FORMALLY ENSURING THE PERMISSIBILITY OF OBLIGATIONS IN SECURITY AND PRIVACY POLICIES

APPROVED BY SUPERVISING COMMITTEE:

_____
Jianwei Niu, Ph.D., Chair

_____
Ravi Sandhu, Ph.D.

_____
Gregory White, Ph.D.

_____
Jeffery von Ronne, Ph.D.

_____
Ninghui Li, Ph.D.

_____
Limin Jia, Ph.D.

Accepted: _____
Dean, Graduate School

DEDICATION

This dissertation is dedicated to my parents and also to my mentor William H. Winsborough for their unconditional affection and guidance.

FORMALLY ENSURING THE PERMISSIBILITY OF OBLIGATIONS IN SECURITY AND PRIVACY POLICIES

by

OMAR HAIDER CHOWDHURY, B. SC.

DISSERTATION
Presented to the Graduate Faculty of
The University of Texas at San Antonio
In Partial Fulfillment
Of the Requirements
For the Degree of

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT SAN ANTONIO
College of Sciences
Department of Computer Science
August 2013

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

ACKNOWLEDGEMENTS

I am deeply indebted to my deceased Ph.D. advisor Will Winsborough. He has taught me not only how to become an independent researcher but also how to be a better person. Without his vision, patience, insight, and motivation, it would have been difficult to finish my dissertation. I sincerely thank him for giving me the right tools to survive in this journey. He was the best role model a Ph.D. student can have.

I am also grateful to my Ph.D. advisor Jianwei Niu. She has immensely helped me to bring stability in my academic career during the torrid time of Will's death. She has helped me to focus on the right things in the right time which has helped me to eventually complete this journey. I am also thankful for her guidance, motivations, and patience.

I also want to thank my collaborators, Ting Yu, Keith Irwin, Anupam Datta, Ninghui Li, Jeffery von Ronne, Limin Jia, and Deepak Garg for their constant guidance and insightful scrutiny of my work. Their suggestions and insightful comments have helped me improve the quality of my work significantly.

I also want to thank my committee members Ravi Sandhu and Gregory White for spending their precious time on giving me suggestions on how to improve the quality of my dissertation.

I also want to thank my collaborator, lab mate, and good friend Murillo Pontual. I am deeply indebted to him for introducing me to Will. It has been nice experience working with Murillo because he was always ready to do the heavy lifting whenever it was necessary.

I also want to thank my best friend Andreas Gampe. He has helped me to crystallize a lot of abstract technical ideas in many of my papers, in some as a co-author. I am also deeply indebted to him for proof-reading my dissertation and also for providing suggestions on proving a lot of the theorems in this dissertation.

I also want to thank my colleagues and friends Shamim Ashik, Hui Shen, Jared Bennatt, Mark Robinson, and Haining Chen with whom I have had a lot of useful brainstorming sessions. I also want to specially thank my friends Jeff McAdams, Jane Liang, Giovanni Del Valle, Arsen

Melkonyan, Arpine Soghoyan, Keith Harrison, Emanuelle Vasconcelos, and BazouMana Kone for their company and a lot of good times we had together.

I am also deeply indebted to my parents for their unconditional love and guidance. Their sacrifice and attention to all my needs have helped me see this day. I am also thankful to my sister Shanta and my younger brother Risad for their inspiration, love, and motivation.

I am also thankful to in-laws for their support. I am also greatly thankful to my sister-in-law Fariba for her useful and critical advices. I am also thankful to my brother-in-laws Atanu and Deco for their motivations and support.

Last but not by any means least, I am really grateful to my lovely wife, Samira, for her inspiration, unconditional love, and support. She has been always there with me in good times and specifically in difficult times. I want to sincerely thank her for making my journey easier, enjoyable, and fun.

August 2013

FORMALLY ENSURING THE PERMISSIBILITY OF OBLIGATIONS IN SECURITY AND PRIVACY POLICIES

Omar Haider Chowdhury, Ph. D.
The University of Texas at San Antonio, 2013

Supervising Professor: Jianwei Niu, Ph.D.

Our society is becoming increasingly dependent on computer information systems for the management of personal information (*e.g.*, medical records, financial data.). Organizations are required to manage and share such information in a manner that conforms to specific privacy regulations (*e.g.*, the Health Insurance Portability and Accountability Act (HIPAA), the Gramm-Leach-Bliley Act (GLBA).). Privacy policies like HIPAA can impose restrictions based on the finite execution history (*present requirements*) and can also impose future requirements (*obligations*). Existing work on checking compliance only investigates whether a certain action respects the present requirements of the policy or investigates whether a certain pending obligation is violated. However, when an obligation is violated they cannot report whether the user was not diligent or whether the policy did not permit the obligation. To this end, we formally specify a property of the policy which we call the Δ-property that statically guarantees that any incurred obligations can be met. When an obligation is violated according to a policy that has the Δ-property, it is safe to assume that the obligation violation is not due to a malformed policy. We prove that checking whether a policy has the Δ-property is undecidable in general. We then develop a sound, semi-automated technique to check whether a policy has the Δ-property under some constraints. We demonstrate the efficacy of our technique by verifying that our interpretation of the HIPAA privacy rule has the Δ-property.

Organizations that intend to be compliant with privacy policies need to rely on their own access control policies to safeguard their resources against unauthorized access. For instance, having access control policy to ensure only valid organization employees have access to the individual's personal information. These access control policies can allow access to a resource provided that

the requesting user or some other user promises to perform some obligations. We are particularly interested in user obligations that can depend on and affect the authorization state of the system. Existing work introduces the property "*accountability*" that ensures that all the incurred user obligations are authorized. However, they assume that obligations cannot further incur other obligations (*i.e.*, no cascading obligations). As a result, it significantly reduces the expressive power of their obligation model as it cannot express several real life scenarios. We show that deciding accountability in the most general case is NP-hard. We then consider several special yet practical cases of cascading obligations and provide a decision procedure for accountability in their presence.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1: INTRODUCTION

Organizations (*e.g.*, banks, hospitals, credit card companies.) collect personal information from individuals to provide them with various services (*e.g.*, financial, healthcare.). The individuals have the expectation from the organizations that they will not disclose the customers' personal information in any unauthorized fashion. To ensure that the customers personal information is not used or disclosed by organizations in some unauthorized fashion, there are federal privacy regulations like the Health Insurance Portability and Accountability Act (HIPAA) [62], Gramm-Leach-Bliley Act (GLBA) [2], Sarbanes-Oxley Act (SOX) [116]. The federal privacy regulations mandate how the collected personal information can be lawfully used or disclosed by organizations. These federal privacy regulations carry the force of law and violation of these federal regulations can bring down heavy financial penalties on the organizations. For instance, Cignet Health Center of Prince George's County, Maryland, was fined a staggering $1.3 million for failing to be compliant with §164.524 of HIPAA [112] a total of 41 times. It is thus incumbent on the organizations to have means for *efficiently* checking compliance[1] with applicable privacy regulations.

Moreover, the organizations are using computer information systems to manage, store, and share the collected personal information. In the healthcare industry, the federal government has introduced programs such as the Medicare and Medicaid EHR Incentive Program [1] that compensate the organizations which use certified electronic healthcare systems. To this end, it is paramount for the organizations to have the means to check compliance of their computer information systems with applicable federal privacy regulations in an *automated* fashion.

A privacy policy like HIPAA can generally impose two kinds of requirements, *present requirements* and *obligations* [4,23,28,39–41,54,67,69,70,86,95,97,100,107,110]. Present requirements restrict when a contemplated action is permissible based on the finite history leading up to the con-

[1]The final arbiter of compliance with a binding privacy regulation is–in the United States, at least–the judicial system. The judges and jurors that make up the judiciary are not bound to interpret these regulations according any particular logical formalization. In that sense, we use the term "compliance" in a non-restrictive way for expressing conformance or permissibility of an action with respect to a formalized privacy policy. We use this term for being consistent with the terminology of the existing work in the literature.

templated action. Obligations on the other hand are actions that a principal or a system might be required to perform at some point of time in the future. An action will be compliant with a privacy policy like HIPAA provided that it satisfies the present requirements imposed by the privacy policy and does not prevent obligations that might be incurred from being fulfilled. An example of the present requirements can be found in §164.502(e)(1)(i) of HIPAA. It specifies that a covered entity (hospital) can disclose a patient's protected health information (*PHI*) to the covered entity's business associate if the covered entity has received a satisfactory assurance from the business associate ensuring that the business associate will protect the patient's *PHI*. According to this rule, a covered entity receiving the satisfactory assurance from its business associate is a present condition imposed by the policy rule. An example of an obligatory requirement can be found in §164.524 of HIPAA, which states that when an individual requests the covered entity to access her own *PHI*, then the covered entity is obligated to provide the individual access to her *PHI*. Requiring the covered entity to give access to the *PHI* to the patient is an example of an obligatory requirement.

Although whether an action satisfies the present requirements imposed by the privacy policy can be checked efficiently, giving the assurance that obligations will be fulfilled on the contrary is not possible. This is due to the fact that it is not possible to force a principal to take an action although it is feasible to check whether an incurred obligation is violated. The majority of the current work on checking compliance of privacy policies [13, 55, 73, 79] only checks whether the current contemplated action satisfies the present requirements imposed by the policy or whether a certain incurred obligation is violated. However, when an obligation goes violated, they cannot differentiate between the following two cases: (1) the policy did not permit the obligation, (2) the principal was not diligent enough to fulfill the obligation. At a first approximation, this distinction might not seem interesting or important, but we will justify that this detailed information is important.

