REASONING ABOUT HISTORY BASED ACCESS CONTROL POLICY USING PAST TIME OPERATORS OF INTERVAL TEMPORAL LOGIC

PhD Thesis

Sami Alsarhani

Software Technology Research Laboratory
Faculty of Technology
De Montfort University
England

Supervisors:

Dr Antonio Cau
Dr Francois Siewe

This thesis is submitted in partial fulfillment of the requirements for the Doctor of Philosophy.

July 17, 2014
Draft: v0.4

Abstract

Interval Temporal Logic (ITL) is a flexible notation for the propositional and first-order logical reasoning about periods of time that exist in specifications of hardware and software systems. ITL is different from other temporal logics since it can deal with both sequential and parallel composition and provides powerful and extensible specification and verification methods for reasoning about properties such as safety, time projection and liveness. Most imperative programming constructs can be seen as ITL formula that form the basis of an executable framework called Tempura that is used for the development and testing of ITL specifications.

ITL has only future operators, but the use of past operators make specifications referring to history more succinct; that is, there are classes of properties that can be expressed by means of much shorter formulas. What is more, statements are easier to express (simplicity) when past operators are included. Moreover, using past operators does not increase the complexity of interval temporal logic regarding the formula size and the simplicity. This thesis introduces past time of interval temporal logic (ITL^p) where, instead of future time operators Chop, Chopstar, and Skip, we have past operators past Chop, past Chopstar and past Skip. The syntax and semantics of past time ITL are given together with its axiom and proof system. Furthermore, Security Analysis Toolkit for Agents (SANTA) operators such always-followed-by and the strong version of it has been given history based semantics using past time operators. In order to evaluate past time interval temporal logic, the problem of specification, verification of history based access control policies has been selected. This problem has already been solved using future time of interval temporal logic ITL but the drawback is that policy rules are not succinct and simple. However, the use of past time operators of ITL produces simple and succinct policy rules. The verification technique used to proof the safety property of history based access control policies is adapted for past time ITL to show that past time operators

of interval temporal logic can specify and verify a security scenario such as history based access control policy.

Declaration

I declare that the work described in my thesis is original work undertaken by me between January 2009 to December 2013, for the degree of Doctor of Philosophy, at De Montfort University, United Kingdom.
This thesis is written by me and produced using LATEX.

Acknowledgement

I wish to sincerely acknowledge the help, motivation and encouragement provided to me by all my supervisors. In particular, I wish to thank Dr Antonio Cau, my first supervisor, with whom I had regular contact and interesting discussions. I also wish to acknowledge the help from my second supervisor Dr Francois Siewe, with whom I had many useful discussions. Last but not least, I wish to acknowledge the support I received from Professor Hussein Zedan, my brother, with whom I had interesting discussions as well as a lot of encouragement and support.

Contents

List of Tables

List of Figures

Chapter 1

INTRODUCTION

This Chapter introduces the following:

- *Background.*
- *Problem statement.*
- *Motivation and Research Objectives.*
- *Research Questions.*
- *Research Approach.*
- *Scope of the research.*
- *Success Criteria.*
- *Thesis Outline.*

1.1 Background

According to Ambrose Bierce [50], "Logic is the art of thinking and reasoning in strict accordance with the limitations and incapacities of human misunderstanding". When we study ordinary logic, it can be seen from two viewpoints. One of these is "where logical systems are seen as being self-contained and not meant to model any real-world objects or actions" [50].

These systems have typically been studied by philosophers and logicians when they are trying to explore and extend various notions of truth, and provability.
We tend to follow the alternative view of logic, which is "a formalization of some aspect of the real world". In this view, some abstraction of objects or actions that either do or do not occur in real life are being described or modelled logically. However, we can use another language to describe such aspects: for example the English language, which has been widely used to describe and reason about problems and situations. Nonetheless, the disadvantages of the natural language - which include ambiguities and inconsistencies - must be overcome.
For this reason, a formal language with well-defined semantics, a consistent reasoning mechanism, and logical systems are required to accurately describe these objects and actions. However, as more complex or detailed properties are represented, a deeper logical system must be used as that logic represents a particular abstraction of the real world [50].

1.1.1 What is Temporal Logic

In classical propositional logic, formula are evaluated within a single fixed world (or state), which generally supports the reasoning with propositions; that is, with statements to be evaluated as either true or false. However, temporal logic (TL) is the logic of studied reasoning that focuses on propositions whose truth values depend on time [120, 46]. Temporal logics, which can be seen as an extension of classical logic in its simplest form [46], was originally developed in order to represent tense in natural languages [116].

Another definition of temporal logic is given by Manna et.al. [86], which describes it as a special branch of logic that deals with the development of situations in time. Whereas classical logic is adequate for describing a static situation, temporal logic gives us the possibility of discussing how a situation changes with the

passage of time.
In addition, to the timeless logic operators, temporal logic contains additional operators such as ($\bigcirc$), which means in the next moment of time, ($\Box$), which means at every future moment of time, and ($\Diamond$), which means at some future moment. These additional operators allow us to establish formulas that cannot be established with classical logical operators; for example:

$$\Box(try\_to\_print \Rightarrow \bigcirc(printed \vee try\_to\_print))$$

this formula means:
" whenever we try to print a document then, at the next moment in time, either the document will be printed or we try again to print it " [48].
Despite the fact that classical logics do not include a time element, temporal logics characterize changes which depend on time and this makes temporal logics a richer notation than classical logics.

1.1.2 Why temporal logic?

Formal languages, and their well-defined semantics, are increasingly used to describe systems behaviour precisely, clearly and unambiguously [48].
For example, it is important to verify that a system behaves as required. These requirements can be captured as a formal specification in an appropriately chosen formal language with the selected specification providing the basis for formal verification. Formal verification provides a comprehensive approach to potentially establishing the correctness of the system in every possible situation. Alternatively, we may want to use the logical specification of a system in other ways such as treating it as a program and directly executing it and again the well-developed logical machinery helps us with this [48].
Temporal logic (TL) within computer science plays a significant role in a number of areas, particularly the formal specification and verification of concurrent and distributed systems [111].
Temporal logic has achieved much of its popularity because a number of useful properties - such as: "safety", which ensures something bad never happens to the system; "liveness", which asserts that something good will eventually happens to the system; as well as "weak fairness", which states that any transition that is continuously enabled eventually happens and "strong fairness", which states that any transition that is enabled infinitely will often occur eventually [48] - can be formally

and concisely specified.

1.1.3 Why past temporal logic?

In computer science, most theoretical studies of temporal logic only use future time constructs. This contrasts with the temporal logics studied in linguistics, philosophy and other areas where past time and future time have been on an equal footing [116]. This is a surprising fact as computer scientists recognize that past time constructs can be very useful when it comes to expressing certain properties. For example, using "$\square$" for "at all future moments" and "$\widehat{\Diamond}$" for "at some past moment". We can easily state that "in all cases the occurrence of a problem must have been preceded by a cause"; namely, "no problem will ever occur without a cause", which is an important safety property under some forms. One simply writes:

$$\square(problem \supset \widehat{\Diamond}\ cause) \tag{1.1}$$

However, it has been shown that formulas using past time constructs can often be replaced by equivalent pure future formulas [53, 83]. For example, formula (1.1) is equivalent to formula (1.2):

$$\neg(\neg\ cause\ U\ problem) \tag{1.2}$$

which uses the "Until" construct U for "the problem holds at the current or a future position, and the cause has to hold until that position and after that position's cause does not have to hold any more".
These two formulas (1.2) and (1.1) are formally stated as being the same [80]; however, the second formula (1.2) is more intricate than the first one (1.1). This is even more obvious when one tries to express a statement like "in all cases the occurrence of a problem must have been preceded by cause1 and cause1 must have been preceded by a cause2" as has been shown in (1.3):

$$\square(problem \supset \widehat{\Diamond}\ (cause1\ \wedge\ \widehat{\Diamond}\ cause2) \tag{1.3}$$

Formula (1.3) is very complicated when ignoring the past construct as shown in (1.4):

$$\neg(\neg(\neg cause_2\ U\ cause_1)\ U\ problem) \tag{1.4}$$

Clearly, a formulation like (1.3) is much easier to understand than the more intricate one (1.4).
The practical consideration has a formal counterpart as linear temporal logics with past time operators are very useful in practice as shown in the given examples. However, it has been agreed that in various relevant cases such usefulness does not involve any increase in expressiveness to linear temporal logic, that is, all the formulations which can be expressed using past time operators can be rewritten using future pure ones [79, 93, 90].
This research is looking at the possible advantages of using the past time operators of interval temporal logic to reason about history based access control policy. History-based policies are a very expressive (not easy to express) class of policies that can define policy decisions dependent on previously observed behaviours [69]. This class of policies consists of phases, each phase consists of a sequence of states and ITL with the introduced past operators (; and * specifically) is a convinced language to reason about it. Next, we will give an overview of history based access control policies.

1.1.4 Background of history based access control policy

During the last decade, access control models have been developed in order to protect important resources from unwanted access. Historically, access control mechanisms have been rigid; whilst they may be either dynamic, or static, (dependent upon a particular access or environment), they either deny or grant access consistently. This is necessary to ensure that the systems' security is well defined and that it satisfies given specifications [109].
Several models for access control have been proposed including stack inspection, adopted by Java and $C\#$. In this model, a policy grants static access rights to the code, while actual runtime rights depend on the static rights of the code frames on the call stack. As access controls only rely on the current call stack, stack inspection suffers from two main weaknesses. First, it is difficult to place the needed checks at the relevant points in the code, and even more difficult to guarantee that they suffice for enforcing the intended security policy. Second, stack inspection may fail to enforce some security constraints, because it relies on the call stack only. Indeed, the access rights of a certain method are no longer affected by the execution of an untrusted one, after it has been popped from the call stack [51].
Conversely, as Abadi and Fournet note, stack inspection is not useful in providing

protection to the caller. Thus, if the trusted code calls the untrusted code and proceeds with the results returned by the latter using the same permissions it has used for the call, undesired results may occur: in proceeding with the results returned by the untrusted code, stack inspection forgets that security may depend on how,(i.e., with which permissions) the results were computed in the first place. This is because the stack frame containing the permissions is popped upon return so that the permissions are no longer available on the stack.
Abadi and Fournet have introduced a novel access control mechanism called History Based Access Control (HBAC) which records the history of previously executed codes [1]. Therefore, each and every code executed before a security-sensitive operation must be sufficiently authorized to execute that operation.
History based access control considers instead a suitable abstraction of the whole execution, and the actual rights of the running code depend on the static rights of all the pieces of code (possibly partially) executed so far.
At runtime, both stack inspection and the history based mechanism involve a set of currently enabled permissions; it is a subset of the statically authorized permissions of the class of the currently-executing code. When a method is invoked, the initial permission set for the method body is the intersection of the caller's current set and the static permissions of the called code. Note that it is the class of the dynamically dispatched code that matters, not the class of the target object. In stack inspection, the method call has no effect on the current permission set of the caller. In the history based mechanism, the caller's permissions become the intersection of their initial value with the final permissions of the called method.
The typical runtime mechanisms for enforcing history-based policies are reference (execution) monitors [127]. Usually, policies constrain the behaviour of the reference monitors in the information system. More precisely, access control policies determine the choice of the reference monitor to permit or deny the execution of a request. A complete specification of the reference monitor can be given in the form of an access control matrix that fully determines the access rights at any point in time during the system execution. This matrix will depend not only on the current state of the information system, but also on the history of execution [69].

1.2 Problem statement and research motivation

Reasoning about histories is gaining an increased importance in several areas; for example, history based access control policies and log file analysis among others.

The future time operators of interval temporal logic can be used to reason about histories [69]. The normal approach to reason about history with future time operators is to go to the final state of the interval and refer from this final state to the prefix states as history.
The final state in the future interval is $\sigma_{|\sigma|}$ and to reach this state you need a temporal operator such as fin. This operator means the final state of this interval and from this final state we refer to the remaining states which are $\sigma_4, \sigma_3, \sigma_2, \sigma_1, \sigma_0$, as history states as is shown in Figure 1.1.

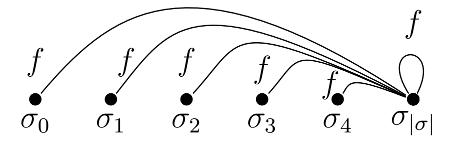

Figure 1.1: Future ITL expressing past

However, when the introduced past time operators of interval temporal logic are used to reason about history, τ_0 is the current state in the past interval, which is the final state of the future interval, and from this state we can refer to the past interval which are states $\tau_1, \tau_2, \tau_3, \tau_4, \tau_{|\tau|}$ so, we do not need the additional operators to reach the final state such as fin as is shown in Figure 1.2.

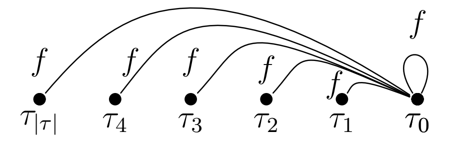

Figure 1.2: Past ITL expressing past

In addition, the "numbering of states" is changed when the past operators are used. The normal approach of numbering the states in the future time and the past time is from the left state to the right state, $\sigma_0 \cdots i \cdots \sigma_{|\sigma|}$ where i is the current state in this interval.
However, on our approach the numbering of states is from the right state to the left one as it is shown in the Figure from the state τ_0 to states $\tau_1, \tau_2, \tau_3, \tau_4, \tau_{|\tau|}$ respectively.

In this research, we are introducing the past time operators of interval temporal logic to reason about history based access control policy and compare it with the future time operators. Depending on the discussion above, the introduced past operators change the "numbering of states" and reduce the symbols used such as the temporal operator fin. The introduced past time operators of interval temporal logic give us two main advantages:
In particular, the change of state numbering makes many past statements easier to express (simplicity). This is clear because in the introduced "numbering of states" the current state of the past interval τ_0 is the final state of future interval; moreover, we do not need an additional operator to reach the final state of the future interval and this reduces the symbols used.
The second advantage is that reasoning about history with past time operators is more succinct; that is, there are classes of properties which can be expressed by means of much shorter formulas and less symbols (fin operator in the given example). When succinctness is achieved, simplicity is achieved; that is, the formula is easier to express due to the formula being shorter and containing less symbols. This study is therefore focusing on the benefits of introducing past time operators to investigate if they resolve the problems identified.

1.3 Research Questions

The first research question, RQ1, is rather global. Answering this question results in a broad understanding of the aims and results of our research.
RQ1: What are the benefits of the past time operators for interval temporal logic?
RQ2: What are the benefits of the introduced "numbering of states" with past time to interval temporal logic ITL?
RQ3: Can past time operators be used to reason about and express history based access control policies?
RQ4: Does the past time operators making the specification and verification process more succinct and thus easier to express (simpler)?

1.4 Research Approach

The adopted research methodology follows the constructive research approach [32], which refers to the contribution to knowledge being developed as a new solution for an identified problem.
A formal framework was developed based on past-time ITL for a known problem which is the formal specification and verification of history-based access control systems. The methodology of the proposed approach consists of the following six steps:

- Step 1: Literature review :
  The research study starts with a critical review of published works on the following:
  - Firstly, the specification in temporal logic.
  - Secondly, temporal logic history including the classification of temporal logic.
  - Finally, the history-based access control Section.
- Step 2: Past time interval temporal logic (ITL^p):
  This step introduces the following:
  - Interval temporal logic (ITL), its syntax and semantics.
  - Past time interval temporal logic ITL^p, its syntax and semantics.
  - The propositional axioms and rules of interval temporal logic and past time interval temporal logic. These axioms and rules have been proven sound in appendix A.
- Step 3: Past time interval temporal logic (ITL^p) to reason about history-based access control policies:
  In this step:
  - Justification of our choice of past time interval temporal logic.
  - Description of the computational model and its components.
  - SANTA Policy language with future time.
  - SANTA Policy language with the history based semantics using past-time operators of ITL.

  - The verification rules.

- Step 4: The scenario:
  In this step, in order to evaluate our past time interval temporal logic ITL^p for reasoning about histories.
  - Specification of history based access control policies.
  - Verifications properties of histories.

- Step 5: Conclusion and future work:
  This step draws conclusions about introducing ITL^p to reason about history-based access control policy. Limitations and potential further research are discussed as well as the future impact of this study.

1.5 Scope

This thesis introduces past time operators of ITL. The syntax and semantics of past time operators of ITL are given together with its axiom and proof system. In order to evaluate past time operators of interval temporal logic, the problem of specification and verification of history based access control policies has been selected. The reason why we chose history based access control policies are that they are a class of policies that can define policy decisions dependent on previously observed behaviours within the system. The past time operators with their history semantics will be used to specify these previously observed behaviours.
The verification technique used to proof the safety property of history based access control policies is adapted for past time operators of ITL to show that past time operators of interval temporal logic can specify and verify a security scenario of history based access control policy such as GPS.

1.6 Success Criteria

In order to measure the success of our research, the following success criteria have been formulated:

- The past time operators of interval temporal logic is suitable for reasoning about and expressing history based access control policies.
- The change of numbering of states with past time operators of interval temporal logic make the reasoning about history easier.
- The formal specification and verification of history based access control policies when using past time operators of ITL is more succinct and thus easier to express (simpler).

1.7 Thesis Outline

The PhD thesis outline is as follows:

- Chapter two is the literature review and in the first Section the specification has been defined, what it is, its advantages and disadvantages. In the second Section, an overview of temporal logic is given in detail including how we can classify temporal logic systems and what temporal logic applications are in general and in computer science in particular.
  In the third Section, which is history based access control, the access control elements, types and categories have been explained. The models used for history based access control policy are listed to justify our choice of SANTA.
- In Chapter three, the interval temporal logic ITL operators and their syntax and semantics have been explained with the derived formula and constructs. Then the past time interval temporal logic operators have been proposed with their syntax and semantics with the derived formula of past time operators. The axioms and rules in logic are explained, then the propositional axioms and rules for ITL and for ITL^p are listed here. These axioms and rules have been proven sound in Appendix A.

- The fourth Chapter shows how to use ITL^p to reason about history based access control policies. The logic languages and its ability to reason about history based access control policy is discussed to justify our choice of past time interval temporal logic ITL^p. Then, SANTA and SANTA with past operators which is given a history based semantics using the past operators, are used to give the syntax and semantics of policy rules. Additionally, the formal syntax and semantics of policies and compound policies are explained in this Chapter. A list of verification rules which are used to verify that a policy satisfies a certain property is given.

- In the fifth Chapter, to evaluate our work, a General Practice System GPS scenario is described. An introduction has been given to this scenario, then a description of General Practice System GPS including the system subjects, objects and actions. The specification of policies has been listed. The semantics of GPS policies has been described in detail, and has been used with the proof rules to verify the safety property of GPS policies.

- In Chapter six we will bring to a conclusion the work done in this thesis and discuss the weaknesses and limitations of this work. Moreover, further research needed in this area is proposed and the future impact is discussed; recommendations have been given to help any researcher who wants to work in this field.

Chapter 2

LITERATURE REVIEW

In this Chapter:

- *Specification.*
- *Temporal logic history.*
- *History based access control.*

2.1 Introduction

This Chapter gives the background to interval temporal logic and its uses. It discusses the specification and its uses and the most relevant terms related to specification, such as “formal specification” and “formal specification approaches”.
In the second Section temporal logic history, and how we can classify temporal logic, as well as the temporal logic applications, has been discussed.
In the last Section of the literature review, the access control has been defined, its elements, types and categories. Finally, the languages that are suitable to express history based access control policies have been listed and discussed.

2.2 Specification

Specification is a vital activity of the software engineering process since it provides a conceptual model of the system if there is a description of the required behaviour of a software system. But the question raised here is what we mean by specification. What is the difference between requirements and specifications?. To answer this question we should define requirements and specifications.
There is a distinct difference between requirements and specifications. While requirement is a condition needed by a user to solve a problem or achieve an objective, a specification is a document that specifies the requirements, design and behaviour in a verifiable, complete and precise manner [132].
Another term that is commonly used in books and papers is requirements specification. That is a document that specifies the requirements for a system or component. It includes functional requirements, performance requirements, interface requirements, design requirements, and development standards. Therefore, the requirements specification is simply the requirements written down on paper.
So, the first step toward developing accurate and complete specifications is to establish correct requirements. This is not an easy task and is more of an art than a science.
Requirements and specifications are very important components in the development of any system. Requirements analysis is the first step in the system design process, where a user’s requirements should be clarified and documented to generate the corresponding specifications. While it is a common tendency for designers to be anxious about starting the design and implementation, discussing requirements with the customer is a vital activity in the construction of any systems. For ex-

ample, errors developed during the requirements and specifications stage may lead to errors in the design stage. When an error is discovered, the engineers must revisit the requirements and specifications to fix the problem. This leads not only to wasted time but also the possibility of other requirements and specifications errors. For these reasons, software requirements are defined in functional terms, and refined and updated as the project progresses to the different stages of its life cycle. Correct, accurate and complete documentation and understanding of software requirements are the most important factors in the success of meeting the required goals and achieving the functional validation of the produced software [117].
So, the relation between requirements and specification is that the specification is a formal documented form of requirements. This document may include, but is not limited to, drawings, patterns, and an itemized description of the system. The specification document can be checked for conformity with the set of requirements at any stage of the software development. Written specifications can be of several types such as system requirements specification, software requirements specifications, and software design specifications, etc. These specifications are the design outputs, and set the criteria to evaluate and verify the system's behaviours as per given requirements [117].

There are three uses of software specification:

1. Statement of user needs: this means that it captures information about the problem; it does not propose or promise any particular solution.
2. Statements of the implementation requirements: they are also used during the verification activity to check if the implementation complies with them.
3. Reference during product maintenance where implementations are modified and consequently their compliance with specifications must be checked.

2.2.1 Formal specification

In the past, specifications may have been written in natural language or informal language. Because of that, producing formal specifications was not part of common software engineering practice [71]. Software developers were not usually familiar with using formal specifications languages, and training in using these languages was both time consuming and expensive [138]. However, today the specifications

are written in formal specification languages such as temporal logic, so we are translating a non mathematical description, such as English and diagrams, into formal specification language [140]. What is more, the formal specification, which uses mathematical notation, is used precisely to describes the functionality, structure and interfaces of software systems. This process does not include the programming languages details needed to produce an implementation [132].
The reason behind this is that the system developer works at a higher level of abstraction than the programmer, so, they have the chance to define system functionality concisely without worrying about other aspects of implementation that they have nothing to do with, such as the functional behaviour of the system, algorithms, efficiency and memory management [138, 132].
This abstraction decreases the specification error rate and removes the confusion that such details bring to the specification reader, and allows him to recognize the defined functionality. This permits the verification of implementation [138, 132].
A formal specification provides a dependable point of reference for researchers who want to study the customer's needs, those who execute the programs in order to ensure that the needs are met, those who evaluate the outcome of the execution, and those who write instruction manuals for the system. Formal specification of a system can be concluded in the early stages of program development, since it is not dependent on the program code. This formal specification has to be modified as the design progresses and the designers better understand the customer's needs. But it is a powerful tool creating a mutual understanding among all parties involved in the system.
According to Gehani [55], formal specifications are used for several reasons which are :

- Ambiguities, omissions and contradictions can be found in the informal formulation of the problem throughout the formalisation process.
- The formal model can be proven correct with mathematical methods.
- Analysis can be made to a formally specified system to have or not to have wanted properties.
- A formal specified system can be embedded within a larger system with more confidence.

- The formal model (partly) leads to automated development methods and tools like simulations.
- Formally specified systems' designs can be easily compared with each other.

Formal specification approaches:

To write detailed formal specifications for any software systems, five basic approaches have been used, these are:

- Algebraic approach:
  This approach emerged in the mid-70s as a technique to deal with data structures in an implementation-independent manner. In this approach, we are giving an implicit definition of operations by relating the behaviour of different operations without defining state, but no explicit representation of concurrency. An example of this approach is OBJ language [56] and PLUSS. In this sense, equational logic [133], a branch of first-order logic, constitutes that part which deals exclusively with sentences in the form of identities chosen as the specification formalism and universal algebra and category theory provided the underlying semantical techniques [40].
- Model-based approach:
  In this approach, we build the system model using familiar mathematical constructs such as sets and sequences. The system operations have been defined as modifications of the systems state [132]. Unlike algebraic specification, the state of the system is not hidden and the state changes are straightforward to define, but again there is no explicit representation of concurrency; this is the approach most widely used by Z notation [35] and Vienna Development Method (VDM) [81].
- Process Algebraic approach:
  This approach is an explicit model of concurrent processes and representing behaviour by means of constraints on allowable observable communication between the processes (e.g. $\pi - Calculus$ [92, 135] and Calculus of Communicating Systems (CCS) [91]).
- Logic-based approach:
  Here, we are describing properties of systems, including low-level specification

of program behaviour and specification of system timing behaviour. An example of this approach is Temporal and Interval Temporal Logics [18, 23, 88].

- Net-based approach:
  In this approach, we are giving an implicit concurrent model of the system in terms of (causal) data flow through a network, including the representing conditions under which data can flow from one node in the net to another. An example of this approach is Petri nets, and predicate transition nets [135, 21].

2.3 Temporal Logic (TL)

Temporal logic has become one of the most important formalisms for specifying, verifying and reasoning about systems that interact with their environment [45]. The formal language with its proof theory, decision algorithms and associated methods of practical application, has found many uses in dealing with programs [65].
Temporal logic is considered to be a very suitable formal method for specifying and verifying concurrent and reactive systems [41, 89, 128]. By 'temporal logic' we mean " a family of logics and logical techniques which can be applied to a wide range of problems, both abstract and concrete " [134]. Temporal logic formulas can describe sequences of state changes and properties of behaviours, and, hence, can span a wide range of problems in various fields with a richer notation [74].
As temporal languages are increasingly employed to cover a variety of uses, as mentioned above, there is growing interest to include the use of past operators to the temporal logic languages [80, 79, 93, 90, 37, 49, 57].
In the next Section, we will give an overview of temporal logic starting from the models of time.

2.3.1 Time in temporal logic

Time has been studied in disciplines such as physics, philosophy and computer science. It has been one of the most paradoxical concepts of philosophy throughout history [128, 74]. The concept of time has been studied in order to introduce a satisfactory definition of time since there is no common understanding of time that has been given till now. The main reason is that each definition has covered some aspects of time whilst excluding others. The time concept has been studied in various disciplines in order to introduce a common language for time.
In many science applications such as physics, mathematics and first order predicate calculus, which is used to reason about expressions containing the time variable, time has been represented as another variable. Therefore, there is apparently no need for a special temporal logic [128, 74].
In philosophy, temporal logic has been an important subject, as some of the ancient philosophers used some form of temporal logic to analyse the structure of time. Plato defined it as the 'moving image of eternity' while Aristotle described it as 'the number of motion with respect to earlier and later' [139]. Philosophers found it useful to introduce special temporal operators for the analysis of temporal connec-

tives in languages. The verbs 'incipit' (it begins) and 'desinit' (it ends) are found in Aristotle's Physics books 6 and 8[105]. These new operators were soon seen as potentially valuable in analysing the structure of time [89].

Classical logic deals with timeless propositions, so logic formulas can characterize only static states and properties. Temporal propositions typically contain some reference to time conditions, so temporal logic formulas can be used to describe sequences of state changes and properties of behaviours. Therefore, temporal logic can cover a wide range of problems in different fields and areas with richer notations [74].
The various temporal logics can be used to reason about qualitative temporal properties:

- Safety: nothing bad happens to the system.
- Liveness: something good eventually happens to the system.
- Fairness: something good happens fairly.

Depending on the view of time (whether time is linear or branching, or whether time is discrete or continuous) and the types of temporal semantics (interval semantics, point semantics, linear semantics, branching semantics and partial order semantics), we can classify temporal logic. In the next Section, we will discuss the classification of temporal logic systems in details. For an appropriate definition of any temporal logic, the following are necessary:

- Syntax: the language for describing the time or temporal systems;
- Semantics: the model of time to derive the meaning of a logic formula.

The main question we need to ask is what is the system structure of time that should be used (model of time) [85]

2.3.2 Temporal Logic System classification:

Most temporal logic systems can be classified along a number of axes. We will list the most popular axes that can be used to classify temporal logic systems which are:

- Propositional versus first-order;
- Linear versus branching;
- Points (instances) versus intervals;
- Discrete versus continuous; and
- Past versus future tense.

as are shown in the next Figure 2.1.

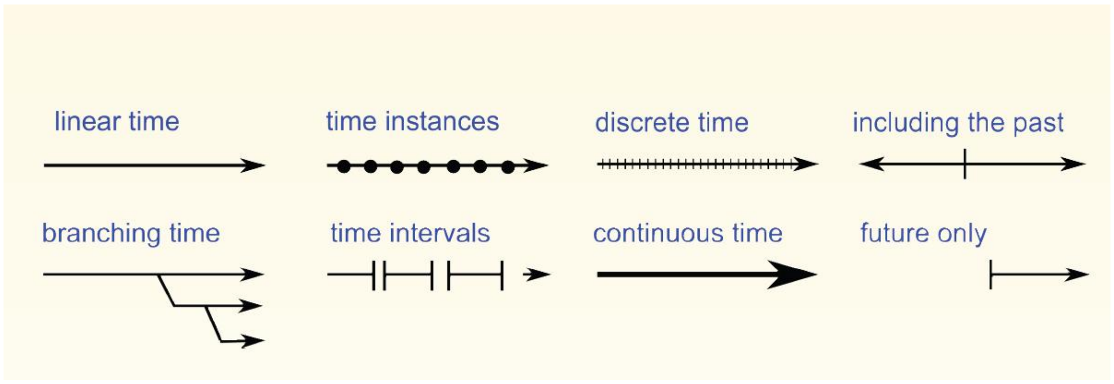

Figure 2.1: Time Models [75]

Next, the most common criteria to distinguish between temporal logic systems is described [41, 75].

Propositional versus First order:

Propositional temporal logic is similar to the classical propositional logic. In propositional temporal logic, problems are expressed in generic language such as the set of propositional letters, the classical propositional connectives $\neg, \vee$ and $\wedge$ and a set of temporal operators [41, 75].
When creating a program from formal specifications it is crucial to use propositional temporal logics since they have the finite model property. The created model is similar to a finite state machine; but, the model accepts infinite strings.
First order temporal logic (FOTL) is similar to predicate logic. Different kinds of FOTL have been suggested; however the generic language consists of predicate symbols, variables, constants, Boolean connectives, quantifiers and temporal operators [41, 75].

A difference can also arise as a result of enabling or disabling restrictions on the interaction of quantifiers and temporal operators. Lack of restrictions or freedom in some cases might lead to logics that cannot be decided. For instance, enabling modal operators within the freedom of quantifiers can cause a serious problem. On the other hand, one can have a restricted FOTL composed of propositional temporal logic together with a first order language for defining the atomic propositions by disabling such quantification over temporal operators [41].

Computational versus Linear Time:

There are two main contrasting views that have tried to explain the structure of time. One view is that the course of time is linear because time flows in only one direction and the other view is that time has a branching tree like nature. According to the theory of linear time, at any instant there is only one possible future moment [74, 41, 128].
According to the branching theory of time, at each moment of time, time can split into alternate courses portraying different possible futures, which mean that at any moment, time has many futures but only one linear past[74].
So, if linear temporal logic has the linear structure of time we call it linear time logic (LTL); however we call it branching (computational) time logic if it has the branching time structure[74, 128]. Depending on the two views stated above, we can classify a system of temporal logic as either a linear time logic or a system of branching time logic.

The nature of time assumed in the semantics is normally reflected in the temporal modalities of a temporal logic system. When it comes to a linear time logic, the flow of events can be explained along a single time line in temporal modalities. On the other hand, in branching time logic systems, modalities enable quantification over possible futures. We can get different logics by changing the structure of the language of the logic in both linear and branching time temporal logic systems [74, 41].
Linear Temporal Logic (LTL):
Linear-time temporal logic, or LTL for short, is a widely accepted formalism for the specification and verification of concurrent and reactive systems [82]. It models time as a sequence of states, extending infinitely into the future. This sequence of states is sometimes called a computation path, or simply a path. In general, the future is not determined, so we consider several paths, representing different possible futures,

any one of which might be the actual path that is realized as shown in Figure 2.2 [66, 128].

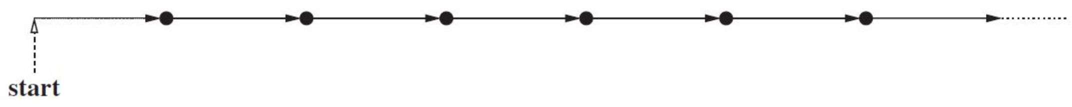

Figure 2.2: LTL [47]

-*Formula of LTL*:
The formula in LTL is defined inductively as follows:

- $\top$ and $\perp$ are formulas.
- All atomic propositions $p \in$ FP are linear temporal logic formulas.
- If F is a formula, then $\neg F$ is a formula.
- If $F_1, ..., F_n$ are formulas, where n ≥ 2, then $(F_1 \wedge ... \wedge F_n)$ and $(F_1 \vee ... \vee F_n)$ are formulas.
- If F and G are formulas, then $(F \rightarrow G)$ and $(F \leftrightarrow G)$ are formulas.
- If F is a formula, then $\bigcirc F$, $\Diamond F$, and $\Box F$ are formulas.
- If F and G are formulas, then $F \,\mathcal{U}\, G$ and $F \,\mathcal{R}\, G$ are formulas.

The symbols $\bigcirc, \Diamond, \Box, \mathcal{U}, \mathcal{R}$ are called temporal operators.
Now we explain their meaning informally. The formulas of LTL are true or false on computation paths, that is sequences of states $s_0, s_1, ...$ The formula $\Box$F means that F is true at all states along the path.

The formula $\Diamond$F means that F is true at some state on the path. The formula $\bigcirc$F means that F is true at the next state after the initial one, that is, at s_1.The formulas $F\ \mathcal{U}\ G$ and $F\ \mathcal{R}\ G$ will be formally defined below because they are a bit more complex [41, 66, 128].
Any two formula F and G called equivalent ($F \equiv G$) if for every path σ we have $\sigma \models F$ if and only if $\sigma \models G$. Examples of linear time temporal logic formula:

- Liveness: Every request is followed by a grant.
  $\Box(request \rightarrow \bigcirc Grant)$
- Safety: p never happens.
  $\Box\neg p$
- Fairness: p happens infinitely often.
  $(\Box \bigcirc p) \rightarrow f$
- Another natural example, we may want to express that a professor and a student cannot be borrowers from the library at the same time:
  $\Box\neg(borrower\_student \wedge borrower\_prof)$
- $\Box\,(S \rightarrow \Diamond T)$
  The informal meaning of this formula is:
  Whenever S holds, in the future T is bound to hold [121].

Computational Temporal Logic (CTL):
Computational Temporal Logic, is a branching time logic, which means that its structure model of time is tree like and has many branches (paths), any one of which might be the actual computation path. In this model of time we should specify the path before any computation as shown in Figure 2.3 [66, 128].

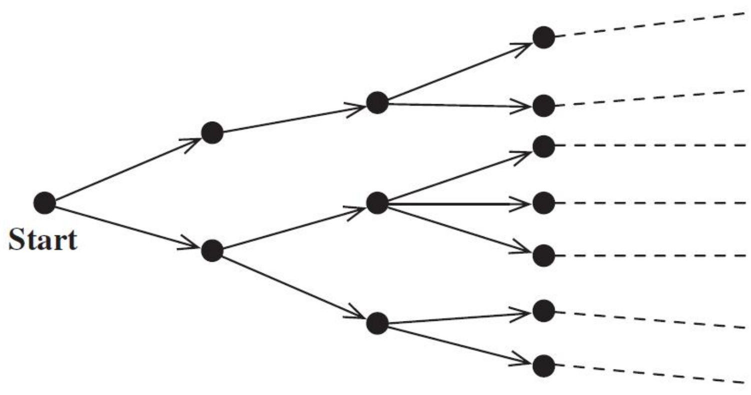

Figure 2.3: CTL [47]

-*Formula of CTL*:
The formula in CTL is defined in pairs inductively as follows:
-Firstly Path part:

- $\mathcal{A}$: means all paths (inevitably)
- $\mathcal{E}$: means on some path (possibly)

-*Formula of LTL*:
The formula in CTL has the tree like (branches), if the branch is computed then inside the branch it has the same syntax of LTL formula, and it is defined inductively as follows:

- $\top$ and $\bot$ are formulas.
- All atomic propositions $p \in$ FP are linear temporal logic formulas.
- If F is a formula, then $\neg F$ is a formula.
- If $F_1, ..., F_n$ are formulas, where n ≥ 2, then $(F_1 \wedge ... \wedge F_n)$ and $(F_1 \vee ... \vee F_n)$ are formulas.

- If F and G are formulas, then $(F \rightarrow G)$ and $(F \leftrightarrow G)$ are formulas.

- If F is a formula, then $\bigcirc F$, $\Diamond F$, and $\Box F$ are formulas.

- If F and G are formulas, then $F \mathcal{U} G$ and $F \mathcal{R} G$ are formulas.

The symbols $\bigcirc, \Diamond, \Box, \mathcal{U}, \mathcal{R}$ are called temporal operators.
Now we explain their meaning informally. The formulas of LTL are true or false on computation paths, that is sequences of states $s_0, s_1, ...$ The formula $\Box$F means that F is true at all states along the path.
The formula $\Diamond$F means that F is true at some state on the path. The formula $\bigcirc$F means that F is true at the next state after the initial one, that is, at s_1. The formulas $F \mathcal{U} G$ and $F \mathcal{R} G$ explained in LTL Section [41, 66, 48, 128].
Another example of branching time temporal logic formula:
Safety: bad thing never happens:
$\mathcal{A}\Box(\neg bad\_thing)$
Fairness: p happens infinitely often.
$\mathcal{E}(\Box \bigcirc p) \rightarrow f$
$\mathcal{E}\Diamond(P \ \wedge \ \neg \ q)$
Which means: There exists a state where p holds but q does not hold. $\mathcal{A}\Box(p \ \rightarrow \ \mathcal{A}\Diamond \ q)$
Which means: Whenever p holds, eventually q holds. $\mathcal{A}\Box(\mathcal{E}\Diamond \ q)$
Which informally means: That at all the paths q holds after some time.

Time Instants (points) versus Intervals:

The choice between time instants and time intervals has been a centre of focus in philosophy when using temporal logic. Temporal logics normally represent time either as point based or intervals. Until the last decade, logic scholars were greatly interested in point based temporal logics. Prior and Pnueli considered time as a discrete sequence of points in their model of temporal logic and used it in system specification and verification [74]. Modelling the refinement of a system specification is a widely recognized problem when using a point-based temporal logic [44].
However, the interval based approach is more efficient than the point based approach since it can provide efficient representation of temporal facts. For example, the interval notion is necessary to show continuous processes and to make temporal statements in AI applications; because of this, temporal statements are based on

intervals [74]. In a point based temporal logic model, the formula evaluated as true or false of points in time is as shown in Figure 2.4.

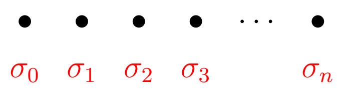

Figure 2.4: Points based

However, in interval based temporal logic the formula is evaluated over intervals of time as shown in Figure 2.5.

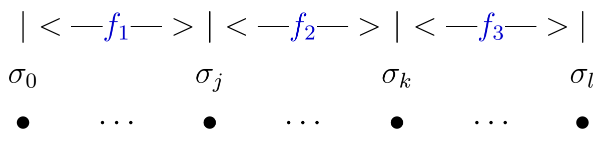

Figure 2.5: Interval based

The claim is that use of intervals greatly simplifies the formulation of certain correctness properties [41].

There are many scientists who proposed use of the interval in many areas; however when it comes to philosophical logic, Simons and Galton suggested the need for intervals with regard to conceptual structures in natural language [54, 130]. Formal tools for reasoning in artificial intelligence have sprung up from Interval based temporal logics. Major contributions in this area were carried out by Allen [2, 3, 4, 5]. Allen proposed thirteen relations between intervals, called Allen's relations. He provided an axiomatisation and representation result of interval structures, and interval-based theory of actions and events.

Interval based logics have been used in other areas of computer science. One of the first applications of interval temporal logic (ITL) in computer science for design of hardware components was developed by Moszkowski [95]. Interval Temporal Logic is a linear temporal logic over (in)finite time. ITL has been applied in different problems, from specification and verification of hardware devices [60, 96, 97] and temporal logic programming [98, 37] to the specification of multimedia documents

[19] and human computer interaction [17]. Interest in ITL also comes from its natural notation and expressiveness. Operators such as chop, projection and star support sequential composition, multiple time granularities and repetitive behaviour in system specifications. Additionally, high-level, imperative-like operators such as loops, conditionals and assignments can easily be defined, and so ITL naturally lends itself to execution [57]. ITL's features make this logic an attractive alternative to the problems faced by conventional point-based temporal logics. It is accepted that the specification of properties in such point-based temporal logics could be difficult for non-temporal logic experts. Thus, successful verification of a misformulated property may give unjustified confidence in a system design. According to Pnueli and Vardi [112, 136], specification languages need the full power of regular expressions which is the term used to describe a codified method of searching, defined by Stephen Kleene in 1956 [73]. It is known that chop and chopstar bring this expressive power to ITL [57]. Furthermore, there is an increasing industrial interest in ITL; for example, Verisity have adopted ITL concepts in their temporal language [64] and a temporal logic called Sugar has been introduced by IBM containing ITL (like) operators and these works targets are making the logic more usable for industrial design engineers [11].
The nature of interval temporal logic can be viewed from two distinct perspectives, according to philosophy. Intervals can be viewed as points, which are the only primitive objects, or they are primitive objects in the logic. The majority of interval based logics construct intervals out of points, for example [2].
The following is an example of instant (points) time temporal logic formula:

$$(\sigma, 6) \models \Diamond(p)$$

This formula can be defined informally as:
there exists a point where p holds.
The following is an illustration of interval time temporal logic formula:

$$A \ ; \ B$$

The above formula can be defined as:
The interval decomposed (chopped) into a prefix interval and suffix interval, such that A holds over the prefix interval and B over the suffix interval, or A holds for that interval if it is infinite.

The issue below is linked to the underlying structure of time.

Discrete or Continuous (Dense):

A more fundamental choice is that between Discrete or Continuous of a flow of time. It implies that it would be composed of a sequence of instances where each non-final point is followed by another immediate point. We can therefore say that a property is correct in the following moment and also correct all time or at some future time. This can be formulated in first-order logic:

$$\forall\ x, y(x < y \rightarrow \exists\ z(x < z \wedge z \leq y \wedge \forall\ w(x < w \wedge w \leq y \rightarrow z \leq w)))$$

Temporal logics mostly used for program reasoning consider time as discrete where the present instant matches to the program's present state and by the finite model property. Hence the temporal structure which matches with a series of states of a program execution is the non negative integers as it is shown in Figure 2.6.

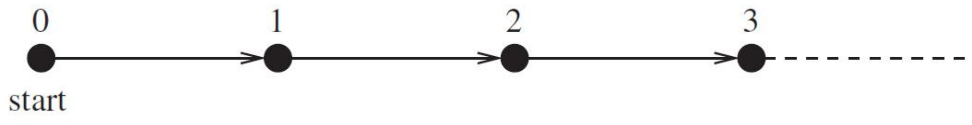

Figure 2.6: Discrete time [47]

Here, each of the black circles represents a classical propositional state, and the arrows represent the accessibility relation, in our case the 'step' to the next moment in time. Note that we also have one state identified as the 'start of time'.
Dense refers to a linear ordering in which we can find another different point between any two distinct points. This can be mathematically represented as:

$$\forall\ x, y(x < y \rightarrow \exists\ z(x < z < y))$$

The idea of the flow of time can be modelled using rational or real numbers, which can represent the flow of dense time [137, 74, 41] as is shown in Figure 2.7.

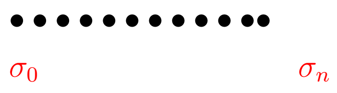

Figure 2.7: Continuous time

Philosophers have been studying tense logics interpreted over a dense time structure. Cau [22] proposed the application of dense time temporal logics to reasoning about concurrent programs. Dense time temporal logics can also be used in real time programs where strict, quantitative performance requirements are placed on programs [41].

Past versus Future:

In this Section, we are trying to answer the old question of whether temporal logic with the past is more succinct than pure-future temporal logic. Logicians have used temporal modal operators to explain the happening of events both in the past and future. Also specifications which could be expressed in natural language sometimes use references to events that happened in the past. In addition the temporal logics studied by linguists and philosophers are where past and future time have been used on an equal footing [116]. In temporal logic systems, for reasoning about concurrency, past time operators do not enhance the expressive power because program implementations have a specific starting time and for this reason, these logic systems usually do not have past time operators [80].
Generally, it can be said that the use of past time operators do not add any expressive power to linear time temporal logic [53]. Also, any past time formulas can be translated into equivalent pure future ones [52]. So, there is no need to use the past operators depending on this theorem [79].
Temporal languages are increasingly employed to cover the variety of uses and there is a growing interest in using past time operators for temporal logic languages [80, 79, 93, 90, 37, 49, 57, 84].

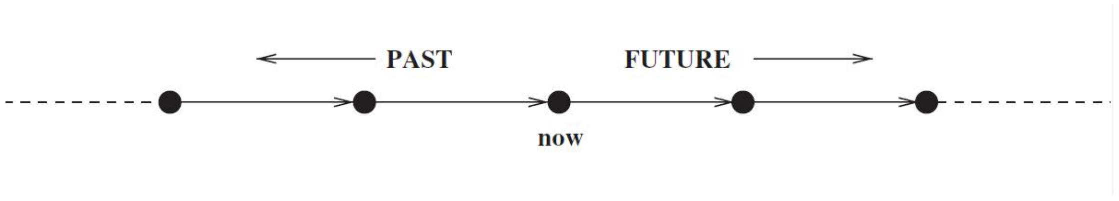

Figure 2.8: Past VS Future [47]

At the moment, past time operators play an important role in compositional specification similar to that of history variables [41]. Finally, the usefulness of past-time constructs is most apparent in the classification of temporal properties [145, 87, 26]. We adopt the view that past time operators make the specification and verification of systems much shorter and contain less symbols, hence it is easier to express. Regarding succinctness and simplicity, past time operators do add expressive power to temporal logic, but from a practical point of view not from a theoretical point of view, as has been shown in many articles such as [79, 93, 90]. All the proposed works in past time ITL are combining the use of past time operators with the future time one [37, 19]. However, in this thesis, our work aims to introduce the past time operators of ITL and use it to reason about history based access control policies separately.
The following are examples of past time formula:
$\widehat{\bigcirc}$ (p)
This formula can be defined informally as:
p holds in the previous state.
$\widehat{\Box}(grant \rightarrow \widehat{\bigcirc}\, request)$
The informal meaning for this formula is (every given grant is previously requested).
The following is an example of future time formula:
$\bigcirc$ (p)
The formula can be defined informally as:
p holds in the next state.
$\Box(request \rightarrow \bigcirc\, grant)$
The informal meaning for this formula is (every request will be granted in the next state).

2.3.3 Temporal Logic Application

Generally, temporal logic is applicable in various areas and sciences such as Philosophy, Computer Science, Artificial Intelligence(AI) and natural language [58]. In philosophy, temporal logic is used as a formalism to clarify philosophical issues about time; however, in computer Science it is used as a tool for handling the temporal aspects of the execution of computer programs and in artificial intelligence (AI) as a language for encoding temporal knowledge and finally, as a framework to define the semantics of temporal expressions in natural language.

Computer Science Application:

Temporal logic has been used in many areas within Computer Science, including the specification and verification of reactive systems. Manna and Pnueli [89] recognized that temporal logic is well-suited for their formal specification and verification of reactive systems. The range of reactive systems is wide. It comprises embedded systems, process control systems, and all types of interactive, concurrent or distributed hardware and software systems.
Generally, temporal logic is applied in computer science in the following areas:

1. Formal specification:
   Temporal logic formulas are used to make accurate, formal and mandatory specification of systems and components [114, 113, 111, 77, 128, 89].

2. Formal verification:
   The rules of a temporal logic proof calculus are used to verify the validity of a temporal logic specification with regard to more abstract system specifications [113, 128, 89].

3. Requirements description:
   In the initial stages of creating a system design, a set of temporal logic formulas are normally used to express the consequences of the requisites restricting the functional system behaviour [111].

4. Specification checks:
   In case different methods other than temporal logic are used to create the specifications, temporal logic may also be used to specify the requisites and plausibility states. At the moment there are several procedures that can work

with the tool-based checking of formal system specifications concerned with temporal logic conditions [111].

Also there are uses of temporal logic for the synthesis of programs from temporal specifications [115, 76], knowledge representation and reasoning [42, 141, 7], and temporal databases [28, 27, 36].

2.4 Access control policies

In this Section, we will introduce history based access definition, elements, types and categories. Also, a comparison between set of policy languages is proposed to express history based access control policy in order to support our choice of policy language and the computation model used.

2.4.1 Introduction

History-based policies [1] are a special class of policies where the policy decisions depend on previously observed behaviours within the system. This has some advantages however it also has some drawbacks, in the next Section we will discuss this special class of policies and discuss the languages which are suitable to express these policies.

2.4.2 Access control

Access control is one of the earliest approaches used to implement security policies, which is still largely practised at present. To give a general description of access control, it is a process of arbitrating requests to the desired resources and data maintained by a system and evaluating whether the access should be granted or denied [123].

In spite of the fact that access control guarantees that every single attempt to get access to a system or its resources should be controlled, with a set of predefined policies [123], access control is one of the major security mechanisms used to achieve confidentiality (information is not disclosed to non permitted persons, processes or devices [63]), integrity(unauthorized persons, processes or devices cannot modify information [63]) and privacy(data is protected so that it is used only by authorized people or for business purposes, based on legal requirements, corporate policies and end-user choices [63]) in software systems [43]. An access control system design is considered by three stage components which are: access control policies, models, and mechanisms [123].

Access Control Policies

Access control policies are security requirements, that describe how access is managed, and who can access which information, and what shall be the conditions for the access of this information. These policies are implemented via a mechanism that arbitrates access requests with the system and makes grant/deny decisions [43]. Access control policies are derived from, and must comply with, security requirements [63].

Access Control Mechanism

Access control mechanisms provides the details of the low level functions for implementation of access control policies.. The access control mechanism must also work as a reference monitor [123], and a trusted component intercepting every single request received by the system [63].

Access Control Model

Access control models are a formal representation of an access control system. They provide the mechanism of how to reason about the supported policies, and provide the proof of the security policies of the access control system. Access control models provide a level of abstraction between policies and mechanisms, enabling the design of implementation mechanisms to enforce multiple policies in various computing environments [123].
Several proposed models exist to represent access control. For example, one of the main models is the stack inspection run using Java and C#. This model represents the policy to grant static access rights to the code, while the actual run-time rights depend upon the static rights of the code frames in the stack. One of the other main access control models is history-based access control. In this access control method proposed by Abadi and Fournett, consideration is taken into account of the entire process execution, and the actual rights of the running code depend upon the static rights of all parts of the code executed in time. The mechanisms that are used in order to enforce history-based policies are execution (reference) monitors which observe computations and abort them if policy is at risk of violation [1].

2.4.3 Access Control Policies elements

An access control policy is comprised of a set of access control rules. A rule can have various modes for example allow, deny and oblige. The work in this dissertation focuses on rules related to allow and deny mode. An allow rule authorizes to perform an action on an object, deny rules are used to restrict a subject to perform an action on an object. When a request is generated by a subject, enforcement authority uses these to assess and make decisions. A typical access rule is expressed as a 3-tuple $(subject, object, action)$, such as a subject can perform an action on an object [34]. An access control policy may require satisfying some supplementary requirements before access is granted to a request. For example, in case of health care, the location of the originated request might affect the grant or deny decision [34]. If a particular request can only be allowed access if it is made from an emergency room, hence location (emergency room) can be specified as a condition for the access control rule. In requirements specification, we are concerned with the actions for which each actor (subject) is responsible, the conditions under which each action can occur (constraints and preconditions). Each of these access control elements can be mapped to a requirements specification element. This mapping suggests it is possible to derive access control policies from requirements to ensure that access control policies comply with the requirements.

