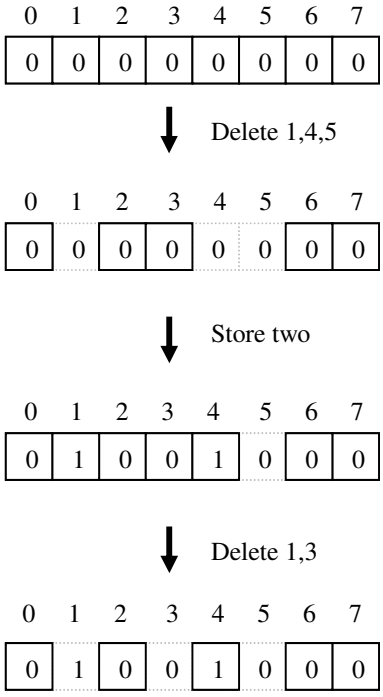

Fig. 1 Allocation sequence with generation markers

In a system with generation markers, there is a causal connection between every pair of consecutive generations at each storage location. The storage of data in the i-th storage location generation g can only commence if the data present in the i-th storage location generation (g-1) has already been stored and deleted. Therefore, for every storage location i, the storage of data in generation (g-1) happened-before the storage of data in generation g. Generally; due to the transitivity property of the happened-before relation, the event of storing data in a storage location is causally dependant on the storing of all previous generations in that storage location.

Let $s_{i,g}$ be the i-th storage location generation g. Let $e_{si,g}$ be the event of storing data in the i-th storage location at the g-th generation. Then for all i and g; $e_{si,g-1} \rightarrow e_{si,g}$. Due to the transitivity of $\rightarrow$, for all generations g,

$$\forall h < g(e_{si,h} \rightarrow e_{si,g}) \quad (1)$$

We now consider the storage of data in storage locations with generation g = 0. When g = 0, there cannot exist any storage location which has been deleted and then overwritten with another data item, because this would have increased the generation number above 0. The only place where new storage locations can be allocated with generation number 0 is at the end of the storage.

Let $s_{i,0}$ be the i-th storage location in a first-fit storage, generation 0. Let $e_{si,0}$ be the event of storing data at generation 0 in the i-th storage location. Then, for all i,

$$\forall j < i(e_{sj,0} \rightarrow e_{si,0}) \quad (2)$$

Two different types of causal event sets have now been defined from the first available storage with generation markers; the causality between storage of storage locations with g = 0, and causality between storage of increasing generations at each storage location. These sets intersect. Each causality set for increasing generations start at g = 0. Each such element is also part of the g = 0 causal set. With these two types of causal connections in the first available storage with generation markers, a causal connection relating all storage locations in the set has been found.

Example 2. Consider the storage location set in Fig. 2. In the figure, the storage locations are shown horizontally, and generations vertically. Deleted data are shown in lighter colour. For each storage location, the topmost item is always the current data stored in the location. There are now causal connections, where each generation within a storage location happened-before the next generation, and each storage location at generation 0 happened-before the next location at generation 0. The resulting Direct Acyclic Graph of the creation events of the existing storage locations is shown in Fig. 3.

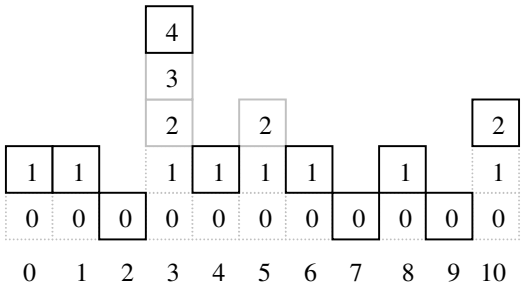

Fig. 2 Storage locations with generations

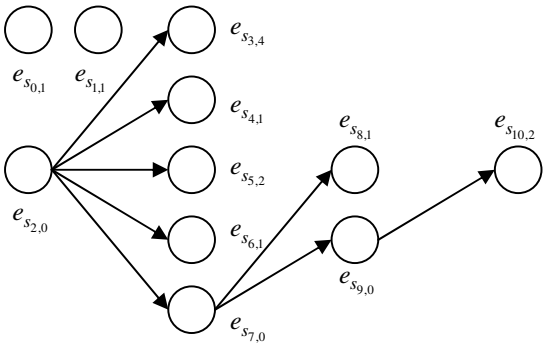

Fig. 3 Causality of existing (non-overwritten) storage locations

Example 2 shows how the happened-before relation imposes a strict partial order on the set of storing events. The partial order follows from the properties of the happened-before relation; it is irreflexive, transitive and asymmetric. [6] The relation does not however relate all elements of the set of

storing events. Elements with $g > 0$ are not related to other elements with $g > 0$, as shown in Fig. 3. A total order is therefore not imposed on the set of storing events.

The Master File Table of the NTFS file system is a first-fit storage with generation markers. Each file stored in NTFS has its own entry in the Master File Table. Data stored in the MFT entry include the file name, the list of data runs where the file data is stored, timestamps and other data such as information on whether the data is compressed or encrypted using the compression and encryption features in the file system. Allocation of file entries within the MFT occurs on a first-fit basis. [9] Each file entry contains a generation marker, termed *entry sequence number*, which identifies the generation of that entry. This number is increased whenever the file entry is reused. Thus, causal connections exist between file entries in the Master File Table of the NTFS file system, following the reasoning above.

V. Implementation

An implementation of the above reasoning above was made in a program named TimeStampLogic. The program uses the utilities in a modified version of the Sleuthkit [10] to find file instances in an NTFS file system image. It then parses the output of these utilities and produces internal representations of file instances, which can then be analyzed. The analysis consists of two modules; SequenceChecker and LogSequenceChecker. SequenceChecker tests the generation causality of the file entries, by the reasoning in Section IV. All entries are compared with the last preceding entry with the base generation sequence number. Since the allocation of Master File Table entries are done in a first-fit fashion, all entries have been stored at a later time than the last preceding entry with the base generation sequence number. LogSequenceChecker tests the causality of the updating of file entries based on the log file sequence numbers stored in the Master File Table file entries, by the reasoning in Section III. The instances are sorted by the associated Log File Sequence Number (LSN) and then printed in sorted order.