Recall that Cignet Health Center was fined for violating the §164.524 of HIPAA 41 times. Note that §164.524 of HIPAA imposes obligatory requirement on the covered entity (*e.g.*, hospital). Consider the situation where the HIPAA policy does not allow the covered entity to discharge

the imposed obligation. In that case, the covered entity should not be blamed for not fulfilling the obligation as the policy is not well-formed. Moreover, individual(s) violating federal privacy regulations can be prosecuted. Thus, before prosecuting somebody, it is desirable to have formal assurance that any incurred obligation according to the privacy policy can be carried out without violating the policy itself. If providing such formal assurance is possible for a privacy policy, then whenever any incurred obligation is violated, it is sound to consider that it was violated due to the lack of diligence from the principal under the assumption that the principal is unconstrained in causing actions permitted by the policy to occur. In the literature, there are very few work [11] that focuses on providing formal guarantee that incurred obligations can be discharged in a privacy policy compliant fashion. To this end, in this thesis we develop a static privacy policy analysis method that provides a formal guarantee that all the incurred obligation by the principal or the system can be carried out in a privacy policy compliant fashion.

Organizations that are required to be compliant with federal privacy regulations like HIPAA also have access control policies for protecting their resources from unauthorized access. For instance, recall §164.502(e)(1)(i) of HIPAA which specifies that a covered entity (hospital) can disclose a patient's protected health information (*PHI*) to the covered entity's business associate if the covered entity has received a satisfactory assurance from the business associate ensuring that the business associate will protect the patient's *PHI*. Now, it should be the case that only authorized employee of the covered entity should get access to the patients' *PHI*. To ensure this, the covered entity should put forward an access policy that denies access to a patient's *PHI* to all unauthorized employee of the covered entity. Access control policies decide who can access what resource under which conditions. *Generally*, access control policies make the decision of whether to allow an access based on the current state and past history. However, research has shown that integrating obligations in access control policies can be particularly useful. For instance, researchers have shown application of obligations on policy management [17, 18], risk management and tackling insider threat [10,25], managing pervasive systems [117], usage control [105], data protection [63],

etc. As mentioned before, obligations specifically *user obligations*[2], are also actions which require appropriate authorizations to be carried out and additionally they can alter the authorization state of the system. Thus, we can have a situation where a user incurs an obligation and it is not carried out which in turn impacts the performance evaluation of the user. Now, as before, the obligation violation could possibly be the result of one of the following two possibilities: (1) the user was not authorized to perform the obligation or (2) the user was not diligent to discharge her obligation. In case the user did not possess proper authorization to carry out her obligation, then the user should not be blamed and her performance evaluation should not be impacted negatively. However, in the case the user was not diligent enough to fulfill her obligation, then it is desirable that the user's performance evaluation should be negatively impacted. To this end, to have the assurance that whenever an obligation is not realized, it is always due to the lack of diligence of the user, one should make sure that all users who incurred obligations should have the appropriate authorizations to fulfill their obligations.

To provide such an assurance, existing work proposes a property of the authorization state and pending obligations called "*accountability*" [67, 109, 110]. The accountability property roughly specifies the requirement that all incurred obligations should be authorized during the appropriate time interval. They propose to maintain the accountability property as an invariant and thus disallowing any actions that violate this property. However, their definition and decision procedure for accountability require a simplifying assumption that obligations cannot further incur obligations (*no cascading obligations*). However, there are practical scenarios which can be naturally captured by cascading obligations. For instance, consider a scenario where, when a sales assistant submits a purchase order, the clerk incurs an obligation to issue a check in the amount identified in the purchase order. As soon as the clerk issues the check, the manager incurs an obligation that requires him to check the consistency of the purchase order. If the purchase order is consistent and the manager approves it, then the accountant incurs another obligation to approve the check. To

[2] *System obligations* are future actions that are required to be carried out by the system and can be assumed to be fulfilled.

this end, the goal of this thesis in this regard is to generalize the accountability definition to support cascading obligations, study the complexity of the accountability decision problem in presence of cascading obligations, and develop a decision procedure for accountability that can support special yet practical cases of cascading obligations.

We have seen situations where it is important to have guarantees that incurred obligations can be carried out in a way that conform to applicable privacy and security policies. Thus, the overarching goal of this thesis is to develop a static and a dynamic technique which can provide formal assurance that incurred obligations can be discharged in a policy conforming fashion. We now provide more details in the context of providing formal assurance that obligations incurred in both privacy and security policies can be discharged in a policy conforming fashion.

1.1 Formally Ensuring Permissibility of Obligations in Privacy Policies

Recall that we have already motivated the necessity to have efficient compliance checking algorithm for checking compliance with applicable privacy regulations. Privacy regulations like HIPAA can impose both present requirements and also obligatory requirements. An action will be compliant with the privacy policy if it is compliant with both present requirements and obligatory requirements. Present requirements impose restrictions based on the system's finite execution history whereas obligatory requirements impose future restrictions. Although present requirements can be enforced, obligatory requirements cannot be enforced. Thus, it is necessary to have formal guarantees that any incurred obligation will be permitted by the policy. Otherwise, it is impossible to pin-point whether an obligation violation is due to lack of diligence or is due to a malformed policy. To mitigate this, we propose a static policy analysis method which gives the guarantee that all the incurred obligation will be permitted by the policy. We first introduce the readers to our privacy policy specification languages. We then introduce the reader to the notion of privacy policy compliance and finally introduce our sound, semi-automated policy analysis technique.

1.1.1 Specification Language for Privacy Policy

The privacy regulations like HIPAA [62], GLBA [2], SOX [116], are specified in natural language. To develop efficient compliance checking algorithms that can automatically check whether a certain action is compliant with the applicable privacy policies or to give formal guarantees that a policy permits any incurred obligations, it is first necessary to express the privacy requirements imposed by the regulation in some formal policy specification language. Several frameworks have been proposed for specifying and analyzing privacy policies [3, 11–13, 20, 31, 35, 37, 55, 73, 79, 94]. To this end, we consider two possible policy specification languages. One of the candidate privacy policy specification languages we consider is a generic, off-the-shelf access control policy specification language, XACML [127]. *OASIS's eXtensible Access Control Markup Language* (*XACML*) [127] is one of the most popular access control specification languages. Along with the rich specification language, XACML [127] also has a robust enforcement engine that can enforce policies specified in the language. Although, XACML is an expressive specification language it lacks features needed to specify privacy policies like HIPAA. This is natural as XACML is designed for specifying access control policies rather than privacy policies like HIPAA. We thus assess XACML's adequacy for expressing HIPAA. More precisely, we investigate what features a specification language requires to sufficiently specify HIPAA. We also discuss which of these features XACML possesses and also propose extensions of XACML to support the missing features.

One of the apparent advantages of extending XACML to support privacy policies like HIPAA is that one uniform specification language and enforcement mechanism can be used to specify and enforce the access control policies and the privacy policies of the system. Managing the access control policies and the privacy policies separately is cumbersome as an action can be mandated by both policies. However, if we use XACML for expressing both policies, then XACML's enforcement mechanism will combine the permissibility decision of an action by using the policy combination algorithms (PCAs). Furthermore, organizations can have their own business privacy policy on top of the federal privacy regulations. In this case, the organization's privacy policy

would be the composition of their business privacy policy and the federal privacy regulations. This composition of privacy policies can be very easily achieved by XACML.

Our evaluation of XACML as a candidate specification language is based on a set of features required for expressing HIPAA, proposed by DeYoung *et al.* [35]. In our evaluation of XACML, we found that XACML has some features (*e.g.*, attributes, policy/policy rule combination) rich enough to support HIPAA. However, it lacks some other necessary features (*e.g.*, event history, obligations, subjective belief, reference to other rules) to adequately capture the HIPAA privacy rules. We thus propose the necessary extensions for XACML to specify HIPAA.

In our analysis of XACML's candidacy as a possible specification language, we found that features like temporal conditions, quantification, which are necessary to specify HIPAA, cannot be expressed in XACML in a flexible and extensible way. To mitigate this, *the next candidate specification language* we consider is specialized for expressing privacy policies like HIPAA and is inspired by the specification languages proposed by Barth *et al.* [11] and DeYoung *et al.* [35]. We denote this privacy policy specification language with ***FOPSL***, which is short for First-order Policy Specification Language. ***FOPSL*** is based on a restricted fragment of first order temporal logic (FOTL). We demonstrate the efficacy of ***FOPSL*** by expressing all 84 disclosure related HIPAA clauses in it (see Appendix A). Due to its well-defined formal semantics and its expressive power, we use ***FOPSL*** as the language of our choice for our static policy analysis technique which gives formal assurance that incurred obligations can be met in a privacy policy conforming way. As we shall show, in general the problem of policy analysis is undecidable. Thus, we impose some restrictions, some of which are on the specification language, to make the problem decidable. The language ***FOPSL*** is thus developed keeping the necessary restrictions required for policy analysis on mind. Although ***FOPSL*** has some restrictions, it is still expressive enough to capture the HIPAA privacy rule which we use in our case study.

1.1.2 Privacy Policy Compliance

Once a privacy policy like HIPAA is specified in a formal language like ***FOPSL***, the next step is to formally define what it means for an action to be compliant with a privacy policy. An organization or an individual complies with a privacy policy if it only takes actions permitted by the policy and performs all actions required by the policy. We consider a particular action to be compliant if it does not cause the organization or person in question to no longer be in compliance with the privacy policy. Barth *et al.* [11] present a framework, Contextual Integrity (*CI*), for specifying privacy regulations like HIPAA. They also introduce two notions of *compliance*, weak compliance (*WC*) and strong compliance (*SC*). WC ensures that all actions are compliant with the *present requirements* of the policy, whereas SC ensures that *obligatory* (*future*) *requirements* incurred due to performing an action will be consistent with the present conditions of the policy [33]. We borrow this notion of compliance and extend it to specify what it means for an action to be compliant with a policy written in ***FOPSL***. Note that Barth *et al.*'s definitions of compliance (WC and SC) are not sufficient for our case as they are restricted to only propositional linear temporal logic (*pLTL*), which cannot be feasibly used for specifying privacy regulations like HIPAA.