2.4.4 Access Control Policies types

Access control policies are typically categorized into two main approach types which are dynamic and static. The dynamic approach type is where access control to protected resources is supervised by an execution (reference) monitor. An execution (reference) monitor is defined as a software component that is used to supervise the execution of programs, and thereby decide whether or not authorisation to use a resource is granted or not according to the security policy in place. In contrast in the static approach type, we attempt to determine at a compile time if a program obeys the security policy [9].

Stack Inspection Access Control

Stack inspection is an access control mechanism which grants authorisations based on the contents of the runtime call stack. The Current implementations of stack inspection uses a lazy evaluation approach; this limits the stack inspection to the point when stack inspection tests are performed. This is done by retrieving and inspecting call stack data. This conventional strategy appears to be quite efficient, because the security state needs not be updated upon method invocations. However, this strategy has a few drawbacks such as during the runtime inspection the runtime overheads may grow to a large extent; secondly, inter procedural program transformation may be prevented because these optimizations may change the call stack structure [9].

History-based Access Control Policy

History-based policies [1], are an expressive class of policies that can define policy decisions dependent on previously observed behaviours within the system. Abadi and Fournet advocates history based access control as a suitable alternative to stack inspection. History based access control is considered as more expressive than conventional methods such as stack inspection [69]. The motivation behind this is that history-based access control overcomes some of the known weaknesses of stack inspection such as imprecise records of the execution history, unclear security goals attained, and invalidation of interprocedural optimizations. Another motivation which makes history based access more attractive is based on some efficient techniques, which illustrate some choices that can be made while re-implementing the access control mechanism of Java and the Common Language Runtime (CLR) [20] to deal with execution histories, rather than with call stacks. Discussion on two high-level programming constructs, that permit changing the set of current access rights, is also provided. First of these language constructs is called grant, which amplifies the rights of the callee, by yielding the static access rights of the caller; this method is similar to Java's privileged method calls. The second language construct is accept, in which a caller entrusts its callee, by restoring, when the callee returns its set of rights before the call is made. Also, worth mentioning is that this construct can be used for recovering of the stack inspection, while providing history based access control [9].

2.4.5 Access Control policies categories

To ensure security and what should and what should not be allowed, various access control policies can be applied. There are different definitions of what security means and so many different criteria can be applied. Access control policies are grouped by three different criteria:

Discretionary Access Control (DAC)

Discretionary Access Control (DAC)(authorisation-based) provides access depending upon the identity of the initiator of the request, and on access rules which state which requestors are allowed or not [123]. These are Discretionary privileges as users have the facility of passing their privileges to other users; the granting and removal of the privileges is regulated by an administrative policy. Early discretionary access control models, such as the access control matrix model [78, 59] and the HRU model [62], provide a basic framework for describing DAC policies. The HRU model has been formalised by Harrison, Ruzzo, and Ullmann for analysing the complexity of access control policies and they identify six primitive operations that describe changes to the state of a system [123].
A more specific assessment of the access control problem indicates the benefits of separating users from non-users. Users are inert entities for whom authorisations are specified with specifics of who can and cannot connect to the system. Once connected to the system, users originate processes (subjects) that execute on their behalf and, accordingly, submit requests to the system. This is different for discretionary policies which do not provide this distinction and treat each process (submitted on behalf of a user) based on the user's authorisation rights. This is a drawback of the discretionary policies as they are vulnerable to malicious programs, which can run processes based on authorisation rights of other users, such as Trojan Horses can bypass these authorisations as they are embedded in programs.
Trojan horse is a malicious program with apparent useful functionality but actually has harmful functions embedded inside, which use the authorisations of the invoking processes. Trojans could even delete all files of the users (this destructive behaviour is not uncommon in the case of viruses) [123].

Mandatory Access Control (MAC)

Mandatory security policies enforce access control on the basis of regulations mandated by a central authority [123]. The most common form of mandatory policy is the multilevel security policy, based on the classifications of subjects and objects in the system. Objects are passive entities storing information. Subjects are active entities that request access to the objects. Note that there is a distinction between subjects of the mandatory policy and the authorisation subjects considered in the discretionary policies. While authorisation subjects typically correspond to users (or groups thereof), mandatory policies make a distinction between users and subjects.

Users are human beings who can access the system, while subjects are processes (i.e., programs in execution) operating on behalf of users. This distinction allows the policy to control the indirect accesses (leakages or modifications) caused by the execution of processes [123]. An example of MAC policy is the lattice-based multilevel security policy [33], policies represented by the Bell-LaPadula model [12, 13, 122] and the Biba model [15, 122] are MAC policies. MAC policies protect indirect information leakages (e.g., Trojan Horse attacks), but are still vulnerable to covert channel attacks [123, 110]. Covert channels are channels that are not intended for normal communication, but can still be exploited to infer information [123, 129].

Role-Based Access Control (RBAC)

The RBAC model is an alternative to traditional DAC and MAC models and has received increased attention in commercial applications, such as the Oracle 9i DBMS [119]. The concept of role-based access control (RBAC) began with multi-user and multi-application on-line systems pioneered in the 1970s. The central notion of RBAC is that permissions are associated with roles, and users are assigned to appropriate roles. This greatly simplifies management of permissions [43, 125]. Roles are created for the various job functions in an organisation and users are assigned roles based on their responsibilities and qualifications. Users can be easily reassigned from one role to another. Roles can be granted new permissions as new applications and systems are incorporated, and permissions can be revoked from roles as needed. A role is properly viewed as a semantic construct around which access control policy is formulated. The particular collection of users and permissions brought together by a role is transitory. The role is more stable because an organisation's activities or functions usually change less frequently [125, 123].

2.4.6 History based access control policy languages

In the next Section, we will now discuss the proposed policy language to be used for history based access control. As the name suggests, the history-based language will deal with the history of the execution and will not only depend upon the current state of the system. A rule-based approach will specify the access control rights in relation to the policy rights.

According to Becker et al. [10] a policy language is expected to be:

- Expressive::
  A policy language can be judged on its ability to express informal requirements. A good policy language will be easy to use and provides simple statements to express specific requirements.

- Clear and Readable:
  The language syntax should be humanly readable, and must be as simple as possible making it easier to learn. Becker et al. [10] in his work reasons that most of the logic based languages are too difficult to read and thus make it difficult to learn; he mentioned about XML-based languages, that they are too verbose making them programmer-unfriendly.

- Intuitive and Unambiguous Semantics:
  The statements written in the policy language must have a concise meaning. Any good policy language shall allow the user to write statements in a clear way and with a concise meaning. Languages without formal foundation normally struggle to provide clear meaning of their statements as, for example, natural language descriptions are mostly ambiguous making them unsuitable for writing specifications and requirements.

- Effective Decision Procedure:
  A policy language must evaluate the query efficiently and provide accurate decisions.

- Extensibility:
  Any policy language must provide ease of extensions to its existing syntax to cater for any additional language. Also, it is important that the extension procedures must be simple and not too difficult as to discourage extensibility.

The main technical features of what a policy language is expected to be is listed above. Now we introduce and test the ability of policy languages to express history-based access control policies in order to support our choice of SANTA policy language with the proposed model to reason about history-based access control policies.

Flexible authorization manager authorization language (FAM)

According to Jajodia et al. [67], most policy languages specify two specific types of policies:

- Open Policies: The access is allowed to anyone unless denied. An example of this type is blacklist.
- Closed Policies: The access is denied to anyone unless explicitly allowed. An example of this type is the E-mail.

The positive and negative authorisations rule has been proposed as well as the decision rules by Jajodia et al. [67]. These decision rules are normally used to resolve the conflicts between authorisations. Also, they introduce the rule-based Flexible Authorization Manager (FAM) authorization language, where positive and negative authorisations for a subject (or group) to perform an action on a specific object can be expressed. Next, we will give an example of positive and negative authorization rules with the informal meaning:
$cando(mail, faculty, +read)$
This authorisation states that faculty is a subject authorised to execute read which is an action on the object mail.
$cando(personal, faculty, -read)$
This authorisation states that faculty is a subject not authorised to execute the action read on the object personal. An authorisation is a triple of the form:
$(o, s, \langle sign \rangle\, a)$ where $o \in AO$, $s \in AS$, $a \in A$ and "sign" is "+" or "-".
What is more, FAM also allows expressing authorisations based on a previous access using so-called done rules (history access). These are essentially facts that are created by the system (FAM) during runtime and reflect the access executed by a user. However, the final decision of granting access or denying it is resolved by a decision rule introduced before. To explain this, assume the following rule:
$do(file, s, +a) \leftarrow dercando(file, s, +a) \& \neg dercando(file, s, -a)$
This specifies that if it can be derived that s is allowed to perform action a on $file$, and it cannot be derived that s is denied to perform action a on $file$, then s is effectively allowed to perform a on file. In FAM, despite there being no notion of time or temporal dependency between the done events, the history-based access control requirements is nevertheless supported and can be expressed. In the architecture of authorisation framework there is a component called history table, and in this table each row shows a single executed access. A row is structured as $(Object, User, Role, Action, Time)$.
The history is represented formally by the predicate *done* with a matching list of parameters. However, it is not clear how the history table is actually updated. The temporal relations are difficult to express at a higher level of abstraction because the

time is represented explicitly. So, it can be concluded that Flexible authorization manager authorization language is not suitable to reason about history based access control policies.

Temporal role-based access control

Role-based Access Control (RBAC) defines a role as a set of privileges associated with a specific position within an organisation. Authorizations are assigned to roles which are then assigned to users. This user is allowed to execute all accesses for which the role is authorized. The policy management when using this mechanism is much easier regarding the separation of assigning the roles to users and the authorisations to roles, so these assignments can be manipulated independently. The model which addresses temporal constraints for an Role-Based Access Control (RBAC) is then proposed [14] and known as Temporal Role-Based Access Control (TRBAC). In these constraints ,the users can be restricted to execute roles at certain time periods represented by using a countable set of contiguous intervals called calendars; these calendars are numbered by integers called indexes of the intervals. $Hours, Days, Weeks, Months$, and $Years$, are examples of calenders. Moreover, we can combine these calenders to represent more general periodic expressions, for example, the set of Sundays or the set of The second hour of the ninth day of each month.
The Role Enabling Base (REB) is a Role that contains temporal constraints on the enabling time. This Role is enabled or disabled at run-time by means of Run-time Request Expressions of the form $p : E$ (prioritized event expression) after duration expression Δt only. A system trace is modelled as a sequence of snapshots which correspond to the current set of events and the status of roles. Some Role Enabling Base (REB) specifications may be ambiguous and they may lead to states where there is no unique way of deciding which roles are enabled because of the expressive power provided by TRBAC. Therefore, to ensure that the specifications are unambiguous and consistent, a notion of safeness is introduced as well as a polynomial algorithm to test the safety of REB specifications [14].
The differences between the proposed SANTA and TRBAC is clear. Firstly, TRBAC is mainly concerned with the assignment of roles; however, SANTA is concerned with the assignment of authorisations. Secondly, TRBAC uses explicit time to model temporal dependencies and incorporates temporal constraints on the role enabling only.

It can be concluded that TRBAC is not suitable to express history based access control policies due to the following reasons:
Firstly, TRBAC is mainly concerned with the assignment of roles and not concerned with the assignment of authorisations. Secondly, TRBAC uses explicit time to model temporal dependencies and incorporates temporal constraints on the role enabling only.

Usage control model

We are going to give an overview of the Usage Control Model (UCON):
The Usage Control (UCON) Model [107, 144] is a session-based model where, between the start of the session and its termination, the user is allowed to perform a usage request which consists of a number of actions. This model supports authorisation (concerned with the authorisation of a subject to exercise a specific right), obligation (concerned with actions the user must perform) and conditions (determine the access of a subject depending on the environment of the usage process). The novelty of the approach is that it addresses mutable attributes [108] and the continuity of the enforcement. Mutable attributes are associated with the subjects, objects or the system and are updated as side-effects of usage processes. They can be used for example to count the number of times a resource has been accessed. The continuity of enforcement means that a UCON process can be revoked based on conditions that are expressed in terms of attributes. The UCON model has been first formalised using an extension of the temporal logic of actions (TLA) [77] by Zhang et al. [143, 142]. Here a single usage process is described in the form of a state diagram. System and user actions represent the transitions in the diagram. UCON policies are then defined as logical formula that postulate temporal relationships between system and user actions of a single usage process. The formalization, however, makes a strong assumption in that only a single usage process is specified. It is assumed that the time-line is finite, i.e., it starts with the beginning of the single usage request and ends with the subsequent usage request. This makes it difficult to reason about the interactions of several concurrent usage requests, or even sequences of usage requests, thus complicating the formal analysis of policies. The (side)effects of a usage process are captured in mutable attributes which are assumed to be persistent over usage processes and can influence subsequent usage control decisions.

A UCON usage process is characterized by the triplet (s, o, r), where s is the subject that exercises its right r on the object o. The usage process can be in one of the following states: initial, requesting, denied, accessing, revoked or end. The current state is described by Zhang et al. as a function state mapping from the triplet (s, o, r) to one of these states. A single usage process is defined by the state diagram in Fig. 2.9.

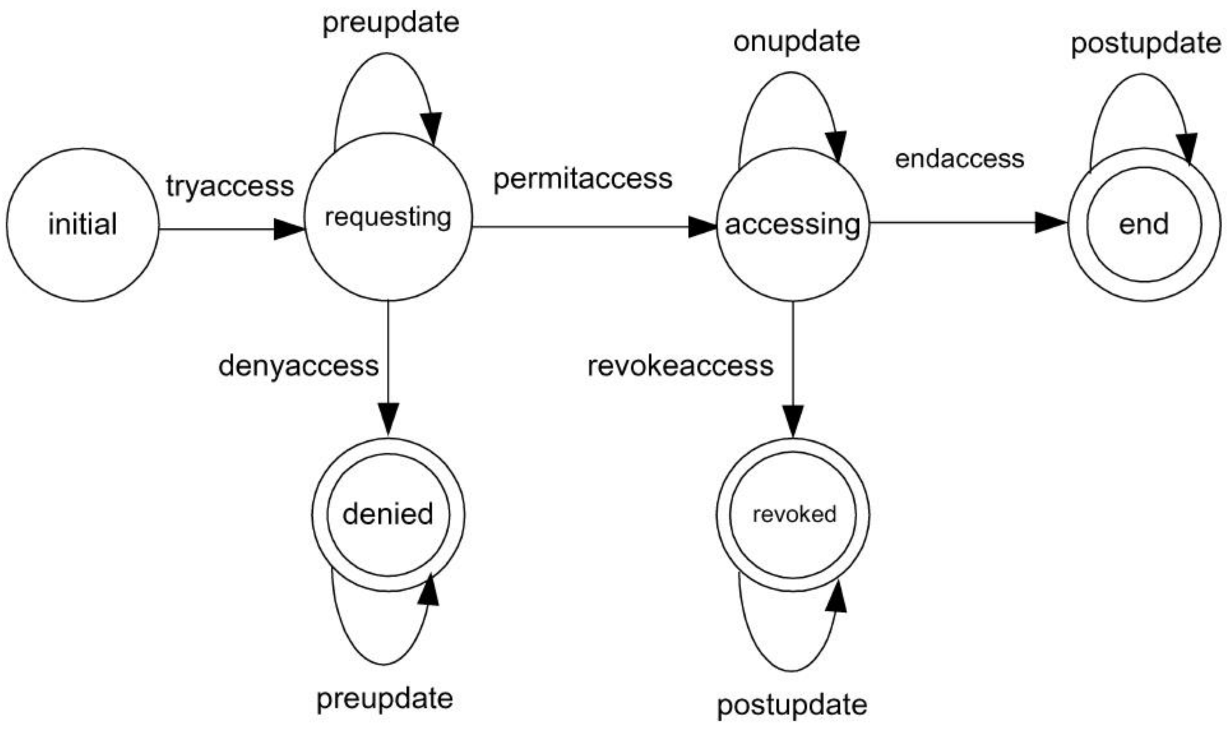

Figure 2.9: Usage control from [142]

In the initial state the subject s performs the action $tryaccess(s, o, r)$ initiating the usage process. The enforcement mechanism, for example a reference monitor (RM), either denies the access ($denyaccess(s, o, r)$) or proceeds by executing actions to update those attributes, which must be updated before the usage process commences. After the RM has updated the relevant attributes ($preupdate(s, o, r)$), it permits the access ($permitaccess(s, o, r)$) and continues to perform all required update actions that must be performed during the ongoing usage process ($onupdate(s, o, r)$). Alternatively the RM may revoke the access if any of the constraints of the UCON model are violated. The subject may end the usage process using the $endaccess(s, o, r)$ action. In both cases, the post update actions ($postupdate(s, o, r)$) are performed to change any mutable attributes that require modification. UCON policies define the enforcement of protection requirements at a relatively low level of abstraction. Janicke et al. [70] have presented an alternative formalisation of the UCON model [107]. They have used Interval Temporal Logic (ITL) for the formalisation which they said it is a more natural logic to express this model than the extended Temporal Logic of Actions (TLA) [142].

In UCON, the time line is assumed to be finite, for instance, it starts with the beginning of the single usage request and ends with the subsequent usage request. So, the interactions of several concurrent usage requests or even sequences of usage requests are difficult to reason about, which makes the formal analysis of policies very complicated. Also, the (side)effects of a usage process are captured in mutable attributes which are assumed to be persistent over usage processes, and can influence subsequent usage control decisions.
Finally, the UCON policies define the enforcement of protection requirements at a relatively low level of abstraction. Because of the above, it can be concluded that UCON is not suitable to express history based access control policies.

SANTA

Security Analysis Toolkit for Agents (SANTA) is a technology to address the arising complexity and its implication on the security of the system [126]. A security policy conveys the safety requirements of the system in an exact and clear way. SANTA provides policies, which are linked to obligations, access control and integrity of a system, and relate the entities with framing of constrains on their interaction. Access control requirements in this model are authorisation requirements, viz. constraints on the actions that a subject can perform on objects, or delegation requirements, namely which subject can delegate which right to another subject. Obligation requirements express that subjects must perform specific actions. Integrity requirements define constraints on the effect that the execution of an action has on subjects and objects. The aim of policies is to express these requirements at a high level of abstraction, hiding the details of the implementation that is necessary for their enforcement. In SANTA, policy rules are used as the basis for policy specifications. Rule-based languages are well established and well suited because most of these requirements are already informally expressed in the form of conditions and consequences. Each rule is expressed in terms of subjects, objects and actions. Subjects are the actors in the system. They can request access to objects that represent the available resources. The term action is used to denote the mode of access [68, 129]. Policy specification expresses the casual protection requirements for the policy language. One of the major tasks of SANTA policy specification is the development of policy rules that precisely elicit the informal requirements. When dealing with complex requirements the accurate capturing of requirements is not a trivial task. For example it is not simple to specify the state of the system or dependencies on the history of the execution. The SANTA policy language provides support for both state and history based dependencies [68].

The advantage of policies specified using SANTA over the majority of other policy languages is that policies can be specified in smaller units, and also these policies are defined using a rich set of SANTA operators. The provided operators allow the policies to be composed along a temporal and structural axis.

Temporal Composition

Siewe [68, 129] first introduced the temporal composition of policies. Temporal composition leads to policies that change dynamically over time or on the occurrence of events and it allows independent specifications of policies for a particular situation. The composition operators are then used to define the conditions of the policy change [68, 129].

Structural Composition

Large-scale systems spanning a large number of organizations, are controlled by the use of policies.. In this case, the composition along the structural axis can effectively elicit the requirements for each unit of the organisation for, e.g., organisation, department, project-group, etc as an individual policy. The policies for the smaller units are then used to compose the overall policy of the larger system. However, there are issues with this compositional approach as conflicts can easily arise in policies due to either common resource sharing or a particular individual placed under more than one policy. In addition, the resolution of these conflicts is difficult because of the dynamics of the system. A specific policy can only be applied to a subset of subject, objects and actions and this forms the basis of structural composition. These sets are referred to as the scope of the policy [68].

Policies in SANTA

Policies in SANTA are an integral part of the system specification. A policy rule is the smallest part of the policy specification, where each rule captures a discrete requirement, such as, “allow member to register”.

The Authorisation rules demonstrate the access control requirements. Three different rules are related with authorization such as positive and negative authorization and finally the decision rule.

Authorisation Rules We use a rule-based approach and specify sets of access control rights in terms of policy Authorisation rules and their compositions. A simple access control policy consists of three types of Authorisation rules:

1. Positive authorisation rules are statements that indicate under which condition an access request should be granted. It is important to note that it is only an indication, which is taken into account for the final access decision of the policy.

2. Negative authorisation rules are statements that indicate under which condition an access request should be denied. Similar to positive authorisations, they are only an indication, which are taken into account for the final access decision of the policy.

3. Decision Rules and Conflict Resolution specify the final access control decision of a policy. Any policy should contain at least one decision rule, as otherwise no access will be granted by the policy. The alternative term “conflict resolution rule” originates from the fact that this rule de-conflicts the policy if a positive and negative authorisation is derived for a specific access. The term decision rule describes more accurately the fact that any access control decision defined by the policy is decided by one or more of these rules, not only decisions in the conflicting case [68, 69].

These simple access control rules are combined to form larger units called simple policies; where simple policies are implemented concurrently. The simple policies are used to define for example the protection requirement applied during a certain phase or situation of the system execution. In the following, we provide the syntax of SANTA policy language.

Subjects $su ::= S_i \mid cs$
Objects $ob ::= O_i \mid co$
Actions $ac ::= A_i \mid ca(e_1, ..., e_n)$
Premise of rule $pr ::= pr_1$ **chop** $pr_2 \mid pr_1$ **and** $pr_2 \mid pr_1$**or** $pr_2 \mid$ **always** $pr \mid$ **sometime** $pr \mid$ **not** $pr \mid$ **next** $pr \mid$ **if** be **then** pr_1 **else** $pr_2 \mid$ **exists** x **in** $se\ :\ pr \mid$ **forall** x **in** $se\ :\ pr \mid$ **last**$(e)\ :\ pr \mid e\ :\ pr \mid be$
Rules $ru ::= [rn\ ::]$ **allow**(su, ob, ac)**when** $pr \mid$ $[rn\ ::]$ **deny**(su, ob, ac)**when** $pr \mid$ $[rn\ ::]$ **decide**(su, ob, ac)**when** pr
Policies $po ::= (ru_1...ru_n) \mid po_2$ **policy** $pn :: po$ **end**$\mid$ po_2 **chop** $po_1 \mid$**if** be **then** po_1 **else** $po_2 \mid$ **aslongas** $be\ :\ po$

Figure 2.10: Syntax of SANTA [69]

Figure 2.10 summarizes the syntax of our policy language where e is an expression, be a Boolean expression, and se a Set expression with their usual operators and semantics. S_i is a subject variable, where i is a arbitrary name, similarly O_i is an object variable, A_i is an action variable and pn is a name for a policy; rn is a name for a rule (optional). Let *Subjects*, *Objects* and *Actions* be, respectively, the universal set of subjects, objects and actions. These can be used as part of SANTA expressions. Let $cs \in Subjects$ be a subject, $co \in Object$ be an object and $ca(\overline{v}) \in Actions$ be an action with interface $\overline{v}$.
It can be concluded that SANTA has the advantage over the majority of other policy Languages; that is the policies can be specified in smaller units, and also these policies are defined using a rich set of SANTA operators. So SANTA policy lan-

guage is appropriate to express history based access control policies. We will show in Chapter 5, that we will give new semantics to SANTA operators using past time operators of ITL. These operators with the new semantics will be used to reason about history based access control policies in the Scenario Chapter.
In the next Section, we will describe the computational model used with SANTA policy language, its components and how these components interact together; this model has been introduced by Cau et al. [24].

Computational model

In a policy based management approach [8, 131], the specification and enforcement of these constraints are loosely coupled from the system.
In Figure 2.11, we proposed the computational model used with SANTA policy language, in this model, the behaviour of the Policy Decision Point (PDP) is determined by the policy, so the specification and verification of policy is crucial for the administration of the system. Policy-based management can be implemented by many real-world implementations that use the Policy Decision Point (PDP)/Policy Enforcement Point (PEP) architecture. The computational model used (which describes the system entities, their behaviour and interactions) represents a suitable abstraction for these real world implementations. The external observation of system behaviour is sufficient for specification, verification and analysis purposes of dynamic security policies. Therefore, the implementation details of the domain-dependent interactions between users and system is not used.
In our system, we have three different entities: subjects, objects and reference monitors. The subject can be defined as any entity that performs actions on objects and it could be a human user, a group or role or a program acting on behalf of a user. The object is any passive entity that represents a shared data structure in the information system. The main function of reference monitors is to control the subject access to objects and whether the subject can perform an action on an object or not. The security policy specifies the concrete conditions under which a reference monitor permits or denies an execution request. The security policy represents an abstract specification of constraints that govern the relation between the subjects and objects in the system. However, the reference monitor behaviour is refines the abstract specification constructively in such a way that the overall system satisfies the policy. If the reference monitor implementation is correct, the properties of the policy are preserved by the system.

In Figure 2.11, in the reference monitor part a description of the reference monitor behaviour and other system components and how they interact with each other is given as a Statechart [61]. Statecharts constitute an extensive generalization of state-transition diagrams. They allow for multilevel states decomposed in an And/Or fashion, and thus support economical specification of concurrency and encapsulation. Concurrency is represented by a dashed line that separates components of a parallel system. The labels on the transitions in Statecharts are of the form Trigger[Condition]/Action, where Trigger determines if and when a transition will be taken and Action is performed when a transition is taken and the Condition is true. An action includes the generation of events [24, 69].

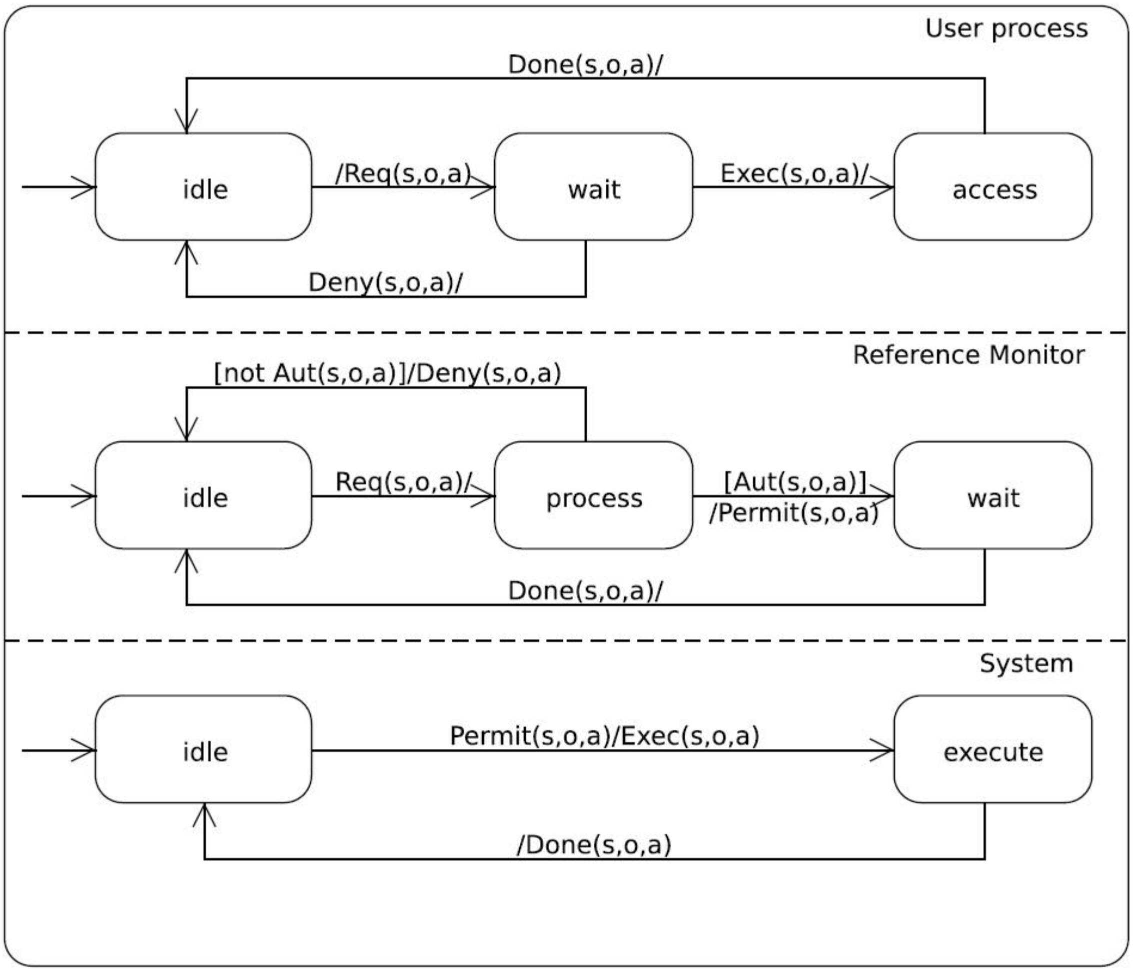

Figure 2.11: Computational model [24]

User model

The process (represented by subject s) in the user model acts on behalf of a user and can be in one of three states:

1. *idle*,
2. *wait* or
3. *access*.

The initial state of user process s is assumed to be in its *idle* state. When the event $Req(s, o, a)$ is raised, this mean that the process s requests the execution of action a on the system object o and transitions to the state $wait(s, o, a)$. The waiting state remains until it is either denied and the event $Deny(s, o, a)$ is raised or the request is executed and the event $Exec(s, o, a)$ is raised before transitioning to the state $access(s, o, a)$ (see user process in Figure 2.11) [24, 69].

Reference monitor model

The behaviour of the reference monitor is represented as:
Initially, the reference monitor (RM) process is in its *idle* state. During a user request $Req(s, o, a)$, the RM move to the state $process(s, o, a)$ and the policy specifies its behaviour. The event $Permit(s, o, a)$ is raised if the policy grants access $(Aut(s, o, a) is true)$, however the event $Deny(s, o, a)$ is raised if it denies the access $(Aut(s, o, a) is false)$ and then the RM returns to its *idle* state consequently (see Figure 2.11) [24].

System model

The access to the objects is facilitated by the system process, depicted in Figure 2.11. We assume that the system is initially in the state $idle(s, o, a)$. In the event that the controller permits the execution, it will transition to the state $execute(s, o, a)$ and raise the event $Exec(s, o, a)$ that synchronizes the state access(s,o,a) of the user process and the state $execute(s, o, a)$ of the system. The concrete behaviour of the user process and the system in these states are not explicitly defined; however, we will assume for the analysis of information flow that every pairing of these states can be characterized into the categories *read*, *write* and $read + write$. The computational model represents a simplification of real information systems, where not only subjects can concurrently make requests, but also the reference monitors and the system facilitating access to the shared objects are distributed and can exhibit concurrent behaviour [24, 69]

Policy rules

History based access control models express a policy in terms of authorization and denial and decision rules. This use of rules makes the specification clearer and easier to understand. A rule typically expresses a single security requirement and forms the basic building block of a policy.
Rules consist of a premise and a consequence. The premise describes a set of system behaviour, which lead to the consequence that represents an assertion on the current system state, such as allowing or denying a particular access. The consequence of a rule defines the decision taken by the reference monitor. The set of system behaviour in the premise is matched against the history of the system execution. Rules therefore can refer to sequences of previously observed states in the system execution, allowing for the expression of history-based policies and dynamic separation of duty constraints. The key idea is to associate a transaction control expression with each information object. This expression constrains the transactions which can be applied to that object to occur in the specified pattern. As operations are actually executed the transaction control expression gets converted to a history. This history serves to enforce separation of duties [124]. Events that can be referred to in the premise of rules are those defined in the computational model (Figure 2.11) or external events that are observable by the RM process.
Authorization defines the access to resources in the system. With respect to the computational model they define whether the execution of an action is permissible. An authorization rule defines the condition under which a subject is allowed to perform an action on an object. In the following we will describe the syntactic elements of the language informally [69].

2.5 Chapter summary

This Chapter has been divided in to four main sections:

- In the specification Section, the definition of software specification is given and enumerates the purposes of software specification. The formal specification definition and approaches have been described, and why informal specifications have been used in the past. Finally, the advantages and disadvantages of formal specifications have been listed.
- Next, an overview of temporal logic is given and the axes along which temporal logic systems can be classified. Temporal logic classification axes have been discussed in detail here, and the application of temporal logic in general and specifically for computer science applications.
- In the last Section of this Chapter, access control policies, mechanism and models have been explained, and what are the elements and types of access control policies have been described in detail.

  An explanation of stack inspection and history-based access control policy has been given in this Section. The categories of access control policies are listed here; Discretionary Access Control (DAC), Mandatory Access Control (MAC), Role-Based Access Control (RBAC). The description also includes the advantages and disadvantages of each model. Finally, history based access control policy languages (Flexible authorization manager authorization language (FAM), Temporal role-based access control (TRBAC),Usage control model (UCON) and Security Analysis Toolkit for Agents (SANTA)) have been introduced and discussed with their models.
  This Chapter contributes basic information about our research such as specification, temporal logic and history based access control policies. This information constructs the basis of our research and gives a good background to the knowledge needed in the following chapters.

Chapter 3

PAST TIME INTERVAL TEMPORAL LOGIC

- *Interval temporal logic, syntax and semantics.*
- *Past time interval temporal logic, syntax and semantics.*
- *Axioms and Rules for ITL^p*

3.1 Introduction

In this Chapter, an overview of interval Temporal Logic (ITL), its syntax and semantics as well as the derived constructs are given. In addition, the main contribution of this thesis which is the past time operators of ITL (ITL^p) is proposed, with the syntax and the semantics of these operators as well as the derived formula and constructs. Since the introduced ITL^p uses the past time operators only, new axioms and rules need to be introduced. Therefore, the axioms and rules for Propositional ITL^p have been introduced and listed in this Chapter, while the soundness proofs of these axioms and rules have been given in Appendix A.

3.2 Interval Temporal Logic (ITL)

Interval Temporal Logic (ITL) is a flexible notation for both propositional and first-order reasoning about periods of time found in descriptions of hardware and software systems. Unlike most temporal logics, ITL can handle both sequential and parallel composition and offers powerful and extensible specification and proof techniques for reasoning about properties involving safety, liveness and projected time [99]. Timing constraints are expressible and furthermore most imperative programming constructs can be viewed as formulas in a slightly modified version of ITL [25].
Interval Temporal Logic (ITL) is

- discrete
- linear temporal logic
- for (in)finite time which includes
- a basic construct for sequential composition and
- an analog of Kleene star.

3.2.1 Syntax of ITL

The key notion of ITL is an *interval*. An *interval* σ is considered to be a (in)finite sequence of states $\sigma_0,\ \sigma_1 \ldots$, where a state σ_i is a mapping from the set of variables *Var* to the set of values *Val*. The length $|\sigma|$ of an *interval* $\sigma_0 \ldots \sigma_n$ is equal to n which is one less than the number of states in the *interval* (this has always been a convention in ITL), i.e., a one state *interval* has length 0.

- Future intervals:

The semantics of *future interval* is as shown in Figure 3.1:

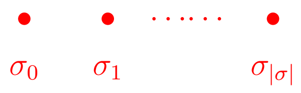

Figure 3.1: Future interval

The syntax of ITL is defined in Table 3.1 where
z is an integer value,
a is a static integer variable (does not change within an interval),
A is a state integer variable (can change within an interval),
v a static or state integer variable,
g is a integer function symbol,
q is a static Boolean variable (does not change within an interval),
Q is a state Boolean variable (can change within an interval),
p is a predicate symbol.

Expressions e ::=	$z \mid a \mid A \mid g(e_1, \ldots, e_n) \mid \bigcirc A \mid \mathsf{fin}\ A$
Formula f ::=	$\mathsf{true} \mid q \mid Q \mid p(e_1, \ldots, e_n) \mid \neg f \mid f_1 \wedge f_2 \mid \forall v.f \mid \mathsf{skip} \mid f_1 ; f_2 \mid f^*$

Table 3.1: Syntax of ITL

Expressions

The syntax is explained with some examples below:
Expressions are built inductively as follows:

- Constants (z):
  We denote Constants by letters of the form z for examples: z_0, z_1 to denote values like 0,4,9 and so on.

- Individual variables:
  - By convention, capital letters are used to denote state variables which are variables whose values can change within an interval for example $A, B, C, ...$
  - Small letters to denote static variables which are variables whose values does not change within an interval for example $a, b, c, ...$
  - Letters of the form v are used to denote a variable which can either be a static or a state variable.

- Functions :
  -$g(e_0, e_1, e_2, .., e_k)$ where $k \geq 0$ and $e_0, e_1, e_2, ..., e_k$ are expressions.
  -$+$ and mod are among common functions used.
  -Constants (such as 0,1 etc.) are treated as zero place functions.
  - Next: $\bigcirc\, e$, where e is an expression.
  - Fin: $\mathsf{fin}\ e$, where e is an expression.

Examples include: $A + B$, $a - b$, $A + a$, $v\ mod\ C$ and so on.
Some examples of syntactically legal expressions are given below:
$I + (\bigcirc\, J + 2)$
This expression adds the value of I in the current state, the value of J in the next state and the constant 2.
$I + (\bigcirc\, J) - (\bigcirc\, I)$
This expression adds the value of I in the current state to the value of J in the next state and subtracts the value of I in the next state from the result [98, 23].

Formula

Formulas are built inductively as follows:

- Predicates $p(e_0, e_1, e_2, .., e_k)$ where $k \geq 0$ and $e_0, e_1, e_2, ..., e_k$ are expressions. Predicates include $\leq$ and other basic relations.

- Equality: $e_1 = e_2$; where e_1 and e_2 are expressions.

- Logical connectives:$\neg f$ and $f_1 \wedge f_2$, where f, f_1 and f_2 are formulas.

- Universal Quantifier : $\forall v.f$ where f is formula.

- Skip: skip is true on an interval σ iff σ has length 1 (unit interval).

- Chop: $f_1 \, ; f_2$, where f_1 and f_2 are a formulas.
- Chopstar: f^*, where f is a formula.

Some examples of syntactically legal formulas are given below:
-$(J = 2) \wedge (K = 4)$
This formula states that the value of J is 2 in the current state and the value of K is 4 in the current state.
-$(I = 2) \wedge (\bigcirc J = I + 2)$
This formula states that the formula is true if I equal to 2 in the current and the value of J in the next state would be I+2.
Note that the operator $\bigcirc$ can be used both for expressions (e.g.,$\bigcirc I$) and for formulas, e.g., $\bigcirc(I = 5)$ [98, 23].

3.2.2 Semantics

The informal semantics of the most interesting constructs are as follows:

- $\bigcirc A$: if interval is non-empty then the value of A in the next state of that interval else an arbitrary value.
- $\mathsf{fin}\ A$: if interval is finite then the value of A in the final state of that interval else an arbitrary value.
- $\neg f$: f does not holds for that interval.
- $f_1 \wedge f_2$: f_1 holds for that interval and f_2 holds for that interval.
- skip unit interval (length 1).
- $f_1 \, ; f_2$ holds if the interval can be decomposed ("chopped") into a prefix and suffix interval, such that f_1 holds over the prefix and f_2 over the suffix, or if the interval is infinite and f_1 holds for that interval.
- f^* holds if the interval is decomposable into a finite number of intervals such that for each of them f holds, or the interval is infinite and can be decomposed into an infinite number of finite intervals for which f holds.

To define the formal semantics, we introduce the following notations:

- Σ denotes the set of sequences of states.

- Σ^ω denotes the set of infinite sequences of states.
- Σ^+ denotes the set of non-empty finite sequences of states.
- $\sigma_{i \to j}$ for $0 \leq i \leq j \leq |\sigma|$ denotes a subinterval $\sigma_i \sigma_{i+1} \cdots \sigma_j$.

Let $\mathcal{E}_\sigma[\![\ldots]\!]$ be the "meaning" (semantic) function from $(\Sigma^+ \cup \Sigma^\omega) \times$ *Expressions* to *Val* and let $\mathcal{M}_\sigma[\![\ldots]\!]$ be the "meaning" function from $(\Sigma^+ \cup \Sigma^\omega) \times$ *Formula* to *Bool* (set of Boolean values, $\{\mathrm{tt}, \mathrm{ff}\}$) and let $\sigma = \sigma_0 \sigma_1 \ldots$ be an interval from $(\Sigma^+ \cup \Sigma^\omega)$. We write $\sigma \sim_v \sigma'$ if the *intervals* σ and σ' are identical with the possible exception of their mappings for the variable v.
The formal semantics of ITL, except the chop and chopstar operators, is listed in Table 3.2.

$$\begin{array}{lcl}
\mathcal{E}_\sigma[\![z]\!] & = & z \\
\mathcal{E}_\sigma[\![a]\!] & = & \sigma_0(a) \text{ and for all } 0 < i \leq |\sigma|, \sigma_i(a) = \sigma_0(a) \\
\mathcal{E}_\sigma[\![A]\!] & = & \sigma_0(A) \\
\mathcal{E}_\sigma[\![g(e_1, \ldots, e_n)]\!] & = & g(\mathcal{E}_\sigma[\![e_1]\!], \ldots, \mathcal{E}_\sigma[\![e_n]\!]) \\
\mathcal{E}_\sigma[\![\bigcirc A]\!] & = & \begin{cases} \sigma_1(A) & \text{if } |\sigma| > 0 \\ \text{undefined} & \text{otherwise} \end{cases} \\
\mathcal{E}_\sigma[\![\mathsf{fin}\ A]\!] & = & \begin{cases} \sigma_{|\sigma|}(A) & \text{if } \sigma \text{ is finite} \\ \text{choose-any-from}(Val) & \text{otherwise} \end{cases} \\
\mathcal{M}_\sigma[\![\mathsf{true}]\!] & = & \mathrm{tt} \\
\mathcal{M}_\sigma[\![q]\!] & = & \sigma_0(q) \text{ and for all } 0 < i \leq |\sigma|, \sigma_i(q) = \sigma_0(q) \\
\mathcal{M}_\sigma[\![Q]\!] & = & \sigma_0(Q) \\
\mathcal{M}_\sigma[\![p(e_1, \ldots, e_n)]\!] = \mathrm{tt} & \text{iff} & p(\mathcal{E}_\sigma[\![e_1]\!], \ldots, \mathcal{E}_\sigma[\![e_n]\!]) \\
\mathcal{M}_\sigma[\![\neg f]\!] = \mathrm{tt} & \text{iff} & \text{not } (\mathcal{M}_\sigma[\![f]\!] = \mathrm{tt}) \\
\mathcal{M}_\sigma[\![f_1 \wedge f_2]\!] = \mathrm{tt} & \text{iff} & (\mathcal{M}_\sigma[\![f_1]\!] = \mathrm{tt}) \text{ and } (\mathcal{M}_\sigma[\![f_2]\!] = \mathrm{tt}) \\
\mathcal{M}_\sigma[\![\mathsf{skip}]\!] = \mathrm{tt} & \text{iff} & |\sigma| = 1 \\
\mathcal{M}_\sigma[\![\forall \boldsymbol{v} \cdot f]\!] = \mathrm{tt} & \text{iff} & (\text{for all } \sigma' \text{ s.t. } \sigma \sim_v \sigma', \mathcal{M}_\sigma[\![f]\!] = \mathrm{tt})
\end{array}$$

Table 3.2: Semantics of ITL

- Chop (;):

The semantics of chop (;) is as follows:
$\mathcal{M}_\sigma[\![f_1 \; ; \; f_2]\!] = \mathrm{tt}$ iff

exists k, such that $0 \leq k \leq |\sigma|$, and
$\mathcal{M}_{\sigma_0 \to \sigma_k}[\![f_1]\!] = \text{tt}$ and $\mathcal{M}_{\sigma_k \to \sigma_{|\sigma|}}[\![f_2]\!] = \text{tt}$.
Future interval σ is a fusion of two future intervals, the first interval $\sigma_0 \ldots \sigma_k$ satisfies f_1 and the second interval $\sigma_k \ldots \sigma_{|\sigma|}$ satisfies f_2

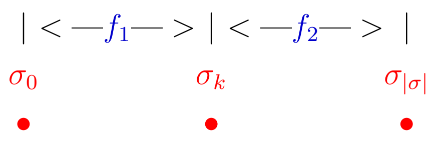

Figure 3.2: Chop of finite interval

or the interval is infinite and
$\mathcal{M}_{\sigma}[\![f_1]\!] = \text{tt}$
Future interval σ is infinite and satisfies f, so f_2 is irrelevant.

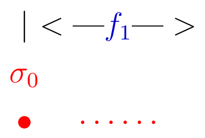

Figure 3.3: Chop of infinite interval

- Chopstar (f^*):

The semantics of chopstar (f^*) is as follows:
$\mathcal{M}_{\sigma}[\![f^*]\!] = \text{tt}$ iff
if σ is finite then exists $l_0, \ldots. l_n$, such that $l_0 = 0 \quad \wedge \; l_n = |\sigma|$
$\wedge$ for all $0 \leq i < n$, $l_i \leq l_{i+1} \wedge \mathcal{M}_{\sigma_{l_i} \to \sigma_{l_{i+1}}}[\![f]\!] \ldots = \text{tt}$.
Finite future interval σ is the fusion of a finite number of finite intervals in the future each satisfying f.

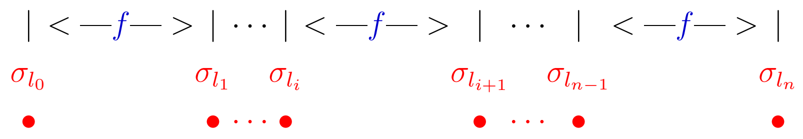

Figure 3.4: Chopstar of finite interval

Else exists l_0,....l_n, such that $l_0 = 0 \quad \wedge \mathcal{M}_{\sigma_{l_n} \rightarrow \sigma_{|\sigma|}}[\![f]\!] = \text{tt}$
$\wedge$ for all $0 \leq i < n$, $l_i \leq l_{i+1} \wedge \mathcal{M}_{\sigma_{l_i} \rightarrow \sigma_{l_{i+1}}}[\![f]\!] \ldots = \text{tt}$.
Infinite future interval σ is the fusion of a finite number of sub-intervals in the future each satisfying f.
Each future sub-interval is finite except the last one which is infinite.

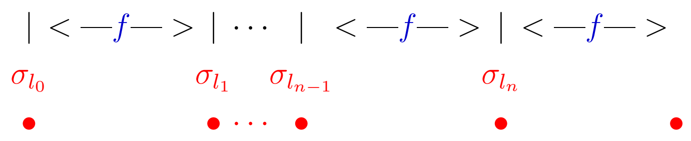
$| <—f—> | \cdots \quad | \quad <—f—> | <—f—>$
σ_{l_0}
σ_{l_1}
$\sigma_{l_{n-1}}$
σ_{l_n}
$\bullet \quad \bullet \cdots \bullet \quad \bullet \quad \bullet$

Figure 3.5: Chopstar of finite interval final infinite

or there exists an infinite number of l_i such that $l_0 = 0 \wedge$ for all $0 \leq i$,$l_i \leq l_{i+1}$ and $\mathcal{M}_{\sigma_{l_i} \rightarrow \sigma_{l_{i+1}}}[\![f]\!] = \text{tt}$.

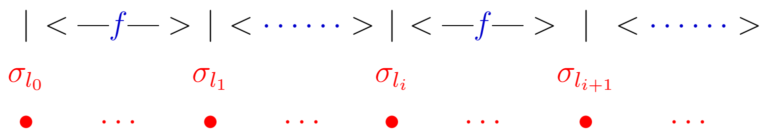
$| <—f—> | < \cdots\cdots > | <—f—> \quad | \quad < \cdots\cdots >$
σ_{l_0}
σ_{l_1}
σ_{l_i}
$\sigma_{l_{i+1}}$
$\bullet \cdots \bullet \cdots \bullet \cdots \bullet \cdots$

Figure 3.6: Chopstar of infinite interval

Infinite future interval σ is the fusion of an infinite number of finite sub intervals in the future each one satisfying f.

- True (true):

$\mathcal{M}_\sigma[\![\mathsf{true}]\!] = \text{tt}$.

- Skip (skip):

$\mathcal{M}_\sigma[\![\mathsf{skip}]\!] = \text{tt}$ iff $|\sigma| = 1$.
Skip (skip), is an interval with only two states.

- Next ($\bigcirc f$):

$$\bigcirc f \mathrel{\widehat{=}} \mathsf{skip} ; f$$

$\mathcal{M}_{\sigma}[\![\bigcirc f]\!] = \text{tt iff } |\sigma| > 0 \wedge \mathcal{M}_{\sigma_1 \rightarrow \sigma_{|\sigma|}}[\![f]\!] \ldots = \text{tt}.$
Next ($\bigcirc f$) holds in the next state of the future interval:

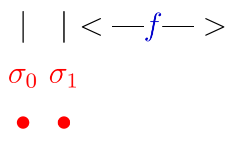

Figure 3.7: Next

- Weak Next ($\textcircled{w} f$):

$$\textcircled{w} f \mathrel{\widehat{=}} \neg \bigcirc \neg f$$

$\mathcal{M}_{\sigma}[\![\textcircled{w} f]\!] = \text{tt iff } |\sigma| = 0 \vee \mathcal{M}_{\sigma_1 \rightarrow \sigma_{|\sigma|}}[\![f]\!] \ldots = \text{tt}.$
Weak Next ($\textcircled{w} f$), holds if the future interval has only one state:

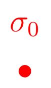

Figure 3.8: Weak Next with one state interval

or f holds in the next state of the future interval:

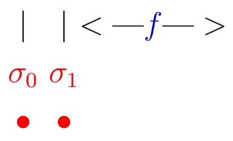

Figure 3.9: Weak Next with more than one state interval

- Sometimes (◇f):

$$\Diamond f \mathrel{\widehat{=}} \textsf{finite} \,;\, f$$

$\mathcal{M}_{\sigma}[\![\Diamond f]\!] = \text{tt}$ iff exists k where $0 \leq k \leq |\sigma|$, such that $\mathcal{M}_{\sigma_k \rightarrow \sigma_{|\sigma|}}[\![f]\!] = \text{tt}$.
Sometime (◇f), holds if there exists a suffix interval in the future satisfying f.

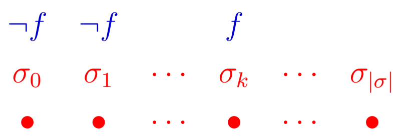

Figure 3.10: Sometimes

- Always (□f):

$$\Box f \mathrel{\widehat{=}} \neg\Diamond\neg f$$

$\mathcal{M}_{\sigma}[\![\Box f]\!] = \text{tt}$ iff for all $0 \leq k \leq |\sigma|$, such that $\mathcal{M}_{\sigma_k \rightarrow \sigma_{|\sigma|}}[\![f]\!] = \text{tt}$.
Always (□f), holds if all the suffix intervals in the future are satisfying f.

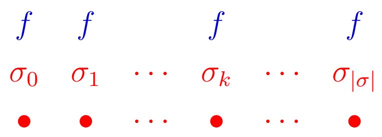

Figure 3.11: Always

- More (more):

$$\textsf{more} \mathrel{\widehat{=}} \bigcirc\,\textsf{true}$$

$\mathcal{M}_{\sigma}[\![\textsf{more} f]\!] = \text{tt}$ iff $|\sigma| > 0$.
More (more) holds if there is a future interval with at least two states.

- Empty (empty):

$$\textsf{empty} \mathrel{\widehat{=}} \neg\textsf{more}$$

$\mathcal{M}_{\sigma}[\![\mathsf{empty}]\!] = \text{tt iff } \ |\sigma| = 0.$
Empty (empty) holds if there is a future interval with only one state.

- Infinite (inf)**:**

$$\mathsf{inf} \mathrel{\widehat{=}} \mathsf{true} \; ; \mathsf{false}$$

$\mathcal{M}_{\sigma}[\![\mathsf{inf}]\!] = \text{tt iff } \ \sigma \text{ is infinite.}$
inf holds if there is a future interval with an infinite number of states.