By inspecting the timestamps associated with each file entry in comparison with the two different sequences produced by SequenceChecker and LogSequenceChecker, it can now be determined if the computer clock has been changed.

VI. Document antedating experiment

In order to test the theory and implementation, a document antedating experiment was performed, in which four subjects were asked to antedate a document. A laptop computer was prepared for the experiment. The hard drive of the computer was wiped. The computer was then started and the system clock was adjusted to approximately two and a half years before the time of the experiment with the BIOS setup program. Then, the Windows XP operating system was installed. After installation, a series of shutdowns, clock forward adjustments and reboots were performed, until the clock was adjusted to civil time in October 2006, when the experiment was performed. The goal of this procedure was to produce data on the hard drive similar to data that would have been produced by real usage of the computer. For each step, the computer was shut down, then started in BIOS setup, where the clock was adjusted forward. The computer was then booted into Windows XP and used for web surfing, downloading files or other normal user activity. After this procedure, the hard drive was copied to an image file on another hard drive using the disk dump utility dd, producing a reference image of the experiment computer.

The experiment computer was then handed to the participating subjects with the following task: "Store a document on this computer in such a way that a person investigating the computer will conclude that the document was produced on May 17th, 2006." When each subject returned the computer, the hard drive was copied to an image file on another hard drive for analysis. Then, the experiment image was copied back to the computer before it was handed to the next subject. The subjects participating in the experiment were:

Subject no	Computer experience level
1	Average computer user, using computer every day
2	Law Enforcement Computer Forensic Investigator
3	Inexperienced office user, mostly used to websurfing
4	Advanced computer user with programming experience

Each image was then analyzed using the TimeStampLogic program. Each subject was also interviewed to determine how they had chosen to perform the task.

VII. Results

In the following, each of the images resulting from imaging the experiment computer after each subject had completed the task is analyzed. The purpose of the analysis is to determine if the document in question has been antedated or not. In a hypothesis based approach to digital investigation (defined by Carrier [11]), this can be formulated as two different hypotheses:

H_0: The document was produced on 17th of May.
H_1: The document was produced later than 17th of May, but has been antedated to 17th of May.

The task for the investigator is then to find evidence supporting or rejecting H_0 and H_1 using TimeStampLogic and other investigative tools.

A. Subject 1

The subject gave the following information about how the task was completed: *I adjusted the clock on my Mac to May 17th. I then produced the document in Microsoft Word on the Mac. When saved on the Mac, I copied the document to my USB stick and inserted it into the PC. I then copied the document from the USB stick to the PC. I believe producing the document on the Mac may have prevented the creation of timestamps inside the Word document.*

When analyzed with TimeStampLogic, the results of this operation did not produce a result significantly different from analysis of the reference image. The introduction of new files when the computer was booted and a new document was copied to it, did not produce any new inconsistencies reported by TimeStampLogic. The document has Modified timestamp on the 17th of May, and Created and Accessed timestamps on the date of the experiment. This is consistent with timestamps produced when files are copied to a medium. Other evidence suggesting that the file had been copied to the medium on the date of the experiment was also found, for example a link-file to an external drive, showing that an external drive had been connected to the computer. If the file had been copied from another computer, the Modified timestamp would then be related to the clock of that computer and not the investigated computer. Since no evidence is available to test clock hypotheses for the other computer, there is no evidence to either support or reject a hypothesis that the production of the document actually occurred on 17th of May civil time. The analysis is therefore inconclusive in this case. The reasonable investigative response in cases like this is to try to get hold of the computer on which the document was produced and do the same type of analysis on that computer.

In response to the subject's claim that timestamps had not been created inside the Word document, it was examined for timestamps in the metadata. Such timestamps were found, identifying that the document had been created and last changed on May 17th. These timestamps would also refer to the clock on the other computer, which will have to be analyzed for evidence.

B. Subject 2

The subject gave the following information about how the task was completed: *I started the PC and connected it to the Internet. I then downloaded and installed OpenOffice on the PC. I then restarted the computer, went into BIOS and adjusted the date back to May 17th. After booting the computer again, I used OpenOffice to create and store the document. I then booted again and adjusted the clock back to current time. I used OpenOffice because I think it doesn't have the same amount of metadata as Microsoft Word. I also think downloading and installing OpenOffice would prevent a proper investigation, since it probably overwrote a lot of deleted data.*

When analyzed with TimeStampLogic, a significant higher number of inconsistencies were reported with both SequenceChecker and LogSequenceChecker. Listing all files on the medium ordered by both the MFT Entry number (SequenceChecker) and Log Sequence Number (LogSequenceChecker), showed several hundred files with Created, Modified and Accessed timestamps on Oct 11th in the time period 07:28-07:40 AM. After these (in terms of entry number and log sequence number), approximately 50 files with Created, Modified and Accessed timestamps on May 17th time period 07:42-07:48 were listed. All timestamps of the document in question were set to May 17th in the period 07:42-07:48.

The timestamps on the document itself were in this case set to May 17th, in contrast to Subject 1. There is however evidence in this case supporting H_1:

- Storing of a significant number of files causally dependant on storing of files occurring on Oct11th, were timestamped May 17th, something that is not possible unless the clock has been adjusted, or the timestamps changed.
- When the date changes from Oct 11th to May 17th, the time of day only moves approximately 2 minutes forward. This indicates that the subject changed the date but did not bother to change the time of day. If the clock adjustment had occurred by an error or some other mysterious event, it is not very likely that it would have ended up at this exact time of day.

The subject's claim that he made the investigation more difficult by installing OpenOffice, does not seem to be correct in the context of using TimeStampLogic to check timestamp consistency. It may be the case that installing a new program would overwrite previously overwritten material, but this does not help, since TimeStampLogic does not rely on the investigator's ability to recover deleted material.