Due to the syntactic restrictions of ***FOPSL***, we can replace all the future temporal subformulas in a policy $\wp$ specified in ***FOPSL*** with logical true and the resulting FOTL formula is a pure past formula which we denote with $weak(\wp)$. The formula $weak(\wp)$ represents the present requirements imposed by the policy $\wp$. Thus, given a finite execution history σ_f and a current contemplated action a, we call the action a a weakly compliant action with respect to σ_f and $\wp$, if every position of finite execution trace $\sigma_f \cdot a$ [3] satisfies the formula $weak(\wp)$. We have developed an algorithm that takes as input a privacy policy $\wp$, a finite execution history (or, log) $\mathcal{L}$, and an action a, and it computes to see whether a is weakly compliant with respect to $\wp$ and $\mathcal{L}$. In the literature there are techniques which can be used to check whether an action is weakly compliant with respect to a finite execution history and a privacy policy specified in pLTL [60, 61, 85, 115]. One possibility is to write quantifications as finite conjunctions and disjunctions and apply the techniques available

[3] $\cdot$ represents the concatenation.

for pLTL. However, ***FOPSL*** allows quantification over on unbounded domains and this approach will not terminate. To achieve termination, we borrow a technique from logic programming called mode checking [9,34,96]. Given a n-ary relation symbol p, the *mode* of p is function m_p that maps each argument position of p to either '+' or '-' where '+' represents input position and '-' represents output position. The implication of the mode of a predicate is that when all the arguments in the input position are ground (concrete values), then the number of concrete values for the output argument positions of the predicate that satisfy the relation will be finite. For instance, let us consider a predicate $mul(x,y,z)$ which holds true when $x \times y = z$. One possible moding of this predicate *mul* is $mul(+,-,+)$. It suggests that provided that we have concrete values for x and z, the number of concrete values for y which will make the condition $x \times y = z$ true is finite. We have developed a mode checking technique which checks the policy in linear time of the policy size and decides whether the policy satisfies our moding rules. Our weak compliance checking algorithm is complete for policies which satisfy our moding restrictions.

Note that using mode checking or some variations of it (*e.g.*, safe range checking [13]) in the context of compliance checking termination is not new [13, 55]. The majority of existing work [13, 15, 16, 73] assumes that the number of satisfiable valuations for each predicate is finite. In other words, they only allow predicates in which all the argument positions are in the output mode. Thus, they store all the necessary valuations that appeared in the finite history in a summary structure (instead of the whole finite execution history) and look it up when it is necessary for making a policy decision. This existing work cannot handle any policies that contain a predicate for which not all argument positions are in the output mode. We have developed a labeling algorithm which labels parts of a policy formula for which a summary structure with all necessary satisfiable valuations can be kept. Moreover, our labeling algorithm can handle policies which contain predicates of which not all arguments positions are in the output mode. In that sense, we have identified a more expressive fragment of the policy than previously known for which it is sufficient to keep only the summary structure (instead of the whole execution trace) to make a sound policy decision. The closest to our weak compliance checking algorithm is the algorithm developed by Garg *et*

al. [55]. They were the first to explicitly use mode checking for policy compliance checking. Their specification language is first order logic instead of FOTL and additionally they require the whole finite history to be stored. However, their algorithm can handle policies that are more expressive than any other prior work. Our specification language is as expressive as theirs. In that sense, our algorithm adopts all the advantages of the previous algorithms and is more space efficient than the algorithm of Garg *et al.* [55] in some cases and also can handle more expressive policies than what is considered in some prior work [13, 15, 16, 73]. Given a finite execution history (or, log) $\mathcal{L}$ and a privacy policy $\wp$, our weak compliance checking algorithm has a runtime complexity of $O(|\mathcal{L}|^{|\wp|})$ and a space complexity of $O(|\wp|)$ in which $|\mathcal{L}|$ and $|\varphi|$, respectively, represents the size of finite history (or, log) and the size of the policy.

Given a finite execution history σ_f and a contemplated action a, we call the action a a strongly compliant action with respect to σ_f and the policy $\wp$, if there exists an infinite extension σ_i such that every position of the infinite trace $\sigma_f \cdot a \cdot \sigma_i$ satisfies $\wp$. Strong compliance ensures that if an obligation is incurred, then there is an extension of the current execution trace in which that obligation can be discharged. To check whether an action is strongly complaint we have to check whether the formula representing the future requirement is satisfiable. ***FOPSL*** is a fragment of the non-monadic FOTL, the satisfiability of which has been shown to be undecidable [64]. In this vein, we will show that to check whether an action is strongly compliant with a policy specified in ***FOPSL*** is in general undecidable.

1.1.3 Policy Analysis Providing Formal Assurance That Obligations can be Met

To check whether an action is compliant with a policy $\wp$ specified in ***FOPSL***, we have to check whether the action is both weakly and strongly compliant with $\wp$. We will prove that checking WC is feasible whereas checking SC is undecidable. Existing work in this area [3, 13, 31, 37, 55, 79], while checking compliance, only considers WC without taking SC into account. Thus, when only WC is checked, if an obligation is violated, it could be due to a malformed policy or lack of diligence from the principal. Existing compliance checking algorithm cannot distinguish between

these two cases.

To mitigate the undecidability of SC, we formally specify the property WC entails SC (denoted by Δ) [11] of a privacy policy. Note that although the Δ-property has been introduced before [11], we are the first to present sound techniques to decide whether a policy has the Δ-property and apply this technique to a practical privacy policy like HIPAA. Prior work presents a semantic definition and a decision procedure for checking the Δ-property restricted to only pLTL policies. As noted before, pLTL cannot be used to concisely specify a policy like HIPAA. We are the first to specify Δ-property in a formal logic which is applicable to a practical privacy policy specification language ***FOPSL*** in which practical privacy policy like HIPAA can be expressed.

A policy has the Δ-property if every weakly compliant action is also strongly compliant. To check compliance of a policy $\wp$ which has the Δ-property, it is sufficient to check only weak compliance. Moreover, when a policy $\wp$ has the Δ-property, it ensures that for every finite execution trace and for all pending obligations, there is an infinite extension in which the obligations can be discharged. Thus, when an obligation is violated for a policy with the Δ-property, it is safe to assume that the obligation was violated due to the principal's lack of diligence.

The Δ-property can be checked once statically before the policy is deployed. Given a privacy policy $\wp$ written in ***FOPSL***, we syntactically generate a first order CTL* with linear past (denoted by FO-CTL$^*_{lp}$) [77] formula $\delta(\wp)$ from $\wp$. We prove that $\delta(\wp)$ is satisfiable in the *most permissive model* $\mathbb{M}_\wp$ if and only if $\wp$ has the Δ-property. The most permissive model with respect to a policy $\wp$ (denoted by $\mathbb{M}_\wp$) is the model in which at each step one action from all the possible actions referred to by $\wp$ is non-deterministically chosen to be performed. We will proof that to check whether a policy $\wp$ specified in ***FOPSL*** has the Δ-property, is in general undecidable. We reduce the Turing Machine halting problem [47] to checking the Δ-property of a policy. Moreover, model checking a FO-CTL$^*_{lp}$ formula with respect to $\mathbb{M}_\wp$ is undecidable in general.

While checking the Δ-property for a policy specified in ***FOPSL*** is in general undecidable, this result is not discouraging as we can develop a sound, semi-automated technique with which we can check the Δ-property for practical privacy policies like HIPAA efficiently based on some

reasonable assumptions. We prove that there are *exactly* two cases in which the Δ-property can be violated. In the first case, taking a weakly compliant action might cause the system to transition to a bad state from which there is no weakly compliant infinite extensions of the current finite trace. In the second case, the policy allows to incur an obligation which is not consistent with the present requirements imposed by the policy [33]. We prove that for policies written in ***FOPSL***, due to a syntactic restriction, the former violation case cannot happen. Thus, it is sufficient to consider the second violation case only. We then present a sound and complete *privacy policy slicing algorithm* which decomposes the original policy analysis problem into multiple smaller policy analysis problems by slicing the policy with respect to one obligation at a time assuming obligations do not interact with each other.

Finally, we use HIPAA as a case study to show the efficacy of our analysis techniques. We first show that the HIPAA policy $\wp_H$ is trivially *satisfiable*. HIPAA does not restrict transmission of any message that does not contain protected health information (*PHI*) of an individual. One can thus satisfy the HIPAA privacy policy by only sending messages not containing any *PHI*. Thus, $\wp_H$ can violate the Δ-property only through allowing a weakly compliant action to incur unsatisfiable obligations. We then slice $\wp_H$ with respect to two different obligations from HIPAA (§160.310, §164.524)[4]. The size of the sliced policies in both cases is only 6.5% of $\wp_H$, which is a significant reduction of the policy size to be considered. We then develop a small model theorem [46] for each sliced policy of $\wp_H$ which reduces the problem of checking the Δ-property with infinite carrier sets to checking the Δ-property for finite carrier sets. A small model theorem for the complete language ***FOPSL*** remains an open question. We then formally verify that the two sliced HIPAA policies have the Δ-property. While there is currently no tool support for model checking CTL$^*_{lp}$, which we leave as future work, we utilize the approach of Barth *et al.* [11] which is applicable in this case.

[4]There are two more obligations in HIPAA which require sending privacy notice to the patient. We assume that privacy notices do not contain any individually identifiable information and thus not regulated by HIPAA. In that case, those obligations are trivially allowed.

1.2 Formally Ensuring Permissibility of Obligations in Security Policies

The organization that is required to be compliant with applicable practical privacy policies like HIPAA will also have some organizational access control policy to protect their resources from unauthorized access. For instance, consider the HIPAA privacy policy rule in §164.508 which requires that a covered entity can disclose a patient's psychotherapy note provided that it has already received a valid authorization that allows the covered entity's action. Now the covered entity must ensure that one of the authorized employee can get access to it rather than any arbitrary employee of the covered entity. To ensure this, the covered entity should put forward an access control policy that only allows the authorized employee to access the patient's psychotherapy notes. Many access control policies contain some notion of actions that are *required* to be performed by a system or its users in some time in the future. Such required actions can be naturally modeled as obligations. Based on who incurs the obligations, we have two types of obligations, namely, *system obligations* and *user obligations*. A user (resp., system) obligation is an action that is to be carried out by a user (resp., the system) in some time in the future. The notion of obligations is not new. Several researchers [3, 11, 18, 31, 69, 94, 95, 97, 101, 122, 127] have proposed frameworks for modeling and managing obligations. The majority of the existing work [3, 11, 18, 31, 94, 101, 122, 127] focuses on policy specification languages for obligations rather than efficient management of obligations [11, 39, 54, 67, 86, 100, 110]. Even for works on the management of obligations, they mainly consider *system obligations*. On the contrary, our goal is to develop techniques necessary for the management of user obligations.