- Finite(finite)**:**

$$\mathsf{finite} \mathrel{\widehat{=}} \neg\mathsf{inf}$$

$\mathcal{M}_{\sigma}[\![\mathsf{finite}]\!] = \text{tt iff } \ \sigma \text{ is finite.}$
finite holds if there is a future interval with a finite number of states.

- Diamond-i ($\Diamond\mkern-13mu{\scriptstyle i}\mkern6mu f$)**:**

$$\Diamond\mkern-13mu{\scriptstyle i}\mkern6mu f \mathrel{\widehat{=}} f \; ; \mathsf{true}$$

$\mathcal{M}_{\sigma}[\![\Diamond\mkern-13mu{\scriptstyle i}\mkern6mu f]\!] = \text{tt iff}$
exists k, such that $0 \leq k \leq |\sigma|$, and $\mathcal{M}_{\sigma_0 \to \sigma_k}[\![f]\!] = \text{tt}$.
Diamond-i ($\Diamond\mkern-13mu{\scriptstyle i}\mkern6mu f$) holds if there exists a prefix interval in the future that satisfies f.

- Box-i ($\boxed{\scriptstyle i}\, f$)**:**

$$\boxed{\scriptstyle i}\, f \mathrel{\widehat{=}} \neg(\Diamond\mkern-13mu{\scriptstyle i}\mkern6mu \neg f)$$

$\mathcal{M}_{\sigma}[\![\Diamond\mkern-13mu{\scriptstyle i}\mkern6mu f]\!] = \text{tt iff}$
for all k, that $0 \leq k \leq |\sigma|$, and $\mathcal{M}_{\sigma_0 \to \sigma_k}[\![f]\!] = \text{tt}$.
Box-i ($\boxed{\scriptstyle i}\, f$) holds if all the prefix intervals in the future are satisfying f.

- Diamond-a ($\Diamond\mkern-13mu{\scriptstyle a}\mkern6mu f$)**:**

$$\Diamond\mkern-13mu{\scriptstyle a}\mkern6mu f \mathrel{\widehat{=}} \mathsf{finite} \; ; f \; ; \mathsf{true}$$

$\mathcal{M}_{\sigma}[\![\Diamond\mkern-13mu{\scriptstyle a}\mkern6mu f]\!] = \text{tt iff}$
exists l, k, such that $0 \leq l \leq k \leq |\sigma|$, and $\mathcal{M}_{\sigma_l \to \sigma_k}[\![f]\!] = \text{tt}$.
Diamond-a ($\Diamond\mkern-13mu{\scriptstyle a}\mkern6mu f$) holds if there exists a sub interval in the future that satisfies f.

- Box-a ($\boxed{\scriptstyle a}\, f$):

$$\boxed{\scriptstyle a}\, f \mathrel{\widehat{=}} \neg(\Diamond\mkern-12mu{\scriptstyle a}\mkern6mu \neg f)$$

$\mathcal{M}_{\sigma}[\![\Diamond\mkern-12mu{\scriptstyle a}\mkern6mu f]\!] = \text{tt}$ iff
for all l, k, that $0 \leq l \leq k \leq |\sigma|$, and $\mathcal{M}_{\sigma_l \rightarrow \sigma_k}[\![f]\!] = \text{tt}$.
Box-a ($\boxed{\scriptstyle a}\, f$) holds if all the sub intervals in the future are satisfying f.

3.2.3 Derived formula

Now, we are using the basic operators such as ; and skip and true to derive and define a new formula, in order to help us in formulating and constructing a logical argument or proof.
The common derived formula listed in Table 3.3 is as follow:

false	$\widehat{=}$	$\neg$true	false value
$\bigcirc f$	$\widehat{=}$	skip ; f	next
$\textcircled{w}\, f$	$\widehat{=}$	$\neg \bigcirc \neg f$	weak next
more	$\widehat{=}$	$\bigcirc$ true	interval with ≥ 2 states
empty	$\widehat{=}$	$\neg$more	one state interval
inf	$\widehat{=}$	true ; false	infinite interval
finite	$\widehat{=}$	$\neg$inf	finite interval
$\Diamond f$	$\widehat{=}$	finite ; f	sometimes in the future
$\Box f$	$\widehat{=}$	$\neg \Diamond \neg f$	always in the future
$\Diamond\mkern-12mu{\scriptstyle i}\mkern6mu f$	$\widehat{=}$	f ; true	some initial future subinterval
$\boxed{\scriptstyle i}\, f$	$\widehat{=}$	$\neg(\Diamond\mkern-12mu{\scriptstyle i}\mkern6mu \neg f)$	all initial future subintervals
$\Diamond\mkern-12mu{\scriptstyle a}\mkern6mu f$	$\widehat{=}$	finite ; f ; true	some subinterval
$\boxed{\scriptstyle a}\, f$	$\widehat{=}$	$\neg(\Diamond\mkern-12mu{\scriptstyle a}\mkern6mu \neg f)$	all subintervals

Table 3.3: Derived formula

Frequently used concrete derived constructs

In this part, the concrete derived constructs are introduced in Table 3.4 as follow:

if f_0 then f_1 else f_2	$\widehat{=}$	$(f_0 \wedge f_1) \vee (\neg f_0 \wedge f_2)$	if then else
if f_0 then f_1	$\widehat{=}$	if f_0 then f_1 else true	if then
fin f	$\widehat{=}$	$\Box$(empty $\supset f$)	final state
halt f	$\widehat{=}$	$\Box$(empty $\equiv f$)	terminate interval when
keep f	$\widehat{=}$	$\boxed{a}$(skip $\supset f$)	all unit subintervals
while f_0 do f_1	$\widehat{=}$	$(f_0 \wedge f_1)^* \wedge$ fin $\neg f_0$	while loop
repeat f_0 until f_1	$\widehat{=}$	f_0 ; (while $\neg f_1$ do f_0)	repeat loop

Table 3.4: Frequently used concrete derived constructs

Frequently used derived constructs related to expressions

In this part, the derived constructs related to expressions are introduced in Table 3.5 as follow:

$A := exp$	$\widehat{=}$	$\bigcirc A = exp$	assignment
$A \approx exp$	$\widehat{=}$	$\Box(A = exp)$	equal in interval
$A \leftarrow exp$	$\widehat{=}$	finite $\wedge$ (fin A) $= exp$	temporal assignment
A gets exp	$\widehat{=}$	keep $(A \leftarrow exp)$	gets
stable A	$\widehat{=}$	A gets A	stability
len(exp)	$\widehat{=}$	$\exists \boldsymbol{I} \cdot (I = 0) \wedge (I$ gets $I + 1) \wedge (I \leftarrow exp)$	interval length

Table 3.5: Frequently used derived constructs related to expressions

3.3 Past Time Interval Temporal Logic

Past Time Interval Temporal Logic (ITL^p) is a flexible notation that uses past time operators only. Past Time Interval Temporal Logic is the main contribution of this thesis, therefore what are the differences between past time interval temporal logic and future time interval temporal logic. Next, we will answer this question and discuss the main differences between ITL^p and ITL.

3.3.1 ITL^p versus ITL

There are many differences between ITL^p and ITL, the first one is that ITL^p uses the past constructs only while, ITL uses future time constructs. The second difference is that past time operators are interpreted over a finite interval because the history interval is assumed to be finite. However, ITL can be interpreted over finite and infinite interval. A part of the contribution introduced in this thesis is that the introduced past time operators change the “numbering of states”; that is the current state of the past interval is the most rights of this interval τ_0 and the remaining states are $\tau_{|\tau|}$....., τ_2, τ_1 as has been shown in the Figure 3.12:

$\tau_{|\tau|}$
τ_1
τ_0

Figure 3.12: States in past interval

However, the “numbering of states” of the future interval is from left to right so, the current state is σ_0 and the remaining states are $\sigma_1, \sigma_2, ...\sigma_{|\sigma|}$ or graphically as has been shown in the Figure 3.13:

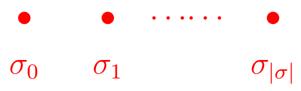

Figure 3.13: States in future interval

This reflects on the interpretation of ITL^p formula being from right to left which opposes the interpretation of ITL formula which is from left to right.

Finally, if the interval has expanded, there is a new state added σ_4 to the interval then, in the future interval, only the final state is changed, so, states σ_0, σ_1, σ_2 and σ_3 do not change as shown in Figure 3.14.

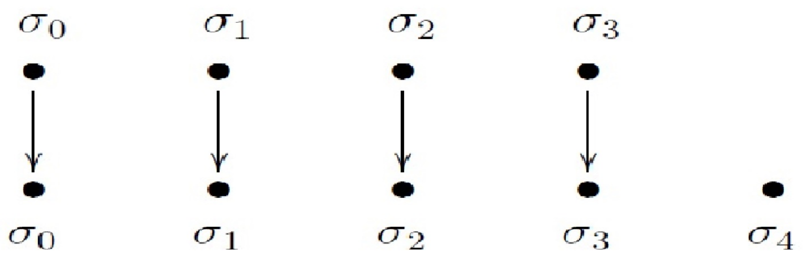

Figure 3.14: Future changed states

However if the past interval has expanded and a new state τ_0 is added, all the states in this interval will be changed, state τ_0 in the top of the Figure is changed to τ_1 in the bottom of the Figure and so on, state τ_1 is changed to τ_2 in the expanded interval, so the interval states τ_0, τ_1, τ_2 and τ_3 are changed to τ_1, τ_2, τ_3 and τ_4 respectively as shown in Figure 3.15.

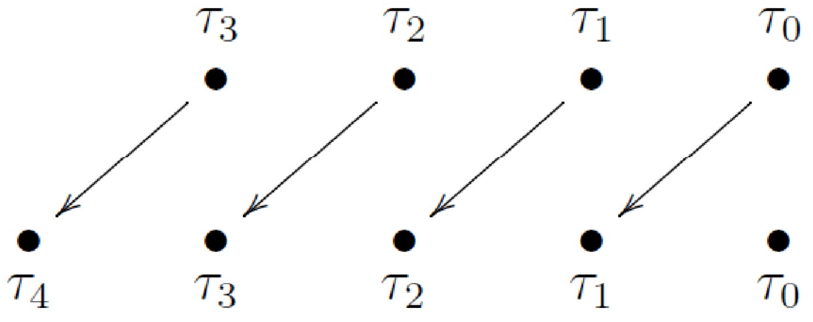

Figure 3.15: Past changed states

Past time interval temporal logic ITL^p is:

- discrete
- linear temporal logic
- for history finite time which includes
- a basic construct for sequential composition and
- an analog of Kleene star

3.3.2 Syntax of ITL^p

The key notion of ITL^p is a history interval. A history interval τ is considered to be a finite sequence of states, $\tau_0, \tau_1, \ldots \tau_n$ where a state τ_i is a mapping from the set of variables Var to the set of values Val. The length $|\tau|$ of an interval $\tau_0 \ldots \tau_n$ is equal to n (one less than the number of states in the history interval, i.e., a one state history interval has length 0).
The semantics of historical interval is shown in in Figure 3.16:

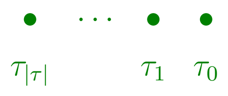

Figure 3.16: Historical interval

The syntax of ITL^p is defined in Table 3.3 where:
z is an integer value,
a is a static integer variable (does not change within an interval),
A is a state integer variable (can change within an interval),
v a static or state integer variable,
g is a integer function symbol,
q is a static Boolean variable (does not change within an interval),
Q is a state Boolean variable (can change within an interval),
p is a predicate symbol.

Expressions e ::=	z |	a |	A |	$g(e_1, \ldots, e_n)$|	$\widehat{\bigcirc} A$ |	$\widehat{\mathsf{fin}}\ A$				
Formula f ::=	$\widehat{\mathsf{true}}$|	q |	Q |	$p(e_1, \ldots, e_n)$|	$\neg h$|	$h_1 \wedge h_2$|	$\forall v.h$|	$\widehat{\mathsf{skip}}$|	$h_2 \,\hat{;}\, h_1$|	$h^{\hat{*}}$

Table 3.6: Syntax of ITL^p

3.3.3 Semantics of ITL^p

The informal semantics of the most interesting constructs are as follows:

- $\widehat{\bigcirc}$ A: if the history interval is non-empty then the value of A in the previous state of that history interval has an arbitrary value.

- $\widehat{\mathsf{fin}}\ A$: The value of A in the final state of that history interval.

- $\widehat{\mathsf{skip}}$: is a history interval (sequence) of 2 states.

- $h_1 \hat{;} h_2$: is called 'h_1 past chop h_2' and denotes sequential composition of two history intervals, i.e., $h_1 \hat{;} h_2$ holds if the interval can be decomposed ("chopped") into a prefix and suffix interval, such that h_1 holds over the prefix and h_2 holds over the suffix.

- $h^{\hat{*}}$: is called 'h past chopstar' and denotes finite iteration of a history interval, i.e., $h^{\hat{*}}$ holds if the interval is decomposable into a finite number of intervals such that for each of them h holds.

In order to define the formal semantics, the following notions are introduced:

- Δ denotes the set of states.
- Δ^+ denotes the set of history non-empty finite sequences of states.
- τ is a history interval, $\tau \in \Delta^+$.
- $|\tau|$ denotes length of τ and is defined as number of states minus 1.
- $\tau_{j \leftarrow i}$ for $0 \leq i \leq j \leq |\tau|$ denotes a history subinterval $\tau_j \cdots \tau_{i+1}\tau_i$.

Let $\mathcal{E}_\tau[\![\ldots]\!]$ be the "meaning" (semantic) function from Val to $(Expressions \times \Delta^+)$ and let $\mathcal{M}_\tau[\![\ldots]\!]$ be the "meaning" function from $Formula \times \Delta^+$ to $Bool$ (set of Boolean values, $\{\text{tt}, \text{ff}\}$) and let $\tau = \tau_{|\tau|} \ldots \tau_0$ be a history interval.

We write $\tau \sim_v \tau^{'}$ if the intervals are identical with the possible exception of their mappings for the variable.
The formal semantics of ITL^p, except the past chop and past chopstar, are listed in Table 3.7.

$\mathcal{E}_\tau[\![z]\!]$	$=$	z
$\mathcal{E}_\tau[\![a]\!]$	$=$	$\tau_0(a)$ and for all $0 < i \leq \lvert\tau\rvert, \tau_i(a) = \tau_0(a)$
$\mathcal{E}_\tau[\![A]\!]$	$=$	$\tau_0(A)$
$\mathcal{E}_\tau[\![g(e_1, \ldots, e_n)]\!]$	$=$	$g(\mathcal{E}_\tau[\![e_1]\!], \ldots, \mathcal{E}_\tau[\![e_n]\!])$
$\mathcal{E}_\tau[\![\widehat{\bigcirc}\, A]\!]$	$=$	$\begin{cases} \tau_1(A) & \text{if } \lvert\tau\rvert > 0 \\ \text{undefined} & \text{otherwise} \end{cases}$
$\mathcal{E}_\tau[\![\widehat{\mathsf{fin}}\ A]\!]$	$=$	$\tau_{\lvert\tau\rvert}(A)$
$\mathcal{M}_\tau[\![\widehat{\mathsf{true}}]\!]$	$=$	tt
$\mathcal{M}_\tau[\![q]\!]$	$=$	$\tau_0(q)$ and for all $0 < i \leq \lvert\tau\rvert, \tau_i(q) = \tau_0(q)$
$\mathcal{M}_\tau[\![Q]\!]$	$=$	$\tau_0(Q)$
$\mathcal{M}_\tau[\![p(e_1, \ldots, e_n)]\!] = \text{tt}$	iff	$p(\mathcal{E}_\tau[\![e_1]\!], \ldots, \mathcal{E}_\tau[\![e_n]\!])$
$\mathcal{M}_\tau[\![\neg h]\!] = \text{tt}$	iff	not $(\mathcal{M}_\tau[\![h]\!] = \text{tt})$
$\mathcal{M}_\tau[\![h_1 \wedge h_2]\!] = \text{tt}$	iff	$(\mathcal{M}_\tau[\![h_1]\!] = \text{tt})$ and $(\mathcal{M}_\tau[\![h_2]\!] = \text{tt})$
$\mathcal{M}_\tau[\![\widehat{\mathsf{skip}}]\!] = \text{tt}$	iff	$\lvert\tau\rvert = 1$
$\mathcal{M}_\tau[\![\forall \boldsymbol{v} \cdot h]\!] = \text{tt}$	iff	(for all τ' s.t. $\tau \sim_v \tau', \mathcal{M}_\tau[\![h]\!] = \text{tt}$)

Table 3.7: Semantics of ITL^p

-Past Chop ($\hat{;}$)

The semantics of past Chop ($\hat{;}$) is as follows:

$$\mathcal{M}_\tau[\![h_2 \,\hat{;}\, h_1]\!] = \text{tt iff}$$

exists k where$0 \leq k \leq \lvert\tau\rvert$, s.t. $\mathcal{M}_{\tau_{\lvert\tau\rvert} \leftarrow \tau_k}[\![h_2]\!] = \text{tt}$ and $\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_1]\!] = \text{tt}$.

History interval τ is finite, then interval τ is a fusion of two past intervals, the first interval $\tau_0 \ldots \tau_k$satisfies h_1 and the second interval $\tau_k \ldots \tau_{\lvert\tau\rvert}$ satisfies h_2.
State τ_k is shared by both.

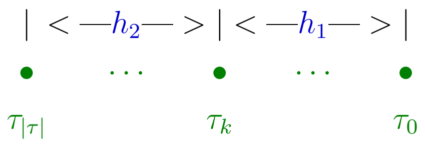

Figure 3.17: Semantics of past Chop

- Past Chopstar($h\ \hat{*}$)

The semantics of past Chopstar($h\ \hat{*}$) is as follows:
$\mathcal{M}_\tau[\![h^{\hat{*}}]\!]$ = tt iff
τ is finite then exists l_0,....l_n, such that $l_0 = 0 \quad \wedge\ l_n = |\tau|$
$\wedge$ for all $0 \le i < n$, $l_i \le l_{i+1} \wedge \mathcal{M}_{\tau_{l_i} \to \tau_{l_{i+1}}}[\![h]\!] \ldots =$ tt.
Finite past interval τ is the fusion of a finite number of finite past intervals each satisfying h.

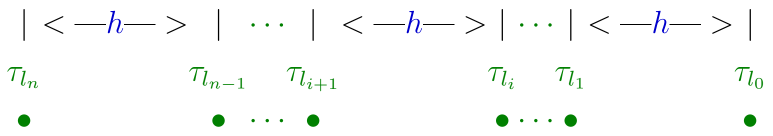

Figure 3.18: Semantics of past Chopstar

- Past Skip ($\widehat{\mathsf{skip}}$):

$\mathcal{M}_\tau[\![\widehat{\mathsf{skip}}]\!]$ = tt iff $|\tau| = 1$.
Past Skip ($\widehat{\mathsf{skip}}$) is a past interval with only two states.

- Previous ($\widehat{\bigcirc}\ h$):

$$\widehat{\bigcirc}\, h \mathrel{\widehat{=}} h \mathbin{\hat{;}} \widehat{\mathsf{skip}}$$

$\mathcal{M}_\tau[\![\widehat{\bigcirc}\, h]\!]$ = tt iff $0 < |\tau| \wedge \mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_1}[\![h]\!] \ldots =$ tt.

Previous ($\widehat{\bigcirc} h$) holds if the previous state of the past interval holds:

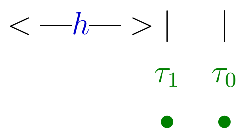

Figure 3.19: Previous

- Weak Previous ($\widehat{\textcircled{w}}$ h):

$$\widehat{\textcircled{w}} h \mathrel{\widehat{=}} \neg \widehat{\bigcirc} \neg h$$

$\mathcal{M}_\tau[\![\widehat{\textcircled{w}} h]\!] = \text{tt}$ iff $|\tau| = 0$.
Weak Previous ($\widehat{\textcircled{w}} h$) holds if the past interval has only one state.

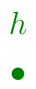

Figure 3.20: Weak Previous with one state interval

$\vee\ \mathcal{M}_{\tau_{|\tau|}\leftarrow\tau_1}[\![h]\!] \ldots = \text{tt}$.
or h holds in the previous state of the past interval:

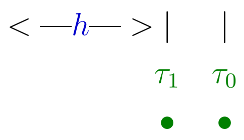

Figure 3.21: Weak Previous with more than one state interval

- Past Sometimes ($\widehat{\Diamond}$ h):

$$\widehat{\Diamond} h \mathrel{\widehat{=}} h \mathbin{\widehat{;}} \widehat{\mathsf{finite}}$$

$\mathcal{M}_\tau[\![\widehat{\Diamond} h]\!] = \text{tt}$ iff exists k where $0 \leq k \leq |\tau|$, such that $\mathcal{M}_{\tau_{|\tau|}\leftarrow\tau_k}[\![h]\!] = \text{tt}$.
Past Sometime ($\widehat{\Diamond}$ h) holds if there exists a suffix interval in the past which is satisfying h.

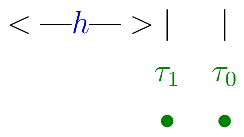

Figure 3.22: Past Sometime

- Past Always ($\widehat{\Box}$h):

$$\widehat{\Box}\, h \mathrel{\widehat{=}} \neg\widehat{\Diamond}\neg h$$

$\mathcal{M}_{\tau}[\![\widehat{\Box}\, h]\!] = \text{tt iff for all } 0 \leq k \leq |\tau|, \text{such that } \mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h]\!] = \text{tt}.$
Past Always ($\widehat{\Box}$f) holds if all the suffix intervals in the past are satisfying h.

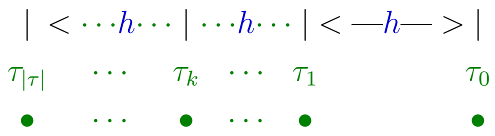

Figure 3.23: Past Always

- Past More ($\widehat{\mathsf{more}}$):

$$\widehat{\mathsf{more}} \mathrel{\widehat{=}} \widehat{\bigcirc}\ \mathsf{true}$$

$\mathcal{M}_{\tau}[\![\widehat{\mathsf{more}}]\!] = \text{tt iff } |\tau| > 0.$
Past More ($\widehat{\mathsf{more}}$) holds if there is a past interval with at least two states.

- Past Empty ($\widehat{\mathsf{empty}}$):

$$\widehat{\mathsf{empty}} \mathrel{\widehat{=}} \neg\widehat{\mathsf{more}}$$

$\mathcal{M}_{\tau}[\![\widehat{\mathsf{empty}}]\!] = \text{tt iff } |\tau| = 0.$
Past Empty ($\widehat{\mathsf{empty}}$) holds if there is a past interval with only one state.

- Past Diamond-i ($\widehat{\Diamond}_i\, h$):

$$\widehat{\Diamond}_i\, h \mathrel{\widehat{=}} \mathsf{true}\, \widehat{;}\, h$$

$\mathcal{M}_{\tau}[\![\hat{\Diamond}_{i}\, h]\!] = \text{tt iff}$
exists k, such that $0 \leq k \leq |\tau|$, and $\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h]\!] = \text{tt}$.
Past Diamond-i ($\hat{\Diamond}_{i}\, h$) holds if there exists a prefix interval in the past that satisfies h.

- Past Box-i ($\hat{\Box}_{i}\, h$):

$$\hat{\Box}_{i}\, h \mathrel{\widehat{=}} \neg(\hat{\Diamond}_{i}\, \neg h)$$

$\mathcal{M}_{\tau}[\![\hat{\Box}_{i}\, h]\!] = \text{tt iff}$
for all k, that $0 \leq k \leq |\tau|$, and $\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h]\!] = \text{tt}$.
Past Box-i ($\hat{\Box}_{i}\, h$) holds if all the prefix intervals in the past are satisfying h.

- Past Diamond-a ($\hat{\Diamond}_{a} h$):

$$\hat{\Diamond}_{a} h \mathrel{\widehat{=}} \text{true} \mathbin{\hat{;}} h \mathbin{\hat{;}} \text{true}$$

$\mathcal{M}_{\sigma}[\![\hat{\Diamond}_{a} h]\!] = \text{tt iff}$
(exists l, k, such that $0 \leq l \leq k \leq |\tau|$, and $\mathcal{M}_{\tau_k \leftarrow \tau_l}[\![h]\!] = \text{tt}$).
Past Diamond-a ($\hat{\Diamond}_{a} h$, holds if there exists a sub interval in the past that satisfies h.

- Past Box-a ($\hat{\Box}_{a} h$):

$$\hat{\Box}_{a} h \mathrel{\widehat{=}} \neg(\hat{\Diamond}_{a} \neg h)$$

$\mathcal{M}_{\tau}[\![\hat{\Box}_{a} h]\!] = \text{tt iff}$
(for all l, k, that $0 \leq l \leq k \leq |\tau|$, and $\mathcal{M}_{\tau_k \leftarrow \tau_l}[\![h]\!] = \text{tt}$).
Past Box-a ($\hat{\Box}_{a} h$) holds if all the sub intervals in the past are satisfying h.

$$\hat{\Box}_{a} h \mathrel{\widehat{=}} \neg(\hat{\Diamond}_{a} \neg h)$$

- Past halt $\widehat{\mathsf{halt}\ h}$:

This operator can be used, in the form $\widehat{\mathsf{halt}\ h}$, to specify that a formula h becomes true only at the end of the past interval.

$$\widehat{\mathsf{halt}\ h} \mathrel{\widehat{=}} \widehat{\Box}(\widehat{\mathsf{empty}} \equiv h)$$

- Past Fin ($\widehat{\mathsf{fin}}\ h$):

$$\widehat{\mathsf{fin}}\ h \mathrel{\widehat{=}} \widehat{\Box}(\widehat{\mathsf{empty}} \supset h)$$

The value of h in the final state of that history interval.

3.3.4 Derived formula

A list of ITL derived formula has been shown in Table 3.3. However, in this Section, the derived formula for ITL^p will be introduced and listed in Table 3.8.

$\widehat{\bigcirc}\, h$	$\widehat{=}$	$h \mathbin{\hat{;}} \widehat{\mathsf{skip}}$	previous
$\widehat{\textcircled{w}} h$	$\widehat{=}$	$\neg \widehat{\bigcirc} \neg h$	weak previous
$\widehat{\mathsf{more}}$	$\widehat{=}$	$\widehat{\bigcirc}\, \mathsf{true}$	history interval with ≥ 2 states
$\widehat{\mathsf{empty}}$	$\widehat{=}$	$\neg \widehat{\mathsf{more}}$	one state history interval
$\widehat{\Diamond} h$	$\widehat{=}$	$h \mathbin{\hat{;}} \mathsf{true}$	all history unit subintervals
$\widehat{\Box}\, h$	$\widehat{=}$	$\neg \widehat{\Diamond} \neg h$	always in the history
$\widehat{\Diamond_{i}}\, h$	$\widehat{=}$	$\mathsf{true} \mathbin{\hat{;}} h$	some initial history subinterval
$\widehat{\boxed{i}}\, h$	$\widehat{=}$	$\neg(\widehat{\Diamond_{i}} \neg h)$	all initial history subintervals
$\widehat{\Diamond_{a}} h$	$\widehat{=}$	$\mathsf{true} \mathbin{\hat{;}} h \mathbin{\hat{;}} \mathsf{true}$	some history subinterval
$\widehat{\boxed{a}} h$	$\widehat{=}$	$\neg(\widehat{\Diamond_{a}} \neg h)$	all history subintervals

Table 3.8: Derived formula for ITL^p

The frequently used concrete derived constructs

The frequently used concrete derived constructs is shown in Table 3.9:

$\mathsf{if}\ w_0\ \mathsf{then}\ h_1\ \mathsf{else}\ h_2$	$\widehat{=}$	$(w_0 \wedge h_1) \vee (\neg w_0 \wedge h_2)$	if then else
$\mathsf{if}\ w_0\ \mathsf{then}\ h_1$	$\widehat{=}$	$\mathsf{if}\ w_0\ \mathsf{then}\ h_1\ \mathsf{else}\ \widehat{\mathsf{true}}$	if then
$\widehat{\mathsf{fin}}\ h$	$\widehat{=}$	$\widehat{\Box}(\widehat{\mathsf{empty}} \supset h)$	final state
$\widehat{\mathsf{halt}}\, h$	$\widehat{=}$	$\widehat{\Box}(\widehat{\mathsf{empty}} \equiv h)$	terminate interval when
$\widehat{\mathsf{keep}}\, h$	$\widehat{=}$	$\widehat{\boxed{a}}(\widehat{\mathsf{skip}} \supset h)$	all unit subintervals
$\widehat{\mathsf{while}}\, w_0 \widehat{\mathsf{do}} h_1$	$\widehat{=}$	$\widehat{\mathsf{fin}}\ \neg w_0 \wedge (w_0 \wedge h_1)^{\hat{*}}$	while loop
$\widehat{\mathsf{repeat}}\, h_0\ \mathsf{until}\ h_1$	$\widehat{=}$	$(\widehat{\mathsf{while}} \neg h_1\ \mathsf{do}\ h_0) \mathbin{\hat{;}} h_0$	repeat loop

Table 3.9: Frequently used concrete derived constructs

Frequently used derived constructs related to expressions

The frequently used concrete derived constructs related to expressions is shown in Table 3.10:

$A =: exp$	$\widehat{=}$	$\widehat{\bigcirc}\, A = exp$	past assignment
$A \widehat{\approx}\ exp$	$\widehat{=}$	$\widehat{\Box}(A = exp)$	equal in history interval
$A \widehat{\leftarrow}\ exp$	$\widehat{=}$	$\widehat{\mathsf{fin}}\ A = exp$	history temporal assignment
$A \widehat{\mathsf{gets}}\ exp$	$\widehat{=}$	$\widehat{\mathsf{keep}}\,(A \widehat{\leftarrow}\ exp)$	past gets
$\widehat{\mathsf{stable}}\, A$	$\widehat{=}$	$A\ \widehat{\mathsf{gets}}\ A$	stability
$\widehat{\mathsf{len}}(exp)$	$\widehat{=}$	$\exists \boldsymbol{I} \bullet (I = 0) \wedge (I \widehat{\mathsf{gets}} I + 1) \wedge (I \widehat{\leftarrow} exp)$	interval length

Table 3.10: Frequently used derived constructs related to expressions

3.4 Relation between past and future time using time reversal

We have shown in Section 3.3.1 the differences between the future time interval temporal logic (ITL) and the past time interval temporal logic (ITL^p); however, we need to know the relation between these two versions of interval temporal logic. To answer this question, we should introduce the time reversal which was used by Moszkowski to verify certain properties expressed in PITL [102].
Time reversal is related to mirror images [116] used for temporal logics to obtain a rule for past-time operators from an analogous one for future-time operators by means of time symmetry [24]. To explain this, let formula f^r denotes the time reversed version of f. Let the time reversed interval of a finite interval σ be denoted by $reverse(\sigma)$ and be defined as:

$$reverse(\sigma_0 \cdots \sigma_{|\sigma|}) \mathrel{\widehat{=}} \sigma_{|\sigma|} \cdots \sigma_0$$

The semantics of time reversal is defined as

$$[\![f^r]\!]_\sigma = \text{tt iff } [\![f]\!]_{reverse(\sigma)} = \text{tt}$$

In addition, the time reversal can be used if the semantics of interval temporal Logic is defined over finite intervals only like the past time interval temporal logic. So, it can be said that:

$$\tau = reverse(\sigma)$$

Therefore, we have three versions of temporal logic which are future time, past time and time reversal. To clarify the relation between these three versions of ITL lets have the future formula:

$$f_1 \,;\, f_2$$

The semantics of this formula for finite time is as follows:
$\mathcal{M}_\sigma[\![f_1 \,;\, f_2]\!] = \text{tt iff}$
exists k, such that $0 \le k \le |\sigma|$, and
$\mathcal{M}_{\sigma_0 \to \sigma_k}[\![f_1]\!] = \text{tt}$ and $\mathcal{M}_{\sigma_k \to \sigma_{|\sigma|}}[\![f_2]\!] = \text{tt}$.
Which is if the future interval σ is a fusion of two future intervals, the first interval

$\sigma_0 \ldots \sigma_k$ satisfies f_1 and the second interval $\sigma_k \ldots \sigma_{|\sigma|}$ satisfies f_2

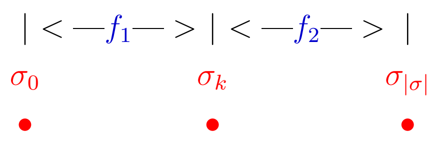

Figure 3.24: Chop future formula

However, if the time reversal is used for the same formula we have:

$$(f_1 \, ; f_2)^r = (f_2)^r \, ; (f_1)^r$$

The semantics of this formula is as follows:
$\mathcal{M}_\sigma[\![(f_1 \, ; f_2)^r]\!] = \mathcal{M}_\sigma[\![(f_2)^r \, ; (f_1)^r]\!] = \text{tt}$ iff
exists k, such that $0 \le k \le |\sigma|$, and
$\mathcal{M}_{\sigma_k \leftarrow 0}[\![f_1]\!] = \text{tt}$ and $\mathcal{M}_{\sigma_{|\sigma|} \leftarrow k}[\![f_2]\!] = \text{tt}$.
Which is if the interval σ is a fusion of two intervals, the first interval $\sigma_0 \ldots \sigma_k$ satisfies f_2 and the second interval $\sigma_k \ldots \sigma_{|\sigma|}$ satisfies f_1

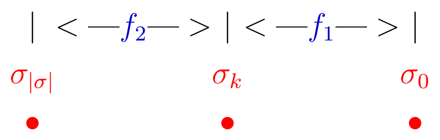

Figure 3.25: Chop with reversal time

When the past time operators are used for the same formula we have:

$$f_2 \,\hat{;}\, f_1$$

The semantics of past Chop ($\hat{;}$) is as follows:

$$\mathcal{M}_\tau[\![f_2 \,\hat{;}\, f_1]\!] = \text{tt iff}$$

exists k where$0 \le k \le |\tau|$, s.t. $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![f_2]\!] = \text{tt}$ and $\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![f_1]\!] = \text{tt}$.

History interval τ is finite, then interval τ is a fusion of two past intervals, the first interval $\tau_0 \ldots \tau_k$satisfies f_1 and the second interval $\tau_k\ldots\tau_{|\tau|}$ satisfies f_2.
State τ_k is shared by both.

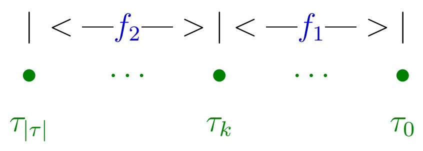

Figure 3.26: Past Chop formula

If we compare Figure 3.25 for the time reversal with Figure 3.26 for the past time formula we can see the two formula are the same.
It can thus be said that the time reversal of interval temporal logic ITL^r can be used to transform the future time interval temporal logic ITL to past time interval temporal logic ITL^p

$$ITL \overset{USING}{\rightarrow} ITL^r \overset{PRODUCE}{\rightarrow} ITL^p$$

and at the same time, the time reversal ITL^r can be used to transform past time interval temporal logic ITL^p to the future time interval temporal logic ITL

$$ITL^p \overset{USING}{\rightarrow} ITL^r \overset{PRODUCE}{\rightarrow} ITL$$

It can be concluded that the time reversal plays an important role in transforming the past time formula to future time formula and vice versa. The importance of this relation between the past time operators and the future time is that the ITL^p has the same expressive power of ITL depending on the fact that any future time formula ITL can be transformed to past time formula ITL^p using the time reversal.
Next, in table 3.11, we will list future time operators of interval temporal logic with the time reversal and the produced past time operators of interval temporal logic.

$Future\ time + time\ reversal$	$Past\ time$
$(f_1 ; f_2)^r = (f_2)^r ; (f_1)^r$	$f_2 \hat{;} f_1$
$(f^*)^r = (f^r)^*$	$f^{\hat{*}}$
$(\mathsf{skip})^r$	$\widehat{\mathsf{skip}}$
$(\bigcirc f)^r$	$\hat{\bigcirc} f$
$(\textcircled{w} f)^r$	$\widehat{\textcircled{w} f}$
$(\Diamond f)^r$	$\hat{\Diamond} f$
$(\Box f)^r$	$\hat{\Box} f$
$(\Diamond\!\!\!\!\!\;\text{i}\ f)^r$	$\hat{\Diamond\!\!\!\!\!\;\text{i}}\ f$
$(\boxed{\text{i}}\ f)^r$	$\hat{\boxed{\text{i}}}\ f$
$(\Diamond\!\!\!\!\!\;\text{a}\ f)^r$	$\widehat{\Diamond\!\!\!\!\!\;\text{a}\ f}$
$(\boxed{\text{a}}\ f)^r$	$\widehat{\boxed{\text{a}}\ f}$

Table 3.11: Future and past operators relation list

3.5 Axioms and Rules in Logic

Several temporal languages have been proposed for the specification and verification of concurrent systems in the last three decades, such as Linear Temporal Logic [88], Computation Tree Logic [30], Interval Temporal Logic [99, 100], Temporal Logic of Actions [77], and many others. In practice, there are two popular verification approaches, the first is model checking and the second is theorem proving.

Model checking is an automatic verification approach based on model theory for concurrent systems that are finite state or have finite state abstractions. This approach was developed independently in the early 1980's by Clarke and Emerson [29] and by Queille and Sifakis [118] and it has been applied successfully to computer hardware and many aspects of software verification. The advantage of model checking is that the verification can be done automatically. However, it suffers from the well-known problem called state explosion [38]. As the number of state variables in the system increases, the size of the system state space grows exponentially. For example, consider a system composed of n processes, each having m states. The asynchronous composition of these processes may have m^n states. In model checking we refer to this problem as the state explosion problem [31].

With the theorem proving approach [16], we assume we have the system S and the property P; to verify whether or not a system S satisfies a property P is to prove whether or not $S \supset P$ is a theorem within the proof system. The advantage is that

theorem proving avoids the state explosion problem and can verify both finite and infinite systems, and can be done semi-automatically; therefore, it is also suitable for data intensive applications. However, within the verification process, lots of assertions need to be inserted in the context of the program modelling the system. So, the use of theorem prover requires considerable expertise to guide and assist the verification process and one of the famous theorem provers is PVS [106].
There are many axiom and proof systems for different temporal logic languages, each of these axiom systems providing a set of basic theorems together with inference rules for generating new theorems.
Theorems describe the universal truths of a formal language, and are typically defined as formula prefixed by the "$\vdash$" symbol [48], for example:

$$\vdash true\_thing$$

Logical axioms (or axiomatisation) are certain formulas in a formal language that are universally valid; these formulas provide rules for generating new universal truths [48], and can be used to prove that a given temporal formula is valid over a given program or in another meaning, used for proving properties of programs [72].
The expressive power (the greater variety and quantity of ideas that can be represented by) of ITL is stronger than these logics (Linear Temporal Logic, Computation Tree Logic, Temporal Logic of Actions) because ITL uses chop (;) and chop-star (*), so it is a useful and powerful formalism for specification and verification of reactive systems [38].
Within the interval based TL community, several researchers have investigated axiom systems with different extensions [39]. Moszkowski [99] presented axiom systems over finite intervals for PITL and first order ITL. The propositional part was claimed to be complete but only an outline of a proof was given. Later he extended this for chop and chop-star with infinite time [100]. Recently, he presented [103] a complete axiom system for Propositional Interval Temporal Logic with infinite time and proven completeness by a reduction to his earlier complete Propositional Interval Temporal Logic axiom system for finite time [101] and conventional propositional linear-time temporal logic (PTL). ITL^p which contains temporal operators $\hat{;}$ and $\hat{*}$ is proposed in Chapter 3, therefore in this Section, the axioms and proof rules suitable for ITL^p are proposed and have been proven sound.

3.5.1 Propositional Axioms and Rules for ITL

In Table 3.12 we list the propositional axioms and rules for ITL.

ChopAssoc	$\vdash\ (f_0; f_1); f_2\ \equiv\ f_0; (f_1; f_2)$
OrChopImp	$\vdash\ (f_0 \lor f_1); f_2\ \supset\ (f_0; f_2) \lor (f_1; f_2)$
ChopOrImp	$\vdash\ f_0; (f_1 \lor f_2)\ \supset\ (f_0; f_1) \lor (f_0; f_2)$
EmptyChop	$\vdash\ \mathsf{empty}; f_1\ \equiv\ f_1$
ChopEmpty	$\vdash\ f_1; \mathsf{empty}\ \equiv\ f_1$
BiBoxChopImpChop	$\vdash\ \boxed{i}(f_0\ \supset\ f_1) \land \Box(f_2 \supset f_3)\ \supset\ (f_0; f_2)\ \supset\ (f_1; f_3)$
StateImpBi	$\vdash\ w\ \supset\ \boxed{i}\, w$
NextImpNotNextNot	$\vdash\ \bigcirc f_0\ \supset\ \neg\bigcirc\neg f_0$
KeepnowImpNotKeepnowNot	$\vdash\ \mathsf{keepnow}\,(f_0)\ \supset\ \neg\,\mathsf{keepnow}\,(\neg f_0)$
BoxInduct	$\vdash\ f_0 \land \Box(f_0\ \supset\ Ⓦ f_0)\ \supset\ \Box f_0$
InfChop	$\vdash\ (f_0 \land \mathsf{inf})\ ;\ f_1\ \equiv\ (f_0 \land \mathsf{inf})$
ChopStarEqv	$\vdash\ f_0^*\ \equiv\ (\mathsf{empty} \lor ((f_0 \land \mathsf{more})\ ;\ f_0^*))$
ChopstarInduct	$\vdash\ (\mathsf{inf} \land f_0 \land \Box(f_0\ \supset\ (f_1 \land \mathsf{fmore})\ ;\ f_0))\ \supset\ f_1^*$
MP	$\vdash\ f_0\ \supset\ f_1,\ \vdash\ f_0\ \Rightarrow \vdash\ f_1$
BoxGen	$\vdash\ f_0\ \Rightarrow \vdash\ \Box f_0$
BiGen	$\vdash\ f_0\ \Rightarrow \vdash\ \boxed{i}\, f_0$

Table 3.12: Axioms and Rules for Propositional ITL

3.5.2 Propositional Axioms and Rules for ITL^p

In Table 3.13 we list the propositional axioms and rules valid for ITL^p.

PastChopAssoc	$\vdash\ (h_0 \,\hat{;}\, h_1) \,\hat{;}\, h_2 \ \equiv\ h_0 \,\hat{;}\, (h_1 \,\hat{;}\, h_2)$
PastOrChopImp	$\vdash\ h_2 \,\hat{;}\, (h_1 \vee h_0) \ \supset\ (h_2 \,\hat{;}\, h_1) \vee (h_2 \,\hat{;}\, h_0)$
PastChopOrImp	$\vdash\ (h_2 \vee h_1) \,\hat{;}\, h_0 \ \supset\ (h_1 \,\hat{;}\, h_0) \vee (h_2 \,\hat{;}\, h_0)$
PastEmptyChop	$\vdash\ h \,\hat{;}\, \widehat{\mathsf{empty}} \ \equiv\ h$
PastChopEmpty	$\vdash\ \widehat{\mathsf{empty}} \,\hat{;}\, h \ \equiv\ h$
PastBiBoxChopImpChop	$\vdash\ \hat{\boxed{\text{i}}}(h_0 \supset h_1) \wedge \widehat{\Box}(h_2 \supset h_3) \ \supset\ (h_2 \,\hat{;}\, h_0) \ \supset\ (h_3 \,\hat{;}\, h_1)$
PastStateImpBi	$\vdash\ w \ \supset\ \hat{\boxed{\text{i}}}\, w$
PastNextImpNotNextNot	$\vdash\ \widehat{\bigcirc}\, h \ \supset\ \neg \widehat{\bigcirc} \neg h$
BoxInduct	$\vdash\ h_0 \wedge \widehat{\Box}(h_0 \supset \hat{\text{ⓦ}} h_0) \ \supset\ \widehat{\Box}\, h_0$
PastChopStarEqv	$\vdash\ h_0^{\hat{*}} \ \equiv\ ((h_0^{\hat{*}} \,\hat{;}\, (h_0 \wedge \widehat{\mathsf{more}})) \vee \widehat{\mathsf{empty}})$
MP	$\vdash\ h_0 \ \supset\ h_1,\ \vdash\ h_0 \ \Rightarrow \vdash\ h_1$
PastBoxGen	$\vdash\ h_0 \supset\ \ \vdash\ \widehat{\Box}\, h_0$
PastBiGen	$\vdash\ h_0 \supset\ \ \vdash\ \hat{\boxed{\text{i}}}\, h_0$
PastChopEmptyAnd	$\vdash\ h \wedge w_0 \quad \equiv \quad h \,\hat{;}\, (\widehat{\mathsf{empty}} \wedge w_0)$

Table 3.13: Axioms and Rules for Propositional ITL^p.

Where w is a state formula, the propositional axioms and rules of ITL^p are assumed to be complete according to Moszkowski introducing the complete axioms system of ITL [104], and we used the axioms and rules with finite time to generate the axioms and rules for ITL^p and proof it for past time, thus, it can be said that we have a complete axioms and rules of ITL^p.
The soundness proof of axioms and rules of ITL^p is given and listed in Appendix A.

3.6 Chapter Summary

This Chapter mainly discusses:
ITL, its syntax and semantics and provides a discussion on past time ITL. Section wise detail of topics covered in this Chapter is:

- The start of this Chapter introduces ITL (Interval temporal language) and its formal definitions.
- The next Section covers the syntax and semantics of Interval temporal logic, these have been explained with the help of examples.
- Past time interval logic, its semantics and syntax, is explained with the help of related examples; a detailed discussion on the past time operators is also presented.
- The last Section of this Chapter introduces an overview of axioms in logic. The axioms and rules for both propositional ITL and its past time counterpart i.e. Propositional ITL^p are listed, and these axioms and rules are proved sound.

This Chapter introduces past time ITL^p, its syntax and semantics, which is the main contribution of this thesis. Past time operators of ITL^p will be used in the next Chapter to give SANTA operators history based semantics.

Chapter 4

ITL^p TO REASON ABOUT HISTORY-BASED ACCESS CONTROL POLICIES

In this Chapter:

- *Choice of temporal logic*
- *Formal semantics of SANTA*
- *History based access control policies using ITL^p*
- *Formal semantics of SANTA*
- *Verification rules*

4.1 Introduction

In this Chapter, a set of temporal languages has been investigated to express history based policies. To represent policy rules formally, the semantic model for the always-followed-by operator will be given in order to investigate the suitability of these languages to reason about history based policies. SANTA policy language has been introduced in Chapter 2, however, in this Chapter, we will redefine SANTA operators and use it to introduce the semantics of individual rules and the semantics of policies as well as the semantics of compound policies. Also, we will list refinement rules used to refine compound policies but we will not use it. What is more, a description of the verification rules used to verify properties is given and we will show how the compositional specification of policies can be exploited using proof-rules that simplify the verification tasks by splitting the proof of a property for a complete specification of rules to proofs of individual weak rules. The refinement rules into enforcer are introduced, these rules can be used to construct an enforcer E for the compound policies.

4.2 Choice of temporal logic

In this next Section, we will investigate a chosen set of temporal languages to express history based access control policies. A formal semantic model for the **always-followed-by** operator (used to represent policy rules) will be used for this investigation.

4.2.1 Formal semantics of policy rules

The semantic model needs to model sequences of "snapshots" of a system. These sequences represent the behaviour of the system. We model these snapshots via a state mapping. A state is a mapping from the set of propositional variables Var to the set of *values* $\{tt, ff\}$.
An interval (behaviour) is a finite sequence of one or more states $\sigma_0\sigma_1\sigma_2 \ldots \sigma_{|\sigma|}$ where $|\sigma|$ denotes the length of an interval σ and is equal to the number of states minus 1. Let Σ denote the set of all possible intervals. Let $\sigma = \sigma_0\sigma_1 \ldots \sigma_{|\sigma|}$ be an interval. Then $\sigma_0 \ldots \sigma_k$ (where $0 \leq k \leq |\sigma|$) denotes a *prefix* interval of $\sigma, \sigma_k \ldots \sigma_{|\sigma|}$ (where $0 \leq k \leq |\sigma|$) denotes a *suffix* interval of σ and $\sigma_k \ldots \sigma_l$ (where $0 \leq k \leq l \leq |\sigma|$) denotes a subinterval of σ.

Let [[...]] be the "meaning" function from policy rule language $\times\Sigma$ to $\{tt, ff\}$. The formal semantics of the **always-followed-by** operator $Pre \mapsto W$, where Pre is a formula denoting the history and W is an access control variable.
The semantics $[[Pre \mapsto W]]_\sigma$ of the **always-followed-by** operator is as follows:
for all k, (where $0 \leq k \leq |\sigma|$ and $[[Pre]]_{\sigma 0 \ldots \sigma_k}=$ tt) $implies[[W]]_{\sigma_k} =$ tt.
Notice that the implication in the semantics of an individual rule means that W can be true in a state even if Pre does not hold in the prefix of that interval. However, for verification purposes, we need to know the value of W in any state of the interval. This is exactly what the **strong always-followed-by** operator does, it explicitly specifies the conditions under which the access decision is $false$ [32, 61]. This operator will be used in the verification of properties on policy rules and is defined as:
$Pre \leftrightarrow W \mathrel{\widehat{=}} Pre \mapsto W \wedge \neg Pre \mapsto \neg W.$
If Pre holds in the prefix interval, then W must hold in the last state of that prefix interval, otherwise W must not hold in that state.
We will discuss the suitability of Propositional Interval Temporal Logic (PITL) [97], Propositional Linear Temporal Logic (PLTL) [88] and the proposed ITL^p to show that ITL^p is the right choice to express history-based access control policies.

4.2.2 Propositional interval temporal logic

Propositional Interval Temporal Logic (PITL) [97] is a temporal logic with a basic construct for the sequential composition of two formula as well as an analogue of Kleene star. Within PITL one can express both finite-state automata and regular expressions. The syntax of PITL is as follow:

Note that the operator $\bigcirc$ can be used both for expressions (e.g.,$\bigcirc J$) and for

Formula f ::=	true|	q |	Q |	$p(e_1, \ldots, e_n)$|	$\neg f$|	$f_1 \wedge f_2$|	$\forall v.f$|	skip|	$f_1 \,;\, f_2$|	f^*

formulas, e.g., $\bigcirc(I = 3)$ [98].
Semantics
The informal semantics of the most interesting constructs are as follows:

- $\neg f$: f does not holds for that interval.
- $f_1 \wedge f_2$: f_1 holds for that interval and f_2 holds for that interval.
- skip unit interval (length 1).

- $f_1 ; f_2$ holds if the interval can be decomposed ("chopped") into a prefix and suffix interval, such that f_1 holds over the prefix and f_2 over the suffix, or if the interval is infinite and f_1 holds for that interval.
- f^* holds if the interval is decomposable into a finite number of intervals such that for each of them f holds, or the interval is infinite and can be decomposed into an infinite number of finite intervals for which f holds.

The formal semantics of PITL is as follows:
Let [[...]] be the "meaning" function from PITL formula $\times\Sigma$ to $\{tt, ff\}$.

- $[[p]]_\sigma$ = tt iff $\sigma_0(p)$=tt.
- $[[\neg f]]_\sigma$ = tt iff *not* $[[f]]_\sigma$=tt.
- $[[f_1 \ \vee \ f_2]]_\sigma$ = tt iff $[[f_1]]_\sigma$=tt or $[[f_2]]_\sigma$=tt.
- $[[\mathsf{skip}]]_\sigma$ = tt iff $|\sigma|$=1.
- $[[f_1 \ ; \ f_2]]_\sigma$ = tt iff (exists k s.t. ($[[f_1]]_{\sigma_0 \ldots \sigma_k}$= tt and $[[f_2]]_{\sigma_k \ldots \sigma_{|\sigma|}}$ = tt)).
- $[[f\ ^*]]_\sigma$ = tt iff (exists $l_0 \ldots l_n$ s.t. $l_0 = 0$ and $l_n = |\sigma|$ and for all $0 \ \leq < \ n$, $l_i \ < \ l_{i+i}$ and $[[f]]_{\sigma_{l_i} \ldots \sigma_{l_{i+1}}}$ = tt).