C. Subject 3

The subject gave the following information about how the task was completed: *I don't know how to manipulate timestamps, so I just went into the control panel and set the date to May 17th. Then I used Microsoft Word to produce the document. Then I set the current date again in the control panel.*

On this image, TimeStampLogic produced the same type of results as on the image from Subject 2. Approximately 10 files were listed with Created, Modified and Accessed timestamps on Oct 12th from 9:17-9:44 PM. After this (in terms of MFT entry sequence and LSN sequence), approximately 10 files were listed with timestamps at May 17th 9:46-9:52 PM. This gives evidence for H_1, for the same reasons as for Subject 2.

In this case, as opposed to the case of Subject 2, the clock change was done in the operating system. Therefore, the event logs of the system were searched to determine if the clock change had logged a system event. No such event was found. Windows XP has a system logging feature that allow logging of clock change events. This particular event is however logged only if Privileged Use logging is enabled, something it is not by default. [12]

It is interesting to note that both Subject 2 and 3 changed the date without changing the time of day. Both in the BIOS of the experiment computer and in the Windows XP control panel, changing date is done by a separate control than changing time of day, even if they are both related to the same underlying clock. A plausible rationale for not changing the time of day could be that it would then be easier to adjust the clock back to the current time, because one would then not have to resynchronize with an external clock. When asked about this, subject 3 said: *I didn't think about that. I just wanted the correct date on the document. The time of day didn't matter to me. I might have thought about it if the time of day were of any importance, for example if it mattered if I were at work at the time or not.*

D. Subject 4

The subject gave the following information about how the task was completed: *I used my own pc for the antedating. I adjusted its clock back to May 17th, and produced the document using Microsoft Word. I then copied the document over to the experiment PC using my USB-stick.*

The story of Subject 4 matches the story of Subject 1, and the results of TimeStampLogic were similar. No additional inconsistencies were found, and the results were inconclusive on the question of whether the document was antedated or not. Also in this case, link files pointing to an external medium identified another computer as the likely source for the document.

E. Summary

In the document antedating experiment, four subjects were asked to antedate a document in such a way that it could not be determine that the document file was antedated. Two of the subjects performed the antedating in such a way that the methods described in this work could produce evidence supporting the hypothesis that the document was antedated and not produced on the date it was timestamped to. Two of the subjects did the antedating itself on another computer and copied the resulting document to the investigated computer. In this case, it could not be determined that the document was antedated, but it could be determined that the document had been copied from another computer, thus another possible item of evidence was found. It is known from the explanation from the subjects, that they produced the antedated document on the other computer by adjusting the clock back to May 17th, which is the same method used by Subject 2 and 3 on the investigated computer. Investigation of the other computer with the methods described in this work would therefore most likely have produced evidence supporting the hypothesis that the document was antedated.

VIII. Concluding remarks

Causality reasoning can be used to check timestamp evidence for consistency with causal ordering of events. Such reasoning can be used to determine if digital information has been antedated or not. The document antedating experiment has shown that causality reasoning can provide evidence of antedating of computer files in practical situations, where subjects have antedated a file. The described methods can be implemented in existing tools for digital investigation such as the Sleuthkit or EnCase. This would provide investigators with the possibility for time and causality reasoning in real cases.

The antedating methods used by the subjects in the experiment are certainly not the only way to antedate a document. Other possible methods can be conceived:

1. Produce the document at current time, then changing its timestamps by special software. This can be done without introducing the software in question on the investigated computer by removing the medium and perform the change on another computer.
2. Finding another file matching the desired timestamps, then replacing the contents of that file with specialized software.
3. Using the same method as used by the subjects in the experiment. Then use special software that adjusts all timestamps on the medium to match the story. Such software could be called anti-TimeStampLogic.

In the case of conceived method 1, TimeStampLogic would probably report the single file as an inconsistency. In the case of conceived method 2 and 3 however, it is not likely that TimeStampLogic would be able to find any inconsistencies. Producing evidence of antedating in these cases would have to rely on other methods, if possible at all. Thus, clock hypothesis testing methods described here are not perfect methods that cannot be avoided by a crafty antedater.

This possibility of evidence manipulation does not however imply that the described methods are not useful in real investigations. Consider the adversaries in a digital investigation, the *Investigator* and the *Perpetrator*. The Investigator usually possesses knowledge of digital investigation and tools that can comb a digital medium for evidence, including tools for digital imaging and data recovery. The Perpetrator on the other hand is likely to be an

average computer user, and does not know how to protect himself from the scrutiny of a digital investigation or where he should go to obtain the necessary tools. The Investigator also has time on his side. Once a digital medium has been forensically imaged, he has plenty of time to investigate its contents. The Perpetrator on the other hand never knows when the Investigator will turn up to seize his data, if ever. He therefore has to be prepared at all times and run the anti-forensic procedure again and again after every action that would leave incriminating evidence. There is no room for mistakes by the Perpetrator. If he makes a small mistake in his anti-forensic procedure, the evidence may be there waiting to be discovered by the Investigator. The Investigator on the other hand can make a lot of mistakes, as long as he doesn't mess up the original data. He can always start from a fresh image at a later time, should he feel that there is more to find or that current results rely on misinterpretations. All in all, the Investigator has a tremendous advantage over the Perpetrator in digital investigations.

The above can also be viewed in light of Locard's exchange principle, in which it is stated that every physical contact yields exchange of matter so that subsequent forensic investigations can prove that the contact occured by analyzing the exchanged matters. By using special software that adjusts all timestamps on the medium to match a predefined story, it is likely that a special timestamp pattern specific to that software would be created. It would then be possible for the investigator to produce evidence of the usage of such software. This would be highly undesirable for the perpetrator, since it would create an impression that he had something to hide.

Applying this reasoning to the methods developed in this work, the conclusion must be that there exist methods by which the investigation methods described in the work can be avoided. This is however difficult to do to in such a way that it cannot be detected. Subjects who want to use antedating are not likely to possess the required knowledge to do this. The described methods are therefore adequate for exposing antedating in most real investigations.

REFERENCES

[1] B. Schatz, G. Mohay, and A. Clark, "A correlation method for establishing provenance of timestamps in digital evidence," *Digital Investigation,* vol. 2006:3S, pp. 98-107, 2006.