Managing user obligations is challenging as system obligations can be assumed to be always fulfilled whereas this is often not the case for user obligations. More generally, we consider user obligations that can require authorization and can also alter the authorization state of the system. As a user obligation is an action, it is subjected to the authorization requirements imposed by the security policy of the system.

While managing user obligations that may depend on and effect authorization, we have to con-

sider the case where a user incurs an obligation that she is not authorized to discharge. This is important due to the fact that when an obligation violation occurs it is difficult to decide whether the violation is the result of the user's negligence or due to insufficient privileges. Existing work [67,109,110] introduces a property of the authorization state and the current obligations in the system named *accountability*. The accountability property ensures that all the incurred obligatory actions are authorized. This enables us to realize that any obligation violation is due to the user's negligence instead of absence of required authorization. Existing work assumes that obligatory actions cannot further incur obligations (*i.e.*, no *cascading obligations*). Cascading obligations can be used to model several real life situations which cannot be captured by existing obligation models [67, 109, 110]. Thus, this assumption significantly reduces the expressive power of their obligation model.

In this thesis, we formally specify the accountability property in presence of cascading user obligations. We also prove that deciding accountability in the presence of cascading user obligations is NP hard. We then consider several special yet practical cases of cascading user obligations. For these special cases of cascading obligations, we provide a tractable decision procedure for accountability and also present associated empirical evaluation results.

Difference Between Δ-property and the Accountability Property

We have used two properties which ensure that incurred obligations in privacy policies and security policies, can be carried out in a policy conforming fashion. Although, the intend of both of the properties are same, there are some subtle differences between these two properties. We briefly discuss them now.

Accountability is a property of the authorization state (for our case, this is keeping track of users and their role assignments), the security policy, and the set of pending obligations. This is a property we intend to maintain as an invariant of the system. We achieve this by disallowing any action that violates it. Moreover, we maintain the accountability property in the runtime, during the execution of the system. Δ-property on the other hand is a property of the privacy policy only

(irrespective of the system state). We check the Δ-property statically once before the privacy policy is deployed. In that sense, Δ-property gives a static guarantee about obligations' permissibility unlike the dynamic accountability property.

Accountability property requires that when an obligation is incurred (not when the obligation is due), it principal who incurred the obligation should have the appropriate authorization to carry out the obligation. On the contrary, the Δ-property requires that whenever an obligation incurred then there exists a way in which the obligatee can carry out the obligation in a policy conforming fashion. The Δ-property is maintained even when the obligation is incurred and the obligatee is not authorized to fulfill the obligation, instead there is a way that the obligation can be carried out by the obligatee. For instance, consider in an abstract setting in which we have a policy that incurs an obligation for user u_o to carry out the obligation o_i. Let us assume, when the o_i is incurred, at that time u_o is not authorized to perform o_i. In that case, the accountability property is violated. However, the Δ-property is not violated as long as there exists a way in which u_o can get the proper authorization to carry out o_i in its associated time frame. In this sense, one might say that the accountability property is stronger than the Δ-property.

1.3 Thesis Statement

> *This thesis demonstrates that, for a class of practical privacy and security policies (e.g., HIPAA), it is possible to provide formal assurance that all incurred obligatory actions are allowed.*

1.4 Contributions

The technical contributions of this thesis can be broadly partitioned into two categories. We discuss each of these just below.

1.4.1 Formally Ensuring Permissibility of Obligations in Privacy Policies.

Privacy Policy Specification Language:

- We evaluate XACML as a possible specification language for expressing HIPAA. To the best of our knowledge, we are the first to consider XACML as a specification language for HIPAA. Our evaluation of XACML as a candidate specification language is based on a set of features required for expressing HIPAA, proposed by DeYoung *et al.* [35]. In our evaluation of XACML, we found that XACML has some of the features (*e.g.*, attributes, policy/policy rule combination) necessary to support HIPAA. However, it lacks some other necessary features (*e.g.*, event history, obligations, subjective belief, reference to other rules) to adequately capture the HIPAA privacy rules. We believe that the support for the missing features will enable XACML to specify HIPAA.

- In our evaluation of XACML as a candidate specification language to specify practical privacy policies like HIPAA, we found out that some of the features necessary (*e.g.*, temporal conditions, quantifications) cannot be easily expressed in XACML in a flexible and extensible way. To mitigate this, we develop a privacy policy specification language based on a restricted fragment of first order temporal logic (FOTL). We call our privacy policy specification language ***FOPSL***. It is inspired by the specification language of Barth *et al.* [11] and DeYoung *et al.* [35]. To show the expressive power of ***FOPSL***, we have expressed all 84 disclosure related clauses in it (see Appendix A).

Privacy Policy Compliance:

- We then formally specify what it means for an action to be compliant with a privacy policy specified in ***FOPSL***. We borrow the notion of privacy policy compliance from a prior work [11] which introduced two notions of compliance, weak compliance (WC) and strong compliance (SC). Their definitions and compliance checking algorithms are inadequate for a policy specification language like ***FOPSL*** as their formalization and compliance checking algorithm assumes the policy to be specified in propositional linear temporal logic (pLTL). However, practical privacy policies like HIPAA cannot be concisely represented in pLTL. We then present a mode driven algorithm for checking weak compliance with a policy spec-

ified in ***FOPSL***. The algorithm has a time complexity of $O(|\mathcal{L}|^{|\wp|})$ where $|\mathcal{L}|$ represents the finite history size and $|\wp|$ represents the policy size. The algorithm has a space complexity of $O(|\wp|)$. We then prove that checking strong compliance of a policy written in ***FOPSL*** is undecidable in general. We show this by reducing the Turing machine halting problem [47] to checking whether a policy has the Δ-property.

Privacy Policy Analysis:

- To overcome the undecidability of SC, we borrow the Δ-property of the policy from the work of Barth *et al.* [11] They however provide a semantic definition of the Δ-property and a decision procedure which is only applicable to policies specified in pLTL. We provide a formal definition of what it means for a policy specified in ***FOPSL*** to have the Δ-property. Given a privacy policy $\wp$ specified in ***FOPSL***, we syntactically generate a first order CTL* with linear past logic (FO-CTL*$_{lp}$) formula which we denote with $\delta(\wp)$. We prove that a policy $\wp$ has the Δ-property if the most permissive model of $\wp$, $\mathbb{M}_{\wp}$, satisfies the $\delta(\wp)$ formula ($\mathbb{M}_{\wp} \models \delta(\wp)$). We are the first to pose the problem of checking whether a policy $\wp$ has the Δ-property, as a FO-CTL*$_{lp}$ model checking problem. However, checking whether $\wp$ has the Δ-property is undecidable in general as FO-CTL*$_{lp}$ model checking problem is undecidable in general.

- We then develop a sound, semi-automated technique to check whether a policy has the Δ-property. To this end, we show that for policies written in ***FOPSL***, a policy $\wp$ can violate the Δ-property, if a weakly compliant action incurs an unsatisfiable obligation. Based on this and the assumption that obligations do not interact with each other, we develop a *privacy policy slicing algorithm*. It takes as input a policy and an obligation, and returns a sub-policy. We prove that to check whether the obligation in question is satisfiable, it is sufficient to analyze the sub-policy. A small model theorem will need to be proved for each policy to which our techniques are applied. Proving the *small model theorem* for an analytical problem ensures that it is sound to consider only a small, bounded size of carriers instead of an unbounded

one while solving the problem. It is not clear whether all policies written in ***FOPSL*** will have a small model theorem, we believe small model theorems can be proved for most practical privacy policies. If such a small model theorem exists, we can rewrite the quantifiers as finite conjunctions and disjunctions and replace the relations with propositions to obtain a pLTL policy as the carriers are small and finite. There are two possible approaches to check whether a policy specified in pLTL has the Δ-property. The first approach is based on CTL$^*_{lp}$ model checking and proposed by us. The other approach is tableau-based and is proposed by Barth *et al.* [11].The complexity of both approaches is EXPSPACE-complete.

- To show the efficacy of our static policy analysis approach to check whether a policy has the Δ-property, we applied our technique on our interpretation of the HIPAA privacy rule. According to our analysis, we have verified that our interpretation of the HIPAA privacy rule has the Δ-property the implication of which is that, to check whether an action is compliant with our interpretation of the HIPAA privacy rule, it is sufficient to check only weak compliance. Moreover, any incurred obligation which has been violated is due to the lack of the principal's diligence rather than a malformed policy.

1.4.2 Formally Ensuring Permissibility of Obligations in Security Policies.

- We enhance the previous concrete user obligation model [67, 109, 110] to specify cascading user obligations. We propose several guidelines which can be used to select the user who incurs the obligation when an action is performed. Moreover, we present a way of specifying the time interval of the new obligation in the access control policy rule.

- We then formally define what it means for a system state to be accountable in presence of cascading user obligations. There are two possible interpretations of accountability when considering cascading user obligations. We define both interpretations, *existential* and *universal*, and give motivations for choosing the existential interpretation. We also show that deciding both variations of accountability in presence of cascading user obligations is NP-

hard.

- We propose several special yet practical cases of cascading user obligations for which accountability can be decided efficiently. We propose a polynomial time algorithm for deciding accountability in presence of those special cases of cascading obligations. We also present empirical evaluations of the accountability determination algorithm.