Now, we need to introduce the following operators in order to help us in expressing the policy rule:

- $\mathsf{more} \ \widehat{=} \ \mathsf{skip} \ ; \ \mathsf{true}$ which is an interval with two or more states.
- $\mathsf{empty} \ \widehat{=} \ \neg\mathsf{more}$ which is an interval with one state only.
- $\Diamond\!\!\!\!\cdot\ \ f \ \widehat{=} \ f \ ; \ \mathsf{true}$ a prefix interval for which f holds.
- $\boxdot \ f \ \widehat{=} \ \neg(\Diamond\!\!\!\!\cdot\ \ \neg \ f \ )$

Now, always-followed-by and strong-always-followed operator are introduced as follows:
always-followed-by ($Pre \ \mapsto \ W$)
$Pre \ \mapsto \ W \ \widehat{=} \ \boxdot(\neg(Pre \ ; \ ((\neg W) \wedge \ \mathsf{empty})))$
strong-always-followed-by ($Pre \ \leftrightarrow \ W$)

$Pre \leftrightarrow W \mathrel{\widehat{=}} \boxdot(\neg(Pre \ ; \ ((\neg W) \wedge \mathsf{empty}))) \wedge \boxdot(\neg(\neg \ Pre \ ; \ (W \ \wedge \ \mathsf{empty})))$
As we see above when introducing always followed by and strong always followed by operators, PITL is suffering from non-elementary complexity. The development of efficient PITL-based verification tools is very difficult due to this problem. So, it can be said that PITL is not a good choice to reason about history based policies.

4.2.3 Propositional linear temporal logic (PLTL)

In the next part, we will investigate whether PLTL [88] is suitable to express history-based access control rules.
The syntax of PLTL formula f is introduced follows:

$$\boxed{Formula \ \ f \quad ::= \mathsf{true}|p|f_1 \vee f_2|\neg f|\bigcirc f|\Box f|f_1 \ U \ f_2}$$

Until operator has the semantics:
-$[[f_1 \ U \ f_2]]_\sigma$ = tt iff there exists $k : 0 \leq k \leq |\sigma|, [[f_2]]_{\sigma_k \ldots \sigma_{|\sigma|}}$= tt and for all $j : 0 \leq j < k, [[f_1]]_{\sigma_j \ldots \sigma_{|\sigma|}}$ =tt.

Therefore the semantics of $\neg(f_1 U f_2)$ is:
$[[\neg(f_1 \ U \ \neg f_2)]]_\sigma$ = tt iff there exists a $k : 0 \leq k \leq |\sigma|, [[f_2]]_{\sigma_k \ldots \sigma_{|\sigma|}}$= tt or not for all $j : 0 \leq j < k, [[f_1]]_{\sigma_j \ldots \sigma_{|\sigma|}}$ =tt.
which is almost the **always-followed-by** operator, i.e., if the formulas f_1 and f_2 are restricted to be state formula then $\neg(f_1 \ U \ \neg f_2)$ corresponds to $(\Box f_1) \mapsto f_2$.
It is clear from above that the premise in Propositional linear temporal logic can only be an always type of property. Hence, the sequential composition of, for example, two phases cannot be expressed in the normal way. Therefore, the sequential access and cardinality on history policy rules are hard to express [24].

4.2.4 Interval Temporal Logic (ITL)

In Chapter 3, Interval temporal logic operators were introduced and now we will use these operators to define SANTA operators always-followed-by and strong-always-followed-by, and use these operators to express history based policies.

Always-followed-by ($\mapsto$):

The operator always-followed-by denoted by the symbol($\mapsto$) is defined as follows:
$f \ \mapsto \ w \ \ \mathrel{\widehat{=}} \ \ \boxdot(f \ \supset \ \mathsf{fin}\,(w))$

where f stands for any ITL formula, and w is a state formula. The intuition of the operator($f \mapsto w$) is that whenever the formula f holds for a prefix interval then the state formula must w holds in the final state of that interval, that is, f is always followed by w as shown in Figure 4.1:
$f \mapsto w \;\widehat{=}\; \boxed{i}(f \supset \mathsf{fin}\,(w))$
iff for all k such that$(0 \leq k \leq |\sigma| \wedge \mathcal{M}_{\sigma_{k \to |\sigma|}}[\![f]\!] = \text{tt}) \supset \mathcal{M}_{\sigma_{|\sigma|}}[\![\mathsf{fin}\,(w)]\!] = \text{tt}$

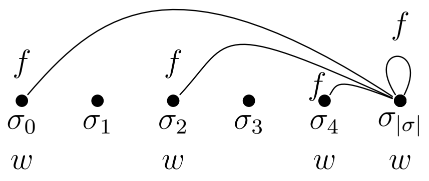

Figure 4.1: Always-followed-by

This operator always-followed-by ($\mapsto$) will be used in the scenario Chapter to express security policy rules of general practice system (GPS).

Strong-always-followed-by ($\leftrightarrow$):

The operator strong always-followed-by, denoted by the symbol ($f \leftrightarrow w$) is defined as follows:
$f \leftrightarrow w \;\widehat{=}\; \boxed{i}(f \equiv \mathsf{fin}\,(w))$
where f stands for any ITL formula, and w is a state formula.

iff for all k such that$(0 \leq k \leq |\sigma| \wedge \mathcal{M}_{\sigma_{k \to |\sigma|}}[\![h]\!] = \text{tt})\ iff\ \mathcal{M}_{\sigma_{|\sigma|}}[\![\mathsf{fin}\,(w)]\!] = \text{tt}$

The intuition of the operator strong always-followed-by ($f \leftrightarrow w$) determines in any state the value of the state formula w. If f holds in the prefix of the reference interval, then w must hold in that state otherwise w must not hold in that state as shown in Figure 4.2, where each bullet represents a state.

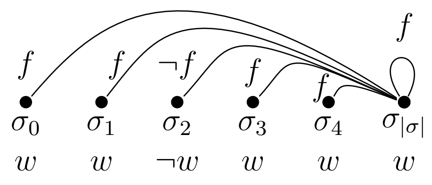

Figure 4.2: Strong-always-followed-by

This operator will be used in the Scenario Chapter to express security policy rules of general practice system (GPS).
As we show above, the two operators always-followed-by (Figure 4.1) and strong-always-followed-by (Figure 4.2) have been defined from the future time operators of ITL. In the following sections and in the GPS scenario, we will use these operators to reason about history based access control policies and compare it with the existing work.

Syntax of SANTA with future time

In the following, we provide the syntax of access control policies.

Subjects $su ::= S_i \mid cs$
Objects $ob ::= O_i \mid co$
Actions $ac ::= A_i \mid ca(e_1, ..., e_n)$
Premise of rule $pr ::= pr_1$ **chop** $pr_2 \mid pr_1$ **and** $pr_2 \mid pr_1$**or** $pr_2 \mid$ **always** $pr \mid$ **sometime** $pr \mid$ **not** $pr \mid$ **next** $pr \mid$ **if** be **then** pr_1 **else** $pr_2 \mid$ **exists** x **in** $se\ :\ pr \mid$ **forall** x **in** $se\ :\ pr \mid$ **last**$(e)\ :\ pr \mid e\ :\ pr \mid be$
Rules $ru ::= [rn\ ::]$ **allow**(su, ob, ac)**when** $pr \mid$ $[rn\ ::]$ **deny**(su, ob, ac)**when** $pr \mid$ $[rn\ ::]$ **decide**(su, ob, ac)**when** pr
Policies $po ::= (ru_1...ru_n) \mid po_2$ **policy** $pn :: po$ **end**$\mid$ po_2 **chop** $po_1\mid$**if** be **then** po_1 **else** $po_2\mid$ **aslongas** $be\ :\ po$

Figure 4.3: Syntax of SANTA policy language

Figure 4.3 above summarizes the syntax of our policy language where e is an expression, be a Boolean expression, and se a Set expression with their usual operators and semantics. S_i is a subject variable, where i is a arbitrary name, similarly O_i is an object variable, A_i is an action variable and pn is a name for a policy; rn is a name for a rule (optional). Let $Subjects$, $Objects$ and $Actions$ be, respectively, the universal set of subjects, objects and actions. These can be used as part of SANTA expressions. Let $cs \in Subjects$ be a subject, $co \in Object$ be an object and $ca(\overline{v}) \in Actions$ be an action with interface $\overline{v}$ [69].
The syntax of policy rule will be explained informally in the next Section.

Policy rules

A rule typically expresses a single security requirement and forms the basic building block of a policy. Rules consist of a premise and a consequence. The premise

describes a set of system behaviours, which lead to the consequence that represents an assertion on the current system state, such as allowing or denying a particular access. The consequence of a rule defines the decision taken by the reference monitor. The set of system behaviours in the premise is matched against the history of the system execution. Rules therefore can refer to sequences of previously observed states in the system execution, allowing for the expression of history-based policies [1] and dynamic separation of duty constraints [124]. Events that can be referred to in the premise of rules are those defined in the computational model represented in the Literature Review Chapter in Figure 2.11 or external events that are observable by the RM process. Authorization defines the access to resources in the system. With respect to the computational model they define whether the execution of an action is permissible. An authorization rule defines the condition under which a subject is allowed to perform an action on an object [69].
Now, we will explain the formal semantics of SANTA.

Formal semantics of SANTA with future time

As SANTA is using a compositional approach to the specification of policies, its underlying formalism should therefore also express specifications of system behaviour compositionally.
In the following we will provides the formal semantics for SANTA.
Semantics of rules:
Policy rules define the behaviour of the access control variables. The consequence of a rule determines the type of the rule and the subjects, objects and actions the rule applies to. The operator always-followed-by is used to capture the relation between the premise of a rule and its consequence.
Let us first define the semantics of a premise. The syntax that is used in the premise is actually a subset of ITL formula.
The semantics of a rule premise is then as follows:
$[\![pr_1 \ \ \mathbf{chop}\ pr_2]\!] \mathrel{\widehat{=}} [\![pr_1]\!] \ ; [\![pr_2]\!]$
$[\![pr_1\ \mathbf{and}\ pr_2]\!] \mathrel{\widehat{=}} [\![pr_1]\!] \wedge [\![pr_2]\!]$
$[\![pr_1\ \mathbf{or}\ pr_2]\!] \mathrel{\widehat{=}} [\![pr_1]\!] \vee [\![pr_2]\!]$
$[\![\mathbf{not}\ pr]\!] \mathrel{\widehat{=}} \neg [\![\ pr]\!]$
$[\![\mathbf{sometime}\ pr]\!] \mathrel{\widehat{=}} \Diamond [\![\ pr]\!]$
$[\![\mathbf{always}\ pr]\!] \mathrel{\widehat{=}} \Box [\![\ pr]\!]$
$[\![\mathbf{next}\ pr]\!] \mathrel{\widehat{=}} \mathsf{skip}\ ; [\![\ pr]\!]$

$[\![\textbf{if } be \textbf{ then } pr_1 \textbf{ else } pr_2]\!] \ \widehat{=}\ ((be \wedge [\![pr_1]\!]) \vee (\neg be \wedge [\![pr_2]\!]))$
$[\![\textbf{exists } x \textbf{ in } se \ : \ pr]\!] \ \widehat{=}\ \exists x.x \in se \wedge [\![pr]\!]$
$[\![\textbf{forall } x \textbf{ in } se \ : \ pr]\!] \ \widehat{=}\ \forall x.x \in se \supset [\![pr]\!]$
$[\![\textbf{last}(e)pr]\!] \ \widehat{=}\ (\Box(\mathsf{empty} \wedge \neg[\![pr]\!])\ ;\ \mathsf{skip}\ ;\ (\mathsf{empty} \wedge [\![pr]\!]))^n$
$[\![e : pr]\!] \ \widehat{=}\ \mathsf{finite}\ \ ;\ \ (len(e)\ \ \wedge\ \ [\![\ pr\ ]\!])$
The semantics of $e \ : \ pr \mapsto w$ includes $\widehat{\mathsf{true}}\ \hat{;}$ to obtain the "history" h of length e from the point where w holds. The operator always-followed-by ($\mapsto$) which has been defined in Section 5.2.4 will be used to define the semantics of individual rules.
The semantics of individual rules is defined as follows:
$[\![\textbf{allow}(su, ob, ac)\ \ \textbf{when}\ \ pr]\!] \ \widehat{=}$
$\forall\ vs \in\ Subjects.\ \ \forall\ vo \in Objects.\ \ \forall\ va \in Actions.$
$[\![\ pr]\!] \mapsto Aut^+(su, ob, ac)$
$[\![\textbf{deny}(su, ob, ac)\ \ \textbf{when}\ \ pr]\!] \ \widehat{=}$
$\forall\ vs\ \ \in Subjects.\ \ \forall\ vo\ \ \in Objects.\ \ \forall\ va\ \ \in Actions.$
$[\![\ pr]\!] \mapsto Aut^-(su, ob, ac)$
$[\![\textbf{decide}(su, ob, ac)\ \ \textbf{when}\ \ pr]\!] \ \widehat{=}$
$\forall\ vs\ \ \in Subjects.\ \ \forall\ vo\ \ \in Objects.\ \ \forall\ va\ \ \in Actions.$
$[\![\ pr]\!] \mapsto Aut(su, ob, ac)$
Let $vs \ \in \ frees(r); vo \ \in \ freeo(r); and\ va \in freea(r)$ be the free variables (subject, object and action, respectively) in the rule r. The propositional state variable **Aut**$^+$(su, ob, ac) captures positive authorisations. If its value is **true** the policy defines a positive authorization for the subject su to perform action ac on object ob. Similarly **Aut**$^-$(su, ob, ac) captures negative authorisations. The propositional state variable **Aut**(su, ob, ac) defines the access control decision taken by the reference monitor [69].

Semantics of policies:

We first define the semantics of a policy that consists of a collection of rules. The implication, in the semantics of an individual rule, $(f \ \ \supset\ \ \mathsf{fin}\,(w))$ means that the state formula w holds in the final state of interval if and only if the formula f holds in some prefix of that interval.
Policies (at semantic level), define the access decision in every state of the reference monitor and are important for its verification.
We adopt a refinement approach using the "strong-always-followed-by" operator denoted by ($\leftrightarrow$) to obtain a complete policy specification.
The operator strong-always-followed-by ($\leftrightarrow$) is defined as follows:

in this operator, a rule of the form ($f \;\leftrightarrow\; \mathsf{fin}\,(w)$) determines in any state the value of the state formula w.
If w holds in the final state of the interval, then f must hold in this interval and if w does not holds in the final state of the interval, then f must not hold in this interval.
The motivation of using a refinement approach is that if we can show that a system satisfies $f \;\leftrightarrow\; \mathsf{fin}\,(w)$ then it also satisfies $f \;\mapsto\; \mathsf{fin}\,(w)$.
Thus, by rewriting the policy specification using the algorithm presented below [69] we strengthen the specification by adding default rules such that the specification is complete.
By this, we mean that the specification defines the value of $\mathbf{Aut}^{+}(s,o,a)$, $\mathbf{Aut}^{-}(s,o,a)$ *and* $\mathbf{Aut}(s,o,a)$ in each state of the system and thus can be enforced by the reference monitor.
The semantics of a policy of the form $ru_1....ru_n$ is a semantically completely specified formula, i.e., the following formula:

$$\begin{array}{ll} \bigwedge & (f(s,o,a) \leftrightarrow Aut^{+}(s,o,a))\wedge \\ s \in Subjects & (g(s,o,a) \leftrightarrow Aut^{-}(s,o,a))\wedge \\ o \in Objects & (h(s,o,a) \leftrightarrow Aut(s,o,a)), \\ a \in Actions & \end{array}$$

where, for each s ∈ Subjects, o ∈ Objects and a ∈ Actions,

1. $f(s,o,a) \;\widehat{=}\; \bigvee_{i=1}^{l} [\![\; pr_i]\!]$ and pr_i appears as a premise in an allow rule of $ru_1....ru_n$. If there are no allow (s,o,a) rules in $ru_1....ru_n$, then $f(s,o,a) = false$.

2. $g(s,o,a) \;\widehat{=}\; \bigvee_{i=1}^{m} [\![\; pr_i]\!]$ and pr_i appears as a premise in a deny rule of $ru_1....ru_n$. If there are no $deny(s,o,a)$ rules in $ru_1....ru_n$, then $g(s,o,a) = false$.

3. $h(s,o,a) \;\widehat{=}\; \bigvee_{i=1}^{k} [\![\; pr_i]\!]$ and pr_i appears as a premise in a decide rule of $ru_1....ru_n$. If there are no decide (s, o, a) rules in $ru_1....ru_n$, then $h(s,o,a) = false$.

For each triple $(s,o,a) \in Subjects \times Objects \times Actions$, the formula $[\![ru_1....ru_n]\!]$ contains exactly one rule of the form $f(s,o,a) \leftrightarrow Aut^{+}(s,o,a)$, one rule of the form $g(s,o,a) \leftrightarrow Aut^{-}(s,o,a)$ and one rule of the form $h(s,o,a) \leftrightarrow Aut(s,o,a)$. Therefore, it fully determines the value of Aut(s,o,a) at each state of the system.

Default rules are automatically provided. For example, if the policy po does not contain a rule for $Aut^+(s,o,a)$, for some subject s, object o and action a, then it defaults to a rule of the form $false \leftrightarrow Aut^+(s,o,a)$ in $[\![ru_1...ru_n]\!]$.
Similarly for $Aut^-(s,o,a)$ and $Aut(s,o,a)$ if there are no explicit rules for them in $ru_1...ru_n$. As such, $[\![ru_1...ru_n]\!]$ grants every right granted by $(ru_1...ru_n)$ and denies everything else [69].

4.2.5 Interval Temporal Logic with past time (ITL^p)

In Chapter 3, the past time operators of ITL^p have been introduced, and in the following Section, we will use these operators to give the SANTA operators always-followed-by and strong-always-followed-by a history based semantics, in order to support our choice of ITL^p to express history based policies.

Always-followed-by ($\mapsto$):

The operator past always-followed-by($\mapsto$) is defined as follows:
$h \ \mapsto \ w \ \ \widehat{=} \ \ \widehat{\Box}(h \ \supset \ (w))$
where h stands for any ITL^p formula, and w is a state formula. The definition of always-followed-by with the history semantics ($\mapsto$) is as follow:

$h \ \mapsto \ w \ \ \widehat{=} \ \ \widehat{\Box}(h \ \supset \ w)$
iff for all k such that$(0 \leq k \leq |\tau| \wedge \mathcal{M}_{\tau_{|\tau|\leftarrow\tau_k}}[\![h]\!] = \text{tt}) \ \supset \ \mathcal{M}_{\tau_k}[\![w]\!] = \text{tt}$
The intuition of the operator($h \ \mapsto \ w$) is that w holds in the initial state of history interval then f must hold previously in the past interval. We only know that w holds at the first state as shown in Figure 4.4:

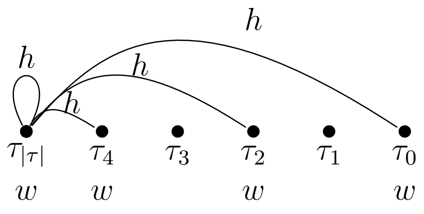

Figure 4.4: Always-followed-by-past

Strong-always-followed-by ($\leftrightarrow$):

The operator past always-followed-by ($h \leftrightarrow w$) is defined as follows:
$h \leftrightarrow w \;\; \widehat{=} \;\; \widehat{\Box}(h \equiv w)$
where h stands for any ITL^p formula, and w is a state formula. The definition of always-followed-by with the history semantics ($h \leftrightarrow w$) is as follow:
iff for all k such that$(0 \le k \le |\tau| \wedge \mathcal{M}_{\tau_{|\tau|\leftarrow\tau_k}}[\![h]\!] = \text{tt})\ iff\ \mathcal{M}_{\tau_k}[\![w]\!] = \text{tt}$

The intuition of the operator ($h \leftrightarrow w$) determines in any state the value of the state formula w.
If w holds in the initial state of the past reference interval, then h must hold for all the past suffix intervals otherwise w must not hold in that state as shown in Figure 4.5:

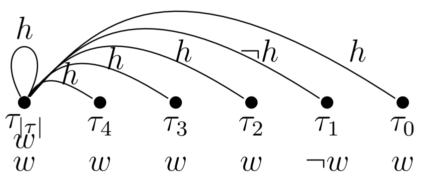

Figure 4.5: Strong-always-followed-by-past

In conclusion, as we show above, the two operators always-followed-by (Figure 4.4) and strong-always-followed-by (Figure 4.5) can be given a history semantics using the past time operators of ITL^p and we will show in the Scenario Chapter that these operators can be used to reason about history based access control policies.

Syntax of SANTA with past time

In the following, we provide the syntax of access control policies.

Subjects $su ::= S_i \mid cs$
Objects $ob ::= O_i \mid co$
Actions $ac ::= A_i \mid ca(e_1, ..., e_n)$
Premise of rule $pr ::= pr_1$ **chop** $pr_2 \mid pr_1$ **and** $pr_2 \mid pr_1$**or** $pr_2 \mid$ **always** $pr \mid$ **sometime** $pr \mid$ **not** $pr \mid$ **next** $pr \mid$ **if** be **then** pr_1 **else** $pr_2 \mid$ **exists** x **in** $se\ :\ pr \mid$ **forall** x **in** $se\ :\ pr \mid$ **last**$(e)\ :\ pr \mid e\ :\ pr \mid be$
Rules $ru ::= [rn\ ::]$ **allow**(su, ob, ac)**when** $pr \mid$ $[rn\ ::]$ **deny**(su, ob, ac)**when** $pr \mid$ $[rn\ ::]$ **decide**(su, ob, ac)**when** pr
Policies $po ::= (ru_1...ru_n) \mid po_2$ **policy** $pn :: po$ **end**$\mid$ po_2 **chop** $po_1 \mid$**if** be **then** po_1 **else** $po_2 \mid$ **aslongas** $be\ :\ po$

Figure 4.6: Syntax of SANTA policy language with past time

Figure 4.6 above summarizes the syntax of our policy language where e is an expression, be a Boolean expression, and se a Set expression with their usual operators and semantics. S_i is a subject variable, where i is a arbitrary name, similarly O_i is an object variable, A_i is an action variable and pn is a name for a policy; rn is a name for a rule (optional). Let $Subjects$, $Objects$ and $Actions$ be, respectively, the universal set of subjects, objects and actions. These can be used as part of SANTA expressions. Let $cs \in Subjects$ be a subject, $co \in Object$ be an object and $ca(\overline{v}) \in Actions$ be an action with interface $\overline{v}$.
The following subsections will explain the semantics of SANTA formally [69].

Formal semantics of SANTA with past time

As SANTA is using a compositional approach to the specification of policies, its underlying formalism should therefore also express specifications of system behaviour compositionally.

In the following we will provide the formal semantics for SANTA.

Semantics of rules:

Policy rules define the behaviour of the access control variables however, the consequence of a rule determines the type of rule and the subjects, objects and actions which the rule applies to. The operator always-followed-by is used to capture the relation between the premise of a rule and its consequence.

The syntax that is used in the premise is actually a subset of ITL^p formula, so, a definition of the premise semantics is needed. The semantics of a rule premise is then as follows:

$[\![pr_2 \ \ \mathbf{chop}\ pr_1]\!] \mathrel{\widehat{=}} [\![pr_2]\!] \mathbin{\hat{;}} [\![pr_1]\!]$

$[\![pr_1\ \mathbf{and}\ pr_2]\!] \mathrel{\widehat{=}} [\![pr_1]\!] \wedge [\![pr_2]\!]$

$[\![pr_1\ \mathbf{or}\ pr_2]\!] \mathrel{\widehat{=}} [\![pr_1]\!] \vee [\![pr_2]\!]$

$[\![\mathbf{not}\ pr]\!] \mathrel{\widehat{=}} \neg [\![\ pr]\!]$

$[\![\mathbf{sometime}\ pr]\!] \mathrel{\widehat{=}} \widehat{\Diamond\!\!\!\mathrm{i}}\,[\![\ pr]\!]$

$[\![\mathbf{always}\ pr]\!] \mathrel{\widehat{=}} \widehat{\boxed{\mathrm{i}}}\,[\![\ pr]\!]$

$[\![\mathbf{next}\ pr]\!] \mathrel{\widehat{=}} \widehat{\mathsf{skip}} \mathbin{\hat{;}} [\![\ pr]\!]$

$[\![\mathbf{if}\ be\ \mathbf{then}\ pr_1\ \mathbf{else}\ pr_2]\!] \ \mathrel{\widehat{=}}\ ((be \wedge [\![pr_1]\!]) \vee (\neg be \wedge [\![pr_2]\!]))$

$[\![\mathbf{exists}\ x\ \mathbf{in}\ se\ :\ pr]\!] \mathrel{\widehat{=}} \exists x . x \in se \wedge [\![pr]\!]$

$[\![\mathbf{forall}\ x\ \mathbf{in}\ se\ :\ pr]\!] \mathrel{\widehat{=}} \forall x . x \in se \supset [\![pr]\!]$

$[\![\mathbf{last}(e) pr]\!] \mathrel{\widehat{=}} (\widehat{\Box}(\widehat{\mathsf{empty}} \wedge \neg [\![pr]\!]) \mathbin{\hat{;}} \widehat{\mathsf{skip}} \mathbin{\hat{;}} (\widehat{\mathsf{empty}} \wedge [\![pr]\!]))^n$

$[\![e : pr]\!] \mathrel{\widehat{=}} \widehat{\mathsf{true}}\ \mathbin{\hat{;}}\ (len(e)\ \wedge\ [\![\ pr\ ]\!])$

The following operators **sometime^p**, **always^p** and **next^p** are the introduced operators; these operators have been given a history based semantics using past time operators and they will help us to reason about history based access control policy in the GPS scenario. The first introduced operator **sometime**$^{\mathbf{p}}$ is needed if an incident happened sometime in the past while **always**$^{\mathbf{p}}$ operator is needed if the incident happened continuously to the end of this interval however, the operator **next**$^{\mathbf{p}}$ is used to express that the incident happened in the previous state in the historical interval.

The following operators, which has been added to SANTA language: has the following semantics:

$[\![\mathbf{sometime^p}\ pr]\!] \mathrel{\widehat{=}} \widehat{\Diamond}[\![\ pr]\!]$

$[\![\mathbf{always^p}\ pr]\!] \mathrel{\widehat{=}} \widehat{\Box}[\![\ pr]\!]$

$[\![\mathbf{next^p}\ pr]\!] \mathrel{\widehat{=}} \widehat{\bigcirc}[\![\ pr]\!]$

The semantics of $e\ \ :\ \ pr \mapsto w$ includes $\widehat{\mathsf{true}}\ \hat{;}$ to obtain the "history" h of length

e from the point where w holds. The operator always-followed-by ($\mapsto$) which has been defined in Section 5.2.4 will be used to define the semantics of individual rules. The semantics of individual rules is defined as follows:
$[\![\mathbf{allow}(su, ob, ac)\ \ \mathbf{when}\ \ pr]\!] \mathrel{\widehat{=}}$
$\forall\ vs \in\ Subjects.\ \forall\ vo \in Objects.\ \forall\ va \in Actions.$
$[\![\ pr]\!] \mapsto Aut^{+}(su, ob, ac)$
$[\![\mathbf{deny}(su, ob, ac)\ \ \mathbf{when}\ \ pr]\!] \mathrel{\widehat{=}}$
$\forall\ vs\ \in Subjects.\ \forall\ vo\ \in Objects.\ \forall\ va\ \in Actions.$
$[\![\ pr]\!] \mapsto Aut^{-}(su, ob, ac)$
$[\![\mathbf{decide}(su, ob, ac)\ \ \mathbf{when}\ \ pr]\!] \mathrel{\widehat{=}}$
$\forall\ vs\ \in Subjects.\ \forall\ vo\ \in Objects.\ \forall\ va\ \in Actions.$
$[\![\ pr]\!] \mapsto Aut(su, ob, ac)$
Let $vs\ \in\ frees(r); vo\ \in\ freeo(r); and\ va \in freea(r)$ be the free variables (subject, object and action, respectively) in the rule r. The propositional state variable **Aut**$^{+}$(su, ob, ac) captures positive authorisations. If its value is **true** the policy defines a positive authorization for the subject su to perform action ac on object ob. Similarly **Aut**$^{-}$(su, ob, ac) captures negative authorisations. The propositional state variable **Aut**(su, ob, ac) defines the access control decision taken by the reference monitor [69].

Semantics of policies: We first define the semantics of a policy that consists of a collection of rules. The implication, in the semantics of an individual rule, ($h\ \supset\ w$) means that w can be true in the initial state of past interval even if h did not hold in the suffix of that past interval.
Policies (at semantic level) define the access decision in every state of the reference monitor and are important for its verification.
We adopt a refinement approach using the "strong-always-followed-by" operator denoted by ($\leftrightarrow$) : to obtain a complete policy specification.
The operator strong-always-followed-by ($\leftrightarrow$) is defined as follows:
in this operator, a rule of the form ($h\ \leftrightarrow\ w$) determines in any state the value of the state formula w.
If w holds in the initial state of the past interval, then h must holds in this interval and if w not holds in the initial state of the past interval, then h must not hold in this interval.
The motivation of using a refinement approach is that if we can show that a system satisfies $h\ \leftrightarrow\ w$ then it also satisfies $h\ \mapsto\ w$.
Thus, by rewriting the policy specification using the algorithm presented below [69]

we strengthen the specification by adding default rules such that the specification is complete.
By this we mean that the specification defines the value of $\mathbf{Aut}^+(s,o,a)$, $\mathbf{Aut}^-(s,o,a)$ *and* $\mathbf{Aut}(s,o,a)$ in each state of the system and thus can be enforced by the reference monitor.
The semantics of a policy of the form $ru_1....ru_n$ is a semantically completely specified formula, i.e., the following formula:

$$\bigwedge_{\substack{s \in Subjects\\ o \in Objects\\ a \in Actions}} \begin{array}{l}(f(s,o,a) \leftrightarrow Aut^+(s,o,a))\wedge\\ (g(s,o,a) \leftrightarrow Aut^-(s,o,a))\wedge\\ (h(s,o,a) \leftrightarrow Aut(s,o,a)),\end{array}$$

where, for each s ∈ Subjects, o ∈ Objects and a ∈ Actions,

1. $f(s,o,a) \mathrel{\widehat{=}} \bigvee_{i=1}^{l} [\![pr_i]\!]$ and pr_i appears as a premise in an allow rule of $ru_1....ru_n$. If there are no allow (s,o,a) rules in $ru_1....ru_n$, then $f(s,o,a) = false$.

2. $g(s,o,a) \mathrel{\widehat{=}} \bigvee_{i=1}^{m} [\![pr_i]\!]$ and pr_i appears as a premise in a deny rule of $ru_1....ru_n$. If there are no $deny(s,o,a)$ rules in $ru_1....ru_n$, then $g(s,o,a) = false$.

3. $h(s,o,a) \mathrel{\widehat{=}} \bigvee_{i=1}^{k} [\![pr_i]\!]$ and pr_i appears as a premise in a decide rule of $ru_1....ru_n$. If there are no decide (s, o, a) rules in $ru_1....ru_n$, then $h(s,o,a) = false$.

For each triple $(s,o,a) \in Subjects \times Objects \times Actions$, the formula $[\![ru_1....ru_n]\!]$ contains exactly one rule of the form $f(s,o,a) \leftrightarrow Aut^+(s,o,a)$, one rule of the form $g(s,o,a) \leftrightarrow Aut^-(s,o,a)$ and one rule of the form $h(s,o,a) \leftrightarrow Aut(s,o,a)$. Therefore, it fully determines the value of Aut(s,o,a) at each state of the system.
Default rules are automatically provided. For example, if the policy *po* does not contain a rule for $Aut^+(s,o,a)$, for some subject s, object o and action a, then it defaults to a rule of the form $false \leftrightarrow Aut^+(s,o,a)$ in $[\![ru_1...ru_n]\!]$.
Similarly for $Aut^-(s,o,a)$ and $Aut(s,o,a)$ if there are no explicit rules for them in $ru_1...ru_n$. As such, $[\![ru_1...ru_n]\!]$ grants every right granted by $(ru_1...ru_n)$ and denies everything else [69].

Semantics of compound policies

The semantics of the other policy construct is as follows: Let $Subjects, Objects, Actions$ be, respectively, the universal set of subjects, objects and actions. Note the SANTA construct **policy** pn : po **end** gives policy po a name pn, i.e. it acts as an abbreviation for po so we do not need to give a semantics to this construct.
$[\![po_2\ \ \textbf{chop}\ po_1]\!] \mathrel{\widehat{=}} [\![po_2]\!] \mathbin{\hat{;}} [\![po_1]\!]$
$[\![\textbf{if}\ be\ \textbf{then}\ po_1\ \textbf{else}\ po_2]\!]\ \mathrel{\widehat{=}}\ ((be \wedge [\![po_1]\!]) \vee (\neg be \wedge [\![po_2]\!]))$
$[\![\textbf{aslongas}\ be\ :\ po]\!] \mathrel{\widehat{=}} (\widehat{\mathsf{empty}} \wedge \neg be) \vee (\widehat{\mathsf{fin}}\ \neg be \wedge (\widehat{\mathsf{skip}} \mathbin{\hat{;}} ([\![po]\!] \wedge \widehat{\Box}\ be)))$

4.3 Verification Rules

We will describe the verification of properties and show how the compositional specification of policies can be exploited using compositional proof-rules that simplify the verification tasks.
The following definition states when a policy satisfies a property.
Definition: We say that a policy po satisfies a property f if and only if $[\![po]\!] \supset f$ is valid.
A safety property informally expresses that nothing bad will happen during execution of a program; however, Alpern and Schneider [6] have defined formally the safety property in a future time temporal logic as:

$$\psi \ \mathrel{\widehat{=}}\ \Box f$$

Because we proposed past time operators of ITL^p we will redefine the safety property to be informally expressed that nothing bad happened during execution of a program using history information and that can be written formally as:

$$\psi \ \mathrel{\widehat{=}}\ \widehat{\Box} f$$

Before the second formula is used in our work, we want to show that it is also a safety property.
Let us take this formula:

$$\psi \ \mathrel{\widehat{=}}\ \Box f$$

it means that f is holds over the future interval, so this future formula looks from the end of the future interval $\sigma_{|\sigma|}$ for every prefix intervals to the current state σ_0, or graphically as in Figure 4.7.

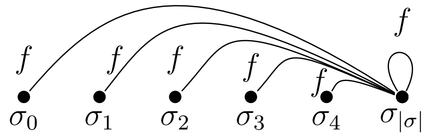

Figure 4.7: Safety property always

Now, let us take this formula:

$$\psi \mathrel{\widehat{=}} \widehat{\boxed{i}}\, f$$

it means that f holds over the past initial intervals, so this past formula means that we are looking from the current state of the past interval τ_0 for every suffix interval to the final state $\tau_{|\tau|}$, or graphically as in Figure 4.8.

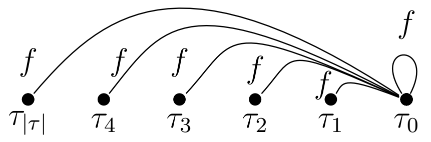

Figure 4.8: Safety property past box-i

As we see from the two figures that the two formula is equivalent and it can be written:

$$\psi \mathrel{\widehat{=}} \Box f \mathrel{\widehat{=}} \widehat{\boxed{i}}\, f$$

Also, we want to show the relation between the operators past always $\widehat{\Box}\, f$ and the operators $\boxed{i}\, f$ and wither the two operators are equal and also a safety property. Let us take this formula:

$$\psi \mathrel{\widehat{=}} \widehat{\Box}\, f$$

it means that f is holds over the past interval, so this past formula look from the end of the past interval $\tau_{|\tau|}$ for every prefix intervals to the current state τ_0, or graphically as in Figure 4.9.

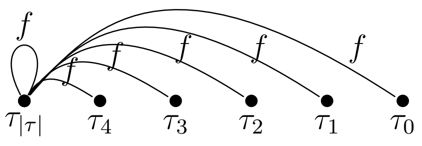

Figure 4.9: Safety property past always

Now, let us take this formula:

$$\psi \ \widehat{=} \ \boxed{\text{i}}\, f$$

it means that f holds over the future initial intervals, so this future formula means that we are looking from the current state of the future interval σ_0 for every suffix interval to the final state $\sigma_{|\sigma|}$, or graphically as in Figure 4.10.

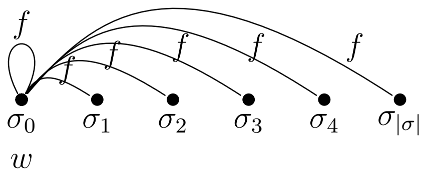

Figure 4.10: Safety property future box-i

As we see from the two Figures 4.9 and 4.10 that the two formulas are equivalent and it can be written:

$$\psi \ \widehat{=} \ \widehat{\Box} f \ \widehat{=} \ \boxed{\text{i}}\, f$$

From above, we have shown in the two Figures 4.7 and 4.8, that we can say that the formulas:

$$\psi \ \widehat{=} \ \Box f$$

and

$$\psi \ \widehat{=} \ \widehat{\boxed{\text{i}}}\, f$$

are equal and can be used as safety property.
Also, we have shown that the two formulas:

$$\psi \quad \widehat{=} \quad \widehat{\Box}\, f$$

and

$$\psi \quad \widehat{=} \quad \boxed{\text{i}}\, f$$

are equal as we have shown in the two Figures 4.9 and 4.10 and can be used as a safety property as well.

4.3.1 Proof Rules:

The following proof rule splits the proof of a property for a complete specification of rules to proofs of individual weak rules. This rule is used when the weak rules have enough information to deduce the property [69].

Proof Rule 1:

$$\frac{[\![ru_1]\!] \supset prop, ..., [\![ru_n]\!] \supset prop}{[\![ru_1 ... ru_n]\!] \supset prop}$$

In case the weak rules does not has enough information, we can use the following stronger rule.

Proof Rule 2:
Let $f(s,o,a), g(s,o,a)$ and $h(s,o,a)$ be defined as in Section 4.4.2

$$\frac{\bigwedge_{\substack{s \in Subjects \\ o \in Objects \\ a \in Actions}} \left(\begin{array}{l} (f(s,o,a) \leftrightarrow Aut^{+}(s,o,a)) \wedge \\ (g(s,o,a) \leftrightarrow Aut^{-}(s,o,a)) \wedge \\ (h(s,o,a) \leftrightarrow Aut(s,o,a)) \end{array} \right) \supset prop}{[\![ru_1 ... ru_n]\!] \supset prop}$$

4.4 Chapter Summary

This Chapter presents a part of our contribution, which is the past time operators and also presents a comparison of existing work.
As part of the preliminaries, we provide an informal description of the underlying computational model of the policies specifications of our work, and put this model into the context of the well-known PDP/PEP model. Security Analysis toolkit (SANTA) policy language and compositions are next introduced and then formal syntax of the language is given with an informal explanation of the syntax. Further, we give a formal semantics to SANTA and define the semantics of individual rules as well as the use of always-followed-by operator and the operator strong-always-followed-by.
The verification rules which are used to verify that a system satisfies properties are given in this Chapter, also how compositional specifications of policies can be exploited using these proof-rules. This will simplify the verification tasks by splitting the proof of a property for a complete specification of rules, to individual proofs of weak rules.
The proposed past time operators of ITL^p used to give the SANTA operators a history-based semantics, and then the existing SANTA and SANTA with history semantics will be used in the Scenario Chapter in order to show that SANTA with past operators is an appropriate choice to reason about history based access control policies.

Chapter 5

SCENARIO: GENERAL PRACTICE SYSTEM (GPS)

In this case study:

- *System description.*
- *Access control policy specification using ITL^p.*
- *Semantics of GPS policies using ITL^p.*
- *Safety property with ITL^p.*
- *Summary of proof*
- *Access control policy specification ITL.*
- *Semantics of GPS policies using ITL.*
- *Safety property with ITL.*
- *evaluation of existed SANTA and SANTA with ITL^p.*
- *Summary of proof*

5.1 Introduction

The current and the following chapters will present two case studies related to our work presented in the previous chapters (3,4 and 5). In this chapters the case study presented is of a General Practice System. Past time Interval Temporal Logic ITL^p is used for defining the syntax and semantics of GPS policies and to reason about them.
We illustrate our approach with a detailed case-study of a General Practice System (GPS) showing the compositional verification of safety. The GPS case study is based on Janicke et al.'s paper "Dynamic Access Control Policies:Specification and Verification" [69]. The GPS case study consists of two sections: Specification, and Verification. In the Specification Section, one specifies a GPS by a set of history-based access control rules. In the Verification Section, one takes this set of rules together with properties that need to hold and check whether this set of rules satisfies these properties.

5.2 System description:

A General Practice System GPS is available to the public via a web interface so users such as doctor, patient, and nurse should register before they can use the system.
Before the first entry to the system, the patient must sign a consent form and legal agreement to allow the General Practice to legally collect and use the personal information of the patient.
After the registration and signing of the consent form and legal agreement, the patient can book an appointment with the chosen doctor if (s)he has not an appointment before, and (s)he can update his/her personal information stored in the system about him/her however, (s)he cannot alter his/her medical records in the system, (s)he can view them only.
A nurse can view the patient medical records if the patient is admitted, but she cannot sign a legal agreement on behalf of any patient. Also (s)he cannot subsequently alter the patient's medical records and doctor's notes.

A doctor can view the personal information and alter the medical records of his patient, also he can add private notes about this patient. However, the doctor cannot treat a patient who is not his patient unless there is a requirement to do so, and this can only be done if the patients has agreed.

5.2.1 Subjects description table

We will list and give a description for all the subjects in the GPS:
S is a member of GPS:
S_{NURSE}: a nurse.
$S_{PATIENT}$: a patient.
S_{DOCTOR}: a doctor.

5.2.2 Objects description table

In table 5.1, we will list all the objects in the GPS and give a description for it:

Object	*Description*
GPS	*General Practice System*
O	*any object in the system*
$O_{medical\_records}$	*Patient medical records*
O_{doctor_notes}	*Doctor notes*
$O_{patient\_personal\_info}$	*Patient personal information in the system*
O_{LA}	*Legal agreement*
O_{CF}	*Consent form*
$O_{APPOINTMENT}$	*appointment with the doctor*
$O_{private\_notes}$	*Doctors private notes*

Table 5.1: Objects description table

5.2.3 Action in the GPS:

In table 5.2, we will list all the actions in the GPS system and give a description for them:

$Action$	$Effect$	$Description$
$done(S_{NURSE}, GPS, access)$	$access(S_{NURSE}, GPS)$	*The nurse has an access to the system*
$done(S_{NURSE}, GPS, view(O_{medical\_records}))$	$view(S_{NURSE}, O_{medical\_records})$	*The nurse can view the patient medical records*
$done(S_{PATIENT}, GPS, access)$	$access(S_{PATIENT}, GPS)$	*A patient has an access to the system*
$done(S_{PATIENT}, GPS, update(O_{patient\_info}))$	$update(S_{PATIENT}, O_{patient\_info})$	*A patient can update personal information stored*
$done(S_{PATIENT}, GPS, treatment)$	$sign(S_{PATIENT}, O_{LA})$	*A patient should sign legal agreement before treatment*
$done(S_{PATIENT}, GPS, treatment)$	$sign(S_{PATIENT}, O_{CF})$	*A patient should sign consent form before treatment*
$done(S_{PATIENT}, GPS, book(O_{APPOINTMENT}, S_{DOCTOR}))$	$has(S_{PATIENT}, O_{APPOINTMENT})$	*A patient has an appointment with the doctor*
$done(S_{PATIENT}, GPS, view(O_{medical\_records}))$	$view(S_{PATIENT}, O_{medical\_records})$	*A patient can view personal medical record in the system*
$done(S_{DOCTOR}, GPS, access)$	$access(S_{DOCTOR}, GPS)$	*A doctor has an access to the system*
$done(S_{DOCTOR}, GPS, view(O_{patient\_info}))$	$view(S_{DOCTOR}, O_{patient\_info})$	*A doctor can view own patients personal info*
$done(S_{DOCTOR}, GPS, alter(O_{medical\_records}))$	$alter(S_{DOCTOR}, O_{medical\_records})$	*A doctor can alter own patients medical records*
$done(S_{DOCTOR}, GPS, add(O))$	$add(S_{DOCTOR}, O_{private\_notes})$	*A doctors can add private notes*
$done(S_{DOCTOR}, GPS, treat(S_{PATIENT}))$	$treat(S_{DOCTOR}, S_{PATIENT})$	*A doctor can treat not his patient when the patient has agree*

Table 5.2: Action in the GPS:

5.3 Past time

In this Section, in order to evaluate the introduced past time operators of interval temporal logic ITL^p, it has been used to reason about general practice system GPS to show that it is convinced choice.

5.3.1 Access control policy specification:

The specification of policy reflects the informal description of the GPS System. We define the policy as a combination of simple policies that correspond to the requirements of each policy. The objective of the scenario is to show how policies are composed to yield a structured specification against which the compositional verification approach can be applied. The rules that refer to history such as R_1, R_2, R_6, R_9, R_{10}, R_{13} and R_{16} are applied in this case study.
We are going to model a GPS policy that consists of three simple policies, Nurse, Patient, and the Doctor. The Nurse's policy is a simple policy containing the rules from R_1 to R_5. The Patient's policy is a simple policy containing the rules from R_6 to R_{12}. The Doctor's policy is a simple policy containing the rules from R_{13} to R_{18}. In addition, we are going to specify the policy rules using the future operators to make a comparison between the past time operators and the future time operators.

5.3.2 Specification with past time

Explanation

Figure 5.1 shows the graphical meaning of the formula:

$$\boldsymbol{true}\ \ \boldsymbol{chop^p}\ \ \boldsymbol{always^p}(registered(S_{PATIENT}))$$

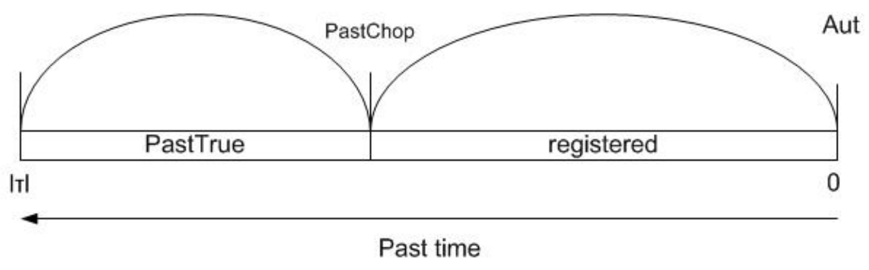

Figure 5.1: AlwaysPastChop

The meaning of this formula is that the past interval is chopped into two interval using $\boldsymbol{chop^p}$, the first interval satisfies the formula $\boldsymbol{always^p}(registered(S_{PATIENT}))$ and before that the second interval satisfies $\boldsymbol{true}$ with the common state in between.

Nurse Rules:

- R_1 :**A nurse has no access to the system before she has registered on the system.**
  $allow(S_{NURSE}, GPS, access)\ \ when\ \ \boldsymbol{true}\ \boldsymbol{chop^p}\ \ \boldsymbol{always^p}(\ registered(S_{NURSE}))$
- R_2 : **A nurse can view the patient medical record when a patient is admitted.**
  $allow(S_{NURSE}, GPS, view(O_{medical\_record}))\ \ when$
  $(\ \boldsymbol{true}\ \ \boldsymbol{chop^p}\ \ \boldsymbol{always^p}(admit(S_{PATIENT})))$ **and**

$medical\_record(O_{medical\_record}, S_{PATIENT})$

- R_3 : **A nurse cannot alter the medical records and doctor's notes.**
  $deny(S_{NURSE}, GPS, alter(O)) when\ 0 : \ (O = O_{medical\_records}$ **or** $O = O_{doctor\_notes})$

- R_4 : **The nurse cannot sign the legal agreement on behalf of a patient.**
  $deny(S_{NURSE}, GPS, sign(O_{LA}))\ \ when$ **true**

- R_5 : **Action is only granted to the nurse if there is a positive authorisation and no negative authorisation.**
  $decide(S_{NURSE}, GPS, A)\ \ when\ \ 0 : \ \ (allow(S_{NURSE}, GPS, A)\ and$
  $not\ deny(S_{NURSE}, GPS, A))$

Patient Rules:

- R_6 : **A patient should be registered to the system before (s)he can access the system.**
  $allow(S_{PATIENT}, GPS, access)$ *when* ***true chop^p always^p***$(registered(S_{PATIENT}))$

- R_7 : **A patient can update his/her own personal information stored in the system.**
  $allow(S_{PATIENT}, GPS, update(O_{patient\_info}))$ *when* $0:$
  $patient\_info(O_{patient\_info}, S_{PATIENT})$

- R_8 : **A patient cannot alter the medical records in the system.**
  $deny(S_{PATIENT}, GPS, alter(O_{medical\_records}))$ *when* ***true***

- R_9 : **The patient should sign the consent form and the legal agreement once before any treatment.**
  $allow(S_{PATIENT}, GPS, treat(S_{PATIENT}))$ *when* ***true chop^p always^p***
  $(\ sign(S_{PATIENT}, O_{LA})$ **and** $sign(S_{PATIENT}, O_{CF}))$

- R_{10} : **The patient can book an appointment with the chosen doctor if (s)he has not an appointment.**
  $allow(S_{PATIENT}, GPS, book(O_{APPOINTMENT}))$ *when*
  true chop^p always^p **not** $booking(S_{PATIENT}, O_{APPOINTMENT})$ ***chop^p skip^p***

- R_{11} : **A patient can view his/her medical record in the system.**
  $allow(S_{PATIENT}, GPS, view(O_{medical\_record}))$ *when* $0:$
  $medical\_record(O_{medical\_record}, S_{PATIENT})$

- R_{12} : **Action is only granted to the patient if there is a positive authorisation and no negative authorisation.**
  $decide(S_{PATIENT}, GPS, A)$ *when* $0:$ $(allow(S_{PATIENT}, GPS, A)$
  and not $deny(S_{PATIENT}, GPS, A))$

Doctor Rules:

- R_{13} : **A doctor has no access to the system before (s)he is registered to the system.**
$allow(S_{DOCTOR}, GPS, access)$ *when* ***true chop^p always^p*** $(registered(S_{DOCTOR}))$

- R_{14} : **The doctor can alter all of the medical records of his patients.**
$allow(S_{DOCTOR}, GPS, alter(O_{medical\_record}))$ *when* $0:$
$doctor(S_{DOCTOR}, S_{PATIENT})$ **and** $medical\_record(O_{medical\_record}, S_{PATIENT})$

- R_{15} : **Doctors can also add private notes about a patient.**
$allow(S_{DOCTOR}, GPS, add(O_{private\_notes}))$ *when* $0:$
$doctor(S_{DOCTOR}, S_{PATIENT})$ **and** $private\_notes(O_{private\_notes}, S_{PATIENT})$

- R_{16} :**A doctor can treat a patient who is not his/her patient if the patient agrees.**
$allow(S_{DOCTOR}, GPS, treat(S_{PATIENT}))$ *when* (***true chop^p always^p*** $(agree(S_{PATIENT})))$**and**$(S_{DOCTOR_1} \neq S_{DOCTOR})$ **and**
always^p$(doctor(S_{DOCTOR_1}, S_{PATIENT}))$ **and**
always^p(**not** $doctor(S_{DOCTOR}, S_{PATIENT}))$

- R_{17} : **The doctor can view his patient's personal information.**
$allow(S_{DOCTOR}, GPS, view(O_{patient\_info}))$ *when* $0:$ $doctor(S_{DOCTOR}, S_{PATIENT})$ **and** $patient\_info(O_{patient\_info}, S_{PATIENT})$

- R_{18} : **Action is only granted to the doctor if there is a positive authorisation and no negative authorisation.**
$decide(S_{DOCTOR}, GPS, A)$ *when* $0:$ $(allow(S_{DOCTOR}, GPS, A)$
and not $deny(S_{DOCTOR}, GPS, A))$

5.3.3 Policies summary

Nurse policy

The nurse policy is a simple policy containing the rules from R_1 to R_4 with the decision rule R_5:
Policy Nurse::

$R_1 : allow(S_{NURSE}, GPS, access)\ when\ \boldsymbol{true\ chop^p\ always^p}(\ registered(S_{NURSE}))$

$R_2 : allow(S_{NURSE}, GPS, view(O_{medical\_record}))\ \ when$
$(\ \boldsymbol{true\ \ chop^p\ \ always^p}(admit(S_{PATIENT})))$ **and**
$medical\_record(O_{medical\_record}, S_{PATIENT})$

$R_3 : deny(S_{NURSE}, GPS, alter(O)) when\ 0:\ \ (O\ =\ O_{medical\_records}$ **or** $O\ =\ O_{doctor\_notes})$

$R_4 : deny(S_{NURSE}, GPS, sign(O_{LA}))\ \ when\ \ \boldsymbol{true}$

$R_5 : decide(S_{NURSE}, GPS, A)\ \ when\ \ 0:$
$(allow(S_{NURSE}, GPS, A) and\ not\ deny(S_{NURSE}, GPS, A))$

end

Patient policy

The patient policy is a simple policy containing the rules from R_6 to R_{11} with the decision rule R_{12}:

Policy Patient::

$R_6 : allow(S_{PATIENT}, GPS, access)\ \ when\ \ \boldsymbol{true}\ \boldsymbol{chop^p}\ \boldsymbol{always^p}(registered(S_{PATIENT}))$

$R_7 : allow(S_{PATIENT}, GPS, update(O_{patient\_info}))\ \ when\ \ 0 :$
$patient\_info(O_{patient\_info}, S_{PATIENT})$

$R_8 : deny(S_{PATIENT}, GPS, alter(O_{medical\_records})\ \ when\ \ \boldsymbol{true}$

$R_9 : allow(S_{PATIENT}, GPS, treat(S_{PATIENT}))\ \ when\ \ \boldsymbol{true}\ \ \boldsymbol{chop^p}\ \ \boldsymbol{always^p}$
$(\ sign(S_{PATIENT}, O_{LA})$ **and** $sign(S_{PATIENT}, O_{CF})$

$R_{10} : allow(S_{PATIENT}, GPS, book(O_{APPOINTMENT}))\ \ when$
$\boldsymbol{true}\ \ \boldsymbol{chop^p}\ \ \boldsymbol{always^p}$ **not** $booking(S_{PATIENT}, O_{APPOINTMENT})\ \boldsymbol{chop^p}\ \boldsymbol{skip^p}$

$R_{11} : allow(S_{PATIENT}, GPS, view(O_{medical\_record}))\ \ when\ \ 0 :$
$medical\_record(O_{medical\_record}, S_{PATIENT})$

$R_{12} : decide(S_{PATIENT}, GPS, A)\ \ when\ \ 0 :\ \ (allow(S_{PATIENT}, GPS, A)$
$and\ not\ deny(S_{PATIENT}, GPS, A))$

end

Doctor policy

The doctor policy is a simple policy containing the rules from R_{13} to R_{17} with the decision rule R_{18} :
Policy Doctor::

$R_{13} : allow(S_{DOCTOR}, GPS, access)$ $when$ ***true chop***

$R_{13} : allow(S_{DOCTOR}, GPS, access)\ \ when\ \boldsymbol{true\ chop^{p}\ always^{p}}\ (registered(S_{DOCTOR}))$

$R_{14} : allow(S_{DOCTOR}, GPS, alter(O_{medical\_record}))\ \ when\ \ 0$
$doctor(S_{DOCTOR}, S_{PATIENT})$ **and** $medical\_record(O_{medical\_record}, S_{PATIENT})$

$R_{15} : allow(S_{DOCTOR}, GPS, add(O_{private\_notes}))\ \ when\ \ 0 :$
$doctor(S_{DOCTOR}, S_{PATIENT})$ **and** $private\_notes(O_{private\_notes}, S_{PATIENT})$

$R_{16} : allow(S_{DOCTOR}, GPS, treat(S_{PATIENT}))\ \ when\ (\ \boldsymbol{true\ \ chop^{p}\ \ always^{p}}$
$(agree(S_{PATIENT})))$**and**$(S_{DOCTOR_1} \neq S_{DOCTOR})$ **and**
$\boldsymbol{always^{p}}(doctor(S_{DOCTOR_1}, S_{PATIENT}))$ **and**
$\boldsymbol{always^{p}}($**not** $doctor(S_{DOCTOR}, S_{PATIENT}))$

$R_{17} : allow(S_{DOCTOR}, GPS, view(O_{patient\_info}))\ \ when\ \ 0 :\ \ doctor(S_{DOCTOR}, S_{PATIENT})$ **and**
$patient\_info(O_{patient\_info}, S_{PATIENT})$

$R_{18} : decide(S_{DOCTOR}, GPS, A)\ \ when\ \ 0 :\ \ (allow(S_{DOCTOR}, GPS, A)$
$and\ not\ deny(S_{DOCTOR}, GPS, A))$

end

5.3.4 Semantics of GPS policies

The following is a mapping from the SANTA policy language used to express the GPS policies into their formal ITL^p semantics. The proofs in Section (5.6) are using the semantic representation of policies.