[2] M. C. Weil, "Dynamic Time & Date Stamp Analysis," *International Journal of Digital Evidence,* vol. 1:2, 2002.

[3] C. Boyd and P. Forster, "Time and date issues in forensic computing - a case study," *Digital Investigation,* vol. 2004:1, pp. 18-23, 2004.

[4] M. W. Stevens, "Unification of relative time frames for digital forensics," *Digital Investigation,* vol. 2004:1, pp. 225-239, 2004.

[5] F. Buchholz, "An Improved Clock Model for Translating Timestamps," James Madison University, Department of Computer Science JUM-INFOSEC-TR-2007-001, 2007.

[6] S. Y. Willassen, "Hypothesis based investigation of digital timestamps," in *IFIP WG 11.9 Workshop*, Kyoto, Japan, 2008.

[7] S. Y. Willassen, "Timestamp evidence correlation by model based clock hypothesis testing," in *E-Forensics 2008*, Adelaide, Australia, 2008.

[8] M. E. Russinovich and D. A. Solomon, *Microsoft Windows internals : Microsoft Windows Server 2003, Windows XP, and Windows 2000*, 4th ed. Redmond, Washington: Microsoft Press, 2005.

[9] B. Carrier, *File system forensic analysis*. Upper Saddle River, N.J.: Addison-Wesley, 2005.

[10] B. Carrier, "Sleuthkit," Available at: www.sleuthkit.org.

[11] B. Carrier, "A hypothesis-based approach to digital forensic investigations," Center for Education and Research in Information Assurance and Security, Purdue University Tech Report 2006-06, 2006.

[12] University of Delaware Police Computer Forensics Lab, "Time Change Captured in Event Log," http://128.175.24.251/forensics/timechange.htm Accessed: Oct 3. 2007

Using Simplified Event Calculus in Digital Investigation

Svein Yngvar Willassen
Department of Telematics
Norwegian University of Science and Technology
O.S. Bragstads plass 2
7491 Trondheim, Norway
+47 92449678
svein@willassen.no

ABSTRACT
In a hypothesis-based approach to digital investigation, the investigator formulates his hypothesis about which events took place, and tests them using the evidence available. A formalism for the description of the investigated system is useful in the hypothesis formulation and testing. Simplified Event Calculus, a form of propositional logic, can be used to define and test hypotheses in a digital investigation. When a system is modelled in this logic, observed states can be used to find action hypotheses and test them in the model. This can assist investigators and fact-finders in reconstruction of events from digital evidence. The logic can also be used to derive invariants for a system that can be utilized in tools checking evidence from these systems for consistency.

Categories and Subject Descriptors
F.4.1 [**Mathematical Logic and formal languages**]: Mathematical Logic – *model theory, temporal logic.*

General Terms
Theory, Legal Aspects, Verification.

Keywords
event calculus, digital investigation, propositional logic

1. INTRODUCTION
Investigations are inquiries into past events. The purpose of an investigation is to find evidence that can establish an understanding of events previously taken place. Investigation of digital media with the purpose of finding evidence is commonly referred to as *digital investigation*. The purpose of digital investigation is to find evidence related to the events under investigation. In recent works, most notably by Carrier, efforts have been made to make the digital investigation process based on scientific principles, by using a hypothesis-based approach. [1] In this approach, the investigator formulates his hypothesis about which events took place, and tests them using the available evidence.

In a hypothesis-based approach, it is useful to use a formalism to describe the system under investigation, and the events that have taken place. Such a formalism needs to be able to describe events occurring on a system and their effect on the state, in such a way that the investigator's hypothesis can be tested for consistency with the evidence on the examined system. Previous works have used variants of Finite State Machines to represent the system under investigation and parts thereof. [1, 2] In this work, a variant of propositional logic will be used for the same purpose. This paper examines if and how Shanahan's Simplified Event Calculus [3] can be used in a hypothesis based approach to digital investigation.

SAC'08, March 16-20, 2008, Fortaleza, Ceará, Brazil.

2. SIMPLIFIED EVENT CALCULUS
In Simplified Event Calculus, the world is modelled with *fluents* and *actions*. Fluents are states that can hold for a specified or unspecified period of time. Actions are occurrences that initiate or terminate a fluent. Occurrences of actions and fluents are defined with the *HoldsAt* and *Happens* predicates, and the affection of actions on fluents is defined by the predicates *Initiates* and *Terminates*. In addition, there is an *Initially* predicate, for initiating fluents from the start. The effect axioms of the Simplified Event Calculus are:

$$\text{HoldsAt}(f,t2) \leftarrow \text{Happens}(a,t1) \wedge \text{Initiates}(a,f,t1) \wedge t1 < t2 \wedge \text{not Clipped}(t1,f,t2) \quad (1)$$

$$\text{Clipped}(t1,f,t2) \leftarrow \text{Happens}(a,t) \wedge \text{Terminates}(a,f,t) \wedge t1 < t < t2 \quad (2)$$

$$\text{HoldsAt}(f,t) \leftarrow \text{Initially}(f) \wedge \text{not Clipped}(0,f,t) \quad (3)$$

Definition 1. An *event calculus program* is the conjunction of,

- A finite set S of Initially clauses of the form,
  $\text{Initially}(f)$
- A finite set A of Happens clauses of the form,
  $\text{Happens}(a,t)$
- A finite set E of Initiates clauses and a finite set of Terminates clauses of the form,
  $\text{Initiates}(a,f_1,t) \leftarrow \Pi$
  $\text{Terminates}(a,f_1,t) \leftarrow \Pi$

where Π does not mention the predicates Initially, Happens, Initiates or Terminates and every occurrence of the HoldsAt predicate is of the form

$$\text{HoldsAt}(f_2, t)$$

- The effect axioms of simplified event calculus
- A finite set of general clauses not mentioning the predicates Initially, Happens, Initiates, Terminates or $<$.