1.5 Roadmap

The thesis is organized in the following way. Chapter 2 briefly overviews the background material necessary to understand the technical contributions of this thesis. In Chapter 3, we introduce both our specification languages (XACML and ***FOPSL***). We then formally specify what it means for an action to be compliant with a policy specified in ***FOPSL*** in Chapter 4. In that section we also show the complexity result of both notions of compliance and also provide mechanism to efficiently check weak compliance. In Chapter 5, we formally define what it means for a policy to have the Δ-property. We also show the complexity of checking the Δ-property of a policy, develop a sound, semi-automated technique for checking Δ-property, and finally use HIPAA as a case study for our policy analysis technique. In Chapter 6, we formally define accountability in presence of cascading obligations, show the complexity of maintaining accountability, and provide an algorithm for checking accountability for special yet practical cases of cascading obligations. We then discuss related work in Chapter 7. Finally, we discuss future work and conclude in Chapter 8.

Chapter 2: BACKGROUND

In this section, we briefly summarize the different background concepts necessary for understanding our technical contributions.

2.1 Temporal Logic

In this section, we briefly overview the different temporal logics used in the formal policy analysis. The notion of temporal logic is important to understand our technical contributions. While defining temporal logic, there are two possible views of time. One of which is where time is viewed as linear (*linear temporal logic* [108, 118]). In this view, for each moment of time there is only one possible future. On the other view, the time is branching and tree like (*computational tree logic* [30, 43]). In the branching view of time, on each moment, time can get split into several possibilities resulting in the situation where each moment of time can have several possible future. The computational tree logic is particularly useful for specifying properties of systems with non-deterministic behavior. Moreover, they are also useful for sanity checking of the models used for verification. We formally specify the syntax and semantics of these logics.

2.1.1 Linear Temporal Logic (LTL) and First Order Temporal Logic (FOTL)

Temporal logic [108] is concerned with characterizing (typically, infinite) sequences of states and/or events. We are principally interested in supporting (many sorted) first-order linear temporal logic (FOTL), though many of the works we will draw upon consider only propositional linear temporal logic (PLTL), in which propositions involve no parameters and quantifiers do not occur.

Linear temporal logics characterize reactive computations in terms of infinite sequences states called *traces*, which we denote by σ. FOTL generalizes propositional linear temporal logic in the same way that first-order logic generalizes propositional logic, namely, by replacing propositional variables by predicate symbols and by introducing quantification and variables. In the formulation

we use, trace elements are states and events are embedded in states. We further discuss the structure of states presently.

Sorts resemble a primitive type system; each variable occurring in a formula is assigned a sort when it is quantified and ranges over values in a unique carrier associated with that sort. Predicate argument positions also have associated sorts with which actual arguments must agree. This association is called the *signature* of the predicate symbol. An FOTL *language* is given by a set of variables, a set of sorts, and a set of predicate symbols over those sorts. In our formulation, events, which induce state transitions, are represented by atomic formulas that hold in the destination state. Our policy language uses only one event predicate, send, which takes a sending agent, a receiving agent, and a message: $\mathrm{send}(p_1, p_2, m)$. The semantics of this event predicate will be made precise later.

FOTL formulas include *non-temporal formulas*, which are constructed from atomic formulas, possibly with variables, logical connectives, and quantifiers over variables, just as in standard many-sorted, first-order logic. As in the latter logic, a variable is *free* if it is not within the scope of any quantification of that variable. A formula is *closed* if it contains no free variables.

FOTL formulas can also contain temporal operators, each of which takes either one or two FOTL subformulas as arguments, depending on the operator. The temporal operators we use are standard and have the following intuitive meanings. Note that, our policy language does not use $\bigcirc$ (next) or $\ominus$ (previous) for technical reasons that will be discussed in a later section. Note that we consider the discrete, point-based semantics of LTL [42]. Moreover, we consider the non-strict versions of the temporal operators where the reasoning of the past and the future operators do not include about present state. *Future Operators.* Henceforth: $\Box\phi$ says that ϕ holds in all future states. Eventually: $\Diamond\phi$ says that ϕ holds in some future state. *Past Operators.* Historically: $\boxminus\phi$ says that ϕ held in all previous states. Once: $\diamondminus\phi$ says that ϕ held in some previous state. Since: $\phi_1 \mathcal{S} \phi_2$ says that ϕ_2 held at some point in the past, and since then ϕ_1 has held in every state; it is sufficient to consider the most recent point at which ϕ_2 held.

A *state* is an interpretation, meaning that each state s is a mapping of predicate symbols p to

relations. Each position in each tuple in $s(p)$ is occupied by a value from the appropriate carrier, based on the signature of p.

A *logical environment* η maps each variable to a value in the carrier that corresponds to the variable's sort (denoted as τ). That a formula ϕ is satisfied by σ at an index i under η is denoted by $\sigma, i, \eta \models \phi$, and can be defined inductively on the structure of ϕ. Note that, we use "rigid" quantification. This signifies that each variable bound to a quantifier ranges over the same domain in all the different worlds. One says that σ satisfies ϕ, written $\sigma \models \phi$, if and only if for all logical environments η, we have $\sigma, 0, \eta \models \phi$.

$$[\![x]\!]\eta = \begin{cases} \eta(x) & \text{if } x : \tau \\ x & \text{otherwise (constants),} \end{cases}$$

We now formally present the syntax and semantics of propositional linear temporal logic (LTL) introduced by Pnueli [90, 108]. This can be extended for the First Order Linear Temporal Logic (FOTL) as we will show later. The propositional linear temporal logic formulas are composed of a finite set AP of atomic propositions, their boolean connectives, and temporal operators. The syntax of LTL can be defined inductively in the following way.

Definition 1 (Syntax of LTL)**.** *Every atomic proposition $p \in AP$ is an LTL formula. When ϕ_1 and ϕ_2 are well-formed LTL formulas, then so are the following:*

- $\top$ *(logical true),* $\bot$ *(logical false)*
- $\neg\phi_1$
- $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$, $\phi_1 \rightarrow \phi_2$, $\phi_1 \iff \phi_2$
- $\phi_1 \mathcal{S} \phi_2$, $\boxminus\phi_1$, $\diamondminus\phi_1$
- $\Diamond\phi_1$, $\Box\phi_1$, $\phi_1 \mathbb{U} \phi_2$

The semantics of temporal logics are often defined in terms of traces of Kripke structures. A *Kripke structure* is a tuple $K = (S, I, R, L, \Omega)$ in which:

- S is a finite set of states.
- $I \subseteq S$ is a set of initial states.
- $R \subseteq S \times S$ is a transition relation that is left-total.
- $L : S \longrightarrow \Omega$ is a function that maps each state to an element of Ω, and Ω is a set the elements of which are called *labels*. Ω can be defined as $\Omega = 2^{AP}$.

In this context, trace σ is a infinite sequence of states, in which for each $i \in \mathbb{N}$, $\sigma_i \in S$, $(s_i, s_{(i+1)}) \in R$, and additionally $\sigma_0 \in I$. The set of infinite and finite sequences of states are given by S^{ω} and S^*, respectively. The semantics of LTL is given with respect to a Kripke structure K, a trace of it σ, and a position in the trace $i \in \mathbb{N}$.

Definition 2 (Semantics of LTL)**.** *The semantics of LTL is given below.*

- $K, \sigma, i \models \top$
- $K, \sigma, i \not\models \bot$
- $K, \sigma, i \models p$ *holds if and only if* $p \in AP$ *and* $p \in L(s_i)$.
- $K, \sigma, i \models \neg\phi_1$ *holds if and only if* $K, \sigma, i \not\models \phi_1$
- $K, \sigma, i \models \phi_1 \wedge \phi_2$ *holds if and only if* $K, \sigma, i \models \phi_1$ *and* $K, \sigma, i \models \phi_2$
- $K, \sigma, i \models \phi_1 \vee \phi_2$ *holds if and only if* $K, \sigma, i \models \phi_1$ *or* $K, \sigma, i \models \phi_2$
- $K, \sigma, i \models \phi_1 \rightarrow \phi_2$ *holds if and only if* $K, \sigma, i \models \phi_1$ *implies that* $K, \sigma, i \models \phi_2$
- $K, \sigma, i \models \phi_1 \Longleftrightarrow \phi_2$ *holds if and only if* $K, \sigma, i \models \phi_1 \rightarrow \phi_2$ *and* $K, \sigma, i \models \phi_2 \rightarrow \phi_1$
- $K, \sigma, i \models \phi_1 \, \mathcal{S} \, \phi_2$ *holds if and only if there exists an* $i_1 \in \mathbb{N}$ *such that* $K, \sigma, i_1 \models \phi_2$ *holds and for all* $i_1 < i_2 \leq i$, $K, \sigma, i_2 \models \phi_1$ *holds.*
- $K, \sigma, i \models \boxminus \phi_1$ *holds if and only if for all* $i_1 \in \mathbb{N}$ *and* $i_1 \leq i$, $K, \sigma, i_1 \models \phi_1$ *holds.*

- $K, \sigma, i \models \diamondsuit \phi_1$ *holds if and only if there exists an* $i_1 \in \mathbb{N}$ *and* $i_1 \leq i$ *such that* $K, \sigma, i_1 \models \phi_1$ *holds.*

- $K, \sigma, i \models \Diamond \phi_1$ *holds if and only if there exists an* $i_1 \in \mathbb{N}$ *and* $i_1 \geq i$ *such that* $K, \sigma, i_1 \models \phi_1$ *holds.*

- $K, \sigma, i \models \Box \phi_1$ *holds if and only if for all* $i_1 \in \mathbb{N}$ *and* $i_1 \geq i$*,* $K, \sigma, i_1 \models \phi_1$ *holds.*

- $K, \sigma, i \models \phi_1 \mathbb{U} \phi_2$ *holds if and only if there exists an* $i_1 \geq i$ *and* $i_1 \in \mathbb{N}$ *such that* $K, \sigma, i_1 \models \phi_2$ *and for all* $i \leq i_2 < i_1$ *and* $i_2 \in \mathbb{N}$*,* $K, \sigma, i_2 \models \Box \phi_1$ *holds.*

We now briefly overview the syntax and semantics of FOTL [64]. Let us assume we have a finite set of predicate symbols represented by R. We have a finite set of constants denoted by C. We denote the finite set of variables with X. As mentioned before, we use η to represent an environment (or, substitution) which maps free variables to values in the appropriate domain. We use D (possibly with subscripts) to represent the associated domain. Although FOTL may also have function symbols, we do not include this in our presentation as our policy language do not allow function symbols. We use $r \in R$ (possibly with subscripts) to represent n-ary predicate symbols, $x \in X$ (possibility with subscripts) to represent variables, we use $c \in C$ (possibly with subscripts) to represent constants. We use t (possibly with subscripts) to represent terms (variables or constants) of the logic. FOTL has the following syntax:

Definition 3 (Syntax of FOTL)**.** *We denote* φ *to represent well-formed FOTL formulas.*

$$\begin{aligned} \varphi \quad ::= \quad & \top \mid \bot \mid r(t_0, t_1, \ldots, t_n) \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \\ & \varphi_1 \mathcal{S} \varphi_2 \mid \forall \vec{x}.(\varphi_1(\vec{x}) \rightarrow \varphi_2(\vec{x})) \mid \exists \vec{x}.\varphi(\vec{x}) \end{aligned}$$

Definition 4 (Semantics of FOTL)**.** *Given a trace* σ*, a position in the trace* $i \in \mathbb{N}$*, an environment* η*, and a formula* φ*, we use* $\sigma, i, \eta \models \varphi$ *to represent that* φ *is satisfied in the* i^{th} *position of the trace* σ *with respect to the environment* η*. This is defined inductively as below. We use* σ_i *to denote*

the i^{th} position of the trace. Note that in our presentation of FOTL semantics we keep the Kripke structure implicit.