Semantics of P_{NURSE}

$[\![P_{NURSE}]\!] \equiv [\![\ R_1 ... R_5\ ]\!] \equiv [\![\ R_1^{'}\ ]\!] \wedge [\![\ R_2^{'}\ ]\!] \wedge [\![\ R_3^{'}\ ]\!] \wedge [\![\ R_4^{'}\ ]\!]$ where:

$$[\![\ R_1\ \wedge\ R_5\ ]\!] \equiv [\![\ R_1^{'}\ ]\!]$$

$$\bigwedge_{\substack{s \in Subjects \\ o \in Objects \\ a \in Actions}} \left(\begin{array}{l} (f_{NURSE}(s,o,a) \leftrightarrow Aut^{+}(s,o,a)) \wedge \\ (g_{NURSE}(s,o,a) \leftrightarrow Aut^{-}(s,o,a)) \wedge \\ (h_{NURSE}(s,o,a) \leftrightarrow Aut(s,o,a)) \end{array} \right)$$

where $f_{NURSE}(s,o,a)$ is defined as:

$f_{NURSE}(s,o,a)$	(s,o,a)	R
$[\![$ ***true chop^p always^p***($registered(S_{NURSE})$)$]\!]$	(S_{NURSE},GPS, $access$)	1

and $g_{NURSE}(s,o,a)$ is defined as:

$g_{NURSE}(s,o,a)$	(s,o,a)	R
$[\![$ false $]\!]$	$(S_{NURSE}, GPS, access)$	1

and $h_{NURSE}(s,o,a)$ is defined as:

$h_{NURSE}(s,o,a)$	(s,o,a)	R
$[\![0:$ $allow(S_{NURSE}$,GPS, $access)$ **and not** $deny(S_{NURSE}$,GPS, $access)]\!]$	(S_{NURSE},GPS, $access$)	5

and $[\![\ R_2 \ \wedge \ R_5\ ]\!] \equiv [\![\ R_2'\ ]\!]$

where $f_{NURSE}(s,o,a)$ is defined as:

$f_{NURSE}(s,o,a)$	(s,o,a)	R
$[\![\ (\ \boldsymbol{true}\ \ \boldsymbol{chop^p}\ \ \boldsymbol{always^p}(admit(S_{PATIENT})))$ **and** $medical\_record(O_{medical\_record}, S_{PATIENT})]\!]$	$(S_{NURSE}$,GPS$, view(O_{medical_record}))$	2

and $g_{NURSE}(s,o,a)$ is defined as:

$g_{NURSE}(s,o,a)$	(s,o,a)	R
$[\![$ false $]\!]$	$(S_{NURSE}$,GPS$, view(O_{medical_record}))$	2

and $h_{NURSE}(s,o,a)$ is defined as:

$h_{NURSE}(s,o,a)$	(s,o,a)	R
$[\![0:\ allow(S_{NURSE}$,GPS$, view(O_{medical_record}))$ **and not** $deny(S_{NURSE}$,GPS$, view(O_{medical_record}))]\!]$	$(S_{NURSE}$,GPS$, view(O_{medical_record}))$	5

and $\llbracket\ R_3 \ \wedge\ R_5\ \rrbracket \equiv \llbracket\ R_3^{'}\ \rrbracket$

where $f_{NURSE}(s, o, a)$ is defined as:

$f_{NURSE}(s, o, a)$	(s, o, a)	R
$\llbracket$ false $\rrbracket$	$(S_{NURSE}$,GPS, $alter(O))$	3

and $g_{NURSE}(s, o, a)$ is defined as:

$g_{NURSE}(s, o, a)$	(s, o, a)	R
$\llbracket 0 :\ (O\ =\ O_{medical\_records})$ **or** $(O\ = O_{doctor\_notes})\rrbracket$	$(S_{NURSE}$,GPS, $alter(O))$	3

and $h_{NURSE}(s, o, a)$ is defined as:

$h_{NURSE}(s, o, a)$	(s, o, a)	R
$\llbracket 0 :\ deny(S_{NURSE}$,GPS, $alter(O))$ **and not** $allow(S_{NURSE}$,GPS, $alter(O))\rrbracket$	$(S_{NURSE}$,GPS, $alter(O))$	5

and $[\![R_4 \ \wedge \ R_5]\!] \equiv [\![R_4^{'}]\!]$
where $f_{NURSE}(s, o, a)$ is defined as:

$f_{NURSE}(s, o, a)$	(s, o, a)	R
$[\![$ false $]\!]$	$(S_{NURSE}$,GPS, $sign(O))$	4

and $g_{NURSE}(s, o, a)$ is defined as:

$g_{NURSE}(s, o, a)$	(s, o, a)	R
$[\![$ true $]\!]$	$(S_{NURSE}$,GPS, $sign(O_{LA}))$	4

and $h_{NURSE}(s, o, a)$ is defined as:

$h_{NURSE}(s, o, a)$	(s, o, a)	R
$[\![0:$ $deny(S_{NURSE}$,GPS, $sign(O))$ **and not** $allow(S_{NURSE}$,GPS, $sign(O))]\!]$	$(S_{NURSE}$,GPS, $sign(O))$	5

Semantics of $P_{PATIENT}$

$[\![P_{PATIENT}]\!] \equiv [\![\ R_6...R_{12}]\!] \equiv [\![\ R_6^{'}\ ]\!] \wedge [\![\ R_7^{'}\ ]\!] \wedge [\![\ R_8^{'}\ ]\!] \wedge [\![\ R_9^{'}\ ]\!] \wedge [\![\ R_{10}^{'}\ ]\!] \wedge [\![\ R_{11}^{'}\ ]\!]$

where $[\![\ R_6^{'}\ ]\!], [\![\ R_7^{'}\ ]\!], [\![\ R_8^{'}\ ]\!], [\![\ R_9^{'}\ ]\!], [\![\ R_{10}^{'}\ ]\!], [\![\ R_{11}^{'}\ ]\!]$
are defined in Appendix B Part 1.

Semantics of P_{DOCTOR}

$[\![P_{DOCTOR}]\!] \equiv [\![\ R_{13}...R_{18}\ ]\!] \equiv [\![\ R_{13}^{'}\ ]\!] \wedge [\![\ R_{14}^{'}\ ]\!] \wedge [\![\ R_{15}^{'}\ ]\!] \wedge [\![\ R_{16}^{'}\ ]\!] \wedge [\![\ R_{17}^{'}\ ]\!]$
where $[\![\ R_{13}^{'}\ ]\!], [\![\ R_{14}^{'}\ ]\!], [\![\ R_{15}^{'}\ ]\!], [\![\ R_{16}^{'}\ ]\!], [\![\ R_{17}^{'}\ ]\!]$ are defined in Appendix B Part 2.

5.3.5 Safety property:

A safety property informally expresses that nothing bad will happen during execution of a program or formally:

$$\psi \ \ \widehat{=} \ \ \widehat{\Box}\, h$$

We will give a specific safety property for every member of the *GPS* and prove that this property holds for each member.

Nurse Safety Property:

The safety property can be defined as:
It is never the case that the nurse can use *GPS* without being registered in the system.
Let $\psi_1 \ \ \widehat{=} \ \widehat{\Box}((Aut(S_{NURSE}, \text{GPS}, a)) \ \supset \ \texttt{true} \ \hat{;} \ \widehat{\Box}(registered(S_{NURSE})))$
denote the nurse safety property and let $a \ \in \ \{access\ ,\ view\ ,\ alter\}$

The proof that the *GPS* policy satisfies ψ_1 can be done by proving that each of the nurse rules in this policy which affect the safety property will not invalidate this property ψ_1, i.e. $[\![R_1...R_5]\!] \supset \psi_1$.
We have to prove that $[\![R_1 \ \wedge \ R_5]\!] \supset \psi_1, [\![R_2 \ \wedge \ R_5]\!] \supset \psi_1, [\![R_3 \wedge R_5]\!] \supset \psi_1$.
Here $[\![R_i]\!]$ denotes only the rules that affect the safety property so the rules $[\![R_4$ and $R_5]\!] \supset \psi_1$ are not included because they cannot affect (invalidate) ψ_1.

We can prove that the nurse policy satisfies the safety property by proving that $R_1^{'}$ and $R_2^{'}$ and $R_3^{'}$ which affect the safety property satisfies ψ_1.

Rule $R_1^{'}$ states:

$R_1^{'}$: A nurse has no access to the system before she has registered on the system.

$$allow(S_{NURSE}, \text{GPS}, access)\ when\ \textbf{\textit{true chop}}^{\boldsymbol{p}}\ \textbf{\textit{always}}^{\boldsymbol{p}}(\ registered(S_{NURSE}))$$

1. From the semantics of P_{NURSE} we have:
   $[\![R_1 \wedge R_5\ ]\!] \equiv [\![\ R_1^{'}\ ]\!] \equiv$
   $$\bigwedge_{\substack{s \in Subjects \\ o \in Objects \\ a \in Actions}} \begin{pmatrix} (f_{NURSE}(s,o,a) \leftrightarrow Aut^{+}(s,o,a)) \wedge \\ (g_{NURSE}(s,o,a) \leftrightarrow Aut^{-}(s,o,a)) \wedge \\ (h_{NURSE}(s,o,a) \leftrightarrow Aut(s,o,a)) \end{pmatrix}$$
   $\mathsf{true}\ \hat{;}\ \widehat{\Box}(\ registered(S_{NURSE}))\ \leftrightarrow\ Aut^{+}(S_{NURSE}, \text{GPS}, access)$
   $\wedge\ \mathsf{false}\ \leftrightarrow\ Aut^{-}(S_{NURSE}, \text{GPS}, access)$
   $\wedge\ Aut^{+}(S_{NURSE}, \text{GPS}, access)\ \wedge\ \neg\ Aut^{-}(S_{NURSE}, \text{GPS}, access)\ \leftrightarrow$
   $Aut(S_{NURSE}, \text{GPS}, access)$

2. from the definition of $(\leftrightarrow)$ we have:
   $h\ \leftrightarrow\ w\ \ \widehat{=}\ \ \widehat{\Box}(h\ \equiv\ w)$

3. $\mathsf{true}\ \hat{;}\ \widehat{\Box}(\ registered(S_{NURSE}))\ \equiv\ Aut^{+}(S_{NURSE}, \text{GPS}, access)$
   $\wedge\ \mathsf{false}\ \equiv\ Aut^{-}(S_{NURSE}, \text{GPS}, access)$
   $\wedge\ Aut^{+}(S_{NURSE}, \text{GPS}, access)\ \wedge\ \neg\ Aut^{-}(S_{NURSE}, \text{GPS}, access)\ \equiv$
   $Aut(S_{NURSE}, \text{GPS}, access)$

4. so, we have:
   $\widehat{\Box}(Aut^{+}(S_{NURSE}, \text{GPS}, access)\ \wedge\ \neg\ Aut^{-}(S_{NURSE}, \text{GPS}, access)$
   $\equiv\ Aut(S_{NURSE}, \text{GPS}, access))$

5. but we know that:
   $\mathsf{false}\ \leftrightarrow\ Aut^{-}(S_{NURSE}, \text{GPS}, access)$
   so, step 3 can be written as:
   $\widehat{\Box}(Aut^{+}(S_{NURSE}, \text{GPS}, access)\ \equiv\ Aut(S_{NURSE}, \text{GPS}, access))$

6. from above we have:
   $\widehat{\Box}(\mathsf{true} \,\hat{;}\, \widehat{\Box}(\ registered(S_{NURSE})) \ \equiv\ Aut(S_{NURSE},\text{GPS}, access))$

7. using ITL^p reasoning, we have:
   $\widehat{\Box}(Aut(S_{NURSE},\text{GPS}, access) \ \supset\ \mathsf{true} \,\hat{;}\, \widehat{\Box}(\ registered(S_{NURSE})))$

8. and this is ψ_1.

Rule $R_2^{'}$ states:

$R_2^{'}$: A nurse can view the patient's medical records when a patient is admitted.
$allow(S_{NURSE}, \text{GPS}, view(O_{medical\_records}))$ *when*
true chop^p always^p$((admit(S_{PATIENT}))$ **and** $medical\_record(S_{PATIENT}))$

1. From the semantics of P_{NURSE} we have:
   $[\![R_2 \wedge R_5]\!] \equiv [\![R_2^{'}]\!] \equiv$
   $$\bigwedge_{\substack{s \in Subjects \\ o \in Objects \\ a \in Actions}} \left(\begin{array}{c} (f_{NURSE}(s,o,a) \leftrightarrow Aut^{+}(s,o,a)) \wedge \\ (g_{NURSE}(s,o,a) \leftrightarrow Aut^{-}(s,o,a)) \wedge \\ (h_{NURSE}(s,o,a) \leftrightarrow Aut(s,o,a)) \end{array} \right)$$
   $\mathsf{true} \, ; \, \widehat{\Box}((admit(S_{PATIENT})) \wedge medical\_record(O_{medical\_record}, S_{PATIENT})) \leftrightarrow$
   $Aut^{+}(S_{NURSE}, \text{GPS}, view(O_{medical\_record}))$
   $\wedge \, \mathsf{false} \leftrightarrow Aut^{-}(S_{NURSE}, \text{GPS}, view(O_{medical\_record}))$
   $\wedge \, Aut^{+}(S_{NURSE}, \text{GPS}, view(O_{medical\_record})) \wedge \neg \, Aut^{-}(S_{NURSE}, \text{GPS}, view(O_{medical\_record}))$
   $\leftrightarrow \; Aut(S_{NURSE}, \text{GPS}, view(O_{medical\_record}))$

2. from the definition of $(\leftrightarrow)$ we have:
   $h \leftrightarrow w \;\widehat{=}\; \widehat{\Box}(h \equiv w)$

3. $\mathsf{true} \, ; \, \widehat{\Box}((admit(S_{PATIENT})) \wedge medical\_record(O_{medical\_record}, S_{PATIENT})) \equiv$
   $Aut^{+}(S_{NURSE}, \text{GPS}, view(O_{medical\_record}))$
   $\wedge \, \mathsf{false} \equiv Aut^{-}(S_{NURSE}, \text{GPS}, view(O_{medical\_record}))$
   $\wedge \, Aut^{+}(S_{NURSE}, \text{GPS}, view(O_{medical\_record})) \wedge \neg \, Aut^{-}(S_{NURSE}, \text{GPS}, view(O_{medical\_record}))$
   $\equiv \; Aut(S_{NURSE}, \text{GPS}, view(O_{medical\_record}))$

4. so, we have:
   $\widehat{\Box}(Aut^{+}(S_{NURSE}, \text{GPS}, view(O_{medical\_record})) \wedge \neg \, Aut^{-}(S_{NURSE}, \text{GPS}, view(O_{medical\_record}))$
   $\equiv \; Aut(S_{NURSE}, \text{GPS}, view(O_{medical\_record})))$

5. but we know that:
   $\mathsf{false} \leftrightarrow Aut^{-}(S_{NURSE}, \text{GPS}, view(O_{medical\_record}))$

6. so, we can write it as:
$\widehat{\Box}(Aut^{+}(S_{NURSE},\text{GPS}, view(O_{medical\_record})) \equiv Aut(S_{NURSE},\text{GPS}, view(O_{medical\_record})))$

7. and we have:
$\widehat{\Box}(\mathsf{true} \,\hat{;}\, \widehat{\Box}((admit(S_{PATIENT})) \wedge medical\_record(O_{medical\_record}, S_{PATIENT})) \equiv$

   $Aut(S_{NURSE},\text{GPS}, view(O_{medical\_record})))$

8. this is equal to:
$\widehat{\Box}(Aut(S_{NURSE},\text{GPS}, view(O_{medical\_record})) \equiv$
$\mathsf{true} \;\hat{;}\; \widehat{\Box}((admit(S_{PATIENT})) \;\wedge\; medical\_record(O_{medical\_record}, S_{PATIENT})))$

9. we assume that view can only be after registration, so we can say that:
$\widehat{\Box}(Aut(S_{NURSE},\text{GPS}, view(O_{medical\_record})) \equiv \mathsf{true} \,\hat{;}\, \widehat{\Box}(\, registered(S_{NURSE})))$

10. and from it we have:
$\widehat{\Box}(Aut(S_{NURSE},\text{GPS}, view(O_{medical\_record})) \supset \mathsf{true} \,\hat{;}\, \widehat{\Box}(\, registered(S_{NURSE})))$

11. we can say that $R_2^{'}$ satisfies ψ_1

Rule R_3' states:

R_3' : A nurse cannot alter the medical records and doctor's notes.

$deny(S_{NURSE}, \text{GPS}, alter(O)) when\ \ 0 : \ \ (O = O_{medical\_records}$ **or** $O = O_{doctor\_notes})$

1. From the semantics of P_{NURSE} we have:
   $[\![R_3 \wedge R_5]\!] \equiv [\![R_3']\!] \equiv$
   $[\![P_{NURSE}]\!] \equiv [\![R_3 \ \wedge \ R_5]\!] \equiv [\![R_3']\!] \equiv$
   $$\bigwedge_{\substack{s \in Subjects \\ o \in Objects \\ a \in Actions}} \left(\begin{array}{c} (f_{NURSE}(s,o,a) \leftrightarrow Aut^{+}(s,o,a)) \wedge \\ (g_{NURSE}(s,o,a) \leftrightarrow Aut^{-}(s,o,a)) \wedge \\ (h_{NURSE}(s,o,a) \leftrightarrow Aut(s,o,a)) \end{array} \right)$$
   $\mathsf{false} \ \leftrightarrow Aut^{+}(S_{NURSE}, \text{GPS}, alter(O))$
   $\wedge \ O \ = \ (O_{medical\_records} \ \vee \ O_{doctor\_notes}) \leftrightarrow Aut^{-}(S_{NURSE}, \text{GPS}, alter(O))$
   $\wedge \ Aut^{-}(S_{NURSE}, \text{GPS}, alter(O)) \ \wedge \ \neg \ Aut^{+}(S_{NURSE}, \text{GPS}, alter(O))$
   $\leftrightarrow \ Aut(S_{NURSE}, \text{GPS}, alter(O))$

2. from the definition of $(\leftrightarrow)$ we have:
   $h \ \leftrightarrow \ w \ \ \widehat{=} \ \ \widehat{\Box}(h \ \equiv \ w)$

3. $\mathsf{false} \ \equiv Aut^{+}(S_{NURSE}, \text{GPS}, alter(O))$
   $\wedge \ O \ = \ (O_{medical\_records} \ \vee \ O_{doctor\_notes}) \equiv Aut^{-}(S_{NURSE}, \text{GPS}, alter(O))$
   $\wedge \ Aut^{-}(S_{NURSE}, \text{GPS}, alter(O)) \ \wedge \ \neg \ Aut^{+}(S_{NURSE}, \text{GPS}, alter(O))$
   $\equiv \ Aut(S_{NURSE}, \text{GPS}, alter(O))$

4. so, we have:
   $\widehat{\Box}(Aut^{-}(S_{NURSE}, \text{GPS}, alter(O)) \ \wedge \ \neg \ Aut^{+}(S_{NURSE}, \text{GPS}, alter(O)) \ \equiv$
   $Aut(S_{NURSE}, \text{GPS}, alter(O)))$

5. We know that:
   $\mathsf{false} \ \equiv \ Aut^{+}(S_{NURSE}, \text{GPS}, alter(O))$
   So, we can simplify it to:
   $\widehat{\Box}(Aut^{-}(S_{NURSE}, \text{GPS}, alter(O)) \ \equiv \ Aut(S_{NURSE}, \text{GPS}, alter(O)))$

6. this yields to:
$\widehat{\Box}(\mathsf{true} \mathbin{\hat{;}} (len(0) \wedge (O = O_{medical\_records} \vee O = O_{Doctor\_notes})) \equiv Aut(S_{NURSE}, \mathrm{GPS}, alter(O)))$

7. and from the reasoning of ITL^p we have:
$\widehat{\Box}((O = O_{medical\_records} \vee O = O_{Doctor\_notes}) \equiv Aut(S_{NURSE}, \mathrm{GPS}, alter(O)))$

8. this is equal to:
$\widehat{\Box}(Aut(S_{NURSE}, \mathrm{GPS}, alter(O)) \equiv (O = O_{medical\_records} \vee O = O_{doctor\_notes}))$

9. by assume that alter can be after registration, so we can say that:
$\widehat{\Box}(Aut(S_{NURSE}, \mathrm{GPS}, alter(O)) \supset (O = O_{medical\_records} \vee O = O_{doctor\_notes}))$

10. from it we can say that R_3' satisfies ψ_1.

Patient Safety Property:

It is never the case that the patient can alter the medical records in the system.
Let $\psi_2 \;\; \widehat{=} \; \widehat{\Box}(\neg Aut(S_{PATIENT}, \text{GPS}, alter(O_{medical\_records})))$
denote the safety property.
A compositional proof that patient policy satisfies ψ_2 can be done using proof rules by proving that each of the rules in this policy satisfies the safety property or formally:
$[\![R'_6 ... R'_{11}]\!] \supset \psi_2$.
We have to prove that $[\![R'_6 \wedge R'_{11}]\!] \supset \psi_2$. Here $[\![R'_i]\!]$ denotes only rules that can affect the safety property. The rules $[\![R'_6, R'_7, R'_9, R'_{10}$ and $R'_{11}]\!]$ cannot invalidate ψ_2.
We can prove that the patient policy satisfies the safety property by proving that R'_8 which affect the safety property satisfies ψ_2.

Rule R'_8 states:

R'_8 : A patient cannot alter the medical records in the system.

$$deny(S_{PATIENT}, \text{GPS}, alter(O_{medical\_records}) \;\; when \;\; \boldsymbol{true}$$

1. From the semantics of $P_{PATIENT}$ we have:

$$[\![\; R_8 \;\wedge\; R_{12} \;]\!] \equiv [\![\; R'_8 \;]\!] \equiv$$

$$\bigwedge_{\substack{s \in Subjects \\ o \in Objects \\ a \in Actions}} \left(\begin{array}{l} (f_{PATIENT}(s,o,a) \leftrightarrow Aut^{+}(s,o,a)) \wedge \\ (g_{PATIENT}(s,o,a) \leftrightarrow Aut^{-}(s,o,a)) \wedge \\ (h_{PATIENT}(s,o,a) \leftrightarrow Aut(s,o,a)) \end{array} \right)$$

$$\begin{array}{l} \mathsf{false} \;\leftrightarrow\; Aut^{+}(S_{PATIENT}, \text{GPS}, alter(O_{medical\_records})) \\ \wedge \;\; \mathsf{true} \;\leftrightarrow\; Aut^{-}(S_{PATIENT}, \text{GPS}, alter(O_{medical\_records})) \\ \wedge \; Aut^{-}(S_{PATIENT}, \text{GPS}, alter(O_{medical\_records})) \\ \wedge \;\; \neg \; Aut^{+}(S_{PATIENT}, \text{GPS}, alter(O_{medical\_records})) \;\; \leftrightarrow \\ Aut(S_{PATIENT}, \text{GPS}, alter(O_{medical\_records})) \end{array}$$

2. from the definition of $(\leftrightarrow)$ we have:
$h \;\leftrightarrow\;\; w \;\;\widehat{=}\;\; \widehat{\Box}(h \;\equiv\; w)$

3. $\mathsf{false} \equiv Aut^{+}(S_{PATIENT},\text{GPS}, alter(O_{medical\_records}))$
$\wedge\ \mathsf{true} \equiv Aut^{-}(S_{PATIENT},\text{GPS}, alter(O_{medical\_records}))$
$\wedge\ Aut^{-}(S_{PATIENT},\text{GPS}, alter(O_{medical\_records}))$
$\wedge\ \neg\ Aut^{+}(S_{PATIENT},\text{GPS}, alter(O_{medical\_records})) \equiv$
$Aut(S_{PATIENT},\text{GPS}, alter(O_{medical\_records}))$

4. and from ITL^{P} reasoning, we have:

   $\widehat{\Box}(Aut^{-}(S_{PATIENT},\text{GPS}, alter(O_{medical\_records}))$
   $\wedge\ \neg\ Aut^{+}(S_{PATIENT},\text{GPS}, alter(O_{medical\_records})) \equiv$
   $Aut(S_{PATIENT},\text{GPS}, alter(O_{medical\_records})))$

5. which can be simplified to:

   $\widehat{\Box}(\mathsf{true} \equiv Aut(S_{PATIENT},\text{GPS}, alter(O_{medical\_records})))$

6. which can be written as:

   $\widehat{\Box}(\neg Aut(S_{PATIENT},\text{GPS}, alter(O_{medical\_records})))$

7. which is ψ_2.

Doctor Safety Property:

It is never the case that anyone can modify the patient medical records without being a doctor of this patient.
Let $\psi_3 \ \widehat{=} \ \widehat{\Box}(Aut(S_{DOCTOR}, \text{GPS}, alter(O_{medical\_records})) \ \supset$
$doctor(S_{DOCTOR}, S_{PATIENT}) \ \wedge \ medical\_records(S_{PATIENT}))$
denote the safety property.
A compositional proof that any policy satisfies ψ_3 can be done using proof rules by proving that each of the rules in this policy satisfies the safety property or formally: $[\![R_{13}...R_{18}]\!] \supset \psi_3$.
We have to prove that $[\![R_{14} \ \wedge \ R_{18}]\!] \ \widehat{=} \ [\![R'_{14}]\!] \supset \psi_3$.

Here $[\![R_i]\!]$ denotes only rules that can affect the safety property. $[\![R'_{13} \, , R'_{15} ... \, R'_{18}]\!] \supset \psi_3$ because they cannot affect ψ_3 .
We can prove that the doctor policy satisfies the safety property by proving that R'_{14} which affect the safety property satisfies ψ_3.
Rule R'_{14} states:
R'_{14} : The doctor can alter all of the medical records of his/her patients.
$allow(S_{DOCTOR}, \text{GPS}, alter(O_{medical\_record})) \ \ when \ \ 0 :$
$doctor(S_{DOCTOR}, S_{PATIENT})$ **and** $medical\_record(O_{medical\_record}, S_{PATIENT})$

1. From the semantics of P_{DOCTOR} we have:

$$[\![\ R_{14} \ \wedge \ R_{18} \ ]\!] \equiv [\![\ R'_{14} \ ]\!] \equiv$$
$$\bigwedge_{\substack{s \in Subjects \\ o \in Objects \\ a \in Actions}} \left(\begin{array}{l} (f_{DOCTOR}(s,o,a) \leftrightarrow Aut^{+}(s,o,a)) \wedge \\ (g_{DOCTOR}(s,o,a) \leftrightarrow Aut^{-}(s,o,a)) \wedge \\ (h_{DOCTOR}(s,o,a) \leftrightarrow Aut(s,o,a)) \end{array} \right)$$
$$doctor(S_{DOCTOR}, S_{PATIENT}) \ \wedge \ medical\_record(O_{medical\_record}, S_{PATIENT}) \ \leftrightarrow$$
$$Aut^{+}(S_{DOCTOR}, \text{GPS}, alter(O_{medical\_record}))$$
$$\wedge \ \mathsf{false} \ \leftrightarrow Aut^{-}(S_{DOCTOR}, \text{GPS}, alter(O_{medical\_record}))$$
$$\wedge \ Aut^{+}(S_{DOCTOR}, \text{GPS}, alter(O_{medical\_record}))$$
$$\wedge \ \neg \ Aut^{-}(S_{DOCTOR}, \text{GPS}, alter(O_{medical\_record})) \ \leftrightarrow$$
$$Aut(S_{DOCTOR}, \text{GPS}, alter(O_{medical\_record}))$$

2. from the definition of $(\leftrightarrow)$ we have:

$h \leftrightarrow w \ \widehat{=} \ \widehat{\Box}(h \equiv w)$

3. and from the reasoning of ITL^p, so we have:
$\widehat{\Box}(Aut^+(S_{DOCTOR},\text{GPS}, alter(O_{medical\_record}))$
$\wedge \neg Aut^-(S_{DOCTOR},\text{GPS}, alter(O_{medical\_record})) \equiv$
$Aut(S_{DOCTOR},\text{GPS}, alter(O_{medical\_record})))$

4. which can be simplified to:
$\widehat{\Box}(Aut^+(S_{DOCTOR},\text{GPS}, alter(O_{medical\_record})) \equiv$
$Aut(S_{DOCTOR},\text{GPS}, alter(O_{medical\_record})))$

5. and we can write as:
$\widehat{\Box}(doctor(S_{DOCTOR}, S_{PATIENT}) \wedge medical\_record(O_{medical\_record}, S_{PATIENT}) \equiv$

$Aut(S_{DOCTOR},\text{GPS}, alter(O_{medical\_record})))$

6. so we have:
$\widehat{\Box}(Aut(S_{DOCTOR},\text{GPS}, alter(O_{medical\_record})) \supset (doctor(S_{DOCTOR}, S_{PATIENT}) \wedge$

$medical\_record(O_{medical\_record}, S_{PATIENT})))$

7. which is ψ_3.

Proof summary

We can proof that any policy satisfies the safety property, this can be done using the proof rules in (Section 4.5.1) by proving that each of the rules in this policy satisfies the property.

NURSE

The nurse policy consists from simple policies which is from R_1 to R_5:
and from the semantics of P_{NURSE} we have:
$[\![P_{NURSE}]\!] \ = \ [\![\ R_1^{'} \ \wedge \ R_2^{'} \ \wedge \ R_3^{'} \ \wedge \ R_4^{'}\ ]\!]$
we prove that:
$[\![R_1]\!] \supset \psi_1$
$[\![R_2]\!] \supset \psi_1$
$[\![R_3]\!] \supset \psi_1$
so each rule that affect the safety property in nurse policy satisfies ψ_1 and from the proof rule 2 we have:
$[\![R_1^{'} \wedge R_2^{'} \wedge R_3^{'}\ ]\!] \ \ \supset \ \ \psi_1$
and from above, we can say that:
$[\![P_{NURSE}]\!] \ \ \supset \ \ \psi_1$

PATIENT

The patient policy consists from simple policies which is from R_6 to R_{12}:
and from the semantics of $P_{PATIENT}$ we have:
$[\![P_{PATIENT}]\!] \ = \ [\![\ R_6^{'} \ \wedge \ R_7^{'} \ \wedge \ R_8^{'} \ \wedge \ R_9^{'} \ \wedge \ R_{10}^{'} \ \wedge \ R_{11}^{'}\ ]\!]$
we prove that:
$[\![\ R_8^{'}\ ]\!] \supset \psi_2$
so each rule that affect the safety property in patient policy satisfies ψ_2 and from the proof rule 2 we can say that:
$[\![P_{PATIENT}]\!] \ \ \supset \ \ \psi_2$

DOCTOR

The doctor policy consists from simple policies which is from R_{13} to R_{18}:
and from the semantics of P_{DOCTOR} we have:
$[\![P_{DOCTOR}]\!] \ = \ [\![\ R'_{13} \ \wedge \ R'_{14} \ \wedge \ R'_{15} \ \wedge \ R'_{16} \ \wedge \ R'_{17} \ ]\!]$
we prove that:
$[\![\ R'_{14} \ ]\!] \supset \psi_3$
so each rule that affect the safety property in doctor policy satisfies ψ_3 and from the proof rule 2, and from it we can say that:
$[\![P_{DOCTOR}]\!] \ \supset \ \psi_3$

5.4 Future time

In this Section, the future time operators of ITL have been used to reason about general practice system GPS and compare it with the introduced past time operators to evaluate it.

5.4.1 Specification with future time

Nurse Rules:

- R_1 :**A nurse has no access to the system before she has registered on the system.**
  $allow(S_{NURSE}, \text{GPS}, access)\ \ when\ \ 0:\ \ \boldsymbol{true\ chop\ always}(\ registered(S_{NURSE}))$

- R_2 : **A nurse can view the patient medical record when a patient is admitted.**
  $allow(S_{NURSE}, \text{GPS}, view(O_{medical\_record}))\ \ when\ \ 0:$
  $(\ \boldsymbol{true\ \ chop\ \ always}(admit(S_{PATIENT})))$ **and**
  $medical\_record(O_{medical\_record}, S_{PATIENT})$

- R_3 : **A nurse cannot alter the medical records and doctor's notes.**
  $deny(S_{NURSE}, \text{GPS}, alter(O)) when\ \ 0:\ \ (O\ =\ O_{medical\_records}$ **or** $O\ =\ O_{doctor\_notes})$

- R_4 : **The nurse cannot sign the legal agreement on behalf of a patient.**
  $deny(S_{NURSE}, \text{GPS}, sign(O_{LA}))\ \ when$ **true**

- R_5 : **Action is only granted to the nurse if there is a positive authorisation and no negative authorisation.**
  $decide(S_{NURSE}, \text{GPS}, A)\ \ when\ \ 0:\ \ (allow(S_{NURSE}, \text{GPS}, A)\ and$
  $not\ deny(S_{NURSE}, \text{GPS}, A))$

Patient Rules:

- R_6 : **A patient should be registered to the system before (s)he can access the system.**
  $allow(S_{PATIENT}, \text{GPS}, access)$ *when* $0:$ ***true chop always***$(registered(S_{PATIENT}))$

- R_7 : **A patient can update his/her own personal information stored in the system.**
  $allow(S_{PATIENT}, \text{GPS}, update(O_{patient\_info}))$ *when* $0:$
  $patient\_info(O_{patient\_info}, S_{PATIENT})$

- R_8 : **A patient cannot alter the medical records in the system.**
  $deny(S_{PATIENT}, \text{GPS}, alter(O_{medical\_records}))$ *when* ***true***

- R_9 : **The patient should sign the consent form and the legal agreement once before any treatment.**
  $allow(S_{PATIENT}, \text{GPS}, treat(S_{PATIENT}))$ *when* $0:$ ***true chop always***
  $(sign(S_{PATIENT}, O_{LA})$ **and** $sign(S_{PATIENT}, O_{CF}))$

- R_{10} : **The patient can book an appointment with the chosen doctor if (s)he has not an appointment.**
  $allow(S_{PATIENT}, \text{GPS}, book(O_{APPOINTMENT}))$ *when* $0:$
  true chop always **not** $booking(S_{PATIENT}, O_{APPOINTMENT})$ ***chop skip***

- R_{11} : **A patient can view his/her medical record in the system.**
  $allow(S_{PATIENT}, \text{GPS}, view(O_{medical\_record}))$ *when* $0:$
  $medical\_record(O_{medical\_record}, S_{PATIENT})$

- R_{12} : **Action is only granted to the patient if there is a positive authorisation and no negative authorisation.**
  $decide(S_{PATIENT}, \text{GPS}, A)$ *when* $0:$ $(allow(S_{PATIENT}, \text{GPS}, A)$
  and not $deny(S_{PATIENT}, \text{GPS}, A))$

Doctor Rules:

- R_{13} : **A doctor has no access to the system before (s)he is registered to the system.**
  $allow(S_{DOCTOR}, \text{GPS}, access)$ $when$ $0:$ ***true chop always*** $(registered(S_{DOCTOR}))$

- R_{14} : **The doctor can alter all of the medical records of his patients.**
  $allow(S_{DOCTOR}, \text{GPS}, alter(O_{medical\_record}))$ $when$ $0:$
  $doctor(S_{DOCTOR}, S_{PATIENT})$ **and** $medical\_record(O_{medical\_record}, S_{PATIENT})$

- R_{15} : **Doctors can also add private notes about a patient.**
  $allow(S_{DOCTOR}, \text{GPS}, add(O_{private\_notes}))$ $when$ $0:$
  $doctor(S_{DOCTOR}, S_{PATIENT})$ **and** $private\_notes(O_{private\_notes}, S_{PATIENT})$

- R_{16} :**A doctor can treat a patient who is not his/her patient if the patient agrees.**
  $allow(S_{DOCTOR}, \text{GPS}, treat(S_{PATIENT}))$ $when$ $0:$ (***true chop always*** $(agree(S_{PATIENT}))$)**and**$(S_{DOCTOR_1} \neq S_{DOCTOR})$ **and**
  always$(doctor(S_{DOCTOR_1}, S_{PATIENT}))$ **and**
  always(**not** $doctor(S_{DOCTOR}, S_{PATIENT})$)

- R_{17} : **The doctor can view his patient's personal information.**
  $allow(S_{DOCTOR}, \text{GPS}, view(O_{patient\_info}))$ $when$ $0:$ $doctor(S_{DOCTOR}, S_{PATIENT})$ **and**
  $patient\_info(O_{patient\_info}, S_{PATIENT})$

- R_{18} : **Action is only granted to the doctor if there is a positive authorisation and no negative authorisation.**
  $decide(S_{DOCTOR}, \text{GPS}, A)$ $when$ $0:$ $(allow(S_{DOCTOR}, \text{GPS}, A)$
  $and\ not\ deny(S_{DOCTOR}, \text{GPS}, A))$

Rules comparison

The nurse policy is a simple policy containing the rules from R_1 to R_4 with the decision rule R_5. Next, we specify the history rules using future time operators. In the nurse policy, we have the rules R_1 and R_2 which are assumed to be history rules; we specify these using past time operators and future time operators to make a comparison between the rules.
R_1 :**A nurse has no access to the system before she has registered on the system.**
Using past operators:
$allow(S_{NURSE}, \text{GPS}, access)\ \ when\ \ \boldsymbol{true\ \ chop^p\ \ always^p}(\ registered(S_{NURSE}))$
Using future operators:
$allow(S_{NURSE}, \text{GPS}, access)\ \ when\ \ 0:\ \ \boldsymbol{true\ chop\ always}(\ registered(S_{NURSE}))$
R_2 : **A nurse can view the patient medical record when a patient is admitted.**
Using past operators:
$R_2 : allow(S_{NURSE}, \text{GPS}, view(O_{medical\_record}))\ \ when$
$(\ \boldsymbol{true\ \ chop^p\ \ always^p}(admit(S_{PATIENT})))$ **and**
$medical\_record(O_{medical\_record}, S_{PATIENT})$
Using future operators:
$allow(S_{NURSE}, \text{GPS}, view(O_{medical\_record}))\ \ when\ \ 0:$
$(\ \boldsymbol{true\ \ chop\ \ always}(admit(S_{PATIENT})))$ **and**
$medical\_record(O_{medical\_record}, S_{PATIENT})$

The patient policy is a simple policy containing the rules from R_6 to R_{11} with the decision rule R_{12}. Next, we specify the history rules using future time operators. In the patient policy, we have the rules R_6, R_9 and R_{10} which are assumed to be history rules; we specify these using past operators and future time operators to make a comparison between the rules.
R_6 : **A patient should be registered to the system before (s)he can access the system.**
Using past operators:
$allow(S_{PATIENT}, \text{GPS}, access)\ \ when\ \ \boldsymbol{true\ \ chop^p\ \ always^p}(registered(S_{PATIENT}))$
Using future operators:
$allow(S_{PATIENT}, \text{GPS}, access)\ \ when\ \ 0:\ \ \boldsymbol{true\ chop\ always}(registered(S_{PATIENT}))$
R_9 : **The patient should sign the consent form and the legal agreement**

once before any treatment.
Using past operators:
$allow(S_{PATIENT}$,GPS,$treat(S_{PATIENT}))$ $when$ $\boldsymbol{true}$ $\boldsymbol{chop^p}$ $\boldsymbol{always^p}$
($sign(S_{PATIENT}, O_{LA})$ **and** $sign(S_{PATIENT}, O_{CF})$
Using future operators:
$allow(S_{PATIENT}$,GPS,$treat(S_{PATIENT}))$ $when$ $0:$ $\boldsymbol{true}$ $\boldsymbol{chop}$ $\boldsymbol{always}$
($sign(S_{PATIENT}, O_{LA})$ **and** $sign(S_{PATIENT}, O_{CF}))$

R_{10} : **The patient can book an appointment with the chosen doctor if (s)he has not an appointment.**
Using past operators:
$allow(S_{PATIENT}$,GPS,$book(O_{APPOINTMENT}))$ $when$
$\boldsymbol{true}$ $\boldsymbol{chop^p}$ $\boldsymbol{always^p}$ **not** $booking(S_{PATIENT}, O_{APPOINTMENT})$ $\boldsymbol{chop}$ $\boldsymbol{skip^p}$
Using future operators:
$allow(S_{PATIENT}$,GPS,$book(O_{APPOINTMENT}))$ $when$ $0:$
$\boldsymbol{true}$ $\boldsymbol{chop}$ $\boldsymbol{always}$ **not** $booking(S_{PATIENT}, O_{APPOINTMENT})$ $\boldsymbol{chop}$ $\boldsymbol{skip}$

The doctor policy is a simple policy containing the rules from R_{13} to R_{17} with the decision rule R_{18}. Next, we specify the history rules using future time operators. In the Doctor policy, we have the rules R_{13} and R_{16} which are assumed to be history rules; we specify these using past operators and future time operators to make a comparison between the rules.

R_{13} : **A doctor has no access to the system before (s)he is registered on the system.**
Using past operators:
$allow(S_{DOCTOR}, \text{GPS}, access)\ when\ \boldsymbol{true\ chop^{p}\ always^{p}}\ (registered(S_{DOCTOR}))$
Using future operators:
$allow(S_{DOCTOR}, \text{GPS}, access)\ when\ 0:\ \boldsymbol{true\ chop\ always}\ (registered(S_{DOCTOR}))$

R_{16} :**A doctor can treat a patient who is not his/her patient if the patient agrees.**
Using past operators:
$allow(S_{DOCTOR}, \text{GPS}, treat(S_{PATIENT}))\ when\ (\ \boldsymbol{true\ chop^{p}\ always^{p}}$
$agree(S_{PATIENT})))\textbf{and}(S_{DOCTOR_1} \neq S_{DOCTOR})$ **and**
$\boldsymbol{always^{p}}(doctor(S_{DOCTOR_1}, S_{PATIENT}))$ **and**
$\boldsymbol{always^{p}}(\textbf{not}\ doctor(S_{DOCTOR}, S_{PATIENT}))$
Using future operators:
$allow(S_{DOCTOR}, \text{GPS}, treat(S_{PATIENT}))\ when\ 0:\ (\ \boldsymbol{true\ chop\ always}$
$agree(S_{PATIENT})))\textbf{and}(S_{DOCTOR_1} \neq S_{DOCTOR})$ **and**
$\boldsymbol{always}(doctor(S_{DOCTOR_1}, S_{PATIENT}))$ **and**
$\boldsymbol{always}(\textbf{not}\ doctor(S_{DOCTOR}, S_{PATIENT}))$

Conclusion of rules comparison

As has been shown above, it can be concluded the past time operators of ITL^{p} are valid to reason about history based access control policy such as GPS. Also, it can be concluded that it is not clear that the specification when using past time operators is easier to express (simplicity) than the specification when using future time operators. Additionally, the formula with past is not shorter (succinctness) than the formula with future time operators, so it can be said succinctness and simplicity are not achieved in the specification level.

5.4.2 Semantics of GPS policies using future time

The following is a mapping from the SANTA policy language used to express the GPS policies into their formal ITL semantics. The proofs in Section (5.10) are using the semantic representation of policies.

Semantics of P_{NURSE}

$[\![P_{NURSE}]\!] \equiv [\![\ R_1 ... R_5\ ]\!] \equiv [\![\ R_1^{'}\ ]\!] \wedge [\![\ R_2^{'}\ ]\!] \wedge [\![\ R_3^{'}\ ]\!] \wedge [\![\ R_4^{'}\ ]\!]$ where:

$$[\![\ R_1\ \wedge\ R_5\ ]\!] \equiv [\![\ R_1^{'}\ ]\!]$$

$$\bigwedge_{\substack{s \in Subjects \\ o \in Objects \\ a \in Actions}} \left(\begin{array}{l} (f_{NURSE}(s,o,a) \leftrightarrow Aut^{+}(s,o,a)) \wedge \\ (g_{NURSE}(s,o,a) \leftrightarrow Aut^{-}(s,o,a)) \wedge \\ (h_{NURSE}(s,o,a) \leftrightarrow Aut(s,o,a)) \end{array} \right)$$

where $f_{NURSE}(s,o,a)$ is defined as:

$f_{NURSE}(s,o,a)$	(s,o,a)	R
$[\![$ ***true chop always***($registered(S_{NURSE})$) $]\!]$	$(S_{NURSE}$,GPS$, access)$	1

and $g_{NURSE}(s,o,a)$ is defined as:

$g_{NURSE}(s,o,a)$	(s,o,a)	R
$[\![$ false $]\!]$	$(S_{NURSE}$,GPS$, access)$	1

and $h_{NURSE}(s,o,a)$ is defined as:

$h_{NURSE}(s,o,a)$	(s,o,a)	R
$[\![0 :\ allow(S_{NURSE}$,GPS$, access)$ **and not** $deny(S_{NURSE}$,GPS$, access)]\!]$	$(S_{NURSE}$,GPS$, access)$	5

and $[\![R_2 \ \wedge \ R_5]\!] \equiv [\![R_2']\!]$

where $f_{NURSE}(s, o, a)$ is defined as:

$f_{NURSE}(s, o, a)$	(s, o, a)	R
$[\![$ ($\boldsymbol{true}$ $\boldsymbol{chop}$ $\boldsymbol{always}(admitt(S_{PATIENT})))$ **and** $medical\_record(O_{medical\_record}, S_{PATIENT})]\!]$	$(S_{NURSE}$,GPS$, view(O_{medical_record}))$	2

and $g_{NURSE}(s, o, a)$ is defined as:

$g_{NURSE}(s, o, a)$	(s, o, a)	R
$[\![$ false $]\!]$	$(S_{NURSE}$,GPS$, view(O_{medical_record}))$	2

and $h_{NURSE}(s, o, a)$ is defined as:

$h_{NURSE}(s, o, a)$	(s, o, a)	R
$[\![0 : \ allow(S_{NURSE}$,GPS$, view(O_{medical_record}))$ **and not** $deny(S_{NURSE}$,GPS$, view(O_{medical_record}))]\!]$	$(S_{NURSE}$,GPS$, view(O_{medical_record}))$	5

and $[\![R_3 \wedge R_5]\!] \equiv [\![R_3']\!]$

where $f_{NURSE}(s,o,a)$ is defined as:

$f_{NURSE}(s,o,a)$	(s,o,a)	R
$[\![\text{false}]\!]$	$(S_{NURSE},\text{GPS},alter(O))$	3

and $g_{NURSE}(s,o,a)$ is defined as:

$g_{NURSE}(s,o,a)$	(s,o,a)	R
$[\![O = O_{medical\_records})$ **or** $O = O_{doctor\_notes}]\!]$	$(S_{NURSE},\text{GPS},alter(O))$	3

and $h_{NURSE}(s,o,a)$ is defined as:

$h_{NURSE}(s,o,a)$	(s,o,a)	R
$[\![0:$ $deny(S_{NURSE},\text{GPS},alter(O))$ **and not** $allow(S_{NURSE},\text{GPS},alter(O))]\!]$	$(S_{NURSE},\text{GPS},alter(O))$	5

and $[\![\ R_4 \ \wedge \ R_5\ ]\!] \equiv [\![\ R_4^{'}\ ]\!]$
where $f_{NURSE}(s,o,a)$ is defined as:

$f_{NURSE}(s,o,a)$	(s,o,a)	R
$[\![$ false $]\!]$	$(S_{NURSE}$,GPS$, sign(O))$	4

and $g_{NURSE}(s,o,a)$ is defined as:

$g_{NURSE}(s,o,a)$	(s,o,a)	R
$[\![$ true $]\!]$	$(S_{NURSE}$,GPS$, sign(O_{LA}))$	4

and $h_{NURSE}(s,o,a)$ is defined as:

$h_{NURSE}(s,o,a)$	(s,o,a)	R
$[\![0:$ $deny(S_{NURSE}$,GPS$, sign(O))$ **and not** $allow(S_{NURSE}$,GPS$, sign(O))]\!]$	$(S_{NURSE}$,GPS$, sign(O))$	5

Semantics of $P_{PATIENT}$

$[\![P_{PATIENT}]\!] \equiv [\![\ R_6...R_{12}]\!] \equiv [\![\ R_6^{'}\ ]\!] \wedge [\![\ R_7^{'}\ ]\!] \wedge [\![\ R_8^{'}\ ]\!] \wedge [\![\ R_9^{'}\ ]\!] \wedge [\![\ R_{10}^{'}\ ]\!] \wedge [\![\ R_{11}^{'}\ ]\!]$

where $[\![\ R_6^{'}\ ]\!], [\![\ R_7^{'}\ ]\!], [\![\ R_8^{'}\ ]\!], [\![\ R_9^{'}\ ]\!], [\![\ R_{10}^{'}\ ]\!], [\![\ R_{11}^{'}\ ]\!]$ are defined in Appendix C Part 1.