For further description of the Simplified Event Calculus, the reader is referred to Shanahan's work on the subject. [3]

3. A SIMPLE FILE SYSTEM MODEL

For the purpose of this paper, we define a simple file system: This file system contains files, and each file has an Accessed timestamp and a Modified timestamp. Files can be Read or Written. Reading a file causes the Accessed timestamp to be updated. Writing a file causes both the Accessed timestamp and the Modified timestamp to be updated. There is only one timestamp of each type for each file, so whenever a timestamp is changed, the previous value is lost.

Changes in the simple file system can now be represented as an event calculus program, where the fluents are file timestamps with an associated clock value, and actions are the operations that might change the timestamps. For the simple file system, the fluents can be called $\text{Accessed}(\text{file}, \tau_a)$ and $\text{Modified}(\text{file}, \tau_m)$. Further, actions can be represented with $\text{Read}(\text{file})$ and $\text{Write}(\text{file})$. The set E of Initiates and Terminates clauses will then contain clauses that Initiates timestamps with the value from current system clock c(t), and Terminates them:

$$\text{Initiates}(\text{Read}(\text{file}),\text{Accessed}(\text{file},c(t)),t) \quad (4)$$

$$\text{Initiates}(\text{Write}(\text{file}),\text{Accessed}(\text{file},c(t)),t) \quad (5)$$

$$\text{Initiates}(\text{Write}(\text{file}),\text{Modified}(\text{file},c(t)),t) \quad (6)$$

$$\text{Terminates}(\text{Read}(\text{file}),\text{Accessed}(\text{file},c(t_1)),t) \leftarrow t_1 < t \quad (7)$$

$$\text{Terminates}(\text{Write}(\text{file}),\text{Accessed}(\text{file},c(t_1)),t) \leftarrow t_1 < t \quad (8)$$

$$\text{Terminates}(\text{Write}(\text{file}),\text{Modified}(\text{file},c(t_1)),t) \leftarrow t_1 < t \quad (9)$$

In most real file systems there is always a value assigned to the time stams of a file. It therefore makes sense to define Initially clauses that initiates fluents for the timestamps, so they will hold from the start:

$$\text{Initially}(\text{Accessed}(\text{file},\tau_0)) \quad (10)$$

$$\text{Initially}(\text{Modified}(\text{file},\tau_0)) \quad (11)$$

In the simple file system model, S is the conjunction of formulae (10) - (11) and E is the conjunction of formulae (4) - (9). With a definition of a set A of Happens clauses, an event calculus program for this simple file system has been completed. Then, SLDNF resolutions can be utilized to search the space of possible event histories and test propositions about fluents at particular moments in time.

Example 1. Let a file be Read at $t = t_R$ and subsequently written at $t = t_W$, so that $t_R < t_W$. Let c(t) be an integer, so that $\tau_0 = 0$, $c(t_R) = 5$ and $c(t_W) = 10$. Let an event program be defined by (4) - (11) and the following clauses in A:

$$\text{Happens}(\text{Read}(\text{file}),t_R)$$

$$\text{Happens}(\text{Write}(\text{file}),t_W)$$

The timestamp fluents at certain moments in time can now be examined by means of SLDNF resolutions. For example, let us determine if the accessed time stamp at time $t = t_{Obs}$, $t_R < t_W < t_{Obs}$ has value 10.

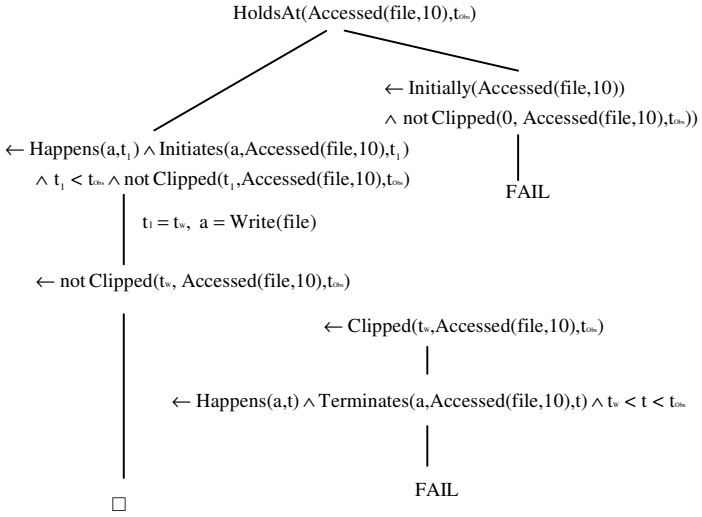

Figure 1. Resolution of HoldsAt(Accessed(file,10),t_{Obs})

Figure 1 shows a resolution for the observation of the Accessed time stamp given a specific observation time. The right hand branch of the resolution, representing the case that the time stamp was initially set to the observed value fails due to the fact that there is no Initially clause setting the Accessed time stamp to 10. The left hand branch of the resolution assumes that an action a happened at time t_1 initiating the fluent Accessed(file,10) at time t_1. The only Happens clause that can satisfy this is $\text{Happens}(\text{Write}(\text{file}),t_W)$. Since the evaluation of the clause $\text{Clipped}(t_W, \text{Accessed}(\text{file},10),t_{Obs})$ fails, the left branch succeeds and we can conclude that $\text{HoldsAt}(\text{Accessed}(\text{file},10),t_{Obs})$ holds.

4. OBSERVATION SETS

The example in the previous section showed how Simplified Event Calculus can be utilized to determine if specific timestamp values holds at a specific moment in time, given known occurrence of actions. This can be extended into taking the final state of the system into account.

Definition 2. Formulated in the Simplified Event Calculus, an *Observation Set O* is a finite set of HoldsAt clauses on the form $\text{HoldsAt}(f, t_{Obs})$

The *observation proposition* is the conjunction of the HoldsAt clauses in the observation set. The observation proposition has the form

$$p = \varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n$$

Where each φ is a HoldsAt clause contained in O, and n is the number of elements in O.