- $\sigma, i, \eta \models \top$
- $\sigma, i, \eta \not\models \bot$
- $\sigma, i, \eta \models r(t_0, \ldots, t_n)$ *if and only if* $r_i \in R$ *and* $\eta(r_i)(\eta(t_0), \ldots, \eta(t_n)) \in D_i$.
- $\sigma, i, \eta \models \neg\varphi$ *if and only if* $\sigma, i, \eta \not\models \varphi$
- $\sigma, i, \eta \models \varphi_1 \wedge \varphi_2$ *if and only if* $\sigma, i, \eta \models \varphi_1$ *and* $\sigma, i, \eta \models \varphi_2$.
- $\sigma, i, \eta \models \varphi_1 \vee \varphi_2$ *if and only if* $\sigma, i, \eta \models \varphi_1$ *or* $\sigma, i, \eta \models \varphi_2$.
- $\sigma, i, \eta \models \exists \vec{x}.\varphi(\vec{x})$ *if and only if there exists* $\vec{t}$ *such that* $\sigma, i, \eta[\vec{x} \mapsto \vec{t}] \models \varphi(\vec{x})$.
- $\sigma, i, \eta \models \forall \vec{x}.(\varphi_1(\vec{x}) \rightarrow \varphi_2(\vec{x}))$ *if and only if for all* $\vec{t}$ *if* $\sigma, i, \eta[\vec{x} \mapsto \vec{t}] \models \varphi_1(\vec{x})$ *holds then* $\sigma, i, \eta[\vec{x} \mapsto \vec{t}] \models \varphi_2(\vec{x})$ *holds.*
- $\sigma, i, \eta \models \varphi_1 \, \mathcal{S} \, \varphi_2$ *if and only if there exists* $k \leq i$, *where* $k \in \mathbb{N}$, *such that* $\sigma, k, \eta \models \varphi_2$ *and for all* j, *where* $j \in \mathbb{N}$ *and* $k < j \leq i$, *it implies that* $\sigma, j, \eta \models \varphi_1$ *holds.*

2.1.2 Computational Tree Logic (CTL)

The branching time temporal logic also known as the computational tree logic (CTL) [30, 43, 81], assumes the structure of time to be branching in nature. Each moment of time, in this view, can have multiple possible futures. The structure of time can thus be viewed as an infinite tree like structure. The tree like view of time assumed in CTL, comes handy to reason about a lot of systems. One possible application of this computation-tree like reasoning is sanity checks of the model which is being verified. For instance, to detect vacuous satisfaction of specifications, one might check the satisfiability of a witness formula [74,75], which is basically an existential formula representing a non-trivial behavior of the model. In the same vein, one might check satisfiability

of a possibility property [75, 82], which ensure that computations of the model can be extended to a computation exhibiting the required behaviors. Another application of computation tree like reasoning is in the field of automated task planning [51, 75, 106]. It also useful for reasoning about programs with non-deterministic behavior [81]. Along with the future temporal operators defined above, CTL additionally has universal (A) and existential (E) path quantifiers. CTL syntax allows each such path quantifiers to be followed by a future temporal operator described above. Past temporal operators (*e.g.*, $\mathcal{S}$, $\diamondminus$, $\boxminus$) are not directly incorporated in the logic. The syntax of CTL can be inductively constructed as follows.

Definition 5 (Syntax of CTL)**.** *Let us consider that AP represents a set of atomic propositions. Every atomic proposition $p \in AP$ is a CTL formula. If ϕ_1 and ϕ_2 are CTL formulas then so are the following:*

- $\neg\phi_1$
- $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$, $\phi_1 \rightarrow \phi_2$, $\phi_1 \Longleftrightarrow \phi_2$
- $A\Box\phi_1$, $E\Box\phi_1$, $A\diamondsuit\phi_1$, $E\diamondsuit\phi_1$, $A\bigcirc\phi_1$, $E\bigcirc\phi_1$
- $A[\phi_1 \mathbb{U} \phi_2]$, $E[\phi_1 \mathbb{U} \phi_2]$

Note that in the literature F and G are used to denote the temporal operators eventually ($\diamondsuit$) and henceforth ($\Box$), respectively.

Now, we turn our attention to the formal semantics of the logic CTL. A CTL formula is interpreted with respect to a *labeled state transition graph M*. In this structure, states are labeled with atomic propositions. Note that in the same vein CTL formula can be interpreted over kripke structures. De Nicola and Vaandrager has shown that there are correspondence between kripke structures and labeled transition graphs [32]. They have provided translation from one to another.

Definition 6 (Labeled State Transition Graph)**.** *A labeled state transition graph is a tuple $M = (S, R, L)$ where:*

- *S is the finite set of states.*
- *$R \subseteq S \times S$ represents the state transition relation that is left total.*
- *$L : S \longrightarrow 2^{AP}$ is the labeling function that assigns each state a set of propositions that are true in that state.*

Before we can formally specify the semantics of CTL, we should first introduce the following concepts. A *full path* (or, just path) π in the structure $M = (S, R, L)$ is an infinite sequence of states $\pi = (s_0, s_1, \ldots)$ such that $\forall i.(s_i \in S)$ and $\forall i.(s_i, s_{i+1}) \in R$. Additionally, we use $M, s \models \phi$ to denote that the formula ϕ holds true in the state s of the structure $M = (S, R, L)$ where .

Definition 7 (Semantics of CTL)**.** *We now inductively define the $\models$ relation as follows.*

- *$M, s \models p$ holds if and only if $p \in L(S)$.*
- *$M, s \models \neg\phi_1$ holds if and only $M, s \not\models \phi_1$.*
- *$M, s \models \phi_1 \wedge \phi_2$ holds if and only if $M, s \models \phi_1$ and $M, s \models \phi_2$.*
- *$M, s \models \phi_1 \vee \phi_2$ holds if and only if $M, s \models \phi_1$ or $M, s \models \phi_2$.*
- *$M, s \models \phi_1 \rightarrow \phi_2$ holds if and only if $M, s \models \phi_1$ implies that $M, s \models \phi_2$.*
- *$M, s \models \phi_1 \Longleftrightarrow \phi_2$ holds if and only if $M, s \models \phi_1 \rightarrow \phi_2$ and $M, s \models \phi_2 \rightarrow \phi_1$.*
- *$M, s \models A\bigcirc\phi_1$ holds if and only if for all $\hat{s}$ such that $(s, \hat{s}) \in R$ and $M, \hat{s} \models \phi_1$.*
- *$M, s \models E\bigcirc\phi_1$ holds if and only if there exists a $\hat{s}$ such that $(s, \hat{s}) \in R$ and $M, \hat{s} \models \phi_1$.*
- *$M, s \models A\Box\phi_1$ holds if and only if for all full paths $\pi = (s_0, s_1, \ldots)$ such that $s_0 = s$ and for all $i \in \mathbb{N}$, $M, s_i \models \phi_1$ holds.*
- *$M, s \models E\Box\phi_1$ holds if and only if there exist a full path $\pi = (s_0, s_1, \ldots)$ such that $s_0 = s$ and for all $i \in \mathbb{N}$, $M, s_i \models \phi_1$ holds.*

- $M,s \models A\Diamond\phi_1$ *holds if and only if for all full paths* $\pi = (s_0, s_1, \ldots)$ *such that* $s_0 = s$ *and there exists an* $i \in \mathbb{N}$ *such that* $M, s_i \models \phi_1$ *holds.*

- $M,s \models E\Diamond\phi_1$ *holds if and only if there exists a full path* $\pi = (s_0, s_1, \ldots)$ *such that* $s_0 = s$ *and there exists an* $i \in \mathbb{N}$ *such that* $M, s_i \models \phi_1$ *holds.*

- $M,s \models A[\phi_1 \mathbb{U} \phi_2]$ *holds if and only if for all full paths* $\pi = (s_0, s_1, \ldots)$ *such that* $s_0 = s$ *there exists an* $i \geq 0$ *such that* $M, s_i \models \phi_2$ *and for all* j *such that* $0 \leq j < i$ *that* $M, s_j \models \phi_i$ *holds.*

- $M,s \models E[\phi_1 \mathbb{U} \phi_2]$ *holds if and only if there exists a full path* $\pi = (s_0, s_1, \ldots)$ *such that* $s_0 = s$ *there exists an* $i \geq 0$ *such that* $M, s_i \models \phi_2$ *and for all* j *such that* $0 \leq j < i$ *that* $M, s_j \models \phi_i$ *holds.*

Expressive power. The expressive power of CTL and LTL are not comparable [44, 81]. There are requirements (or, specifications) that can be easily expressed in one but not the another. For instance, consider the LTL formula $\Diamond\Box p$ which specifies that there is a point in time in the future after which p will always hold true. One cannot express this requirement in CTL. Additionally consider the CTL formula $A\Box(E\Diamond p)$ which specifies that for all paths and each point of time in it, there will be a path from that point where p would eventually hold true. There is no LTL equivalent of this formula.