Semantics of P_{DOCTOR}

$[\![P_{DOCTOR}]\!] \equiv [\![\ R_{13}...R_{18}\ ]\!] \equiv [\![\ R_{13}^{'}\ ]\!] \wedge [\![\ R_{14}^{'}\ ]\!] \wedge [\![\ R_{15}^{'}\ ]\!] \wedge [\![\ R_{16}^{'}\ ]\!] \wedge [\![\ R_{17}^{'}\ ]\!]$
where $[\![\ R_{13}^{'}\ ]\!], [\![\ R_{14}^{'}\ ]\!], [\![\ R_{15}^{'}\ ]\!], [\![\ R_{16}^{'}\ ]\!], [\![\ R_{17}^{'}\ ]\!]$ are defined in Appendix C part 2.

5.4.3 Safety property:

A safety property informally express that nothing bad will happen during execution of a program or formally:

$$\psi \quad \widehat{=} \quad \boxed{i}\, f$$

We will give a specific safety property for every member of the GPS and prove that this property holds for each member.

Nurse Safety Property:

The safety property can be defined as:
It is never the case that the nurse can use GPS without being registered on the system.
Let $\psi_1 \quad \widehat{=} \boxed{i}(\mathsf{fin}\,(Aut(S_{NURSE},\text{GPS}, a)) \supset \ \mathsf{true}\ ;\ \Box(registered(S_{NURSE})))$
denote the nurse safety property and let $a\ \in\ \{access\ ,\ view\ ,\ alter\}$

The proof that GPS policy satisfies ψ_1 can be done by proving that each of the nurse rules in this policy which affect the safety property will not invalidate this property ψ_1, i.e. $[\![R_1...R_5]\!] \supset \psi_1$.
We have to prove that $[\![R_1\ \wedge\ R_5]\!] \supset \psi_1, [\![R_2\ \wedge\ R_5]\!] \supset \psi_1, [\![R_3 \wedge R_5]\!] \supset \psi_1$.
Here$[\![R_i]\!]$ denotes only the rules that affect the safety property.$[\![R_4...R_5]\!] \supset \psi_1$ because they cannot affect (invalidate) ψ_1.

We can prove that the nurse policy satisfies the safety property by proving that $R_1^{'}$ and $R_2^{'}$ and $R_3^{'}$ which affect the safety property satisfy ψ_1.

Rule $R_1^{'}$ states:

$R_1^{'}$: A nurse has no access to the system before she has registered on the system.

$$allow(S_{NURSE},\text{GPS}, access)\ when\ 0\ \boldsymbol{true\ chop\ always}(\ registered(S_{NURSE}))$$

1. From the semantics of P_{NURSE} we have:
   $[\![R_1 \wedge R_5\ ]\!] \equiv [\![\ R_1^{'}\ ]\!] \equiv$
   $$\bigwedge_{\substack{s \in Subjects \\ o \in Objects \\ a \in Actions}} \begin{pmatrix} (f_{NURSE}(s,o,a) \leftrightarrow Aut^{+}(s,o,a))\wedge \\ (g_{NURSE}(s,o,a) \leftrightarrow Aut^{-}(s,o,a))\wedge \\ (h_{NURSE}(s,o,a) \leftrightarrow Aut(s,o,a)) \end{pmatrix}$$
   $(\mathsf{true}\ ;\ \Box(\ registered(S_{NURSE}))) \ \leftrightarrow\ Aut^{+}(S_{NURSE},\text{GPS}, access)$
   $\wedge\ \mathsf{false}\ \leftrightarrow\ Aut^{-}(S_{NURSE},\text{GPS}, access)$
   $\wedge\ 0:\ (Aut^{+}(S_{NURSE},\text{GPS}, access)\ \wedge\ \neg\ Aut^{-}(S_{NURSE},\text{GPS}, access))\ \leftrightarrow$
   $Aut(S_{NURSE},\text{GPS}, access)$

2. from the definition of($\leftrightarrow$), we have:
   $f \leftrightarrow w\ \ \widehat{=}\ \ \boxed{i}(f \equiv \mathsf{fin}\,(w))$

3. which can be written as:
   $\boxed{i}((\mathsf{true}\ ;\ \Box(\ registered(S_{NURSE})))\ \equiv\ \mathsf{fin}\ \ (Aut^{+}(S_{NURSE},\text{GPS}, access)))$
   $\wedge\ \boxed{i}(\mathsf{false}\ \equiv\ \mathsf{fin}\ \ (Aut^{-}(S_{NURSE},\text{GPS}, access)))$
   $\wedge\ \boxed{i}((\mathsf{true};(len(0)\wedge Aut^{+}(S_{NURSE},\text{GPS}, access) \wedge \neg\ Aut^{-}(S_{NURSE},\text{GPS}, access)))\ \equiv$
   $\mathsf{fin}\ \ Aut(S_{NURSE},\text{GPS}, access))$

4. By definition, we know that:
   $\mathsf{true}\,;(len(0) \wedge (Aut^{+}(S_{NURSE},\text{GPS}, access)\ \wedge\ \neg\ Aut^{-}(S_{NURSE},\text{GPS}, access)))$
   $\equiv\ \mathsf{fin}\,(Aut^{+}(S_{NURSE},\text{GPS}, access)\ \wedge\ \neg\ Aut^{-}(S_{NURSE},\text{GPS}, access))$

5. so, we can write the first one as:
   $\boxed{i}(\mathsf{true}\ ;\ \Box(\ registered(S_{NURSE})))\ \equiv\ \mathsf{fin}\ \ (Aut^{+}(S_{NURSE},\text{GPS}, access))$
   $\wedge\ \boxed{i}(\mathsf{false}\ \equiv\ \mathsf{fin}\ \ (Aut^{-}(S_{NURSE},\text{GPS}, access)))$
   $\wedge\ \boxed{i}(\mathsf{fin}\,(Aut^{+}(S_{NURSE},\text{GPS}, access)\ \wedge\ \neg\ Aut^{-}(S_{NURSE},\text{GPS}, access))\ \equiv$
   $\mathsf{fin}\ \ Aut(S_{NURSE},\text{GPS}, access))$

6. so, we have:
   $\boxed{\scriptstyle i}(\mathsf{fin}\,(Aut^{+}(S_{NURSE},\text{GPS}, access) \wedge \neg Aut^{-}(S_{NURSE},\text{GPS}, access)) \equiv \mathsf{fin}\;\; Aut(S_{NURSE},\text{GPS}, access))$

7. we know that:
   $\mathsf{fin}\,(Aut^{+}(S_{NURSE},\text{GPS}, access) \wedge \neg Aut^{-}(S_{NURSE},\text{GPS}, access)) \equiv \mathsf{fin}\,(Aut^{+}(S_{NURSE},\text{GPS}, access)) \wedge \mathsf{fin}\,(\neg Aut^{-}(S_{NURSE},\text{GPS}, access))$

8. also, we know that:
   $\mathsf{fin}\,(\neg Aut^{-}(S_{NURSE},\text{GPS}, access)) \equiv \neg \mathsf{fin}\,(Aut^{-}(S_{NURSE},\text{GPS}, access))$

9. so, we have:
   $\boxed{\scriptstyle i}(\mathsf{fin}\,(Aut^{+}(S_{NURSE},\text{GPS}, access)) \wedge \neg \mathsf{fin}\;(Aut^{-}(S_{NURSE},\text{GPS}, access)) \equiv \mathsf{fin}\;(Aut(S_{NURSE},\text{GPS}, access)))$

10. we know that $\mathsf{false} \equiv \mathsf{fin}\,(Aut^{-}(S_{NURSE},\text{GPS}, access))$

11. so, we can write:
    $\boxed{\scriptstyle i}(\mathsf{fin}\,(Aut^{+}(S_{NURSE},\text{GPS}, access)) \equiv \mathsf{fin}\;(Aut(S_{NURSE},\text{GPS}, access)))$

12. and from above we have:
    $\boxed{\scriptstyle i}(\mathsf{true}\;;\;\Box(\;registered(S_{NURSE})) \equiv \mathsf{fin}\;(Aut(S_{NURSE},\text{GPS}, access)))$

13. using ITL reasoning, we have:
    $\boxed{\scriptstyle i}(\mathsf{fin}\;(Aut(S_{NURSE},\text{GPS}, access)) \supset \mathsf{true}\;;\;\Box(\;registered(S_{NURSE})))$

14. and this is ψ_1.

Rule $R_2^{'}$ states:

$R_2^{'}$: A nurse can view the patient's medical records when a patient is admitted.

$allow(S_{NURSE},\text{GPS}, view(O_{medical\_records}))\ \ when\ \ 0:$

$\boldsymbol{true\ \ chop\ \ always}((admitt(S_{PATIENT}))$ **and** $medical\_record(S_{PATIENT}))$

1. From the semantics of P_{NURSE} we have:

   $[\![R_2 \wedge R_5\ ]\!] \equiv [\![\ R_2^{'}\ ]\!] \equiv$
   $$\bigwedge_{\substack{s \in Subjects\\ o \in Objects\\ a \in Actions}} \left(\begin{array}{l} (f_{NURSE}(s,o,a) \leftrightarrow Aut^{+}(s,o,a))\wedge \\ (g_{NURSE}(s,o,a) \leftrightarrow Aut^{-}(s,o,a))\wedge \\ (h_{NURSE}(s,o,a) \leftrightarrow Aut(s,o,a)) \end{array} \right)$$

   $\mathsf{true}\ ;\ \Box((admitt(S_{PATIENT})) \wedge medical\_record(O_{medical\_record}, S_{PATIENT}))\ \leftrightarrow$
   $Aut^{+}(S_{NURSE},\text{GPS}, view(O_{medical\_record}))$
   $\wedge\ \mathsf{false}\ \leftrightarrow\ Aut^{-}(S_{NURSE},\text{GPS}, view(O_{medical\_record}))$
   $\wedge\ 0:\ Aut^{+}(S_{NURSE},\text{GPS}, view(O_{medical\_record}))\ \wedge$
   $\neg\ Aut^{-}(S_{NURSE},\text{GPS}, view(O_{medical\_record}))$
   $\leftrightarrow\ Aut(S_{NURSE},\text{GPS}, view(O_{medical\_record}))$

2. from the definition of$(\leftrightarrow)$ we have:

   $$f\ \leftrightarrow\ w\ \ \widehat{=}\ \ \boxed{i}(f\ \equiv\ \mathsf{fin}\,(w))$$

3. we have:

   $\boxed{i}(\mathsf{true}\ ;\ \Box((admitt(S_{PATIENT})) \wedge medical\_record(O_{medical\_record}, S_{PATIENT}))\ \equiv$
   $\mathsf{fin}\,(Aut^{+}(S_{NURSE},\text{GPS}, view(O_{medical\_record}))))$
   $\wedge\ \boxed{i}(\mathsf{false}\ \equiv\ \mathsf{fin}\,(Aut^{-}(S_{NURSE},\text{GPS}, view(O_{medical\_record}))))$
   $\wedge\ \boxed{i}(\mathsf{true}\ ;\ (len(0) \wedge\ Aut^{+}(S_{NURSE},\text{GPS}, view(O_{medical\_record}))$
   $\wedge\ \neg\ Aut^{-}(S_{NURSE},\text{GPS}, view(O_{medical\_record})))$
   $\equiv\ \mathsf{fin}\ \ (Aut(S_{NURSE},\text{GPS}, view(O_{medical\_record}))))$

4. By definition, we have:

   $\mathsf{true}\ ;\ (len(0) \wedge\ Aut^{+}(S_{NURSE},\text{GPS}, view(O_{medical\_record}))\ \wedge$
   $\neg\ Aut^{-}(S_{NURSE},\text{GPS}, view(O_{medical\_record})))\ \equiv$

$\mathsf{fin}\ (Aut^{+}(S_{NURSE},\text{GPS}, view(O_{medical\_record}))\ \wedge$
$\neg\ Aut^{-}(S_{NURSE},\text{GPS}, view(O_{medical\_record})))$

5. so, we can substitute it, and we have:
$\boxed{\text{i}}(\mathsf{true}\,;\Box((admitt(S_{PATIENT}))\wedge medical\_record(O_{medical\_record}, S_{PATIENT}))\ \equiv$
$\mathsf{fin}\,(Aut^{+}(S_{NURSE},\text{GPS}, view(O_{medical\_record}))))$
$\wedge\ \boxed{\text{i}}(\mathsf{false}\ \equiv\ \mathsf{fin}\,(Aut^{-}(S_{NURSE},\text{GPS}, view(O_{medical\_record}))))$
$\wedge\ \boxed{\text{i}}(\mathsf{fin}\ (Aut^{+}(S_{NURSE},\text{GPS}, view(O_{medical\_record}))\ \wedge$
$\neg\ Aut^{-}(S_{NURSE},\text{GPS}, view(O_{medical\_record})))$
$\equiv\ \mathsf{fin}\ (Aut(S_{NURSE},\text{GPS}, view(O_{medical\_record}))))$

6. so, we have:
$\boxed{\text{i}}(\mathsf{fin}\ (Aut^{+}(S_{NURSE},\text{GPS}, view(O_{medical\_record}))\ \wedge$
$\neg\ Aut^{-}(S_{NURSE},\text{GPS}, view(O_{medical\_record})))$
$\equiv\ \mathsf{fin}\ (Aut(S_{NURSE},\text{GPS}, view(O_{medical\_record}))))$

7. we know that:
$\mathsf{fin}\ (Aut^{+}(S_{NURSE},\text{GPS}, view(O_{medical\_record}))\ \wedge$
$\neg\ Aut^{-}(S_{NURSE},\text{GPS}, view(O_{medical\_record})))\ \equiv$
$\mathsf{fin}\ (Aut^{+}(S_{NURSE},\text{GPS}, view(O_{medical\_record})))\ \wedge$
$\mathsf{fin}\ (\neg\ Aut^{-}(S_{NURSE},\text{GPS}, view(O_{medical\_record})))$

8. also, we know that:
$\mathsf{fin}\ (\neg\ Aut^{-}(S_{NURSE},\text{GPS}, view(O_{medical\_record})))\ \equiv$
$\neg\ \mathsf{fin}\ (Aut^{-}(S_{NURSE},\text{GPS}, view(O_{medical\_record})))$

9. so, we have:
$\boxed{\text{i}}(\mathsf{fin}\ (Aut^{+}(S_{NURSE},\text{GPS}, view(O_{medical\_record})))\ \wedge$
$\neg\ \mathsf{fin}\ (Aut^{-}(S_{NURSE},\text{GPS}, view(O_{medical\_record})))$
$\equiv\ \mathsf{fin}\ (Aut(S_{NURSE},\text{GPS}, view(O_{medical\_record}))))$

10. but we know that $\mathsf{false}\ \equiv\ Aut^{-}(S_{NURSE},\text{GPS}, view(O_{medical\_record}))$

11. substitute it, we can write it as:
$\boxed{\text{i}}(\mathsf{fin}\ (Aut^{+}(S_{NURSE},\text{GPS}, view(O_{medical\_record})))$

$\equiv \mathsf{fin}\ (Aut(S_{NURSE},\text{GPS}, view(O_{medical\_record}))))$

12. and we have:
$\boxed{\text{i}}(\mathsf{fin}\,(\mathsf{true}\ ;\ \Box((admitt(S_{PATIENT}))\ \wedge$
$medical\_record(O_{medical\_record}, S_{PATIENT})))\ \equiv$
$\mathsf{fin}\,(\ Aut(S_{NURSE},\text{GPS}, view(O_{medical\_record}))))$

13. this is equal to:
$\boxed{\text{i}}(\mathsf{fin}\,(Aut(S_{NURSE},\text{GPS}, view(O_{medical\_record})))\ \equiv$
$\mathsf{true}\ ;\ \Box((admitt(S_{PATIENT}))\ \wedge\ medical\_record(O_{medical\_record}, S_{PATIENT})))$

14. we assume that view can only be after registration, so we can say that:
$\boxed{\text{i}}(\mathsf{fin}\,(Aut(S_{NURSE},\text{GPS}, view(O_{medical\_record})))\ \equiv\ \mathsf{true}\,;\Box(\ registered(S_{NURSE})))$

15. and from it we have:
$\boxed{\text{i}}(\mathsf{fin}\,(Aut(S_{NURSE},\text{GPS}, view(O_{medical\_record})))\ \supset\ \mathsf{true}\,;\Box(\ registered(S_{NURSE})))$

16. we can say that R_2' satisfies ψ_1

Rule R_3' states:

R_3' : A nurse cannot alter the medical records and doctor's notes.

$deny(S_{NURSE},\text{GPS}, alter(O)) when\ \ 0 : \ (O = O_{medical\_records}$ **or** $O = O_{doctor\_notes})$

1. From the semantics of P_{NURSE} we have:
   $[\![R_3 \wedge R_5]\!] \equiv [\![R_3']\!] \equiv$
   $[\![P_{NURSE}]\!] \equiv [\![R_3 \ \wedge \ R_5]\!] \equiv [\![R_3']\!] \equiv$
   $$\bigwedge_{\substack{s \in Subjects \\ o \in Objects \\ a \in Actions}} \begin{pmatrix} (f_{NURSE}(s,o,a) \leftrightarrow Aut^{+}(s,o,a))\wedge \\ (g_{NURSE}(s,o,a) \leftrightarrow Aut^{-}(s,o,a))\wedge \\ (h_{NURSE}(s,o,a) \leftrightarrow Aut(s,o,a)) \end{pmatrix}$$
   $\mathsf{false} \ \leftrightarrow Aut^{+}(S_{NURSE},\text{GPS}, alter(O))$
   $\wedge \ O \ = \ (O_{medical\_records} \ \vee \ O_{doctor\_notes}) \leftrightarrow Aut^{-}(S_{NURSE},\text{GPS}, alter(O))$
   $\wedge \ 0 : \ Aut^{-}(S_{NURSE},\text{GPS}, alter(O)) \ \wedge \ \neg \ Aut^{+}(S_{NURSE},\text{GPS}, alter(O))$
   $\leftrightarrow \ Aut(S_{NURSE},\text{GPS}, alter(O))$

2. from the definition of$(\leftrightarrow)$ we have:
   $f \ \leftrightarrow \ w \ \ \widehat{=} \ \ \boxed{i}(f \ \equiv \ \mathsf{fin}\,(w))$

3. we have:
   $\boxed{i}(\mathsf{false} \ \equiv \ \mathsf{fin}\,(Aut^{+}(S_{NURSE},\text{GPS}, alter(O))))$
   $\wedge \boxed{i}(\ O \ = \ (O_{medical\_records} \vee O_{doctor\_notes}) \equiv \ \mathsf{fin}\,(Aut^{-}(S_{NURSE},\text{GPS}, alter(O))))$
   $\wedge \ \boxed{i}(0 : \ Aut^{-}(S_{NURSE},\text{GPS}, alter(O)) \ \wedge \ \neg \ Aut^{+}(S_{NURSE},\text{GPS}, alter(O))$
   $\equiv \ \mathsf{fin}\,(Aut(S_{NURSE},\text{GPS}, alter(O))))$

4. So, we have:
   $\boxed{i}(\mathsf{false} \ \equiv \ \mathsf{fin}\,(Aut^{+}(S_{NURSE},\text{GPS}, alter(O))))$
   $\wedge \boxed{i}(\ O \ = \ (O_{medical\_records} \vee O_{doctor\_notes}) \equiv \ \mathsf{fin}\,(Aut^{-}(S_{NURSE},\text{GPS}, alter(O))))$
   $\wedge \boxed{i}(\mathsf{true};(len(0)\wedge Aut^{-}(S_{NURSE},\text{GPS}, alter(O)) \wedge \neg \ Aut^{+}(S_{NURSE},\text{GPS}, alter(O)))$
   $\equiv \ \mathsf{fin}\,(Aut(S_{NURSE},\text{GPS}, alter(O))))$

5. By definition, we have:
   $\mathsf{true};(len(0)\wedge Aut^{-}(S_{NURSE},\text{GPS}, alter(O)) \wedge \neg \ Aut^{+}(S_{NURSE},\text{GPS}, alter(O))) \ \equiv$

$\mathsf{fin}\ \ (Aut^{-}(S_{NURSE},\text{GPS}, alter(O))\ \wedge\ \neg\ Aut^{+}(S_{NURSE},\text{GPS}, alter(O)))$

6. So, we have:
   $\boxed{\text{i}}(\mathsf{false}\ \equiv\ \mathsf{fin}\,(Aut^{+}(S_{NURSE},\text{GPS}, alter(O))))$
   $\wedge\ \boxed{\text{i}}(\ O\ =\ (O_{medical\_records} \vee O_{doctor\_notes}) \equiv\ \mathsf{fin}\,(Aut^{-}(S_{NURSE},\text{GPS}, alter(O))))$
   $\wedge\ \boxed{\text{i}}(\mathsf{fin}\ \ (Aut^{-}(S_{NURSE},\text{GPS}, alter(O))\ \wedge\ \neg\ Aut^{+}(S_{NURSE},\text{GPS}, alter(O)))$
   $\equiv\ \mathsf{fin}\,(Aut(S_{NURSE},\text{GPS}, alter(O))))$

7. So, we have:
   $\boxed{\text{i}}(\mathsf{fin}\ \ (Aut^{-}(S_{NURSE},\text{GPS}, alter(O))\ \wedge\ \neg\ Aut^{+}(S_{NURSE},\text{GPS}, alter(O)))$
   $\equiv\ \mathsf{fin}\,(Aut(S_{NURSE},\text{GPS}, alter(O))))$

8. We know that:
   $\mathsf{fin}\ \ ((Aut^{-}(S_{NURSE},\text{GPS}, alter(O))\ \wedge\ \neg\ Aut^{+}(S_{NURSE},\text{GPS}, alter(O))))\ \equiv$
   $\mathsf{fin}\ \ (Aut^{-}(S_{NURSE},\text{GPS}, alter(O))) \wedge\ \mathsf{fin}\ \ (\neg\ Aut^{+}(S_{NURSE},\text{GPS}, alter(O)))$

9. Also, we know that:
   $\mathsf{fin}\ (\neg\ Aut^{+}(S_{NURSE},\text{GPS}, alter(O)))\ \equiv\ \neg\ \mathsf{fin}\ \ (Aut^{+}(S_{NURSE},\text{GPS}, alter(O)))$

10. So, we have:
    $\boxed{\text{i}}(\mathsf{fin}\ \ (Aut^{-}(S_{NURSE},\text{GPS}, alter(O)))\wedge\ \neg\ \mathsf{fin}\ \ (Aut^{+}(S_{NURSE},\text{GPS}, alter(O)))$
    $\equiv\ \mathsf{fin}\,(Aut(S_{NURSE},\text{GPS}, alter(O))))$

11. We know that $\mathsf{false}\ \equiv\ Aut^{+}(S_{NURSE},\text{GPS}, alter(O))$ so we can write:
    $\boxed{\text{i}}(\mathsf{fin}\ \ (Aut^{-}(S_{NURSE},\text{GPS}, alter(O)))\ \equiv\ \mathsf{fin}\,(Aut(S_{NURSE},\text{GPS}, alter(O))))$

12. we know that:
    $(O\ =\ (O_{medical\_records}\ \vee\ O_{doctor\_notes})) \equiv\ \mathsf{fin}\,(Aut^{-}(S_{NURSE},\text{GPS}, alter(O)))$

13. so, we have:
$\boxed{i}((O = (O_{medical\_records} \vee O_{doctor\_notes})) \equiv \mathsf{fin}\,(Aut(S_{NURSE},\mathrm{GPS}, alter(O))))$

14. this yields to:
$\boxed{i}(\mathsf{fin}\,(Aut(S_{NURSE},\mathrm{GPS}, alter(O))) \equiv (O = (O_{medical\_records} \vee O_{doctor\_notes})))$

15. which can be written as:
$\boxed{i}(\mathsf{fin}\,(Aut(S_{NURSE},\mathrm{GPS}, alter(O))) \supset (O = (O_{medical\_records} \vee O_{doctor\_notes})))$

16. by assume that alter can be after registration, so we can say that:
$\boxed{i}(\mathsf{fin}\,(Aut(S_{NURSE},\mathrm{GPS}, alter(O))) \supset (O = (O_{medical\_records} \vee O_{doctor\_notes})))$

17. from it we can say that $R_3^{'}$ satisfies ψ_1.

Patient Safety Property:

It is never the case that the patient can alter the medical records in the system.
Let $\psi_2 \ \widehat{=} \ \boxed{\scriptstyle\rm i}(\mathsf{fin}\,(\neg Aut(S_{PATIENT},\mathrm{GPS}, alter(O_{medical\_records}))))$
denote the safety property.
A compositional proof that patient policy satisfies ψ_2 can be done using proof rules by proving that each of the rules in this policy satisfies the safety property ψ_2.
$[\![R_8^{'}...R_{11}^{'}]\!] \supset \psi_2$.
We have to prove that $[\![R_8^{'} \wedge R_{11}^{'}]\!] \supset \psi_2$. Here $[\![R_i^{'}]\!]$ denotes only rules that can affect the safety property. The rules $[\![R_9^{'}...R_{11}^{'}]\!]$ cannot invalidate ψ_2.
We can prove that the patient policy satisfies the safety property by proving that $R_8^{'}$ which affect the safety property satisfies ψ_2.

Rule $R_8^{'}$ states:

$R_8^{'}$: A patient cannot alter the medical records in the system.

$$deny(S_{PATIENT},\mathrm{GPS}, alter(O_{medical\_records}) \ \ when \ \ 0: \ \ \boldsymbol{true}$$

1. From the semantics of $P_{PATIENT}$ we have:

$$[\![\ R_8 \ \wedge \ R_{12} \ ]\!] \equiv [\![\ R_8^{'} \ ]\!] \equiv$$

$$\bigwedge_{\substack{s \in Subjects \\ o \in Objects \\ a \in Actions}} \left(\begin{array}{l} (f_{PATIENT}(s,o,a) \leftrightarrow Aut^{+}(s,o,a)) \wedge \\ (g_{PATIENT}(s,o,a) \leftrightarrow Aut^{-}(s,o,a)) \wedge \\ (h_{PATIENT}(s,o,a) \leftrightarrow Aut(s,o,a)) \end{array} \right)$$

$$\begin{array}{l} \mathsf{false} \ \leftrightarrow \ Aut^{+}(S_{PATIENT},\mathrm{GPS}, alter(O_{medical\_records})) \\ \wedge \ \mathsf{true} \ \leftrightarrow \ Aut^{-}(S_{PATIENT},\mathrm{GPS}, alter(O_{medical\_records})) \\ \wedge \ 0: \ Aut^{-}(S_{PATIENT},\mathrm{GPS}, alter(O_{medical\_records})) \\ \wedge \ \neg \ Aut^{+}(S_{PATIENT},\mathrm{GPS}, alter(O_{medical\_records})) \ \leftrightarrow \\ Aut(S_{PATIENT},\mathrm{GPS}, alter(O_{medical\_records})) \end{array}$$

2. from the definition of$(\leftrightarrow)$ we have:
$f \ \leftrightarrow \ w \ \ \widehat{=} \ \ \boxed{\scriptstyle\rm i}(f \ \equiv \ \mathsf{fin}\,(w))$

3. we have:
$\boxed{i}(\mathsf{false} \equiv \mathsf{fin}\,(Aut^{+}(S_{PATIENT}, \text{GPS}, alter(O_{medical\_records}))))$
$\wedge\ \boxed{i}(\mathsf{true} \equiv \mathsf{fin}\,(\ Aut^{-}(S_{PATIENT}, \text{GPS}, alter(O_{medical\_records}))))$
$\wedge\ \boxed{i}(0:\ Aut^{-}(S_{PATIENT}, \text{GPS}, alter(O_{medical\_records}))\ \wedge$
$\neg\ Aut^{+}(S_{PATIENT}, \text{GPS}, alter(O_{medical\_records}))$
$\equiv\ \mathsf{fin}\,(Aut(S_{PATIENT}, \text{GPS}, alter(O_{medical\_records}))))$

4. so, we have:
$\boxed{i}(\mathsf{false} \equiv \mathsf{fin}\,(Aut^{+}(S_{PATIENT}, \text{GPS}, alter(O_{medical\_records}))))$
$\wedge\ \boxed{i}(\mathsf{true} \equiv \mathsf{fin}\,(Aut^{-}(S_{PATIENT}, \text{GPS}, alter(O_{medical\_records}))))$
$\wedge\ \boxed{i}(\mathsf{true}\,;(len(0) \wedge\ Aut^{-}(S_{PATIENT}, \text{GPS}, alter(O_{medical\_records}))\ \wedge$
$\neg\ Aut^{+}(S_{PATIENT}, \text{GPS}, alter(O_{medical\_records})))$
$\equiv\ \mathsf{fin}\,(Aut(S_{PATIENT}, \text{GPS}, alter(O_{medical\_records}))))$

5. By definition, we have:
$\mathsf{true}\,;(len(0) \wedge\ Aut^{-}(S_{PATIENT}, \text{GPS}, alter(O_{medical\_records}))\ \wedge$
$\neg\ Aut^{+}(S_{PATIENT}, \text{GPS}, alter(O_{medical\_records})))\ \equiv$
$\mathsf{fin}\ \ (Aut^{-}(S_{PATIENT}, \text{GPS}, alter(O_{medical\_records}))\ \wedge$
$\neg\ Aut^{+}(S_{PATIENT}, \text{GPS}, alter(O_{medical\_records})))$

6. so, we have:
$\boxed{i}(\mathsf{false} \equiv \mathsf{fin}\,(Aut^{+}(S_{PATIENT}, \text{GPS}, alter(O_{medical\_records}))))$
$\wedge\ \boxed{i}(\mathsf{true} \equiv \mathsf{fin}\,(Aut^{-}(S_{PATIENT}, \text{GPS}, alter(O_{medical\_records}))))$
$\wedge\ \boxed{i}(\mathsf{fin}\ \ (Aut^{-}(S_{PATIENT}, \text{GPS}, alter(O_{medical\_records}))\ \wedge$
$\neg\ Aut^{+}(S_{PATIENT}, \text{GPS}, alter(O_{medical\_records})))$
$\equiv\ \mathsf{fin}\,(Aut(S_{PATIENT}, \text{GPS}, alter(O_{medical\_records}))))$

7. which can be written as:
$\boxed{i}(\mathsf{fin}\ \ (Aut^{-}(S_{PATIENT}, \text{GPS}, alter(O_{medical\_records}))\ \wedge$
$\neg\ Aut^{+}(S_{PATIENT}, \text{GPS}, alter(O_{medical\_records})))$
$\equiv\ \mathsf{fin}\,(Aut(S_{PATIENT}, \text{GPS}, alter(O_{medical\_records}))))$

8. We know that:
$\mathsf{fin}\ \ (Aut^{-}(S_{PATIENT}, \text{GPS}, alter(O_{medical\_records}))\ \wedge$

$\neg\ Aut^{+}(S_{PATIENT},\text{GPS}, alter(O_{medical\_records})))\ \equiv$
$\mathsf{fin}\ \ (Aut^{-}(S_{PATIENT},\text{GPS}, alter(O_{medical\_records})))\ \wedge$
$\mathsf{fin}\ \ (\neg\ Aut^{+}(S_{PATIENT},\text{GPS}, alter(O_{medical\_records})))$

9. Also, we know that:
$\mathsf{fin}\ \ (\neg\ Aut^{+}(S_{PATIENT},\text{GPS}, alter(O_{medical\_records})))\ \equiv$
$\neg\ \mathsf{fin}\ \ (Aut^{+}(S_{PATIENT},\text{GPS}, alter(O_{medical\_records})))$

10. So, we have:
$\boxed{i}(\mathsf{fin}\ \ (Aut^{-}(S_{PATIENT},\text{GPS}, alter(O_{medical\_records})))\ \wedge$
$\neg\ \mathsf{fin}\ \ (Aut^{+}(S_{PATIENT},\text{GPS}, alter(O_{medical\_records})))$
$\equiv\ \mathsf{fin}\,(Aut(S_{PATIENT},\text{GPS}, alter(O_{medical\_records}))))$

11. We know that:
$\mathsf{false}\ \equiv\ \mathsf{fin}\,(Aut^{+}(S_{PATIENT},\text{GPS}, alter(O_{medical\_records})))$

12. So, we have:
$\boxed{i}(\mathsf{fin}\ \ (Aut^{-}(S_{PATIENT},\text{GPS}, alter(O_{medical\_records})))$
$\equiv\ \mathsf{fin}\,(Aut(S_{PATIENT},\text{GPS}, alter(O_{medical\_records}))))$

13. we know that:
$\mathsf{true}\ \equiv\ \mathsf{fin}\,(\ Aut^{-}(S_{PATIENT},\text{GPS}, alter(O_{medical\_records})))$

14. so, we have:
$\boxed{i}(\mathsf{true}\ \equiv\ \mathsf{fin}\,(Aut(S_{PATIENT},\text{GPS}, alter(O_{medical\_records}))))$

15. which can be written as:
$\boxed{i}(\mathsf{fin}\,(\neg Aut(S_{PATIENT},\text{GPS}, alter(O_{medical\_records}))))$

16. which is ψ_2.

Doctor Safety Property:

It is never the case that anyone can modify the patient medical records without being a doctor of this patient.
Let $\psi_3 \ \widehat{=} \ \boxdot(\mathsf{fin}\,(Aut(S_{DOCTOR}, \text{GPS}, alter(O_{medical\_records}))) \ \supset$
$doctor(S_{DOCTOR}, S_{PATIENT}) \ \wedge \ medical\_records(S_{PATIENT}))$
denote the safety property.
A compositional proof that any policy satisfies ψ_3 can be done using proof rules by proving that each of the rules in this policy satisfies the safety property ψ_3.

$[\![R_{13}...R_{18}]\!] \supset \psi_3$.
We have to prove that $[\![R_{14} \ \wedge \ R_{18}]\!] \supset \psi_3$.
Here $[\![R_i]\!]$ denotes only rules that can affect the safety property. $[\![R^{'}_{13}\ , R^{'}_{15}...\ R^{'}_{18}]\!] \supset$ ψ_3 because they cannot affect ψ_3 .
We can prove that the doctor policy satisfies the safety property by proving that $R^{'}_{14}$ which affects the safety property satisfies ψ_3.
Rule $R^{'}_{14}$ states:
$R^{'}_{14}$: The doctor can alter all of the medical records of his/her patients.
$allow(S_{DOCTOR}, \text{GPS}, alter(O_{medical\_record})) \ \ when \ \ 0:$
$doctor(S_{DOCTOR}, S_{PATIENT})$ **and** $medical\_record(O_{medical\_record}, S_{PATIENT})$

1. From the semantics of P_{DOCTOR} we have:

$$[\![\ R_{14} \ \wedge \ R_{18}\ ]\!] \equiv [\![\ R^{'}_{14}\ ]\!] \equiv$$
$$\bigwedge_{\substack{s \in Subjects \\ o \in Objects \\ a \in Actions}} \left(\begin{array}{l} (f_{DOCTOR}(s,o,a) \leftrightarrow Aut^{+}(s,o,a)) \wedge \\ (g_{DOCTOR}(s,o,a) \leftrightarrow Aut^{-}(s,o,a)) \wedge \\ (h_{DOCTOR}(s,o,a) \leftrightarrow Aut(s,o,a)) \end{array} \right)$$

$doctor(S_{DOCTOR}, S_{PATIENT}) \ \wedge \ medical\_record(O_{medical\_record}, S_{PATIENT}) \ \leftrightarrow$
$Aut^{+}(S_{DOCTOR}, \text{GPS}, alter(O_{medical\_record}))$
$\wedge \ \mathsf{false} \ \leftrightarrow Aut^{-}(S_{DOCTOR}, \text{GPS}, alter(O_{medical\_record}))$
$\wedge \ 0: \ \ Aut^{+}(S_{DOCTOR}, \text{GPS}, alter(O_{medical\_record}))$
$\wedge \ \neg \ Aut^{-}(S_{DOCTOR}, \text{GPS}, alter(O_{medical\_record})) \ \leftrightarrow$
$Aut(S_{DOCTOR}, \text{GPS}, alter(O_{medical\_record}))$

2. from the definition of($\leftrightarrow$) we have:
$f \ \leftrightarrow \ w \ \ \widehat{=} \ \ \boxed{\scriptstyle\text{i}}(f \ \equiv \ \mathsf{fin}\,(w))$

3. we have:
$\boxed{\scriptstyle\text{i}}(doctor(S_{DOCTOR}, S_{PATIENT}) \wedge medical\_record(O_{medical\_record}, S_{PATIENT}) \ \equiv$

$\mathsf{fin}\,(Aut^{+}(S_{DOCTOR}, \text{GPS}, alter(O_{medical\_record}))))$
$\wedge \ \boxed{\scriptstyle\text{i}}(\mathsf{false} \ \equiv \ \mathsf{fin}\,(Aut^{-}(S_{DOCTOR}, \text{GPS}, alter(O_{medical\_record}))))$
$\wedge \ \boxed{\scriptstyle\text{i}}(0 : \ Aut^{+}(S_{DOCTOR}, \text{GPS}, alter(O_{medical\_records})) \ \wedge$
$\neg \ Aut^{-}(S_{DOCTOR}, \text{GPS}, alter(O_{medical\_records}))$
$\equiv \ \mathsf{fin}\,(Aut(S_{DOCTOR}, \text{GPS}, alter(O_{medical\_records}))))$

4. so, we have:
$\boxed{\scriptstyle\text{i}}(doctor(S_{DOCTOR}, S_{PATIENT}) \wedge medical\_record(O_{medical\_record}, S_{PATIENT}) \ \equiv$

$\mathsf{fin}\,(Aut^{+}(S_{DOCTOR}, \text{GPS}, alter(O_{medical\_record}))))$
$\wedge \ \boxed{\scriptstyle\text{i}}(\mathsf{false} \ \equiv \ \mathsf{fin}\,(Aut^{-}(S_{DOCTOR}, \text{GPS}, alter(O_{medical\_record}))))$
$\wedge \ \boxed{\scriptstyle\text{i}}(\mathsf{true}\,;\,(len(0) \wedge \ Aut^{+}(S_{DOCTOR}, \text{GPS}, alter(O_{medical\_records})) \ \wedge$
$\neg \ Aut^{-}(S_{DOCTOR}, \text{GPS}, alter(O_{medical\_records})))$
$\equiv \ \mathsf{fin}\,(Aut(S_{DOCTOR}, \text{GPS}, alter(O_{medical\_records}))))$

5. By definition, we have:
$\mathsf{true}\,;\,(len(0) \wedge \ Aut^{+}(S_{DOCTOR}, \text{GPS}, alter(O_{medical\_records})) \ \wedge$
$\neg \ Aut^{-}(S_{DOCTOR}, \text{GPS}, alter(O_{medical\_records}))) \ \equiv$
$\mathsf{fin} \ \ (Aut^{+}(S_{DOCTOR}, \text{GPS}, alter(O_{medical\_records})) \ \wedge$
$\neg \ Aut^{-}(S_{DOCTOR}, \text{GPS}, alter(O_{medical\_records})))$

6. so, we have:
$\boxed{\scriptstyle\text{i}}(doctor(S_{DOCTOR}, S_{PATIENT}) \wedge medical\_record(O_{medical\_record}, S_{PATIENT}) \ \equiv$

$\mathsf{fin}\,(Aut^{+}(S_{DOCTOR}, \text{GPS}, alter(O_{medical\_record}))))$
$\wedge \ \boxed{\scriptstyle\text{i}}(\mathsf{false} \ \equiv \ \mathsf{fin}\,(Aut^{-}(S_{DOCTOR}, \text{GPS}, alter(O_{medical\_record}))))$
$\wedge \ \boxed{\scriptstyle\text{i}}(\mathsf{fin} \ \ (Aut^{+}(S_{DOCTOR}, \text{GPS}, alter(O_{medical\_records})) \ \wedge$
$\neg \ Aut^{-}(S_{DOCTOR}, \text{GPS}, alter(O_{medical\_records})))$

$\equiv\ \mathsf{fin}\,(Aut(S_{DOCTOR},\text{GPS}, alter(O_{medical\_records}))))$

7. which can be written as:
   $\boxed{i}(\mathsf{fin}\ \ (Aut^{+}(S_{DOCTOR},\text{GPS}, alter(O_{medical\_records}))\ \wedge$
   $\neg\ Aut^{-}(S_{DOCTOR},\text{GPS}, alter(O_{medical\_records})))$
   $\equiv\ \mathsf{fin}\,(Aut(S_{DOCTOR},\text{GPS}, alter(O_{medical\_records}))))$

8. We know that:
   $\mathsf{fin}\ \ (Aut^{+}(S_{DOCTOR},\text{GPS}, alter(O_{medical\_records}))\ \wedge$
   $\neg\ Aut^{-}(S_{DOCTOR},\text{GPS}, alter(O_{medical\_records})))\ \equiv$
   $\mathsf{fin}\ \ (Aut^{+}(S_{DOCTOR},\text{GPS}, alter(O_{medical\_records})))\ \wedge$
   $\mathsf{fin}\ \ (\neg\ Aut^{-}(S_{DOCTOR},\text{GPS}, alter(O_{medical\_records})))$

9. Also, we know that:
   $\mathsf{fin}\ \ (\neg\ Aut^{-}(S_{DOCTOR},\text{GPS}, alter(O_{medical\_records})))\ \equiv$
   $\neg\ \mathsf{fin}\ \ (Aut^{-}(S_{DOCTOR},\text{GPS}, alter(O_{medical\_records})))$

10. we know that:
    $\mathsf{false}\ \equiv\ \mathsf{fin}\,(Aut^{-}(S_{DOCTOR},\text{GPS}, alter(O_{medical\_record})))$

11. so, we have:
    $\boxed{i}(\mathsf{fin}\ \ (Aut^{+}(S_{DOCTOR},\text{GPS}, alter(O_{medical\_records})))$
    $\equiv\ \mathsf{fin}\,(Aut(S_{DOCTOR},\text{GPS}, alter(O_{medical\_records}))))$

12. which can be written as:

    $\boxed{i}(\mathsf{fin}\,(Aut(S_{DOCTOR},\text{GPS}, alter(O_{medical\_records})))$
    $\equiv\ \mathsf{fin}\ \ (Aut^{+}(S_{DOCTOR},\text{GPS}, alter(O_{medical\_records}))))$

13. we know that:

    $doctor(S_{DOCTOR}, S_{PATIENT})\ \wedge\ medical\_record(O_{medical\_record}, S_{PATIENT})\ \equiv$
    $\mathsf{fin}\,(Aut^{+}(S_{DOCTOR},\text{GPS}, alter(O_{medical\_record})))$

14. so, we can write it as:
$\boxdot(\mathsf{fin}\,(Aut(S_{DOCTOR},\mathrm{GPS}, alter(O_{medical\_records})))$
$\equiv\ doctor(S_{DOCTOR}, S_{PATIENT})\ \wedge\ medical\_record(O_{medical\_record}, S_{PATIENT}))$

15. which is ψ_3.

5.5 Comparison between the two proofs

In this Section, we will compare the proofs when using the past time operators with the one when using the future time operators.
First, let us take the proof of safety property of the nurse policy with the past time operators:

$$\psi_1 \;\widehat{=}\; \widehat{\Box}((Aut(S_{NURSE}, GPS, a)) \supset \ \mathsf{true} \ \widehat{;} \ \widehat{\Box}(registered(S_{NURSE}))) \tag{5.1}$$

Then, the proof of safety property of the nurse policy with the future time operators:

$$\psi_1 \;\widehat{=}\; \boxed{\scriptstyle i}(\mathsf{fin}\,(Aut(S_{NURSE}, GPS, a)) \supset \ \mathsf{true} \ ; \ \Box(registered(S_{NURSE}))) \tag{5.2}$$

The difference between the first formula (5.1) and the second one (5.2) is the operator fin which is an additional operator in the second formula (5.2). As we see in the Nurse Safety Property, in Section (5.6.1) with the past time operators and in Section (5.10.1) with the future time operators, when comparing between the two proofs, it is clear that one is shorter and contains less symbols. For instance, the proof that rule $R_1^{'}$ satisfies the safety property, when using the proposed verification rules with the past time operators and is much shorter (only 8 steps) than the proofs when using the future time (14 steps). Also, the proof that the rule $R_2^{'}$ satisfies the safety property, when using the past time operators is done with only 11 steps; however, the same proof is done with 16 steps when using future time. In the next table we will list the rules which we have proved satisfy the safety property and in how many steps the proof is done.

Rules number	*Steps with the past*	*Steps with the future*
$R_1^{'}$	8 *steps*	14 *steps*
$R_2^{'}$	11 *steps*	16 *steps*
$R_3^{'}$	10 *steps*	17 *steps*
$R_8^{'}$	7 *steps*	16 *steps*
$R_{14}^{'}$	7 *steps*	15 *steps*

Table 5.3: Rules proof comparison

Table 5.3 shows that all the proofs with the past time operators have been done

with fewer steps compared to the proofs with future time.
Also, if we compare between the starting proof step, which is the third step of the past time proof:
$\mathsf{true}\ \hat{;}\ \widehat{\Box}(\ registered(S_{NURSE})) \ \equiv\ Aut^{+}(S_{NURSE}, \text{GPS}, access)$
$\wedge\ \mathsf{false}\ \equiv\ Aut^{-}(S_{NURSE}, \text{GPS}, access)$
$\wedge\ Aut^{+}(S_{NURSE}, \text{GPS}, access)\ \wedge\ \neg\ Aut^{-}(S_{NURSE}, \text{GPS}, access)\ \equiv$
$Aut(S_{NURSE}, \text{GPS}, access)$

For the future time proof, the third step is:
$\boxed{i}((\mathsf{true}\ ;\ \Box(\ registered(S_{NURSE})))\ \equiv\ \mathsf{fin}\ (Aut^{+}(S_{NURSE}, \text{GPS}, access)))$
$\wedge\ \boxed{i}(\mathsf{false}\ \equiv\ \mathsf{fin}\ (Aut^{-}(S_{NURSE}, \text{GPS}, access)))$
$\wedge\ \boxed{i}((\mathsf{true};(len(0) \wedge Aut^{+}(S_{NURSE}, \text{GPS}, access) \wedge \neg\, Aut^{-}(S_{NURSE}, \text{GPS}, access)))\ \equiv$
$\mathsf{fin}\ \ Aut(S_{NURSE}, \text{GPS}, access))$

It can be seen that the proof with past time is much shorter and has less number of symbols and this makes the proof easier to express (simpler) than the future time proof. As we have said before, the reason is the additional two operators fin and $\boxed{i}$ which are used in the future time proof. This affects the remaining of the proof and hence it has more steps. So, it can be said that future time uses more operators, fin and $\boxed{i}$, than the past time in the verification process and this make the verification of the safety property with the past time much shorter as it has a less number of symbols (more succinct) than the proof with future time; also it can be said it is easier to express (simpler) regarding succinctness. Consequently, it can be concluded that the succinctness and simplicity are achieved in the verification step.

5.6 Chapter Summary

In order to evaluate the past time operators of ITL^p, the scenario of GPS system is given and specified using the past time operators of ITL^p as well as the existing future time operators of ITL.
The proof rules which have been proposed in Section 4.5.1, have been used to verify the safety property of the GPS scenario and this show the benefits of the proof system ofITL^p. Also, we have used the existing proof rules of ITL to verify the safety property of the GPS system. A comparison between the specification and verification when using past time operators with the specification and verification

of future time operators has been made to evaluate the past time operators. This Chapter evaluates the work done in the contribution Chapters (3 and 4) and shows that the past time operators of ITL^p can be used to reason about history-based access control policy. Additionally, the proposed verification rules (Section 4.3.1) are used to prove that GPS system satisfies the safety property.

Chapter 6

CONCLUSION

In this Chapter:

- *Thesis summary.*
- *Comparison with related work.*
- *Original contributions.*
- *Success criteria revisited*
- *Conclusion.*
- *Limitations.*
- *Future work.*

6.1 Thesis summary

The aim of this work is to contribute the development of interval temporal logic with past time operators ITL^p as formal specification and verification language and use it to reason about history based access control policies.
Towards this goal, this research has explored in the literature review the specification, temporal logic history including the time models and the classification of temporal logic and finally, the access control policies where the policy languages and models to reason about history based access control systems are discussed in order to support our choice of ITL^p. It has been shown that these languages and models are not appropriate to reason about this class of policies except SANTA and the proposed model.
Interval temporal logic with future time ITL and with past time ITL^p have been introduced with their syntaxes and semantics, and the proof systems. Moreover, SANTA operators such as always-followed-by have been given a history semantics using past time operators of ITL^p to use in the Scenario Chapter. In the Scenario Chapter, the specification of GPS system has been described using ITL^p and SANTA operators with the history semantics; however, the verification rules proposed in Section 4.5 has been used to verify the safety property of GPS.
In order to evaluate ITL^p, the specification of the GPS system has been described using ITL and the existing SANTA operators, and the verification rules have been used to verify the safety property of the given scenario. There has then been a comparison between the specification of GPS using past time operators of ITL^p and using the future time operators of ITL to show the advantages of the introduced past time operators.

6.2 Comparison with related work

This thesis presents the past time operators of ITL^p which are different from the interval temporal logic ITL proposed by Moszkowski [98] since the past time operators such as past chop ($\hat{;}$), and past chopstar ($^{\hat{*}}$) have been proposed instead of the future one.
Duan's [37] and Bowman's [19] approaches to interval temporal logic are the closest ones to ours. However, there are many differences between these two works compared to our work in this thesis. The first one is that Duan and Bowman combine the use of past operators with the future one. This use may be accepted in the

syntax level but in the semantics level it will lead the model to become complicated and not clear as the current state is swinging between the future and the past. Also, the two works have used the past time operators of ITL, but they do not have the same semantics. To explain that, take the past chop operators proposed by Duan. First, Duan has used the interpretation notation (σ, i, k, j) rather than an interval in the original ITL where σ is fixed and the formula p interpreted over it. The chop operator is used to partition the whole interval σ to some subintervals where the sub formula p is interpreted over. The notations i, k, j are used to specify the subintervals $\sigma_{(i \ldots j)}$ of σ where σ_k is the current state and a sub formula of p. The sub formula may involve next and previous operators in arbitrary order, so σ_k may swing between σ_i and σ_j. Also, when using the chop operator in the sub interval $\sigma_{(i \ldots j)}$, this produces two subintervals $\sigma_{(i \ldots h)}$ where the current position is σ_k, and $\sigma_{(h \ldots j)}$ where the current position is σ_h as shown in Figure 6.1.