With the definition of an observation set, the relationship between an observation set and the sets S, E and A defining an event calculus program can be investigated. An event calculus program defines the behaviours occurring in a system in terms of the initial state (S), the effect any actions would have on the states (E) and

the actions that actually occurred (A). With known S, E, and A, possible states at a specific moment in time can be tested for consistency with the event calculus program. When S, E and A are known, SLDNF resolutions can be used to test observation propositions and therefore confirm or refute possible observation sets O. The observation set

$$O = \{\text{HoldsAt}(\text{Accessed}(\text{file},10),t_{Obs})\}$$

was in Example 1 determined to be a possible observation set for S, E and A. This shows how a possible observation set can be tested for consistency with an event calculus program.

Things are however different in an investigation situation. In an investigation, the state at the time of the investigation is observable, whereas information about occurred events is unknown. Under the assumption that the investigator has all information about the initial state S, and also thorough knowledge about the workings of the system, E, the investigator can use the knowledge about the observed state O to derive information about occurred events. In this case A is unknown, whereas S, E and O are known. The investigator can now infer knowledge about A from the observation set O and the detailed knowledge about how the system works, S and E.

Returning to Example 1, if the observed set is

$$O = \{\text{HoldsAt}(\text{Accessed}(\text{file},10),t_{Obs})\}$$

and A is unknown, the investigator can now reason that since (from O) the fluent Accessed(file, 10) holds at the time of the observation and since (from S) initially Accessed(file, 0), some action must have occurred that terminated Accessed(file, 0) and initiated Accessed(file, 10). From E, the investigator knows that this must have been an action occurring at $t = t_a$, where $c(t_a) = 10$. The investigator also knows that the action must have been either a Read or a Write action, since (again, from E) these are the only actions that can affect the Accessed fluent. The investigator can therefore formulate two hypotheses about occurred actions, H_1 and H_2 where $c(t_a) = 10$.

$$H_1 = \{\text{Happens}(\text{Read}(\text{file}),t_a)\}$$

$$H_2 = \{\text{Happens}(\text{Write}(\text{file}),t_a)\}$$

These hypotheses can be tested by SLDNF resolution of the observation proposition for both H_1 and H_2, and both hypotheses will be accepted. H_1 and H_2 are hypotheses about actions that actually took place. If hypotheses about occurred actions are accepted by an event program resolution, it means that they are possible explanations for the observed set O. The hypotheses do however, even if they are accepted, not imply full knowledge of the set of actions A. Even if only one hypothesis is accepted, it is still in the unknown if there were any actions in A for which there exist no evidence anymore. In Example 1, it could for example be the case that the file was Read at some moment prior to t_a. The timestamp fluent resulting from this Read would be Terminated by the Read occurring at t_a, and therefore not be observable at t_{Obs}.

5. ACTION HYPOTHESES

Definition 3. An action hypothesis H is a finite set of Happens clauses on the form $\text{Happens}(a,t)$ derived from an observation set O, given finite sets S and E in an event calculus program.

The acceptance of an action hypothesis means that it is a possible set of actions that can explain the observation set O. In order to be able to deduct possible courses of events from an observation set, we would like to find all possible hypotheses H, given an observation set O and knowledge about the system, represented by S and E.

From Definition 2 the elements of an observation set O are HoldsAt clauses representing the fluents that holds at the time of the observation. The observation proposition to be tested in the event calculus program is the conjunction of these HoldsAt clauses and takes the form

$$p = \varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n \qquad (12)$$

where each φ is a HoldsAt clause.

From the effect axioms of the Simplified Event Calculus, these HoldsAt clauses may exist either because they held initially (formula (3)) or because an action occurred that initiated them (formula (1)). There is no other way a HoldsAt clause can come to existence than through formulae (1) or (3). It is therefore possible to find all possible action hypotheses by reasoning on the observation proposition, the Initiates clauses in E and the Initially clauses in S. This reasoning does not have to consider termination of fluents as per the Terminates clause in E, since this will be done by means of SLDNF resolution when each hypothesis is tested for acceptance. The proposition that all fluents in an observation set has been initiated is the conjunction of the initiation of each fluent and takes the form:

$$q = \kappa_1 \wedge \kappa_2 \wedge \ldots \wedge \kappa_n \qquad (13)$$

where each κ is the initiation of the corresponding φ in the observation proposition p. In the following, this proposition will be called the *initiation proposition*.

A fluent may exist because it held initially or because it was Initiated by a clause in E. There may be more than one Initiates clause in E initiating one particular fluent, and these must all be considered. Written in propositional logic, the initiation of a HoldsAt clause takes the form of a disjunction:

$$\kappa_i = \alpha_{i1} \vee \alpha_{i2} \vee \ldots \vee \alpha_{im} \vee \eta_i \qquad (14)$$

Where κ_i is the i-th HoldsAt(f,t2) clause in q, η_i is an Initially(f) clause, m is the number of Initiates(a,f,t1) clauses affecting that fluent and each α_i is a clause on the form $\text{Happens}(a,t1)$ where there exists a clause Initiates(a,f,t1) in E.

The initiation of the fluents in the observation proposition can now be found by inserting (14) into (13), yielding

$$\begin{aligned} q = {} & (\alpha_{11} \vee \alpha_{12} \vee \ldots \vee \alpha_{1m} \vee \eta_1) \\ & \wedge\ (\alpha_{21} \vee \alpha_{22} \vee \ldots \vee \alpha_{2m} \vee \eta_2) \\ & \wedge\ \ldots \\ & \wedge\ (\alpha_{n1} \vee \alpha_{n2} \vee \ldots \vee \alpha_{nm} \vee \eta_n) \end{aligned}$$

q is a conjunction of disjunctive clauses. By reordering it into a disjunction of conjunctive clauses, a set of action hypotheses will be found, where each of the conjunctive clauses in the disjunction is an action hypothesis H.