2.1.2.1 The Logic CTL*

As we have seen in the previous section, the expressive power of LTL and CTL are not comparable. As a result, Emerson and Halpern [44] have introduced CTL* which is also known as the full branching time temporal logic. The logic CTL* is a superset of both CTL and LTL. More precisely, every LTL and every CTL formula is a well-formed formula of the logic CTL*. Note that the semantics of both LTL and CTL formula remains the same when they are interpreted as a CTL* formula. CTL* generalizes CTL by allowing each path quantifiers to be followed by a LTL formula (path formula). We now formally define the syntax and semantics of CTL* [44].

There are two types of formulas in CTL*, namely, *state formulas* and *path formulas*. State formulas are true in a specific state whereas path formulas are true in a specific path. Let us consider that AP represents the set of atomic propositions.

Definition 8 (Syntax of CTL*)**.** *CTL* formulas are state formulas generated in the following way.*

- ***State Formulas****: Every atomic proposition $p \in AP$ is a state formula.*
  - *If ϕ_1 and ϕ_2 are state formulas, then so are : $\neg\phi_1$, $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$.*
  - *If ψ is a path formula then $A\psi$ and $E\psi$ are state formulas.*
- ***Path Formulas****: Every state formula ϕ is also a path formula. If ψ_1 and ψ_2 are path formulas then so are the following.*
  - $\neg\psi_1$, $\psi_1 \wedge \psi_2$, $\psi_1 \vee \psi_2$
  - $\bigcirc\psi_1$, $\Diamond\psi_1$, $\Box\psi_1$, $\psi_1 \mathbb{U} \psi_2$

Similar to CTL, CTL* formulas are interpreted with respect to a labeled state transition graph $M = (S, R, L)$. Furthermore, we use $\pi = (s_0, s_1, \ldots)$ to denote a path where for all $i \geq 0$, $(s_i, s_{i+1}) \in R$. We use π^i to denote the suffix of the path π starting at s_i. We use $M, s \models \phi$ to denote that the state formula ϕ holds in the state s of structure M. Similarly, we use $M, \pi \models \psi$ to denote that the path formula ψ holds in the path π of structure M.

Definition 9 (Semantics of CTL*)**.** *Let us consider ϕ_1 and ϕ_2 to be state formulas whereas ψ_1 and ψ_2 are path formulas. The relation $\models$ can be defined inductively as follows.*

- ***State Formulas****:*
  - *$M, s \models p$ if and only if $p \in L(s)$.*
  - *$M, s \models \neg\phi_1$ if and only if $M, s \not\models \phi_1$*
  - *$M, s \models \phi_1 \wedge \phi_2$ if and only if $M, s \models \phi_1$ and $M, s \models \phi_2$*

- $M, s \models E(\phi_1)$ *if and only if there exists a path* $\pi = (s_0, s_1, \ldots)$ *such that* $s_0 = s$ *and* $M, \pi \models \phi_1$.

- ***Path Formulas**:*

  - $M, \pi \models \neg\psi_1$ *if and only if* $M, \pi \not\models \psi_1$.
  - $M, \pi \models \psi_1 \wedge \psi_2$ *if and only if* $M, \pi \models \psi_1$ *and* $M, \pi \models \psi_2$.
  - $M, \pi \models \bigcirc\psi_1$ *if and only if* $M, \pi^1 \models \psi_1$.
  - $M, \pi \models \psi_1 \mathbb{U} \psi_2$ *if and only if there exists a* $k \geq 0$ *such that* $M, \pi^k \models \psi_2$ *and for all* $0 \leq j < k$ $M, \pi^j \models \phi_1$ *holds.*

The semantics of other CTL* formulas can be easily derived from the semantics above using the following equivalences. $\Diamond\psi \equiv \top \mathbb{U} \psi$. $\Box\psi \equiv \neg\Diamond\neg\psi$. $A(\phi) \equiv \neg E \neg\phi$. $\phi_1 \vee \phi_2 \equiv \neg(\neg\phi_1 \wedge \neg\phi_2)$. $\phi_1 \rightarrow \phi_2 \equiv \neg\phi_1 \vee \phi_2$. $\phi_1 \leftrightarrow \phi_2 \equiv \phi_1 \rightarrow \phi_2 \wedge \phi_2 \rightarrow \phi_1$.

2.1.2.2 The Logic CTL* with Linear Past

As we have seen already, it is not trivial to add past temporal operators in CTL*. Adding past time temporal operators to LTL does not increase the expressive power of LTL [53] but makes it exponentially more succinct [84]. Kupferman *et al.* [76] extends branching time temporal logic CTL* with past time temporal operators. There are two possible views of past in a branching time model, namely, *branching past* and *linear past*. In the branching past view, each moment of time can have several futures and several possible pasts. In the linear past view, each moment of time can have several possible futures but a unique past. In both views however past is assumed to be finite. For our purposes, we chose the linear past view. We call the extended CTL* logic with linear past, CTL^*_{lp}. Note that, Kupferman *et al.* [76] have shown that adding linear past to CTL* does not increase the expressive power of CTL* whereas adding the branching past to CTL* increases the expressive power of CTL*.

Similar to CTL*, CTL^*_{lp} has two types of formulas, namely, state formulas and path formulas. Let us consider that AP represents the set of atomic propositions.

Definition 10 (Syntax of CTL*). *CTL^*_{lp} formulas are state formulas generated in the following way.*

- ***State Formulas**: Every atomic proposition $p \in AP$ is a state formula.*
  - *If ϕ_1 and ϕ_2 are state formulas, then so are : $\neg\phi_1$, $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$.*
  - *If ψ is a path formula then $A\psi$ and $E\psi$ are state formulas.*
- ***Path Formulas**: Every state formula ϕ is also a path formula. If ψ_1 and ψ_2 are path formulas then so are the following.*
  - $\neg\psi_1$, $\psi_1 \wedge \psi_2$, $\psi_1 \vee \psi_2$
  - $\bigcirc\psi_1$, $\Diamond\psi_1$, $\Box\psi_1$, $\psi_1 \mathbb{U} \psi_2$
  - $\ominus\psi_1$, $\diamondminus\psi_1$, $\boxminus\psi_1$, $\psi_1 \mathcal{S} \psi_2$

The semantics of CTL^*_{lp} we use is inspired by Kupferman *et al.* [76] and is defined with respect to a variation of the Kripke structure called *computation trees*. The key feature of computation trees is that each state (called a "node") has exactly one path reaching it from the start state. This is essential for expressing linear past. Note that the computation tree can be viewed as the unwinding of a given Kripke structure [76]. This is due to the fact that CTL^*_{lp} is not sensitive to unwinding unlike its branching past counterpart. We now formally define computation trees and use it to specify the semantics of CTL^*_{lp}.

One way of constructing computation trees is as follows. Let $\mathcal{D}$ be a set the elements of which are called *directions*. A *$\mathcal{D}$-tree* is a set $\mathbb{T} \subseteq \mathcal{D}^*$ such that for all $x \cdot c \in \mathbb{T}$ in which $x \in \mathcal{D}^*$ and $c \in \mathcal{D}$, $x \in \mathbb{T}$ also holds. The elements of $\mathbb{T}$ are called *nodes*; the empty string ε is called the *root* of $\mathbb{T}$. For every $x \in \mathbb{T}$, the nodes $x \cdot c \in \mathbb{T}$ where $c \in \mathcal{D}$ are the successors of x. We consider here trees in which each node has at least one successor. A *path* ρ of a tree $\mathbb{T}$ is a set $\rho \subseteq \mathbb{T}$ such that $\varepsilon \in \rho$ and for every $x \in \rho$ there exists a unique $c \in \mathcal{D}$ such that $x \cdot c \in \rho$. For a path ρ and $j \geq 0$, let ρ_j denote the node of length j in ρ. Given a set $\mathcal{D}$ of directions and a set $\sum$ of alphabets, a *$\sum$-labeled $\mathcal{D}$ tree* is a pair $\langle \mathbb{T}, \mathcal{L} \rangle$ where $\mathbb{T}$ is a $\mathcal{D}$-tree and $\mathcal{L} : \mathbb{T} \to \sum$ is a function that takes an element of $\mathbb{T}$

and maps it to an element of $\sum$. A computation tree is a $\sum$-*labeled* $\mathcal{D}$ *tree* with $\sum = 2^{AP}$ for some set of atomic propositions. The set of directions $\mathcal{D}$ is arbitrary and we can use $\mathcal{D} = \mathbb{N}$ where $\mathbb{N}$ represents the set of natural numbers.

We now formally define the semantics of CTL^*_{lp} with respect to a computation tree $\langle \mathcal{T}, \mathcal{L} \rangle$. We use $x \models \phi$ to represent that a state formula ϕ holds in the node $x \in \mathcal{T}$. We use $\rho, i \models \psi$ to denote that a path formula ψ holds in the position i of the path $\rho \subseteq \mathcal{T}$.