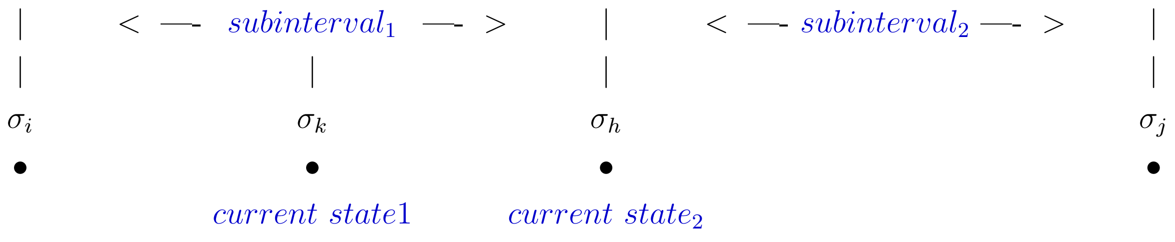

Figure 6.1: Duan Chop

However, Bowman in Multimedia in Executable Interval Temporal Logic (Mexitl), includes the past chop operators ($\tilde{;}$). To explain the chop operators in Mexitl, assume that we have $A \tilde{;} B$ which is satisfied by an interval such that:

1. A holds over the larger interval resulting from moving the start of the interval into the past, and
2. B holds over the original interval according to a past history that is truncated at the start of the interval over which A holds.

The intervals are assumed to be line segments with three reference points. Starting from the left hand side, the start of time is at the leftmost point, the start of the current interval is at the next point and the end of the current interval is at the final point. Therefore, time is divided into a past history interval and a current interval as shown in Figure 6.2.

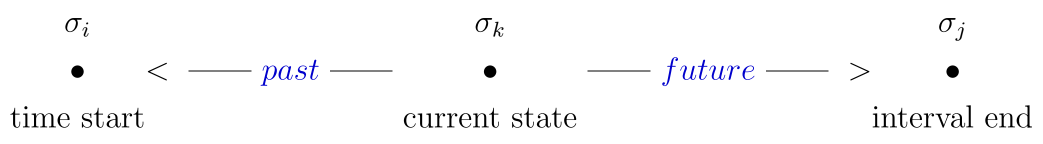

Figure 6.2: Mexitl Chop

In addition, Duan's chop behaves differently to Bowman's chop which also behave differently to the past chop proposed in this work. As we see from above, the two works in additional to our work has incorporated past operators, However, these operators have different semantics as is explained above. Also, they incorporate the uses of past time operators with the future time and this make these operators difficult to understand and to use because the reference point is not fixed.

6.3 Original Contributions

With the massive improvement of all the systems in the science area and in computer science in particular, this necessitates the specification languages such as Interval Temporal Logic ITL to develop to specify history based access policies; these are a very expressive class of policies that can define policy decisions dependent on previously observed behaviours within the system and one of these developments includes the past.
So far, most languages incorporate only future time operators and exclude the use of past time operators. The reason behind this is that in various relevant cases the addition of past operators does not increase the expressiveness power of these temporal languages.
However, supporters reply that the succinctness is achieved whether the expressive power is added or not according to the fact that there are many properties can be expressed by means of much shorter formulas. What is more, when using past operators many statements become easier to express (simplicity) because of a less number of symbols and therefore it can be said that the use of temporal logic when referring to the past is much easier as we will show in simple examples.
As a result, past time operators of ITL^p such as past chop ($\overset{\frown}{;}$) and past chop star

($\hat{*}$) and past skip ($\widehat{\mathsf{skip}}$) were defined to enable more simplicity and succinctness in specifications. Our research work has been to include the past time operators of ITL^p and define the syntax and semantics of these operators. Then, the complete set of axioms and rules of ITL^p are proposed and have proved sound. What is more, the past time operators have been used to give a history semantics to the SANTA operators always-followed-by and the strong version of it in (Section 4.3.3).
Additionally, we proposed the verification rules in (Section 4.5.1); these rules have been used to verify that a system satisfies a property. We have used these verification rules to prove that the GPS policies satisfy the safety property.
The past time operators of ITL^p have been used to reason about history based access control policies of GPS. A scenario of GPS is given to illustrate the use of past time operators of ITL^p in a history based access control systems where the policy decision depends on previously observed behaviour to show the application of these operators and the proposed proof rules.

6.4 Success Criteria Revisited

In order to measure the success of our research, success criteria were formulated in Chapter 1. These criteria are revisited here:

- The past time operators of interval temporal logic are suitable to reason about and express history based access control policies.
  As we have shown in the GPS scenario, the past time Interval Temporal Logic ITL^p is appropriate and suitable to reason about history based access control policies and this give us all the advantages of using ITL^p, e.g., the use of the proof rules to verify the safety property of history interval. This can be seen in the specification of the scenario and the verification of the safety property of the GPS policies.

- The change of numbering of states with past time operators of interval temporal logic make the reasoning about history easier.
  In the problem statement and research motivation (Section 1.2), it has been discussed how the " numbering of states" is changed when the past time operators are used, also in the comparison between the two proofs (Section 5.5),

it has been shown that this change makes the fin operator redundant and simplify the reasoning about history.

- The formal specification and verification of history based access control policies when using past time operators of ITL is more succinct and thus, easier to express (simpler).
  In the given GPS scenario, we have used the past time operators to proof the safety property as it is shown in Sections 5.3.5, then, in Section 5.4.3, we have used the future time operators to proof the same safety property.
  As it has been shown in the comparison between the two proofs (Section 5.5), it can be said that is the use of the past time operators to proof the safety property making it much shorter (succinct) and easier to prove (simplicity)

6.5 Conclusion

The interval temporal logic with past time operators ITL^p has been investigated: past modalities, past chop ($\hat{;}$), past chopstar ($\hat{*}$) and past skip ($\widehat{\mathsf{skip}}$) that allows us to reason about the past has been introduced. Once we used the past time operators we found out that they helped us to clarify several issues. In particular they contribute to reason about a very expressive security scenario such as history based access control policies, where the policy decisions depend on previously observed behaviours within the system. It is known that past time operators of interval temporal logic ITL^p do not increases the expressive power of interval temporal logic ITL; that is, all the property which can be expressed using past time operators can be expressed using future time operators. Also, the specification of the GPS policies using the past time operators (Section 5.3.2) is the same as the specification with the future time (Section5.4.1). However, there are classes of property proof that can be expressed by means of much shorter formulas and also with less symbols (succinctness), hence, the formula is easier to express (simplicity).
This is clear in the safety property with the past time (Section 5.3.4) and in the safety property with the future time (Section 5.4.3).
Succinctness is achieved due to all the safety property proofs having been done in fewer steps as has been shown in Table 5.3; also, all the proof formulas with past time operators are much shorter than the proof formulas with the future time, as is show in the comparison between the proof (Sections 5.5). Also, Succinctness and simplicity can be achieved in the specification level as we will show in the following

example:
If we want to express that "every request is eventually granted" one finds it natural to write it as:

$$\Box(request \supset \Diamond grant) \tag{6.1}$$

But, if we would like to express that "every grant is preceded by a request"

$$\widehat{\Box}(grant \supset \widehat{\Diamond} request) \tag{6.2}$$

formula (6.2) can be expressed without past time operators as:

$$\Box(\mathsf{fin}\,(grant) \supset \Diamond\ request) \tag{6.3}$$

This example shows that the interval temporal logic formula with past operators (6.2) is shorter than the future one (6.3). Thus, it can be said that the specifications of history with the past time interval temporal logic can be more succinct and thus simpler as has been shown in the example provided.
In conclusion, while this research is looking at the possible advantages of using the past time operators of interval temporal logic to reason about security policies, where the policy decisions depend on previously observed behaviours within the system known as history based access control policy, the introduced operators can be applied to several applications such as log file analysis (Section 6.8.2). Past time operators of interval temporal ITL^p and particularly the two operators past Chop ($\hat{;}$) and past Chopstar ($\hat{*}$) have the ability to express any system consisting of phases; and these phases consist of sequence of states such as history based access control policies.
This thesis has made a useful contribution to enable ITL^p to be used in nontrivial scenario such as GPS. However, the area of ITL and particularly ITL^p needs further developments and it is hoped that it will see a growing number and wider spectrum of researchers.

6.6 Limitations

Any class of interval based temporal logic over linear orderings that contain at least one linear ordering with an infinite ascending or descending chain of points such as ITL is suffers from a will-known weakness that is undecidability since the formula

of these logic are evaluated over intervals, that is, pairs of points. As a consequence, formula translated into binary relation over the underlying ordering and, respectively, the validity and satisfiability problems translate into dyadic second-order logic [94]. This situation had discouraged attempts for practical applications and further research on interval temporal logics. The formula is satisfiable if there exists an interpretation of the formula as true, whereas a formula is valid if for every interpretation the formula is true, it can be said that the presence of chop operator makes the satisfiability of ITL formulas undecidable.
However, there are some additional limitations when using the past time operators; one of the these limitations is that we are not combining the use of past time operators together with the future time operators; according combining the past time and future time operators are suffer from the current state problem and this is clear in the work proposed by Duan in [37] and Bowman in [19]. Also, the executable subset AnaTempura interpreter does not include the use of past time operators so, we cannot execute these operators and benefits from the executability advantages.

6.7 Future Work

In the future, we plan to add the past operators to AnaTempura interpreter itself and this give us the choice to use the past operators or not. We also hope to describe the operational semantics of these operators in Tempura to execute the past operators and benefit from the advantages of executing ITL formula and this will enable us to formalize the relation between various execution strategies.

6.8 Future impact

6.8.1 Academic impact

Adding the past time operators to AnaTempura interpreter and benefiting from the advantages of executing ITL formula has a future academic impact in the backtracking which is a well-known technique used when exploring logical properties. In this technique, whenever we reach a point, we have a choices, for example between disjuncts, and we make a selection. If we later find that our choice led to some inconsistency or not a good choice we stop and use the past operators to revert back to the last point where the choice has been made and select a different option. If we find that all the choices form this point where the choice was made have been explored and each leads to an inconsistency, then the past operators can be used to go back to the last choice before this one and so on. If we eventually get back to our initial starting point, then we know that the formula is inconsistent because we have explored all potential models. The past operators in this technique are used instead of the normal procedure which is writing down all the possible choices and trying them one by one; if we find that the choice is a wrong choice we go back to the point where the choice has been made and try another one manually. If the explored logical properties are simple then this process can be done manually, but if the logical properties are too long and we have many choices, then this process cannot be done manually and the past operators should be used to perform the backtracking.

6.8.2 Industrial impact

Current software application often produces some auxiliary text files known as log files. These files, reports all the events that have occurred during the running of programs continuously.
Typically, log files are used by programs in the following way:
The log file is an auxiliary output file, distinct from other outputs of the program and almost all log files are plain text files. On start-up of the program, the log file is either empty, or contains whatever was left from previous runs of the program. During program operation, lines (or groups of lines) are gradually appended to the log file, never deleting or changing any previously stored information. Each record in a log file is caused by a given event in the program, like user interaction, function call, input or output procedure. Records in log files are often parameterized, i.e. they show current values of variables, return values of function calls or any other state information. The information reported in log files is the information that programmers consider important or useful for program monitoring and/or locating faults.
The proposed verification rules when using past time operators (Section 4.5 and Section 4.5.1) can be applied to check that the log files records satisfy a property such as safety property because we can use these records as execution history.

Bibliography

[1] Martin Abadi and Cédric Fournet. Access control based on execution history. In *NDSS*. The Internet Society, 2003.

[2] James Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, November 1983.

[3] James Allen. Towards a general theory of action and time. *Artif. Intell.*, 23(2):123–154, July 1984.

[4] James Allen and Patrick Hayes. A common-sense theory of time. In *Proceedings of the 9th international joint conference on Artificial intelligence - Volume 1*, IJCAI'85, pages 528–531, San Francisco, CA, USA, 1985. Morgan Kaufmann Publishers Inc.

[5] James Allen and Patrick Hayes. Moments and points in an interval-based temporal logic. *Comput. Intell.*, 5(4):225–238, May 1990.

[6] Bowen Alpern and Fred Schneider. Recognizing safety and liveness. *Distributed Computing*, 2:117–126, 1986.

[7] Alessandro Artale, Enrico Franconi, Milenko Mosurovic, Frank Wolter, and Michael Zakharyaschev. Temporal description logic. In *Handbook of Time and Temporal Reasoning in Artificial Intelligence*, pages 96–105. MIT Press, 2001.

[8] Arosha Bandara, Nicodemos Damianou, Emil Lupu, and Morris Sloman. Policy based management. In Jan Bergstra and Mark Burgess, editors, *Handbook of Network and System Administration*, pages 507–564. Elsevier, 2008.

[9] Massimo Bartoletti. *Language-based Security: Access Control and Static Analysis*. PhD thesis, Dipartimento di Informatica, April 2005.

[10] Moritz Becker, Cédric Fournet, and Andrew Gordon. Secpal: Design and semantics of a decentralized authorization language. *Journal of Computer Security*, 18(4):619–665, 2010.

[11] Ilan Beer, Shoham Ben-david, Dana Fisman, Anna Gringauze, and Yoav Rodeh. The temporal logic sugar. In *Computer Aided Verification*, pages 363–367. Springer, 2001.

[12] David Elliott Bell and Leonard LaPadula. Secure computer systems: Mathematical foundations. Technical report, MITRE Corporation, March 1973.

[13] David Elliott Bell and Leonard LaPadula. Secure Computer System: Unified Exposition and MULTICS Interpretation. Technical Report ESD-TR-75-306, The MITRE Corporation, 1976.

[14] Elisa Bertino, Piero Andrea Bonatti, and Elena Ferrari. Trbac: A temporal role-based access control model. *ACM Trans. Inf. Syst. Secur.*, 4(3):191–233, August 2001.

[15] Kenneth Biba. Integrity considerations for secure computer systems. Technical report, MITRE Corp., 04 1977.

[16] Woody Bledsoe and Donald Loveland. *Automated Theorem Proving: After 25 Years*. American Mathematical Society: Annual meeting. American Mathematical Society, 1984.

[17] Howard Bowman. An interpretation of cognitive theory in concurrency theory. Technical Report 8-98, Computing Laboratory, University of Kent at Canterbury, October 1998.

[18] Howard Bowman, Helen Cameron, Peter King, and Simon Thompson. Specification and prototyping of structured multimedia documents using interval temporal logic. In Howard Barringer, Michael Fisher, Dov Gabbay, and Graham Gough, editors, *Advances in Temporal Logic*, volume 16 of *Applied Logic Series*, pages 435–453. Springer Netherlands, 2000.

[19] Howard Bowman, Helen Cameron, Peter King, and Simon Thompson. Mexitl: Multimedia in executable interval temporal logic. *Formal Methods in System Design*, 22(1):5–38, 2003.

[20] Don Box and Chris Sells. *Essential .Net: The Common Language Runtime.* Essential .NET. Addison-Wesley, 2003.

[21] Wilfried Brauer, Wolfgang Reisig, and Grzegorz Rozenberg, editors. *Advances in Petri Nets 1986, Part I on Petri Nets: Central Models and Their Properties*, London, UK, 1987. Springer-Verlag.

[22] Antonio Cau. *Compositional Verification and Specification of Refinement for Reactive Systems in a Dense Time Temporal Logic.* Bericht. Inst. für Informatik und Praktische Math., 1996.

[23] Antonio Cau. Interval temporal logic, February 2012. `http://www.cse.dmu.ac.uk/STRL/ITL/`.

[24] Antonio Cau, Helge Janicke, and Ben Moszkowski. Verification and enforcement of access control policies. *Formal Methods in System Design*, 43(3):450–492, 2013.

[25] Antonio Cau and Hussein Zedan. Refining interval temporal logic specifications. In *Transformation-Based Reactive Systems Development, number 1231 in LNCS*, pages 79–94. AMAST, Springer-Verlag, 1997.

[26] Edward Chang, Zohar Manna, and Amir Pnueli. Characterization of temporal property classes. In *Proceedings of the 19th International Colloquium on Automata, Languages and Programming*, ICALP '92, pages 474–486, London, UK, 1992. Springer-Verlag.

[27] Jan Chomicki. Efficient checking of temporal integrity constraints using bounded history encoding. *ACM Transactions on Database Systems (TODS)*, 20(2):149–186, 1995.

[28] Jan Chomicki and David Toman. Logics for databases and information systems. chapter Temporal logic in information systems, pages 31–70. Kluwer Academic Publishers, Norwell, MA, USA, 1998.

[29] Edmund Clarke and Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs, Workshop*, pages 52–71, London, UK, UK, 1982. Springer-Verlag.

[30] Edmund Clarke, Allen Emerson, and Aravinda Prasad Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, April 1986.

[31] Edmund Clarke, William Klieber, Miloš Nováček, and Paolo Zuliani. Model checking and the state explosion problem. In *Tools for Practical Software Verification*, pages 1–30. Springer, 2012.

[32] Gordana Dodig Crnkovic. Constructive research and info-computational knowledge generation. *Model-Based Reasoning in Science and Technology*, pages 359–380, 2010.

[33] Dorothy Denning. A lattice model of secure information flow. *Commun. ACM*, 19(5):236–243, May 1976.

[34] Dorothy Denning. *Cryptography and data security.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1982.

[35] Antoni Diller. *Z: an introduction to formal methods.* John Wiley & Sons, Inc., 1994.

[36] Clare Dixon, Mari-Carmen Fernández Gago, Michael Fisher, and Wiebe van der Hoek. Temporal logics of knowledge and their applications in security. *Electronic Notes in Theoretical Computer Science*, 186:27–42, 2007.

[37] Zhenhua Duan. *An Extended Interval Temporal Logic and A Framing Technique for Temporal Logic Programming.* Phd thesis, University of Newcastle Upon Tyne, 1996.

[38] Zhenhua Duan and Nan Zhang. A complete axiomatization of propositional projection temporal logic. In *Proceedings of the 2008 2nd IFIP/IEEE International Symposium on Theoretical Aspects of Software Engineering*, TASE '08, pages 271–278, Washington, DC, USA, 2008. IEEE Computer Society.

[39] Zhenhua Duan, Nan Zhang, and Maciej Koutny. A complete proof system for propositional projection temporal logic. *Theoretical Computer Science*, 497:84–107, 2013.

[40] Hartmut Ehrig, Bernd Mahr, Ingo Classen, and Fernando Orejas. Introduction to algebraic specification. part 1: Formal methods for software development. *The Computer Journal*, 35(5):460–467, 1992.

[41] Allen Emerson. Handbook of theoretical computer science (vol. b). chapter Temporal and modal logic, pages 995–1072. MIT Press, Cambridge, MA, USA, 1990.

[42] Ronald Fagin, Joseph Halpern, Yoram Moses, and Moshe Vardi. *Reasoning About Knowledge.* MIT Press, 1995.

[43] David Ferraiolo, D. Richard Kuhn, and Ramaswamy Chandramouli. Role-based access control, artech house. *Inc., Norwood, MA*, 2003.

[44] José Luiz Fiadeiro and Tom Maibaum. Sometimes "tomorrow" is "sometime". In Dov Gabbay and Hans Jürgen Ohlbach, editors, *Temporal Logic*, volume 827 of *Lecture Notes in Computer Science*, pages 48–66. Springer Berlin Heidelberg, 1994.

[45] Marcelo Finger and Mark Reynolds. Imperative history: Two-dimensional executable temporal logic. In Hans Jürgen Ohlbach and Uwe Reyle, editors, *Logic, Language and Reasoning*, volume 5 of *Trends in Logic*, pages 73–98. Springer Netherlands, 1999.

[46] Michael Fisher. Implementing temporal logics: Tools for execution and proof. In *Proceedings of CLIMA VI, LNAI 3900*, pages 129–142. Springer.

[47] Michael Fisher. An introduction to executable temporal logics. *The Knowledge Engineering Review*, 11:43–56, 2 1996.

[48] Michael Fisher. *An Introduction to Practical Formal Methods Using Temporal Logic.* Wiley, 2011.

[49] Michael Fisher and Richard Owens. From the past to the future: Executing temporal logic programs. In *In Proceedings of Logic Programming and Automated Reasoning (LPAR)*, pages 369–380. Springer Verlag, 1992.

[50] Michael Fisher and Richard Owens. An introduction to executable modal and temporal logics. In *Proceedings of the Workshop on Executable Modal and Temporal Logics*, IJCAI '93, pages 1–20, London, UK, 1995. Springer-Verlag.

[51] Cédric Fournet and Andrew Gordon. Stack inspection: Theory and variants. *SIGPLAN Not.*, 37(1):307–318, January 2002.

[52] Dov Gabbay. The declarative past and imperative future. In Behnam Banieqbal, Howard Barringer, and Amir Pnueli, editors, *Temporal Logic in Specification*, volume 398 of *Lecture Notes in Computer Science*, pages 409–448. Springer Berlin Heidelberg, 1989.

[53] Dov Gabbay, Amir Pnueli, Saharon Shelah, and Jonathan Stavi. On the temporal analysis of fairness. In *Proceedings of the 7th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '80, pages 163–173, New York, NY, USA, 1980. ACM.

[54] Antony Galton. *The Logic of Aspect: An Axiomatic Approach.* Clarendon library of logic and philosophy. Clarendon Press, 1984.

[55] Narain Gehani. Specifications: Formal and informal - a case study. *Software: Practice and Experience*, 12(5):433–444, 1982.

[56] Joseph Goguen. *Higher Order Functions Considered Unnecessary for Higher Order Programming.* Computer Science Laboratory Menlo Park, Calif: SRI-CSL. SRI International, Computer Science Laboratory, 1988.

[57] Rodolfo Gomez and Howard Bowman. PITL2MONA: Implementing a Decision Procedure for Propositional Interval Temporal Logic. *Journal of Applied Non-Classical Logics*, 14(1-2):105–148, unknown 2004. Issue on Interval Temporal Logics and Duration Calculi. V. Goranko and A. Montanari guest eds.

[58] Valentin Goranko, Angelo Montanari, and Guido Sciavicco. A road map of interval temporal logics and duration calculi. *Journal of Applied Non-Classical Logics*, 14(1-2):9–54, 2004.

[59] Orlena Gotel and Anthony Finkelstein. An analysis of the requirements traceability problem. In *Requirements Engineering, 1994., Proceedings of the First International Conference on*, pages 94–101. IEEE, 1994.

[60] Joseph Halpern, Zohar Manna, and Ben Moszkowski. A hardware semantics based on temporal intervals. Technical report, Stanford, CA, USA, 1983.

[61] David Harel. Statecharts: A visual formalism for complex systems. *Science of computer programming*, 8(3):231–274, June 1987.

[62] Michael Harrison, Walter Ruzzo, and Jeffrey Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, August 1976.

[63] Qingfeng He. *Requirements-Based Access Control Analysis And Policy Specification.* PhD thesis, North Carolina State University, 2005.

[64] Yoav Hollander, Morley Morley, and Amos Noy. The e language: a fresh separation of concerns. In *Technology of Object-Oriented Languages and Systems, 2001. TOOLS 38. Proceedings*, pages 41–50. IEEE, 2001.

[65] Ullrich Hustadt. Temporal logic: Mathematical foundations and computational aspects, volume 2, dov gabbay, mark reynolds, and marcelo finger. *Journal of Logic, Language and Information*, 10(3):406–410, 2001.

[66] Michael Huth and Mark Ryan. *Logic in Computer Science: Modelling and Reasoning About Systems.* Cambridge University Press, 2000.

[67] Sushil Jajodia, Pierangela Samarati, Maria Luisa Sapino, and V. S. Subrahmanian. Flexible support for multiple access control policies. *ACM Trans. Database Syst.*, 26(2):214–260, June 2001.

[68] Helge Janicke. *The development of secure multi-agent systems.* PhD thesis, De Montfort University, 2007.

[69] Helge Janicke, Antonio Cau, François Siewe, and Hussein Zedan. Dynamic access control policies: Specification and verification. *The Computer Journal*, 2012.

[70] Helge Janicke, Antonio Cau, and Hussein Zedan. A note on the formalisation of ucon. In *Proceedings of the 12th ACM Symposium on Access Control Models and Technologies*, SACMAT '07, pages 163–168, New York, NY, USA, 2007. ACM.

[71] Kristofer Johannisson. *Formal and informal software specifications.* Citeseer, 2005.

[72] Yonit Kesten, Zohar Manna, and Amir Pnueli. Temporal verification of simulation and refinement. In J.W. Bakker, W.-P. Roever, and G. Rozenberg, editors, *A Decade of Concurrency Reflections and Perspectives*, volume 803 of *Lecture Notes in Computer Science*, pages 273–346. Springer Berlin Heidelberg, 1994.

[73] Stephen Cole Kleene. Representation of events in nerve nets and finite automata. In Claude Shannon and John McCarthy, editors, *Automata Studies*, pages 3–41. Princeton University Press, Princeton, NJ, 1956.

[74] Savas Konur. A survey on temporal logics. *CoRR*, abs/1005.3199, 2010.

[75] Thomas Kropf. *Introduction to Formal Hardware Verification.* Springer, 1999.

[76] Orna Kupferman and Moshe Vardi. Synthesis with incomplete informatio. In *In Advances in Temporal Logic*, pages 109–127. Kluwer Academic Publishers, 2000.

[77] Leslie Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(3):872–923, May 1994.

[78] Butler Lampson. Protection. *ACM SIGOPS Operating Systems Review*, 8(1):18–24, January 1974.

[79] François Laroussinie and Nicolas Markey. Temporal logic with forgettable past. In *In LICS02*, pages 383–392. IEEE Computer Society Press, 2002.

[80] François Laroussinie and philippe Schnoebelen. A hierarchy of temporal logics with past. In Patrice Enjalbert, ErnstW. Mayr, and KlausW. Wagner, editors, *STACS 94*, volume 775 of *Lecture Notes in Computer Science*, pages 47–58. Springer Berlin Heidelberg, 1994.

[81] Peter Gorm Larsen. Ten years of historical development "bootstrapping" pdm tools vx. *Journal of Universal Computer Science*, 7(8):692–709, 2001.

[82] Martin Leucker and César Sánchez. Regular linear temporal logic. In *Proceedings of the 4th international conference on Theoretical aspects of computing*, ICTAC'07, pages 291–305, Berlin, Heidelberg, 2007. Springer-Verlag.

[83] Orna Lichtenstein, Amir Pnueli, and Lenore Zuck. The glory of the past. In Rohit Parikh, editor, *Logics of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 196–218. Springer Berlin Heidelberg, 1985.

[84] Orna Lichtenstein, Amir Pnueli, and Lenore Zuck. The glory of the past. In *Proceedings of the Conference on Logic of Programs*, pages 196–218, London, UK, UK, 1985. Springer-Verlag.

[85] Carsten Lutz. Temporal logic. 2006. Summer semester14 lectures.

[86] Zohar Manna and Amir Pnueli. Verification of concurrent programs, part i: The temporal framework. Technical report, Stanford, CA, USA, 1981.

[87] Zohar Manna and Amir Pnueli. A hierarchy of temporal properties. In *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, PODC '87, pages 205–205, New York, NY, USA, 1987. ACM.

[88] Zohar Manna and Amir Pnueli. *The temporal logic of reactive and concurrent systems.* Springer-Verlag New York, Inc., New York, NY, USA, 1992.

[89] Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* The Temporal Logic of Reactive and Concurrent Systems. Springer-Verlag, 1992.

[90] Nicolas Markey. Temporal logic with past is exponentially more succinct, concurrency column. *Bulletin of the EATCS*, 79:122–128, 2003.

[91] Robin Milner. *A Calculus of Communicating Systems.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982.

[92] Robin Milner. The polyadic π-calculus: a tutorial. In Friedrich L. Bauer, Wilfried Brauer, and Helmut Schwichtenberg, editors, *Logic and Algebra of Specification*, volume 94 of *NATO ASI Series*, pages 203–246. Springer Berlin Heidelberg, 1993.

[93] Dario Della Monica, Angelo Montanari, and Pietro Sala. The importance of the past in interval temporal logics: The case of propositional neighborhood logic. In *Logic Programs, Norms and Action*, pages 79–102, 2012.

[94] Angelo Montanari. Back to interval temporal logics. In Maria Garcia de la Banda and Enrico Pontelli, editors, *Logic Programming*, volume 5366 of *Lecture Notes in Computer Science*, pages 11–13. Springer Berlin Heidelberg, 2008.

[95] Ben Moszkowski. Reasoning about digital circuits. Phd, Department of Computer Science, Stanford University, Technical Report STAN-CS-83-970, Stanford, CA, 1983.

[96] Ben Moszkowski. *Reasoning about digital circuits.* PhD thesis, Stanford, CA, USA, 1983. AAI8329756.

[97] Ben Moszkowski. A temporal logic for multilevel reasoning about hardware. *Computer*, 18(2):10–19, 1985.

[98] Ben Moszkowski. *Executing Temporal Logic Programs.* Cambridge University Press, 1986.

[99] Ben Moszkowski. Some very compositional temporal properties. In *Proceedings of the IFIP TC2/WG2.1/WG2.2/WG2.3 Working Conference on Programming Concepts, Methods and Calculi*, PROCOMET '94, pages 307–326, Amsterdam, The Netherlands, The Netherlands, 1994. North-Holland Publishing Co.

[100] Ben Moszkowski. Compositional reasoning about projected and infinite time. In *Engineering of Complex Computer Systems, 1995. Held jointly with 5th CSESAW, 3rd IEEE RTAW and 20th IFAC/IFIP WRTP, Proceedings., First IEEE International Conference on*, pages 238–245, 1995.

[101] Ben Moszkowski. A hierarchical completeness proof for propositional interval temporal logic with finite time. *Journal of Applied Non-Classical Logics*, 14(1-2):55–104, 2004.

[102] Ben Moszkowski. Compositional reasoning using intervals and time reversal. In *Temporal Representation and Reasoning (TIME), 2011 Eighteenth International Symposium on*, pages 107–114, Sept 2011.

[103] Ben Moszkowski. A complete axiom system for propositional interval temporal logic with infinite time. *arXiv preprint arXiv:1207.3816*, 2012.

[104] Ben Moszkowski. A complete axiom system for propositional interval temporal logic with infinite time. *Logical Methods in Computer Science*, 8(3), 2012.

[105] Peter Øhrstrøm and Per Hasle. *Temporal Logic: From Ancient Ideas to Artificial Intelligence.* Mathematics and Its Applications. Springer, 1995.

[106] Sam Owre, John Rushby, and Natarajan Shankar. Pvs: A prototype verification system. In Deepak Kapur, editor, *Automated DeductionCADE-11*, volume 607 of *Lecture Notes in Computer Science*, pages 748–752. Springer Berlin Heidelberg, 1992.

[107] Jaehong Park and Ravi Sandhu. The ucon abc usage control model. *ACM Transactions on Information and System Security (TISSEC)*, 7(1):128–174, February 2004.

[108] Jaehong Park, Xinwen Zhang, and Ravi Sandhu. Attribute mutability in usage control. In Csilla Farkas and Pierangela Samarati, editors, *Research Directions in Data and Applications Security XVIII*, volume 144 of *IFIP International Federation for Information Processing*, pages 15–29. Springer US, 2004.

[109] Sean Peisert and Matt Bishop. Dynamic, flexible, and optimistic access control. Technical report, Technical Report CSE-2013-76, University of California at Davis, 2013.

[110] Charles Pfleeger and Shari Lawrence Pfleeger. *Security in Computing (4th Edition)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2006.

[111] Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, SFCS '77, pages 46–57, Washington, DC, USA, 1977. IEEE Computer Society.

[112] Amir Pnueli. Logics and models of concurrent systems. chapter In transition from global to modular temporal reasoning about programs, pages 123–144. Springer-Verlag New York, Inc., New York, NY, USA, 1985.

[113] Amir Pnueli. Applications of temporal logic to the specification and verification of reactive systems: A survey of current trends. In Willem-Paul De Roever Jacobus De Bakker and Grzegorz Rozenberg, editors, *Current Trends in Concurrency*, volume 224 of *Lecture Notes in Computer Science*, pages 510–584. Springer Berlin Heidelberg, 1986.

[114] Amir Pnueli and Eyal Harel. Applications of temporal logic to the specification of real time systems. In M. Joseph, editor, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 331 of *Lecture Notes in Computer Science*, pages 84–98. Springer Berlin Heidelberg, 1988.

[115] Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '89, pages 179–190, New York, NY, USA, 1989. ACM.

[116] Arthur Prior. *Past, Present and Future.* Clarendon Press, 1967.

[117] Nageshwar Rao Pusuluri. *Software Testing Concepts And Tools.* Dreamtech Press, 2006.

[118] Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in cesar. In *Proceedings of the 5th Colloquium on International Symposium on Programming*, pages 337–351, London, UK, UK, 1982. Springer-Verlag.

[119] Chandramouli Ramaswamy and Ravi Sandhu. Role-based access control features in commercial database management systems. In *In Proceedings of 21st NIST-NCSC National Information Systems Security Conference*, pages 503–511. Citeseer, 1998.

[120] Steve Reeves and Michael Clarke. *Logic for computer science.* International computer science series. Addison-Wesley, 1990.

[121] Kristin Rozier. Survey: Linear temporal logic symbolic model checking. *Computer Science Review Journal*, 5(2):163–203, May 2011.

[122] Peter Ryan. Mathematical models of computer security. In Riccardo Focardi and Roberto Gorrieri, editors, *Foundations of Security Analysis and Design*, volume 2171 of *Lecture Notes in Computer Science*, pages 1–62. Springer Berlin Heidelberg, 2001.

[123] Pierangela Samarati and Sabrina de Vimercati. Access control: Policies, models, and mechanisms. *Foundations of Security Analysis and Design*, pages 137–196, 2001.

[124] Ravi Sandhu. Transaction control expressions for separation of duties. In *Aerospace Computer Security Applications Conference, 1988., Fourth*, pages 282–286, 1988.

[125] Ravi Sandhu, Edward Coyne, Hal Feinstein, and Charles Youman. Role-based access control models. *Computer*, 29(2):38–47, February 1996.

[126] SANTA. Security analysis toolkit for agents (santa), July 2013. `http://www.tech.dmu.ac.uk/STRL/research/software/SANTA.pdf`.

[127] Fred Schneider. Enforceable security policies. *ACM Trans. Inf. Syst. Secur.*, 3(1):30–50, February 2000.

[128] Philippe Schnoebelen. The complexity of temporal logic model checking. *Advances in Modal Logic*, 4:393–436, 2002.

[129] François Siewe. *A compositional framework for the development of secure access control systems.* PhD thesis, De Montfort University, 2005.

[130] Peter Simons. *Parts : A Study in Ontology.* Clarendon Press, 1987.

[131] Morris Sloman. Policy driven management for distributed systems. *Journal of Network and Systems Management*, 2:333–360, 1994.

[132] Ian Sommerville. *Software Engineering.* Addison-Wesley, Harlow, England, 9 edition, 2010.

[133] Alfred Tarski. Equational logic and equational theories of algebras. In K. Schtte H. Arnold Schmidt and H.-J. Thiele, editors, *Contributions to Mathematical Logic Proceedings of the Logic Colloquium, Hannover 1966*, volume 50 of *Studies in Logic and the Foundations of Mathematics*, pages 275 – 288. Elsevier, 1968.

[134] Sara Uckelman. Lecture notes: Temporal logic, spring 2010. *Lecture Notes: Temporal Logic*, March 25 2010.

[135] Wil van der Aalst. Pi calculus versus petri nets: Let us eat "humble pie" rather than further inflate the "pi hype", 2003.

[136] Moshe Vardi. Branching vs. linear time: Final showdown. In *Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS 2001, pages 1–22, London, UK, UK, 2001. Springer-Verlag.

[137] Yde Venema. Temporal logic. In *The Blackwell Guide to Philosophical Logic. Blackwell Philosophy Guides (2001).* Basil Blackwell Publishers, 1998.

[138] Timothy Allen Wahls. *On the Execution of High Level Formal Specifications.* Iowa State University, 1995.

[139] Gerald James Whitrow. Reflections on the history of the concept of time. In Julius Thomas Fraser, Francis Haber, and Gert Heinz Mller, editors, *The Study of Time*, pages 1–11. Springer Berlin Heidelberg, 1972.

[140] Jeannette Wing. A specifier's introduction to formal methods. *Computer*, 23(9):8–23, September 1990.

[141] Frank Wolter and Michael Zakharyaschev. Temporalizing description logics. Technical report, 1998.

[142] Xinwen Zhang, Francesco Parisi-Presicce, Ravi Sandhu, and Jaehong Park. Formal model and policy specification of usage control. *ACM Trans. Inf. Syst. Secur.*, 8(4):351–387, 2005.

[143] Xinwen Zhang, Jaehong Park, Francesco Parisi-Presicce, and Ravi Sandhu. A logical specification for usage control. In *Proceedings of the ninth ACM symposium on Access control models and technologies*, SACMAT '04, pages 1–10, New York, NY, USA, 2004. ACM.

[144] Xinwen Zhang, Ravi Sandhu, and Francesco Parisi-Presicce. Safety analysis of usage control authorization models. In *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security*, ASIACCS '06, pages 243–254, New York, NY, USA, 2006. ACM.

[145] Lenore Zuck. Past temporal logic. *Ann Arbor*, 1001:48106–1346, 1987.

Appendices

Appendix A

In this appendix, the soundness proof of the propositional axioms and rules of past time ITL^p which are listed in Table 3.13 in Section(3.4.2).

PastChopAssoc

PastChopAssoc $\vdash (h_0 \hat{;} h_1) \hat{;} h_2 \quad \equiv \quad h_0 \hat{;} (h_1 \hat{;} h_2)$

- PastChopAssoc is valid iff

  for all τ. $\mathcal{M}_{\tau}[\![(h_0 \hat{;} h_1) \hat{;} h_2]\!] =$ tt iff$\mathcal{M}_{\tau}[\![h_0 \hat{;} (h_1 \hat{;} h_2)]\!] =$ tt

  at the L.H.S. we have:

- $\mathcal{M}_{\tau}[\![(h_0 \hat{;} h_1) \hat{;} h_2]\!] =$ tt iff exists k where $0 \leq k \leq |\tau|$, s.t.

  $\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_2]\!] =$ tt and$\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![(h_0 \hat{;} h_1)]\!] =$ tt

- $\mathcal{M}_{\tau_\tau \leftarrow \tau_k}[\![(h_0 \hat{;} h_1)]\!] =$ tt iff exist j where $k \leq j \leq |\tau|$, s.t.

  $\mathcal{M}_{\tau_j \leftarrow \tau_k}[\![h_1]\!] =$ tt and$\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_j}[\![h_0]\!] =$ tt and

- so we have:

  $\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_2]\!] =$ tt and $\mathcal{M}_{\tau_j \leftarrow \tau_k}[\![h_1]\!] =$ tt and $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_j}[\![h_0]\!] =$ tt.

  at the R.H.S. we have:

- $\mathcal{M}_{\tau}[\![h_0 \hat{;} (h_1 \hat{;} h_2)]\!] =$ tt iff exists j where $0 \leq j \leq |\tau|$, s.t.

  $\mathcal{M}_{\tau_j \leftarrow \tau_0}[\![(h_1 \hat{;} h_2)]\!] =$ tt and $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_j}[\![h_0]\!] =$ tt and

- $\mathcal{M}_{\tau_j \leftarrow \tau_0}[\![(h_1 \mathbin{\hat{;}} h_2)]\!]=$ tt iff exists k where $\quad 0 \leq k \leq j,$ s.t.

  $\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_2]\!] =$ tt $\quad$ and$\mathcal{M}_{\tau_j \leftarrow \tau_k}[\![h_1]\!] =$ tt

  We have that the R.H.S. is equal to the L.H.S., so we have prove a:

  $\vdash \ (h_0 \mathbin{\hat{;}} h_1) \mathbin{\hat{;}} h_2 \qquad \equiv \qquad h_0 \hat{;} \, (h_1 \mathbin{\hat{;}} h_2)$

PastEmptyChop

PastEmptyChop $\vdash (h \hat{;} \widehat{\mathsf{empty}}) \quad \equiv \quad h$

- PastEmptyChop is valid iff

  for all τ $\mathcal{M}_{\tau}[\![h\hat{;}\,\widehat{\mathsf{empty}}]\!] = \mathrm{tt}$ iff$\mathcal{M}_{\tau}[\![h]\!] = \mathrm{tt}$

- $\mathcal{M}_{\tau}[\![h \,\hat{;}\, \widehat{\mathsf{empty}}]\!] = \mathrm{tt}$ iff exists k :$0 \leq k \leq |\tau|$, s.t.

  $\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![\widehat{\mathsf{empty}}]\!] = \mathrm{tt}$ and$\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h]\!] = \mathrm{tt}$

  we have:

- $\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![\widehat{\mathsf{empty}}]\!] = \mathrm{tt}$

  which is a past interval with only one state, so k $=$ 0 and then

  $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_0}[\![h]\!]$

  which is equal to:

  $\mathcal{M}_{\tau}[\![h]\!]$.

PastChopEmpty

PastChopEmpty $\vdash (\widehat{\mathsf{empty}} \,\hat{;}\, h) \quad \equiv \quad h$

- PastChopEmpty is valid iff

  for all $\tau \mathcal{M}_{\tau}[\![\widehat{\mathsf{empty}} \,\hat{;}\, h]\!] = \text{tt}$ iff$\mathcal{M}_{\tau}[\![h]\!] = \text{tt}$

- $\mathcal{M}_{\tau}[\![\widehat{\mathsf{empty}} \,\hat{;}\, h]\!] = \text{tt}$ iff exists k :$0 \leq k \leq |\tau|$ s.t.

  $\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h]\!] = \text{tt}$ and $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![\widehat{\mathsf{empty}}]\!] = \text{tt}$

  since we have:

- $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![\widehat{\mathsf{empty}}]\!] = \text{tt}$

  which is a past interval with only one state, so k $= |\tau|$:

  and thus:

  $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_0}[\![h]\!]$

  which is:

  $\mathcal{M}_{\tau}[\![h]\!]$.

PastNextImpNotNextNot

PastNextImpNotNextNot $\vdash \quad \widehat{\bigcirc}\, h \supset \neg \widehat{\bigcirc} \neg h$

- we know that: $\quad \widehat{\textcircled{w}} h \mathrel{\widehat{=}} \neg \widehat{\bigcirc} \neg h$

- we need to show:

  for all $\tau \quad \mathcal{M}_{\tau}[\![\widehat{\bigcirc}\, h]\!] = \text{tt} \quad$ implies $\quad \mathcal{M}_{\tau}[\![\neg \widehat{\bigcirc} \neg h]\!] = \text{tt}$

- first we take the L.H.S :

  $\mathcal{M}_{\tau}[\![\widehat{\bigcirc}\, h]\!] = \text{tt}$ iff $\quad 0 < |\tau|$ and $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_1}[\![h]\!] = \text{tt}$

- for the R.H.S. we have:

  $\mathcal{M}_{\tau}[\![\widehat{\textcircled{w}} h]\!] = \text{tt}$ iff $\;|\tau| = 0$ or $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_1}[\![h]\!] = \text{tt}$

- $|\tau| = 0 \;\; or \;\; \mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_1}[\![h]\!] = \text{tt}$ iff$|\tau| = 0 \;\; or \;\; (|\tau| > \;\; 0 \;\; and \;\; \mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_1}[\![h]\!] = \text{tt})$because$|\tau| \; \geq \; 0$
- L.H.S implies R.H.S
- so: $\quad \vdash \quad \widehat{\bigcirc}\, h \supset \neg \widehat{\bigcirc} \neg\, h$

PastOrChopImp

$\vdash \quad h_2 \,\hat{;}\, (h_1 \vee h_0) \supset (h_2 \,\hat{;}\, h_1) \vee (h_2 \,\hat{;}\, h_0)$

- for all $\tau \quad \mathcal{M}_{\tau}[\![h_2 \,\hat{;}\, (h_1 \vee h_0)]\!] = \text{tt}$ implies $\mathcal{M}_{\tau}[\![(h_2 \,\hat{;}\, h_1) \vee (h_2 \,\hat{;}\, h_0)]\!] = \text{tt}$

- On the L.H.S. we have: $\mathcal{M}_{\tau}[\![h_2 \,\hat{;}\, (h_1 \vee h_0)]\!] = tt$

- $\mathcal{M}_{\tau}[\![h_2 \,\hat{;}\, (h_1 \vee h_0)]\!] =$ tt iff exists k: $0 \leq k \leq |\tau|$, s.t.

  $\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_1 \vee h_0]\!] = \text{tt}$ and $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_2]\!] = \text{tt}$

- $\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_1 \vee h_0]\!] = \text{tt}$ iff

  $\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_1]\!] = \text{tt}$ or$\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_0]\!] = \text{tt}$

- so we have: exsits k: $0 \leq k \leq |\tau|$ s.t.

  $(\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_1]\!] = \text{tt}$ or$\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_0]\!] = \text{tt})$ and $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_2]\!] = \text{tt}$

- 0n the R.H.S. we have: $\mathcal{M}_{\tau}[\![(h_2 \,\hat{;}\, h_1) \quad \vee \quad (h_2 \,\hat{;}\, h_0)]\!] = tt$
  which is :$\mathcal{M}_{\tau}[\![h_2 \,\hat{;}\, h_1]\!] = tt or \mathcal{M}_{\tau}[\![h_2 \,\hat{;}\, h_0]\!] = tt$

- $\mathcal{M}_{\tau}[\![h_2 \,\hat{;}\, h_0]\!] = \text{tt}$ iff exists k :$0 \leq k \leq |\tau|, s.t.$

  $\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_0]\!] = \text{tt}$ and $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_2]\!] = \text{tt}$

- $\mathcal{M}_{\tau}[\![h_2 \,\hat{;}\, h_1]\!] = \text{tt}$ iff exists k : $0 \leq k \leq |\tau|, s.t.$

  $\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_1]\!] = \text{tt}$ and $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_2]\!] = \text{tt}$

- so we have:
  exists k :$0 \leq k \leq |\tau|, s.t.$
  $\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_0]\!] = \text{tt}$ and $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_2]\!] = \text{tt}$

  or
  exists k :$0 \leq k \leq |\tau|, s.t.$
  $\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_1]\!] = \text{tt}$ and$\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_2]\!] = \text{tt}$

- so we have:
  exists k :$0 \leq k \leq |\tau|, s.t.$
  $\Big((\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_0]\!] = \text{tt or}\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_1]\!] = \text{tt})\, and \mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_2]\!] = \text{tt}\Big)$

  which is equal to
  exists k :$0 \leq k \leq |\tau|, s.t.$
  $\Big((\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_0]\!] = \text{tt and}\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_2]\!] = \text{tt}) or (\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_1]\!] = \text{tt and}\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_2]\!] = \text{tt})\Big)$

  which implies
  $\Big($ exists k :$0 \leq k \leq |\tau|, s.t. \mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_0]\!] = \text{tt and}\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_2]\!] = \text{tt}\Big)$
  or
  $\Big($ exists k :$0 \leq k \leq |\tau|, s.t. \mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_1]\!] = \text{tt and}\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_2]\!] = \text{tt}\Big)$
  which is the R.H.S.

PastChopOrImp

PastChopOrImp $\quad\vdash\quad (h_2 \vee h_1) \overset{\frown}{;} h_0 \supset (h_1 \overset{\frown}{;} h_0) \vee (h_2 \overset{\frown}{;} h_0)$

- for all τ $\quad \mathcal{M}_{\tau}[\![(h_2 \vee h_1) \overset{\frown}{;} h_0]\!] = \text{tt}$ implies $\mathcal{M}_{\tau}[\![(h_1 \overset{\frown}{;} h_0) \vee (h_2 \overset{\frown}{;} h_0)]\!] = \text{tt}$

- On the L.H.S. we have: $\quad \mathcal{M}_{\tau}[\![(h_2 \vee h_1) \overset{\frown}{;} h_0]\!] = \text{tt}$

- $\mathcal{M}_{\tau}[\![(h_2 \vee h_1) \overset{\frown}{;} h_0]\!] = \text{tt}$ iff exists k : $\quad 0 \leq k \leq |\tau|$, s.t.

  $\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_0]\!]$= tt and $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_2 \vee h_1]\!]$= tt

- $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_2 \vee h_1]\!]$= tt iff

  $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_2]\!]$= tt or $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_1]\!]$= tt.

- so we have: exists k :$0 \leq k \leq |\tau|$, s.t

  $\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_0]\!] = \text{tt}$ and $\Big(\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_2]\!] = \text{tt}$ or $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_1]\!] = \text{tt.}\Big)$

- now take the R.H.S., we have:

- $\mathcal{M}_{\tau}[\![(h_1 \overset{\frown}{;} h_0) \vee (h_2 \overset{\frown}{;} h_0)]\!] = \text{tt}$ iff

  $\mathcal{M}_{\tau}[\![h_1 \overset{\frown}{;} h_0]\!] = \text{tt}$ or $\mathcal{M}_{\tau}[\![h_2 \overset{\frown}{;} h_0]\!] = \text{tt.}$

- exists k: $0 \leq k \leq |\tau|$, s.t
  $\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_0]\!] = \text{tt}$ and $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_1]\!] = \text{tt}$
  or exists k: $0 \leq k \leq |\tau|$, s.t
  $\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_0]\!]$= tt and $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_2]\!]$= tt

- so we have for the L.H.S.:
  exists k: $0 \leq k \leq |\tau|$, s.t
  $\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_0]\!]=$ tt and $\Big(\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_2]\!] =$ tt or $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_1]\!] =$ tt$\Big)$

  which is equal to
  exists k :$0 \leq k \leq |\tau|$, s.t

  $\Big(\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_0]\!] =$ tt and$\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_2]\!] =$ tt$\Big)$
  or
  $\Big(\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_0]\!] =$ tt and$\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_1]\!] =$ tt$\Big)$

  which implies

  exists k :$0 \leq k \leq |\tau|$, s.t
  $\Big(\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_0]\!] =$ tt and$\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_2]\!] =$ tt$\Big)$

  or
  exists k :$0 \leq k \leq |\tau|$, s.t $\Big(\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_0]\!] =$ tt and$\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_1]\!] =$ tt$\Big)$

  which is the R.H.S.

PastStateImpBi

PastStateImpBi $\vdash \quad P \supset \widehat{\boxed{\text{i}}}\, P$

- for all $\tau \quad \mathcal{M}_{\tau}[\![P \supset \widehat{\boxed{\text{i}}}\, P]\!] = \text{tt}$.

- $\mathcal{M}_{\tau}[\![P \supset \widehat{\boxed{\text{i}}}\, P]\!] = \text{tt}$ iff

  $\mathcal{M}_{\tau}[\![P]\!] = \text{tt}$ implies $\mathcal{M}_{\tau}[\![\widehat{\boxed{\text{i}}}\, P]\!] = \text{tt}$

- $\mathcal{M}_{\tau}[\![P]\!] = \text{tt}$ iff $\mathcal{M}_{\tau_0}[\![P]\!] = \text{tt}$ because P is state formula.