Consider an event calculus program with S and E as previously defined and O as defined by the following observation proposition, where $c(t_m) \neq \tau_0$ and $c(t_a) \neq \tau_0$:

$$p = \text{HoldsAt}(\text{Modified}(\text{file},c(t_m)),t_{Obs}) \wedge \text{HoldsAt}(\text{Accessed}(\text{file},c(t_a)),t_{Obs}) \quad (15)$$

The initiation of these fluents can then be expressed as a conjunction of disjunctive clauses as follows:

$$q = (\text{Happens}(\text{Write}(\text{file}),t_m) \vee \text{Initially}(\text{Modified}(\text{file},c(t_m)))) \wedge (\text{Happens}(\text{Read}(\text{file}),t_a) \vee \text{Happens}(\text{Write}(\text{file}),t_a) \vee \text{Initially}(\text{Accessed}(\text{file},c(t_a))))$$

Since there is no Initially(Modified(file,c(t_m))) or Initially(Accessed(file,c(t_a))) in S, we know that these clauses are false. q then becomes:

$$q = \text{Happens}(\text{Write}(\text{file}),t_m) \wedge (\text{Happens}(\text{Read}(\text{file}),t_a) \vee \text{Happens}(\text{Write}(\text{file}),t_a))$$

Rewritten as a disjunction of conjunctive clauses:

$$q = \text{Happens}(\text{Write}(\text{file}),t_m) \wedge \text{Happens}(\text{Read}(\text{file}),t_a) \vee \text{Happens}(\text{Write}(\text{file}),t_m) \wedge \text{Happens}(\text{Write}(\text{file}),t_a)$$

So here we obtain two different hypotheses from the fluent initiation:

$$H_1 = \{\text{Happens}(\text{Write}(\text{file}),t_m), \text{Happens}(\text{Read}(\text{file}),t_a)\}$$
$$H_2 = \{\text{Happens}(\text{Write}(\text{file}),t_m), \text{Happens}(\text{Write}(\text{file}),t_a)\}$$

6. DERIVING INVARIANTS

The described methods can be used to test the observation proposition in the general case, and thereby determine properties of the simple file system defined in formulae (4) - (11). The general observation proposition for a file in the simple file system was expressed in (15). Now, if $c(t_m) \neq \tau_0$ and $c(t_a) \neq \tau_0$, there must have occurred actions initiating these fluents. As previously determined, these actions must have been

$$H_1 = \{\text{Happens}(\text{Write}(\text{file}),t_m), \text{Happens}(\text{Read}(\text{file}),t_a)\}$$
$$H_2 = \{\text{Happens}(\text{Write}(\text{file}),t_m), \text{Happens}(\text{Write}(\text{file}),t_a)\} \quad (16)$$

Now, by investigating the three different cases; $t_m < t_a$, $t_m = t_a$ and $t_m > t_a$, properties of this system can be found.

In the case of $t_m = t_a$, (16) is reduced to

$$H_1 = \{\text{Happens}(\text{Write}(\text{file}),t_m), \text{Happens}(\text{Read}(\text{file}),t_m)\}$$
$$H_2 = \{\text{Happens}(\text{Write}(\text{file}),t_m)\} \quad (17)$$

Written as a disjunction

$$(\text{Happens}(\text{Write}(\text{file}),t_m) \wedge \text{Happens}(\text{Read}(\text{file}),t_m)) \vee \text{Happens}(\text{Write}(\text{file}),t_m)$$

Which is equivalent to

$$\text{Happens}(\text{Write}(\text{file}),t_m)$$

Thus the only hypothesis is,

$$H_1 = \{\text{Happens}(\text{Write}(\text{file}),t_m)\}$$

The case of $t_m < t_a$ must be investigated further. The resolution in Figure 6.1 shows that H_2 is refuted if $t_m < t_a$. The resolution in Figure 6.2 shows that HoldsAt(Modified(file, c(t_m)), t_{Obs}) in H_1 is accepted for $t_m < t_a$. The resolution for HoldsAt(Accessed(file, c(t_a)), t_{Obs}) would look exactly like the one shown in Figure 6.2 and is omitted here. From these resolutions, it can be concluded that only H_1 is accepted.

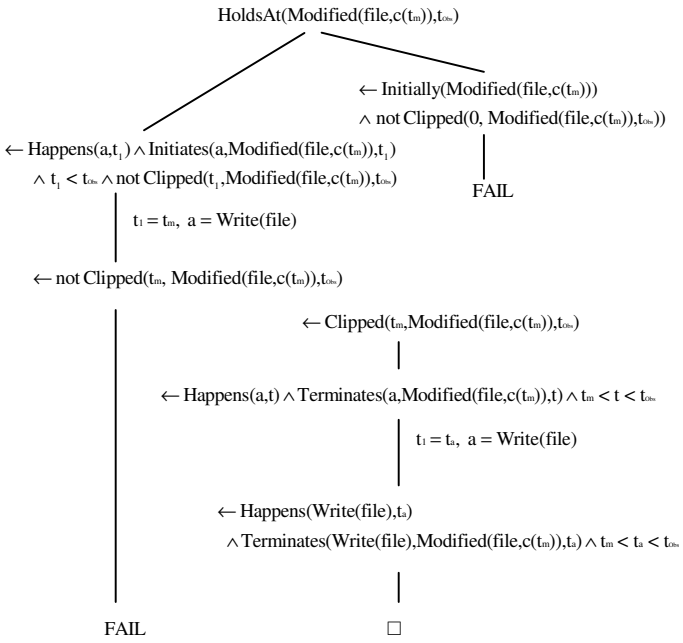

Figure 6.1 – Proposition fails for H_2 when $t_m < t_a$

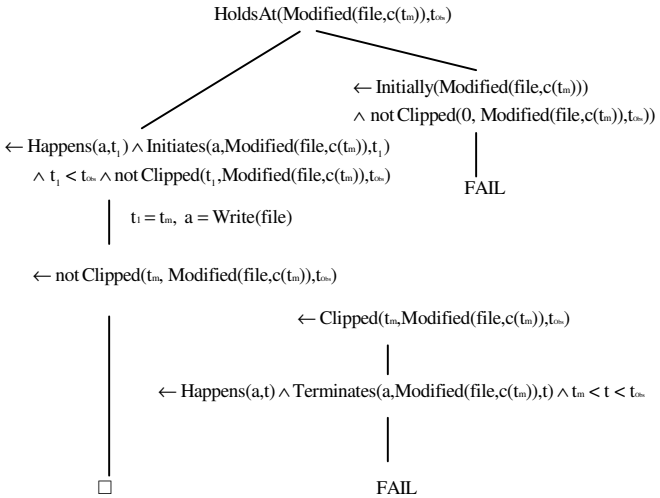

Figure 6.2 – Proposition does not fail for H_1 when $t_m < t_a$

In the case of $t_a < t_m$, H_1 is refuted, as shown in Figure 6.3. A resolution for H_2 would look exactly like the resolution in Figure 6.3, with Accessed and Read replaced with Modified and Write.