- $\mathcal{M}_{\tau}[\![\widehat{\boxed{\text{i}}}\, P]\!] = \text{tt}$ iff for all k : $0 \leq k \leq |\tau|, s.t.$

  $\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![P]\!] = tt.$

- (for all $k : 0 \leq k \leq |\tau|, s.t \mathcal{M}_{\tau_k \leftarrow \tau_0}[\![P]\!] = tt$) iff $\mathcal{M}_{\tau_0}[\![P]\!] = \text{tt}$ because P is state formula

  which is the L.H.S., so

  $\vdash \quad P \supset \widehat{\boxed{\text{i}}}\, P$

PastBiGen

PastBiGen $\quad \vdash \quad h_0 \;\; implies \quad \vdash \quad \widehat{\boxed{\text{i}}}\, h_0$

- $(for\ all\ \tau \quad \mathcal{M}_{\tau}[\![h_0]\!] = \text{tt})$ implies $(for\ all\ \ \tau \quad \mathcal{M}_{\tau}[\![\widehat{\boxed{\text{i}}}\, h_0]\!] = \text{tt}\ )$

- take the R.H.S., we have:

  $for\ all\ \tau, \quad \mathcal{M}_{\tau}[\![\widehat{\boxed{\text{i}}}\, h_0]\!] = \text{tt iff}$

  $for\ all\ \tau, \quad \text{k} : 0 \leq k \leq |\tau| \mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_0]\!] = tt.$

- but we know from the L.H.S. that:

  $for\ all\ \ \tau, \quad \mathcal{M}_{\tau}[\![h_0]\!] = \text{tt}.$

- Instantiate in L.H.S. for $\tau, \tau_k \leftarrow \tau_0$ so we have:

  $\vdash \quad h_0 \Rightarrow \ \vdash \quad \widehat{\boxed{\text{i}}}\, h_0$

PastBoxGen

PastBoxGen $\vdash \quad h_0$ *implies* $\vdash \quad \widehat{\Box}\, h_0$

- $(\mathit{for\ all}\ \ \tau,\ \ \mathcal{M}_{\tau}[\![h_0]\!] = \mathrm{tt})$ implies $(\mathit{for\ all}\ \ \tau,\ \ \mathcal{M}_{\tau}[\![\widehat{\Box}\, h_0]\!] = \mathrm{tt}\ )$

- take the R.H.S., we have:

  $\mathit{for\ all}\ \ \tau,\ \ \mathcal{M}_{\tau}[\![\widehat{\Box}\, h_0]\!] = \mathrm{tt}$ iff

  $\mathit{for\ all}\ \ \tau$,k : $0 \leq k \leq |\tau|,\ \mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_0]\!] = \mathrm{tt}$

- but we know from the L.H.S. that:

  $\mathit{for\ all}\ \ \tau,\ \ \mathcal{M}_{\tau}[\![h_0]\!] = \mathrm{tt}.$

- Instantiate in L.H.S. for $\tau, \tau_\tau \leftarrow \tau_k$ so we have:

  $\vdash \quad h_0 \Rightarrow\ \vdash \quad \widehat{\Box}\, h_0$

PastChopStarEqv

PastChopStarEqv $\vdash \quad h_0^{\hat{*}} \equiv ((h_0^{\hat{*}} \hat{;} (h_0 \wedge \widehat{\mathsf{more}})) \vee \widehat{\mathsf{empty}})$

PastChopStarEqv is valid iff

for all τ, $\mathcal{M}_{\tau}[\![h_0^{\hat{*}}]\!] = \text{tt}$ iff $\mathcal{M}_{\tau}[\![(h_0^{\hat{*}} \hat{;} (h_0 \wedge \widehat{\mathsf{more}})) \vee \widehat{\mathsf{empty}}]\!] = \text{tt}$.

- $\mathcal{M}_{\tau}[\![(h_0^{\hat{*}} \hat{;} (h_0 \wedge \widehat{\mathsf{more}})) \vee \widehat{\mathsf{empty}}]\!] = \text{tt}$ iff

  $\mathcal{M}_{\tau}[\![{h_0}^{\hat{*}} \hat{;}(h_0 \wedge \widehat{\mathsf{more}})]\!] = \text{tt}$

  or $\mathcal{M}_{\tau}[\![\widehat{\mathsf{empty}}]\!] = \text{tt}$

- $\mathcal{M}_{\tau}[\![\widehat{\mathsf{empty}}]\!] = \text{tt}$ iff $|\tau| = 0$

- $\mathcal{M}_{\tau}[\![h_0^{\hat{*}} \hat{;} (h_0 \wedge \widehat{\mathsf{more}})]\!] = \text{tt}$ iff exists k : $0 \leq k \leq |\tau|$, s.t.

  $\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![(h_0 \wedge \widehat{\mathsf{more}})]\!] = \text{tt}$ and $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_0^{\hat{*}}]\!] = \text{tt}$

- $\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![(h_0 \wedge \widehat{\mathsf{more}})]\!] = \text{tt}$ iff

  $\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_0]\!] = \text{tt}$ *and* $\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![\widehat{\mathsf{more}}]\!] = \text{tt}$

- $\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![\widehat{\mathsf{more}}]\!] = \text{tt}$ iff $k \geq 1$

- $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_0^{\hat{*}}]\!] = \text{tt}$

  exists l_0,....l_n, such that $l_0 = 0$ and $l_n = |\tau|$ and

  for all l 0 $\leq i < n$, $l_i \leq l_{i+1}$ and $\mathcal{M}_{\tau_{l_i} \rightarrow \tau_{l_{i+1}}}[\![h_0]\!] \ldots = \text{tt}$

  PastChopStarEqv $\vdash \quad h_0^{\hat{*}} \equiv ((h_0^{\hat{*}} \hat{;} (h_0 \wedge \widehat{\mathsf{more}})) \vee \widehat{\mathsf{empty}})$

PastBiBoxChopImpChop

PastBiBoxChopImpChop $\vdash$ $\widehat{\boxed{\text{i}}}(h_0 \supset h_1) \wedge \widehat{\Box}(h_2 \supset h_3) \supset (h_2 \,\hat{;}\, h_0) \supset (h_3 \,\hat{;}\, h_1)$

- PastBiBoxChopImpChop is valid iff
- $for\ all\ \tau,\ \ \mathcal{M}_{\tau}[\![\widehat{\boxed{\text{i}}}(h_0 \supset h_1) \wedge \widehat{\Box}(h_2 \supset h_3)]\!] = \text{tt}$

  implies

  $(\mathcal{M}_{\tau}[\![h_2 \,\hat{;}\, h_0]\!] = \text{tt implies } \mathcal{M}_{\tau}[\![h_3 \,\hat{;}\, h_1]\!] = \text{tt})$

- take the L.H.S., we have:

  $\mathcal{M}_{\tau}[\![\widehat{\boxed{\text{i}}}(h_0 \supset h_1) \wedge \widehat{\Box}(h_2 \supset h_3)]\!] = \text{tt iff}$

  $\mathcal{M}_{\tau}[\![\widehat{\boxed{\text{i}}}(h_0 \supset h_1)]\!] = \text{tt and } \mathcal{M}_{\tau}[\![\widehat{\Box}(h_2 \supset h_3)]\!] = \text{tt.}$

- $\mathcal{M}_{\tau}[\![\widehat{\boxed{\text{i}}}(h_0 \supset h_1)]\!] = \text{tt iff}$

  $for\ all\ k\ :\ 0 \le k \le |\tau|, \mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_0 \supset h_1]\!] = tt$
- $\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_0 \supset h_1]\!] = tt\ iff \mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_0]\!] = tt$ implies $\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_1]\!] = tt$

- for the second part:

  $\mathcal{M}_{\tau}[\![\widehat{\Box}(h_2 \supset h_3)]\!] = \text{tt iff}$

  $for\ all\ k\ :\ 0 \le k \le |\tau|, \mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_2 \supset h_3]\!] = \text{tt}$

- $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_2 \supset h_3]\!] = \text{tt iff}$

$\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_2]\!] = \text{tt}$ implies $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_3]\!] = \text{tt}$.

- so, we have:
  $for\ all\ k\ :\ 0 \leq k \leq |\tau|$, if $\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_0]\!] = tt$ then $\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_1]\!] = tt.$

  and
  $for\ all\ k\ :\ 0 \leq k \leq |\tau|$, if $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_2]\!] = \text{tt}$ then $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_3]\!] = \text{tt}$.

- for the R.H.S. we have:

  $\mathcal{M}_{\tau}[\![h_2 \,\hat{;}\, h_0]\!] = \text{tt}\ \ implies\ \ \mathcal{M}_{\tau}[\![h_3 \,\hat{;}\, h_1]\!] = \text{tt}$

- take the first part, we have:

  $\mathcal{M}_{\tau}[\![h_2 \,\hat{;}\, h_0]\!] = \text{tt}$ iff exists k : $0 \leq k \leq |\tau, |$

  $\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_0]\!] = tt$ and $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_2]\!]$

- take the second part, we have:

  $\mathcal{M}_{\tau}[\![h_3 \,\hat{;}\, h_1]\!] = \text{tt}$ iff exists k : $0 \leq k \leq |\tau|$,

  $\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_1]\!] = tt$ and $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_3]\!]$

- so on the R.H.S. we have:

  $exists\ k\ :\ 0 \leq k \leq |\tau|, (\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_0]\!] = tt$ and $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_2]\!])$ implies
  $exists\ k\ :\ 0 \leq k \leq |\tau|, (\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_1]\!] = tt$ and $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_3]\!])$

- we know that from L.H.S.

$\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_0]\!] = tt$ implies $\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![h_1]\!] = tt$

and

$\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_2]\!] = tt$ implies $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_3]\!] = tt$

So the R.H.S. holds and therefore
$\vdash \qquad \widehat{\boxed{\text{i}}}(h_0 \supset h_1) \wedge \widehat{\Box}(h_2 \supset h_3) \supset (h_2 \,\hat{;}\, h_0) \supset (h_3 \,\hat{;}\, h_1)$

PastBoxInduct

PastBoxInduct $\vdash \quad h_0 \wedge \widehat{\Box}(h_0 \supset \widehat{\textcircled{w}} h_0) \supset \widehat{\Box}\, h_0$

PastBoxInduct is valid iff

$for\ all \quad \tau \quad \mathcal{M}_{\tau}[\![h_0 \wedge \widehat{\Box}(h_0 \supset \widehat{\textcircled{w}} h_0)]\!] = \mathrm{tt}$ implies $\mathcal{M}_{\tau}[\![\widehat{\Box}\, h_0]\!] = \mathrm{tt}$

- take the L.H.S., we have:

  $\mathcal{M}_{\tau}[\![h_0 \wedge \widehat{\Box}(h_0 \supset \widehat{\textcircled{w}} h_0)]\!] = \mathrm{tt}$ iff

  $\mathcal{M}_{\tau}[\![h_0]\!] = \mathrm{tt}$ and $\mathcal{M}_{\tau}[\![\widehat{\Box}(h_0 \supset \widehat{\textcircled{w}} h_0)]\!] = \mathrm{tt}$

  $for\ all \quad \tau \quad \mathcal{M}_{\tau}[\![h_0]\!] = \mathrm{tt}$ is given (*)

- $\mathcal{M}_{\tau}[\![\widehat{\Box}(h_0 \supset \widehat{\textcircled{w}} h_0)]\!] = \mathrm{tt}$ iff

  for all k : $0 \leq k \leq |\tau|$, $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_0 \supset \widehat{\textcircled{w}} h_0]\!] = \mathrm{tt}$.

- $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_0 \supset \widehat{\textcircled{w}} h_0]\!] = \mathrm{tt}$ iff

  $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_0]\!] = \mathrm{tt}$ implies $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![\widehat{\textcircled{w}} h_0]\!] = \mathrm{tt}$

- $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![\widehat{\textcircled{w}} h_0]\!] = \mathrm{tt}$ iff

  $k = |\tau| \mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![=]\!]0$ or $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_{k+1}}[\![h_0]\!] = \mathrm{tt}$

  so we have for L.H.S.

  $\mathcal{M}_{\tau}[\![h_0]\!] = tt$ and for all k,$0 \leq k \leq |\tau|$,

  $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h_0]\!] = tt$ implies

$k \;=\; |\tau| or \mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_{k+1}} [\![h_0]\!] = \text{tt}$

$\mathcal{M}_{\tau} [\![h_0]\!] = tt\ and$

$k = 0\;\;:$

- $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_0} [\![h_0]\!] = tt$ implies

  $0 = |\tau|\ or\ \mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_1} [\![h_0]\!] = tt$

$$0 = |\tau|\ or\ \mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_1} [\![h_0]\!] = tt$$

$k = 1\;\;:$

- $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_1} [\![h_0]\!] = tt$ implies

  $1 = |\tau|\ or\ \mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_2} [\![h_0]\!] = tt$

$$1 = |\tau|\ or\ \mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_2} [\![h_0]\!] = tt$$

.

.

.

.

.

.

$k = |\tau|\;\;:$

- $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k} [\![h_0]\!] = tt$ implies

$$|\tau| = |\tau|\ or\ \mathcal{M}_{\tau_{|\tau|} \leftarrow |\tau|+1} [\![h_0]\!] = tt$$

so we have

$\mathcal{M}_{\tau} [\![h_0]\!] = \text{tt and}$

for k

- $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k} [\![h_0]\!] = tt$ which is the R.H.S.

$$1 = |\tau| \textit{ or } \mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_2}[\![h_0]\!] = tt$$

$$\vdash \qquad h_0 \wedge \widehat{\Box}(h_0 \supset \widehat{\text{ⓦ}} h_0) \supset \widehat{\Box}\, h_0$$

PastChopEmptyAnd

PastChopEmptyAnd $\vdash h \wedge w_0 \quad \equiv \quad h \,\hat{;}\, (\widehat{\mathsf{empty}} \wedge w_0)$

- To prove this formula is sound, take the R.H.S. we have:
  $\mathcal{M}_{\tau}[\![h \,\hat{;}\, (\widehat{\mathsf{empty}} \wedge w_0)]\!] = \text{tt}$ iff exists k where
  $0 \leq k \leq |\tau|$, s.t.

- $\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![\widehat{\mathsf{empty}} \wedge w_0]\!] = \text{tt}$ and
  $\mathcal{M}_{\tau_{|\tau|} \leftarrow \tau_k}[\![h)]\!] = \text{tt}$
- $\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![\widehat{\mathsf{empty}} \wedge w_0]\!] = \text{tt}$ iff
  $\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![\widehat{\mathsf{empty}}]\!] = \text{tt}$ and $\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![w_0]\!] = \text{tt}$
- $\mathcal{M}_{\tau_k \leftarrow \tau_0}[\![\widehat{\mathsf{empty}}]\!] = \text{tt}$ which is past interval with only one state, so $k = 0$
- from the past empty , we know that $k = 0$, so $\mathcal{M}_{\tau_{\tau_0}}[\![w_0]\!] = \text{tt}$
- because w_0 is state formula, we can write:
  $\mathcal{M}_{\tau}[\![h]\!] = \text{tt}$ so in R.H.S. we have:
  $\mathcal{M}_{\tau}[\![h]\!] = \text{tt}$ and $\mathcal{M}_{\tau_{\tau_0}}[\![w_0]\!] = \text{tt}$
- for the L.H.S. we have $\mathcal{M}_{\tau}[\![h \wedge w_0]\!] = \text{tt}$
- we know that w_0 is state formula, so we can write the L.H.S. as:
  $\mathcal{M}_{\tau}[\![h]\!] = \text{tt}$ and $\mathcal{M}_{\tau_{\tau_0}}[\![w_0]\!] = \text{tt}$ which the R.H.S.
- **PastChopEmptyAnd** $\vdash h \wedge w_0 \quad \equiv \quad h \,\hat{;}\, (\widehat{\mathsf{empty}} \wedge w_0)$

Appendix B Part 1:

In this appendix, the complete semantics of general practice system policy GPS for the Patient and the Doctor policies are listed here.

In the first part we will give the complete semantics of Patient policy which are from Rule $R_6^{'}$ to $R_{11}^{'}$.

The semantics of Patient policy

$[\![R_6 \ \wedge \ R_{12}]\!] \equiv [\![R_6^{'}]\!] \equiv$

$$\bigwedge_{\substack{s \in Subjects \\ o \in Objects \\ a \in Actions}} \left(\begin{array}{l} (f_{PATIENT}(s,o,a) \leftrightarrow Aut^{+}(s,o,a)) \wedge \\ (g_{PATIENT}(s,o,a) \leftrightarrow Aut^{-}(s,o,a)) \wedge \\ (h_{PATIENT}(s,o,a) \leftrightarrow Aut(s,o,a)) \end{array} \right)$$

where $f_{PATIENT}(s,o,a)$ is defined as:

$f_{PATIENT}(s,o,a)$	(s,o,a)	R
$[\![\ \textbf{\textit{true chop}}^{\textbf{\textit{p}}} \ \textbf{\textit{always}}^{\textbf{\textit{p}}}(registered(S_{PATIENT})) \ ]\!]$	$(S_{PATIENT}$,GPS$, access)$	6

and $g_{PATIENT}(s,o,a)$ is defined as:

$g_{PATIENT}(s,o,a)$	(s,o,a)	R
$[\![$ **false** $]\!]$	$(S_{PATIENT}$,GPS$, access)$	6

and $h_{PATIENT}(s,o,a)$ is defined as:

$h_{PATIENT}(s,o,a)$	(s,o,a)	R
$[\![0: \ allow(S_{PATIENT}$,GPS$, access)$ **and not** $deny(S_{PATIENT}$,GPS$, access)]\!]$	$(S_{PATIENT}$,GPS$, access)$	12

and $[\![\ R_7\ \wedge\ R_{12}\ ]\!] \equiv [\![\ R_7{'}\ ]\!] \equiv$

$$\bigwedge_{\substack{s \in Subjects \\ o \in Objects \\ a \in Actions}} \left(\begin{array}{l} (f_{PATIENT}(s,o,a) \leftrightarrow Aut^{+}(s,o,a))\wedge \\ (g_{PATIENT}(s,o,a) \leftrightarrow Aut^{-}(s,o,a))\wedge \\ (h_{PATIENT}(s,o,a) \leftrightarrow Aut(s,o,a)) \end{array} \right)$$

where $f_{PATIENT}(s,o,a)$ is defined as:

$f_{PATIENT}(s,o,a)$	(s,o,a)	R
$[\![0:\ \ patient\_info(O_{patient\_nfo}, S_{PATIENT})]\!]$	$(S_{PATIENT}$,GPS$, update(O_{patient_info}))$	7

and $g_{PATIENT}(s,o,a)$ is defined as:

$g_{PATIENT}(s,o,a)$	(s,o,a)	R
$[\![$ **false** $]\!]$	$(S_{PATIENT}$,GPS$, update(O_{patient_info}))$	7

and $h_{PATIENT}(s,o,a)$ is defined as:

$h_{PATIENT}(s,o,a)$	(s,o,a)	R
$[\![0:\ \ allow(S_{PATIENT}$,GPS$, update(O_{patient_info}))$ **and not** $deny(S_{PATIENT}$,GPS$, update(O_{patient_info}))]\!]$	$(S_{PATIENT}$,GPS$, update(O_{patient_info}))$	12

and $[\![\ R_8 \ \wedge \ R_{12} \ ]\!] \equiv [\![\ R_8' \ ]\!] \equiv$

$$\bigwedge_{\substack{s \in Subjects \\ o \in Objects \\ a \in Actions}} \begin{pmatrix} (f_{PATIENT}(s,o,a) \leftrightarrow Aut^{+}(s,o,a)) \wedge \\ (g_{PATIENT}(s,o,a) \leftrightarrow Aut^{-}(s,o,a)) \wedge \\ (h_{PATIENT}(s,o,a) \leftrightarrow Aut(s,o,a)) \end{pmatrix}$$

where $f_{PATIENT}(s,o,a)$ is defined as:

$f_{PATIENT}(s,o,a)$	(s,o,a)	R
$[\![$ **false** $]\!]$	$(S_{PATIENT}$,GPS, $alter(O_{medical\_records}))$	8

and $g_{PATIENT}(s,o,a)$ is defined as:

$g_{PATIENT}(s,o,a)$	(s,o,a)	R
$[\![$ **true** $]\!]$	$(S_{PATIENT}$,GPS, $alter(O_{medical\_records}))$	8

and $h_{PATIENT}(s,o,a)$ is defined as:

$h_{PATIENT}(s,o,a)$	(s,o,a)	R
$[\![0:$ $deny(S_{PATIENT}$,GPS, $alter(O_{medical\_records}))$ **and not** $allow(S_{PATIENT}$,GPS, $alter(O_{medical\_records}))]\!]$	$(S_{PATIENT}$,GPS, $alter(O_{medical\_records}))$	12

and $\llbracket\ R_9\ \wedge\ R_{12}\ \rrbracket \equiv \llbracket\ R_9^{'}\ \rrbracket \equiv$

$$\bigwedge_{\substack{s \in Subjects \\ o \in Objects \\ a \in Actions}} \left(\begin{array}{l} (f_{PATIENT}(s,o,a) \leftrightarrow Aut^{+}(s,o,a)) \wedge \\ (g_{PATIENT}(s,o,a) \leftrightarrow Aut^{-}(s,o,a)) \wedge \\ (h_{PATIENT}(s,o,a) \leftrightarrow Aut(s,o,a)) \end{array} \right)$$

where $f_{PATIENT}(s,o,a)$ is defined as:

$f_{PATIENT}(s,o,a)$	(s,o,a)	R
$\llbracket$ ***true chop***$^{\boldsymbol{p}}$ ***always***$^{\boldsymbol{p}}$($sign(S_{PATIENT}, O_{LA})$ **and** $sign(S_{PATIENT}, O_{CF})$) $\rrbracket$	$(S_{PATIENT}$,GPS, $treat(S_{PATIENT}))$	9

and $g_{PATIENT}(s,o,a)$ is defined as:

$g_{PATIENT}(s,o,a)$	(s,o,a)	R
$\llbracket$ **false** $\rrbracket$	$(S_{PATIENT}$,GPS, $treat(S_{PATIENT}))$	9

and $h_{PATIENT}(s,o,a)$ is defined as:

$h_{PATIENT}(s,o,a)$	(s,o,a)	R
$\llbracket 0:$ $allow(S_{PATIENT}$,GPS, $treat(S_{PATIENT}))$ **and not** $deny(S_{PATIENT}$,GPS, $treat(S_{PATIENT}))\rrbracket$	$(S_{PATIENT}$,GPS, $treat(S_{PATIENT}))$	12

and $[\![\; R_{10} \; \wedge \; R_{12} \;]\!] \equiv [\![\; R_{10}^{'} \;]\!] \equiv$

$$\bigwedge_{\substack{s \in Subjects \\ o \in Objects \\ a \in Actions}} \begin{pmatrix} (f_{PATIENT}(s,o,a) \leftrightarrow Aut^{+}(s,o,a)) \wedge \\ (g_{PATIENT}(s,o,a) \leftrightarrow Aut^{-}(s,o,a)) \wedge \\ (h_{PATIENT}(s,o,a) \leftrightarrow Aut(s,o,a)) \end{pmatrix}$$

where $f_{PATIENT}(s,o,a)$ is defined as:

$f_{PATIENT}(s,o,a)$	(s,o,a)	R
$[\![$ ***true chop^{p} always^{p}*** **not** *booking*$(S_{PATIENT}, O_{APPOINTMENT})$ ***chop skip^{p}*** $]\!]$	$(S_{PATIENT}$,GPS, *book*$(O_{APPOINTMENT}))$	10

and $g_{PATIENT}(s,o,a)$ is defined as:

$g_{PATIENT}(s,o,a)$	(s,o,a)	R
$[\![$ **false**$]\!]$	$(S_{PATIENT}$,GPS, *book*$(O_{APPOINTMENT}))$	10

and $h_{PATIENT}(s,o,a)$ is defined as:

$h_{PATIENT}(s,o,a)$	(s,o,a)	R
$[\![0:$ *allow*$(S_{PATIENT}$,GPS, *book*$(O_{APPOINTMENT}))$ **and not** *deny*$(S_{PATIENT}$,GPS, *book*$(O_{APPOINTMENT}))]\!]$	$(S_{PATIENT}$,GPS, *book*$(O_{APPOINTMENT}))$	12

and $\llbracket\ R_{11}\ \wedge\ R_{12}\ \rrbracket \equiv \llbracket\ R'_{11}\ \rrbracket \equiv$

$$\bigwedge_{\substack{s \in Subjects \\ o \in Objects \\ a \in Actions}} \left(\begin{array}{l} (f_{PATIENT}(s,o,a) \leftrightarrow Aut^{+}(s,o,a)) \wedge \\ (g_{PATIENT}(s,o,a) \leftrightarrow Aut^{-}(s,o,a)) \wedge \\ (h_{PATIENT}(s,o,a) \leftrightarrow Aut(s,o,a)) \end{array} \right)$$

where $f_{PATIENT}(s,o,a)$ is defined as:

$f_{PATIENT}(s,o,a)$	(s,o,a)	R
$\llbracket\ 0 :\ \ medical\_record(O_{medical\_record}, S_{PATIENT})\ \rrbracket$	$(S_{PATIENT}$,GPS, $view(O_{medical\_record}))$	11

and $g_{PATIENT}(s,o,a)$ is defined as:

$g_{PATIENT}(s,o,a)$	(s,o,a)	R
$\llbracket$ **false** $\rrbracket$	$(S_{PATIENT}$,GPS, $view(O_{medical\_record}))$	11

and $h_{PATIENT}(s,o,a)$ is defined as:

$h_{PATIENT}(s,o,a)$	(s,o,a)	R
$\llbracket 0 :\ \ allow(S_{PATIENT}$,GPS, $view(O_{medical\_record}))$ **and not** $deny(S_{PATIENT}$,GPS, $view(O_{medical\_record}))\rrbracket$	$(S_{PATIENT}$,GPS, $view(O_{medical\_record}))$	12

Appendix B Part 2:

In this part we will list the semantics of Doctor policy which is a part from the GPS policy.

The Doctor policy consists from the rules from rule R'_{13} to rule R'_{17}.

The semantics of Doctor policy

$$[\![R_{13} \wedge R_{18}]\!] \equiv [\![R'_{13}]\!] \equiv$$

$$\bigwedge_{\substack{s \in Subjects \\ o \in Objects \\ a \in Actions}} \left(\begin{array}{l} (f_{DOCTOR}(s,o,a) \leftrightarrow Aut^{+}(s,o,a)) \wedge \\ (g_{DOCTOR}(s,o,a) \leftrightarrow Aut^{-}(s,o,a)) \wedge \\ (h_{DOCTOR}(s,o,a) \leftrightarrow Aut(s,o,a)) \end{array} \right)$$

where $f_{DOCTOR}(s,o,a)$ is defined as:

$f_{DOCTOR}(s,o,a)$	(s,o,a)	R
$[\![$ $\boldsymbol{true}$ $\boldsymbol{chop^p}$ $\boldsymbol{always^p}$ $(registered(S_{DOCTOR}))$ $]\!]$	$(S_{DOCTOR}$,GPS$, access)$	13

and $g_{DOCTOR}(s,o,a)$ is defined as:

$g_{DOCTOR}(s,o,a)$	(s,o,a)	R
$[\![$ **false** $]\!]$	$(S_{DOCTOR}$,GPS$, access)$	13

and $h_{DOCTOR}(s,o,a)$ is defined as:

$h_{DOCTOR}(s,o,a)$	(s,o,a)	R
$[\![0:$ $allow(S_{DOCTOR}$,GPS$, access)$ **and not** $deny(S_{DOCTOR}$,GPS$, access)]\!]$	$(S_{DOCTOR}$,GPS$, access)$	18

and $[\![\; R_{14} \; \wedge \; R_{18} \;]\!] \equiv [\![\; R'_{14} \;]\!] \equiv$

$$\bigwedge_{\substack{s \in Subjects \\ o \in Objects \\ a \in Actions}} \left(\begin{array}{l} (f_{DOCTOR}(s,o,a) \leftrightarrow Aut^{+}(s,o,a)) \wedge \\ (g_{DOCTOR}(s,o,a) \leftrightarrow Aut^{-}(s,o,a)) \wedge \\ (h_{DOCTOR}(s,o,a) \leftrightarrow Aut(s,o,a)) \end{array} \right)$$

where $f_{DOCTOR}(s,o,a)$ is defined as:

$f_{DOCTOR}(s,o,a)$	(s,o,a)	R
$[\![0: \; doctor(S_{DOCTOR}, S_{PATIENT})$ **and** $medical\_record(O_{medical\_record}, S_{PATIENT})]\!]$	$(S_{DOCTOR}$,GPS, $alter(O_{medical\_record}))$	14

and $g_{DOCTOR}(s,o,a)$ is defined as:

$g_{DOCTOR}(s,o,a)$	(s,o,a)	R
$[\![$ **false** $]\!]$	$(S_{DOCTOR}$,GPS, $alter(O_{medical\_record}))$	14

and $h_{DOCTOR}(s,o,a)$ is defined as:

$h_{DOCTOR}(s,o,a)$	(s,o,a)	R
$[\![0: \; allow(S_{DOCTOR}$,GPS, $alter(O_{medical\_record}))$ **and not** $deny(S_{DOCTOR}$,GPS, $alter(O_{medical\_record}))]\!]$	$(S_{DOCTOR}$,GPS, $alter(O_{medical\_record}))$	18

and $[\![\; R_{15} \; \wedge \; R_{18} \;]\!] \equiv [\![\; R'_{15} \;]\!] \equiv$

$$\bigwedge_{\substack{s \in Subjects \\ o \in Objects \\ a \in Actions}} \left(\begin{array}{l} (f_{DOCTOR}(s,o,a) \leftrightarrow Aut^{+}(s,o,a)) \wedge \\ (g_{DOCTOR}(s,o,a) \leftrightarrow Aut^{-}(s,o,a)) \wedge \\ (h_{DOCTOR}(s,o,a) \leftrightarrow Aut(s,o,a)) \end{array} \right)$$

where $f_{DOCTOR}(s,o,a)$ is defined as:

$f_{DOCTOR}(s,o,a)$	(s,o,a)	R
$[\![0:$ $doctor(S_{DOCTOR}, S_{PATIENT})$ **and** $private\_notes(O_{private\_notes}, S_{PATIENT})]\!]$	$(S_{DOCTOR}$,GPS, $add(O_{private\_notes}))$	15

and $g_{DOCTOR}(s,o,a)$ is defined as:

$g_{DOCTOR}(s,o,a)$	(s,o,a)	R
$[\![$ **false** $]\!]$	$(S_{DOCTOR}$,GPS, $add(O_{private\_notes}))$	15

and $h_{DOCTOR}(s,o,a)$ is defined as:

$h_{DOCTOR}(s,o,a)$	(s,o,a)	R
$[\![0:$ $allow(S_{DOCTOR}$,GPS, $add(O_{private\_notes}))$ **and not** $deny(S_{DOCTOR}$,GPS, $add(O_{private\_notes}))]\!]$	$(S_{DOCTOR}$,GPS, $add(O_{private\_notes}))$	18

and $\llbracket\ R_{16}\ \wedge\ R_{18}\ \rrbracket \equiv \llbracket\ R'_{16}\ \rrbracket \equiv$

$$\bigwedge_{\substack{s \in Subjects \\ o \in Objects \\ a \in Actions}} \left(\begin{array}{l} (f_{DOCTOR}(s,o,a) \leftrightarrow Aut^{+}(s,o,a)) \wedge \\ (g_{DOCTOR}(s,o,a) \leftrightarrow Aut^{-}(s,o,a)) \wedge \\ (h_{DOCTOR}(s,o,a) \leftrightarrow Aut(s,o,a)) \end{array} \right)$$

where $f_{DOCTOR}(s,o,a)$ is defined as:

$f_{DOCTOR}(s,o,a)$	(s,o,a)	R
$\llbracket$ ($\boldsymbol{true}$ $\boldsymbol{chop^{p}}$ $\boldsymbol{always^{p}}(agree(S_{PATIENT})))$**and**$(S_{DOCTOR_1} \neq S_{DOCTOR})$ $\boldsymbol{always}(doctor(S_{DOCTOR_1}, S_{PATIENT}))$**and** $\boldsymbol{always}($**not**$doctor(S_{DOCTOR}, S_{PATIENT}))$ $\rrbracket$	$(S_{DOCTOR}$,GPS$, treat(S_{PATIENT}))$	16

and $g_{DOCTOR}(s,o,a)$ is defined as:

$g_{DOCTOR}(s,o,a)$	(s,o,a)	R
$\llbracket$ **false** $\rrbracket$	$(S_{DOCTOR}$,GPS$, treat(S_{PATIENT}))$	16

and $h_{DOCTOR}(s,o,a)$ is defined as:

$h_{DOCTOR}(s,o,a)$	(s,o,a)	R
$\llbracket 0:\ \ allow(S_{DOCTOR}$,GPS$, treat(S_{PATIENT}))$ **and not** $deny(S_{DOCTOR}$,GPS$, treat(S_{PATIENT}))\rrbracket$	$(S_{DOCTOR}$,GPS$, treat(S_{PATIENT}))$	18

and $[\![\ R_{17} \ \wedge \ R_{18} \ ]\!] \equiv [\![\ R'_{17} \ ]\!] \equiv$

$$\bigwedge_{\substack{s \in Subjects \\ o \in Objects \\ a \in Actions}} \left(\begin{array}{l} (f_{DOCTOR}(s,o,a) \leftrightarrow Aut^{+}(s,o,a)) \wedge \\ (g_{DOCTOR}(s,o,a) \leftrightarrow Aut^{-}(s,o,a)) \wedge \\ (h_{DOCTOR}(s,o,a) \leftrightarrow Aut(s,o,a)) \end{array} \right)$$

where $f_{DOCTOR}(s,o,a)$ is defined as:

$f_{DOCTOR}(s,o,a)$	(s,o,a)	R
$[\![\ 0: \ \ doctor(S_{DOCTOR}, S_{PATIENT})$ **and** $patient\_info(O_{patient\_info}, S_{PATIENT}) \ ]\!]$	$(S_{DOCTOR}$,GPS$, view(O_{patient_info}))$	17

and $g_{DOCTOR}(s,o,a)$ is defined as:

$g_{DOCTOR}(s,o,a)$	(s,o,a)	R
$[\![$ **false** $]\!]$	$(S_{DOCTOR}$,GPS$, view(O_{patient_info}))$	17

and $h_{DOCTOR}(s,o,a)$ is defined as:

$h_{DOCTOR}(s,o,a)$	(s,o,a)	R
$[\![0: \ \ allow(S_{DOCTOR}$,GPS$, view(O_{patient_info}))$ **and not** $deny(S_{DOCTOR}$,GPS$, view(O_{patient_info}))]\!]$	$(S_{DOCTOR}$,GPS$, view(O_{patient_info}))$	18

Appendix C Part 1:

In this appendix, the complete semantics of general practice system policy using the future time operators are listed here.

In the first part, we will list the patient policy rules which consists from the rules from rule R'_6 to rule R'_{11}

The semantics of Patient policy using future operators

$[\![R_6 \ \wedge \ R_{12}]\!] \equiv [\![R'_6]\!] \equiv$

$$\bigwedge_{\substack{s \in Subjects \\ o \in Objects \\ a \in Actions}} \left(\begin{array}{l} (f_{PATIENT}(s,o,a) \leftrightarrow Aut^{+}(s,o,a)) \wedge \\ (g_{PATIENT}(s,o,a) \leftrightarrow Aut^{-}(s,o,a)) \wedge \\ (h_{PATIENT}(s,o,a) \leftrightarrow Aut(s,o,a)) \end{array} \right)$$

where $f_{PATIENT}(s,o,a)$ is defined as:

$f_{PATIENT}(s,o,a)$	(s,o,a)	R
$[\![$ ***true chop always***$(registered(S_{PATIENT}))$ $]\!]$	$(S_{PATIENT}$,GPS,$access)$	6

and $g_{PATIENT}(s,o,a)$ is defined as:

$g_{PATIENT}(s,o,a)$	(s,o,a)	R
$[\![$ **false** $]\!]$	$(S_{PATIENT}$,GPS,$access)$	6

and $h_{PATIENT}(s,o,a)$ is defined as:

$h_{PATIENT}(s,o,a)$	(s,o,a)	R
$[\![0:$ $allow(S_{PATIENT}$,GPS,$access)$ **and not** $deny(S_{PATIENT}$,GPS,$access)]\!]$	$(S_{PATIENT}$,GPS,$access)$	12

and $[\![\; R_7 \;\; \wedge \;\; R_{12} \;]\!] \equiv [\![\; R_{7}' \;]\!] \equiv$

$$\bigwedge_{\substack{s \in Subjects \\ o \in Objects \\ a \in Actions}} \left(\begin{array}{l} (f_{PATIENT}(s,o,a) \leftrightarrow Aut^{+}(s,o,a)) \wedge \\ (g_{PATIENT}(s,o,a) \leftrightarrow Aut^{-}(s,o,a)) \wedge \\ (h_{PATIENT}(s,o,a) \leftrightarrow Aut(s,o,a)) \end{array} \right)$$

where $f_{PATIENT}(s,o,a)$ is defined as:

$f_{PATIENT}(s,o,a)$	(s,o,a)	R
$[\![0: \;\; patient\_info(O_{patient\_nfo}, S_{PATIENT})]\!]$	$(S_{PATIENT}$,GPS, $update(O_{patient\_info}))$	7

and $g_{PATIENT}(s,o,a)$ is defined as:

$g_{PATIENT}(s,o,a)$	(s,o,a)	R
$[\![$ **false** $]\!]$	$(S_{PATIENT}$,GPS, $update(O_{patient\_info}))$	7

and $h_{PATIENT}(s,o,a)$ is defined as:

$h_{PATIENT}(s,o,a)$	(s,o,a)	R
$[\![0: \;\; allow(S_{PATIENT}$,GPS, $update(O_{patient\_info}))$ **and not** $deny(S_{PATIENT}$,GPS, $update(O_{patient\_info}))]\!]$	$(S_{PATIENT}$,GPS, $update(O_{patient\_info}))$	12

and $[\![R_8 \wedge R_{12}]\!] \equiv [\![R_8']\!] \equiv$

$$\bigwedge_{\substack{s \in Subjects \\ o \in Objects \\ a \in Actions}} \begin{pmatrix} (f_{PATIENT}(s,o,a) \leftrightarrow Aut^{+}(s,o,a)) \wedge \\ (g_{PATIENT}(s,o,a) \leftrightarrow Aut^{-}(s,o,a)) \wedge \\ (h_{PATIENT}(s,o,a) \leftrightarrow Aut(s,o,a)) \end{pmatrix}$$

where $f_{PATIENT}(s,o,a)$ is defined as:

$f_{PATIENT}(s,o,a)$	(s,o,a)	R
$[\![$ **false** $]\!]$	$(S_{PATIENT}$,GPS, $alter(O_{medical\_records}))$	8

and $g_{PATIENT}(s,o,a)$ is defined as:

$g_{PATIENT}(s,o,a)$	(s,o,a)	R
$[\![$ **true** $]\!]$	$(S_{PATIENT}$,GPS, $alter(O_{medical\_records}))$	8

and $h_{PATIENT}(s,o,a)$ is defined as:

$h_{PATIENT}(s,o,a)$	(s,o,a)	R
$[\![0:$ $deny(S_{PATIENT}$,GPS, $alter(O_{medical\_records}))$ **and not** $allow(S_{PATIENT}$,GPS, $alter(O_{medical\_records}))]\!]$	$(S_{PATIENT}$,GPS, $alter(O_{medical\_records}))$	12

and $[\![\ R_9\ \wedge\ R_{12}\ ]\!] \equiv [\![\ R_9'\ ]\!] \equiv$

$$\bigwedge_{\substack{s \in Subjects\\ o \in Objects\\ a \in Actions}} \left(\begin{array}{l} (f_{PATIENT}(s,o,a) \leftrightarrow Aut^{+}(s,o,a))\wedge \\ (g_{PATIENT}(s,o,a) \leftrightarrow Aut^{-}(s,o,a))\wedge \\ (h_{PATIENT}(s,o,a) \leftrightarrow Aut(s,o,a)) \end{array} \right)$$

where $f_{PATIENT}(s,o,a)$ is defined as:

$f_{PATIENT}(s,o,a)$	(s,o,a)	R
$[\![$ ***true chop always***(*sign*$(S_{PATIENT}, O_{LA})$ **and** *sign*$(S_{PATIENT}, O_{CF})$) $]\!]$	$(S_{PATIENT}$,GPS$, treat(S_{PATIENT}))$	9

and $g_{PATIENT}(s,o,a)$ is defined as:

$g_{PATIENT}(s,o,a)$	(s,o,a)	R
$[\![$ **false** $]\!]$	$(S_{PATIENT}$,GPS$, treat(S_{PATIENT}))$	9

and $h_{PATIENT}(s,o,a)$ is defined as:

$h_{PATIENT}(s,o,a)$	(s,o,a)	R
$[\![0:$ *allow*$(S_{PATIENT}$,GPS$, treat(S_{PATIENT}))$ **and not** *deny*$(S_{PATIENT}$,GPS$, treat(S_{PATIENT}))]\!]$	$(S_{PATIENT}$,GPS$, treat(S_{PATIENT}))$	12

and $[\![\ R_{10} \ \wedge \ R_{12} \ ]\!] \equiv [\![\ R_{10}^{'} \ ]\!] \equiv$

$$\bigwedge_{\substack{s \in Subjects \\ o \in Objects \\ a \in Actions}} \left(\begin{array}{l} (f_{PATIENT}(s,o,a) \leftrightarrow Aut^{+}(s,o,a)) \wedge \\ (g_{PATIENT}(s,o,a) \leftrightarrow Aut^{-}(s,o,a)) \wedge \\ (h_{PATIENT}(s,o,a) \leftrightarrow Aut(s,o,a)) \end{array} \right)$$

where $f_{PATIENT}(s,o,a)$ is defined as:

$f_{PATIENT}(s,o,a)$	(s,o,a)	R
$[\![$ ***true chop always* not** *booking*$(S_{PATIENT}, O_{APPOINTMENT})$ ***chop skip*** $]\!]$	$(S_{PATIENT}$,GPS, *book*$(O_{APPOINTMENT}))$	10

and $g_{PATIENT}(s,o,a)$ is defined as:

$g_{PATIENT}(s,o,a)$	(s,o,a)	R
$[\![$ **false**$]\!]$	$(S_{PATIENT}$,GPS, *book*$(O_{APPOINTMENT}))$	10

and $h_{PATIENT}(s,o,a)$ is defined as:

$h_{PATIENT}(s,o,a)$	(s,o,a)	R
$[\![0:$ *allow*$(S_{PATIENT}$,GPS, *book*$(O_{APPOINTMENT}))$ **and not** *deny*$(S_{PATIENT}$,GPS, *book*$(O_{APPOINTMENT}))]\!]$	$(S_{PATIENT}$,GPS, *book*$(O_{APPOINTMENT}))$	12

and $[\![\ R_{11} \ \wedge \ R_{12} \ ]\!] \equiv [\![\ R'_{11} \ ]\!] \equiv$

$$\bigwedge_{\substack{s \in Subjects \\ o \in Objects \\ a \in Actions}} \left(\begin{array}{l} (f_{PATIENT}(s,o,a) \leftrightarrow Aut^{+}(s,o,a)) \wedge \\ (g_{PATIENT}(s,o,a) \leftrightarrow Aut^{-}(s,o,a)) \wedge \\ (h_{PATIENT}(s,o,a) \leftrightarrow Aut(s,o,a)) \end{array} \right)$$

where $f_{PATIENT}(s,o,a)$ is defined as:

$f_{PATIENT}(s,o,a)$	(s,o,a)	R
$[\![\ 0 : \ \ medical\_record(O_{medical\_record}, S_{PATIENT}) \ ]\!]$	$(S_{PATIENT}$,GPS,$view(O_{medical\_record}))$	11

and $g_{PATIENT}(s,o,a)$ is defined as:

$g_{PATIENT}(s,o,a)$	(s,o,a)	R
$[\![$ **false**$]\!]$	$(S_{PATIENT}$,GPS,$view(O_{medical\_record}))$	11

and $h_{PATIENT}(s,o,a)$ is defined as:

$h_{PATIENT}(s,o,a)$	(s,o,a)	R
$[\![0 : \ \ allow(S_{PATIENT}$,GPS,$view(O_{medical\_record}))$ **and not** $deny(S_{PATIENT}$,GPS,$view(O_{medical\_record}))]\!]$	$(S_{PATIENT}$,GPS,$view(O_{medical\_record}))$	12

Appendix C Part 2:

As a part of general practice system policies, the complete semantics of the doctor policy rules using future time are listed here. The doctor policy consists from the rules from rule $R^{'}_{13}$ to rule $R^{'}_{17}$

The semantics of Doctor policy

$$[\![R_{13} \wedge R_{18}]\!] \equiv [\![R^{'}_{13}]\!] \equiv$$

$$\bigwedge_{\substack{s \in Subjects \\ o \in Objects \\ a \in Actions}} \left(\begin{array}{l} (f_{DOCTOR}(s,o,a) \leftrightarrow Aut^{+}(s,o,a)) \wedge \\ (g_{DOCTOR}(s,o,a) \leftrightarrow Aut^{-}(s,o,a)) \wedge \\ (h_{DOCTOR}(s,o,a) \leftrightarrow Aut(s,o,a)) \end{array} \right)$$

where $f_{DOCTOR}(s,o,a)$ is defined as:

$f_{DOCTOR}(s,o,a)$	(s,o,a)	R
$[\![$ ***true chop always*** $(registered(S_{DOCTOR}))$ $]\!]$	$(S_{DOCTOR}, \text{GPS}, access)$	13

and $g_{DOCTOR}(s,o,a)$ is defined as:

$g_{DOCTOR}(s,o,a)$	(s,o,a)	R
$[\![$ **false** $]\!]$	$(S_{DOCTOR}, \text{GPS}, access)$	13

and $h_{DOCTOR}(s,o,a)$ is defined as:

$h_{DOCTOR}(s,o,a)$	(s,o,a)	R
$[\![0:$ $allow(S_{DOCTOR}, \text{GPS}, access)$ **and not** $deny(S_{DOCTOR}, \text{GPS}, access)]\!]$	$(S_{DOCTOR}, \text{GPS}, access)$	18

and $[\![\ R_{14}\ \wedge\ R_{18}\ ]\!] \equiv [\![\ R'_{14}\ ]\!] \equiv$

$$\bigwedge_{\substack{s \in Subjects \\ o \in Objects \\ a \in Actions}} \begin{pmatrix} (f_{DOCTOR}(s,o,a) \leftrightarrow Aut^{+}(s,o,a))\wedge \\ (g_{DOCTOR}(s,o,a) \leftrightarrow Aut^{-}(s,o,a))\wedge \\ (h_{DOCTOR}(s,o,a) \leftrightarrow Aut(s,o,a)) \end{pmatrix}$$

where $f_{DOCTOR}(s,o,a)$ is defined as:

$f_{DOCTOR}(s,o,a)$	(s,o,a)	R
$[\![0:$ $doctor(S_{DOCTOR}, S_{PATIENT})$ **and** $medical\_record(O_{medical\_record}, S_{PATIENT})]\!]$	$(S_{DOCTOR},\text{GPS}, alter(O_{medical\_record}))$	14

and $g_{DOCTOR}(s,o,a)$ is defined as:

$g_{DOCTOR}(s,o,a)$	(s,o,a)	R
$[\![$ **false** $]\!]$	$(S_{DOCTOR},\text{GPS}, alter(O_{medical\_record}))$	14

and $h_{DOCTOR}(s,o,a)$ is defined as:

$h_{DOCTOR}(s,o,a)$	(s,o,a)	R
$[\![0:$ $allow(S_{DOCTOR},\text{GPS}, alter(O_{medical\_record}))$ **and not** $deny(S_{DOCTOR},\text{GPS}, alter(O_{medical\_record}))]\!]$	$(S_{DOCTOR},\text{GPS}, alter(O_{medical\_record}))$	18

and $[\![\ R_{15}\ \wedge\ R_{18}\ ]\!] \equiv [\![\ R'_{15}\ ]\!] \equiv$

$$\bigwedge_{\substack{s \in Subjects\\ o \in Objects\\ a \in Actions}} \left(\begin{array}{l} (f_{DOCTOR}(s,o,a) \leftrightarrow Aut^{+}(s,o,a))\wedge \\ (g_{DOCTOR}(s,o,a) \leftrightarrow Aut^{-}(s,o,a))\wedge \\ (h_{DOCTOR}(s,o,a) \leftrightarrow Aut(s,o,a)) \end{array} \right)$$

where $f_{DOCTOR}(s,o,a)$ is defined as:

$f_{DOCTOR}(s,o,a)$	(s,o,a)	R
$[\![0:\ \ doctor(S_{DOCTOR},S_{PATIENT})$ **and** $private\_notes(O_{private\_notes},S_{PATIENT})]\!]$	$(S_{DOCTOR},\text{GPS},add(O_{private\_notes}))$	15

and $g_{DOCTOR}(s,o,a)$ is defined as:

$g_{DOCTOR}(s,o,a)$	(s,o,a)	R
$[\![$ **false** $]\!]$	$(S_{DOCTOR},\text{GPS},add(O_{private\_notes}))$	15

and $h_{DOCTOR}(s,o,a)$ is defined as:

$h_{DOCTOR}(s,o,a)$	(s,o,a)	R
$[\![0:\ \ allow(S_{DOCTOR},\text{GPS},add(O_{private\_notes}))$ **and not** $deny(S_{DOCTOR},\text{GPS},add(O_{private\_notes}))]\!]$	$(S_{DOCTOR},\text{GPS},add(O_{private\_notes}))$	18

and $[\![\ R_{16}\ \wedge\ R_{18}\ ]\!] \equiv [\![\ R'_{16}\ ]\!] \equiv$

$$\bigwedge_{\substack{s \in Subjects \\ o \in Objects \\ a \in Actions}} \left(\begin{array}{l} (f_{DOCTOR}(s,o,a) \leftrightarrow Aut^{+}(s,o,a))\wedge \\ (g_{DOCTOR}(s,o,a) \leftrightarrow Aut^{-}(s,o,a))\wedge \\ (h_{DOCTOR}(s,o,a) \leftrightarrow Aut(s,o,a)) \end{array} \right)$$

where $f_{DOCTOR}(s,o,a)$ is defined as:

$f_{DOCTOR}(s,o,a)$	(s,o,a)	R
$[\![$ ($\textbf{\textit{true}}$ $\textbf{\textit{chop}}$ $\textbf{\textit{always}}(agree(S_{PATIENT})))$**and**$(S_{DOCTOR_1} \neq S_{DOCTOR})$ $\textbf{\textit{always}}(doctor(S_{DOCTOR_1}, S_{PATIENT}))$**and** $\textbf{\textit{always}}($**not**$doctor(S_{DOCTOR}, S_{PATIENT}))\ ]\!]$	$(S_{DOCTOR}, \text{GPS}, treat(S_{PATIENT}))$	16

and $g_{DOCTOR}(s,o,a)$ is defined as:

$g_{DOCTOR}(s,o,a)$	(s,o,a)	R
$[\![$ **false** $]\!]$	$(S_{DOCTOR}, \text{GPS}, treat(S_{PATIENT}))$	16

and $h_{DOCTOR}(s,o,a)$ is defined as:

$h_{DOCTOR}(s,o,a)$	(s,o,a)	R
$[\![0:\ \ allow(S_{DOCTOR}, \text{GPS}, treat(S_{PATIENT}))$ **and not** $deny(S_{DOCTOR}, \text{GPS}, treat(S_{PATIENT}))]\!]$	$(S_{DOCTOR}, \text{GPS}, treat(S_{PATIENT}))$	18

and $[\![\ R_{17} \ \wedge \ R_{18} \ ]\!] \equiv [\![\ R'_{17} \ ]\!] \equiv$

$$\bigwedge_{\substack{s \in Subjects \\ o \in Objects \\ a \in Actions}} \left(\begin{array}{l} (f_{DOCTOR}(s,o,a) \leftrightarrow Aut^{+}(s,o,a)) \wedge \\ (g_{DOCTOR}(s,o,a) \leftrightarrow Aut^{-}(s,o,a)) \wedge \\ (h_{DOCTOR}(s,o,a) \leftrightarrow Aut(s,o,a)) \end{array} \right)$$

where $f_{DOCTOR}(s,o,a)$ is defined as:

$f_{DOCTOR}(s,o,a)$	(s,o,a)	R
$[\![$ 0 : *doctor*$(S_{DOCTOR}, S_{PATIENT})$ **and** $patient\_info(O_{patient\_info}, S_{PATIENT})$ $]\!]$	$(S_{DOCTOR}$,GPS, $view(O_{patient\_info}))$	17

and $g_{DOCTOR}(s,o,a)$ is defined as:

$g_{DOCTOR}(s,o,a)$	(s,o,a)	R
$[\![$ **false** $]\!]$	$(S_{DOCTOR}$,GPS, $view(O_{patient\_info}))$	17

and $h_{DOCTOR}(s,o,a)$ is defined as:

$h_{DOCTOR}(s,o,a)$	(s,o,a)	R
$[\![$0 : *allow*$(S_{DOCTOR}$,GPS, $view(O_{patient\_info}))$ **and not** *deny*$(S_{DOCTOR}$,GPS, $view(O_{patient\_info}))]\!]$	$(S_{DOCTOR}$,GPS, $view(O_{patient\_info}))$	18