Thus, H_2 is also refuted for $t_a < t_m$, showing that in the simple file system, $t_a < t_m$ cannot occur.

It has then been shown that for observations of every file in the simple file system:

$$t_m \not> t_a$$

$$t_m = t_a \Rightarrow \text{Happens}(\text{Write}(\text{file}),t_m)$$

$$t_m < t_a \Rightarrow \text{Happens}(\text{Read}(\text{file}),t_a) \wedge \text{Happens}(\text{Write}(\text{file}),t_m)$$

These results would facilitate event reconstruction in a forensic investigation of the simple file system, since the sequence of events on can now be determined directly from the file timestamp configuration.

The requirements on timestamp evidence can be explicitly stated as:

$$t_m \leq t_a$$

This result is also interesting, since it impose restrictions on the formulation of a hypothesis for the system clock. Any occurrences of $c(t_m) > c(t_a)$ in the observation set must necessarily mean that the clock has been adjusted backwards. If many files are available in the observation set, it can also be determined when the clock must have been adjusted.

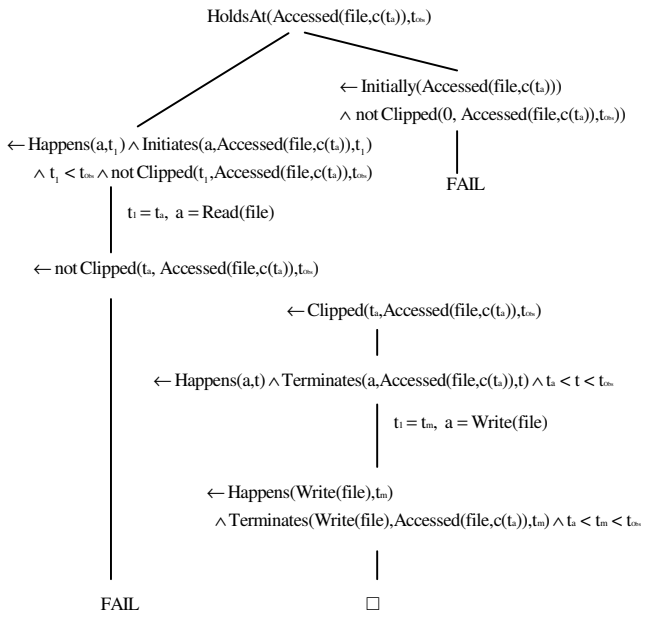

Figure 6.3 – Resolution of H_1 when $t_a < t_m$.

7. RESULTS

In the previous sections, a simple file system model was defined in Simplified Event Calculus, and its properties were examined by SLDNF resolutions. When the set of actions A was known, the resolutions could be used to determine if a set of observations was consistent with the model. When the set of action A was unknown but a set of observations O was known, the model was used to identify hypotheses of possible actions that could have occurred and test if these were consistent with the observation set O.

Simplified Event Calculus can be used to identify hypotheses of possible actions and test them for consistency with an observation set for an investigated system, as long as it is possible to determine how the system works (expressed by clauses in E) and the initial state (expressed by the clauses in S). This method can be used in digital investigations to identify hypotheses of which actions occurred on a system, and test them for consistency with the available evidence.

In Section 6, Simplified Event Calculus was used to derive invariants that must always hold for the simple file system. The method shown can be used for any system with known function and initial state to derive invariants. In digital investigation, these invariants can be used in tools that checks evidence from these specific systems for consistency. Specifically, in systems where timestamps are part of the model, the invariants can be used to determine adjustments of the system clock the timestamps are generated from.

8. CONCLUDING REMARKS

Simplified Event Calculus is a reasonable tool for building a model of a system and determining its properties in a hypothesis-based approach to digital investigation. By building such a model, the investigator can find and test possible hypotheses about actions that occurred on the system, and derive invariants for the system.

The approach in this work has been purely theoretical. Whether the Simplified Event Calculus can be used to model a real system under investigation, or if it is practical to do so, is an area of further study. In order to provide a full model of a real system in Simplified Event Calculus one must understand the system in full, something that can probably only be accomplished by studying the implementation details of the system. It might however be reasonable to construct a partial model only by studying the effects of operations on the real system, if it can be justified that the actions included in the model are the only actions of relevance in the investigation.

9. REFERENCES

[1] B. Carrier, A hypothesis-based approach to digital forensic investigations. *CERIAS Tech Report 2006-06*, Purdue University, 2006

[2] P. Gladyshev and A. Patel, Finite State Machine Approach to Digital Event Reconstruction, *Digital Investigation,* vol. 1, p. 19, 2004.

[3] M. Shanahan, *Solving the frame problem : a mathematical investigation of the common sense law of inertia*. Cambridge, Mass.: MIT Press, 1997.

VITA

Svein Yngvar Willassen received the siv.ing degree in Telematics at the Norwegian University of Science and Technology in 1998. From 1999 to 2002, Willassen was a Special Investigator at the Norwegian National Computer Crime Unit. From 2002 to 2005 he was the Computer Forensics Manager at Ibas AS. In these years, Willassen performed a large number of digital investigations in criminal investigations as well as in the private sector. He also participated in international work such as the authoring of the Interpol Computer Crime Manual in the Interpol European Working Party on IT Crime and the production of guidelines on Digital Evidence analysis in the International Organization on Computer Evidence.

Willassen's research interests are within the areas of computer crime and digital forensic investigation, in particular investigation of mobile phones. He is the author of the forensic tool SIMCon, and has served as expert witness in a significant number of cases involving digital investigations.