Definition 11 (Semantics of CTL^*_{lp})**.** *Let us consider* ϕ_1 *and* ϕ_2 *to be state formulas whereas* ψ_1 *and* ψ_2 *are path formulas. The relation* $\models$ *can be defined inductively as follows.*

- ***State Formulas****:*
  - $x \models p$ *if and only if* $p \in \mathcal{L}(x)$.
  - $x \models \neg\phi_1$ *if and only if* $x \not\models \phi_1$
  - $x \models \phi_1 \wedge \phi_2$ *if and only if* $x \models \phi_1$ *and* $x \models \phi_2$
  - $x \models \phi_1 \vee \phi_2$ *if and only if* $x \models \phi_1$ *or* $x \models \phi_2$
  - $x \models \phi_1 \rightarrow \phi_2$ *if and only if when* $x \models \phi_1$ *holds then* $x \models \phi_2$ *holds*
  - $x \models E(\phi_1)$ *if and only if there exists a path* ρ *and an* $i \geq 0$ *such that* $\rho_i = x$ *and* $\rho, i \models \phi_1$.
  - $x \models A(\phi_1)$ *if and only if for all paths* ρ *and there exists* $i \geq 0$ *such that* $\rho_i = x$ *and* $\rho, i \models \phi_1$.
- ***Path Formulas****:*
  - $\rho, i \models \phi_1$ *for a state formula* ϕ_1 *if and only if* $\rho_i \models \phi_1$.
  - $\rho, i \models \neg\psi_1$ *if and only if* $\rho, i \not\models \psi_i$.
  - $\rho, i \models \psi_1 \wedge \psi_2$ *if and only if* $\rho, i \models \psi_1$ *and* $\rho, i \models \psi_2$.
  - $\rho, i \models \bigcirc\psi_1$ *if and only if* $\rho, i+1 \models \psi_1$.
  - $\rho, i \models \ominus\psi_1$ *if and only if* $i \geq 0$ *and* $\rho, i-1 \models \psi_1$.

– $\rho, i \models \psi_1 \,\mathbb{U}\, \psi_2$ *if and only if there exists* $k \geq i$ *such that* $\rho, k \models \psi_2$ *and for all* $i \leq j < k$ *so that* $\rho, j \models \psi_1$.

– $\rho, i \models \psi_1 \,\mathcal{S}\, \psi_2$ *if and only if there exists* $0 \leq k \leq i$ *such that* $\rho, k \models \psi_2$ *and for all* $k < j \leq i$ *so that* $\rho, j \models \psi_1$.

The semantics of other CTL^*_{lp} formulas can be easily derived from the semantics above and the following equivalences. $\diamondsuit\psi \equiv \top \,\mathcal{S}\, \psi$. $\boxminus\psi \equiv \neg\diamondsuit\neg\psi$. $\diamondsuit\psi \equiv \top \,\mathbb{U}\, \psi$. $\square\psi \equiv \neg\diamondsuit\neg\psi$. $\phi_1 \leftrightarrow \phi_2 \equiv (\phi_1 \rightarrow \phi_2) \wedge (\phi_2 \rightarrow \phi_1)$.

The logic CTL^*_{lp} can be extended to FO-CTL^*_{lp} in the same way how first order logic generalizes propositional logic.

2.2 Safety and Liveness Properties

The information systems we consider can record and transmit individuals' private data. They can be modeled as reactive systems [58]. The system's behavior can in turn be characterized in terms of sets of infinite sequences of events and/or states. Such a sequence is called a *trace*. Sets of traces are called *temporal properties* [24]. Every temporal property is given by the intersection of two properties, one of which is called a safety property, the other a liveness property [5, 6]. We now briefly summarize these properties (safety and liveness).

2.2.1 Safety Properties.

Safety properties [80] are temporal properties that say some bad thing never happens. When the safety property is violated, the violation can be detected within a finite number steps of when the actual violation occurs. As a result, safety properties are amenable to efficient enforcement through classical techniques like monitoring [60, 85, 115]. An example of a safety property can be the following requirement, which mentions that a clinic can disclose a patient's medical records to a third party provided that the patient has authorized this disclosure. When the clinic attempts to disclose a patient's medical records to a third party and it has not received any authorization from

the patient in this regard, then the disclosure action would violate this safety property. We can thus efficiently enforce this policy by denying the disclosure action of the clinic when it has not received an authorization from the patient.

Definition 12 (Safety Property Definition by Alpern and Schneider [6])**.** *A property P is a safety property if the following holds:* $\forall \sigma : \sigma \in S^{\omega} : (\sigma \models P \Longleftrightarrow (\forall i : i \geq 0 : (\exists \beta : \beta \in S^{\omega} : \sigma[\ldots i] \cdot \beta \models P)))$. *Here,* S^{ω} *represents the set of infinite sequences, each element of which, is an element of the state set S. Moreover,* $\sigma[\ldots i]$ *represents the finite prefix of length* $i+1$ *of* σ.

2.2.2 Liveness Properties.

Liveness properties [80] are temporal properties that say something good will eventually happen. The violation of the liveness properties cannot be detected within a finite number of steps. Thus, it is not amenable to enforcement through classical monitoring. All LTL representable liveness properties can be expressed as an LTL formula of the form $\Box\Diamond p$ in which p is a pure past LTL formula. The formula $\Box\Diamond p$ signifies that p happens infinitely often. However, policies of our form do not allow arbitrary liveness properties of the form $\Box\Diamond p$. We allow response properties in our policies which are restricted variations of general liveness properties. They have the form $\Box(a \rightarrow \Diamond b)$ and specify that whenever a is true then b will become true eventually. The proposition b can be viewed as an obligatory requirement (obligation) [109] when the triggering condition a becomes true. Note that in the general response properties a and b are arbitrary past formula. However, in our policies a and b are of specific form which we discuss later. We want to emphasize that in the response property ($\Box(a \rightarrow \Diamond b)$) when a is logical true then the response property reduces to the general liveness property of form $\Box\Diamond b$. An example of a response property can be the following which specifies that when a patient requests a copy of her medical records, the clinic will provide them to the patient. The patient requesting can be viewed as a becoming true and the clinic responding can be viewed as b becoming true.

Definition 13 (Liveness Property Definition by Alpern and Schneider [6])**.** *A property P is a liveness property if the following holds:* $\forall \alpha : \alpha \in S^{*} : (\exists \beta : \beta \in S^{\omega} : \alpha \cdot \beta \models P)$. *Here,* S^{*} *represents the*

set of finite sequences, each element of which, is an element of the state set S.

Violations of liveness properties, and hence also response properties cannot be detected within a finite number of steps. To mitigate this problem and for making enforcement feasible, sometimes *bounded liveness* is used. Bounded liveness is actually a safety property and it requires that the obligatory requirement be fulfilled within a certain number of steps or certain period of time, making detection of violation feasible. If we rewrite the privacy rule (1) (discussed above) to require that the clinic responds to the patient's request within 30 days, it would be an example of a bounded liveness property.

2.3 Model Checking

Model checking [29, 123] is used to automatically and formally verify finite concurrent system. It takes an input a finite concurrent system and a desired property. It then checks to see whether the finite concurrent system satisfies the given property. The finite system is generally modeled as finite state machines (FSM). As the specification language of the property, different temporal logics can be used (*e.g.*, LTL, CTL, CTL*). Generally, the (explicit) model checking algorithms exhaustively explore the reachable state space of the finite concurrent system based on the transition relation to determine if a given property holds. In the case, the finite concurrent system does not adhere to the desired property, the model checking generates an example trace as a counter example that demonstrates how the finite concurrent system falsifies the desired property. This counter example can then be used to enhance and refine the model so that it satisfies the desired property.

Table 2.1: Complexity of Model Checking For Different Property Specification Logic

Logic	Model Checking Complexity (with respect to formula size)
LTL	PSPACE-complete
CTL	PTIME-complete
CTL*	PSPACE-complete
CTL^*_{lp}	EXPSPACE

One of the disadvantages of using model checking to verify a certain finite state system is its

high time complexity. It is inherent to model checking as it explores all possible runs of the system to find example of the property falsification. In the most of the cases, the reachable state space is too big to be explored causing "*state explosion*". There are several techniques, including abstraction, partial order reductions, bisimulation, that have been proposed to deal with this state explosion problem possibly sacrificing soundness or completeness. When successful, model checking gives static assurances that whenever the model-checked finite state system is executed it will not violate the desired properties. As a result of which, there is no need for any dynamic checking to see whether the system is violating the desired properties. Model checking can be used to verify both safety and liveness properties. In the table 2.1, we show the complexity of performing model checking for various specification language.

2.4 Runtime Monitoring

Recall that, model checking is an automated algorithmic verification technique that statically checks to see whether a concurrent finite system specified as finite state machines satisfies a given property, typically specified as a temporal logic formula. The specification it can verify can be either a safety property, a liveness property, or a combination of the two. Model checking algorithms checks all possible runs of the system by constructing the reachable state space and inspects it. The reachable state space can be large causing the *state explosion problem*.

One possible alternative that has got a lot of attention from both industry and academia is "runtime monitoring" [60, 85, 115]. Model checking gives static guarantee that whenever the finite concurrent system is executed it will not violate the desired properties whereas in the runtime monitoring approach there is a separate component called a "monitor" that inspects the behavior of the finite concurrent system. It suppresses or prohibits those actions/events requested by the finite concurrent system that violates the desired property. In that sense, it is one possibility of guaranteeing that the finite concurrent system does not violate the desired property. However, there are several disadvantages of this approach. The first is the overhead of instrumenting the finite concurrent system to emit necessary events that can be monitored by the runtime component,

the monitor. The next disadvantage is that a runtime monitor can typically enforce only safety properties. Thus, the runtime monitoring technique does not give any guarantees about liveness properties.

Havelund and Rośu [60, 61] propose a dynamic programming based algorithm for monitoring pure past, propositional linear temporal logic formula. Note that, a pure past propositional LTL formula can only specify safety properties. They try to verify whether a pure past, propositional LTL formula satisfies a given observed history (finite run of the system). More formally, for a given pure past, propositional LTL formula ϕ they check to see whether a given finite history h and a position in it $i \in \mathbb{N}$ satisfies ϕ written as $h, m \models \phi$. They use the intuition that pure past LTL formula has a recursive semantics. More precisely, deciding $h, m \models \phi$ can be done easily if one knows: (1) the truth value of $h, m-1 \models \phi_j$ for all proper subformulas ϕ_j of ϕ and (2) the truth value of $h, m \models \phi_i$ for all proper subformulas ϕ_i of ϕ. As a result of which, they use two bit arrays, *old* and *new*, each of size $|\phi|$ where ϕ is the formula they are monitoring. Each entry of the bit array *old* contains the truth value of a proper subformula of ϕ in the previous step (1 means *true* and 0 means *false*). Each entry of the bit array *new* contains the truth value of a proper subformula of ϕ in the current step. Given the value of the *old* bit array, the *new* bit array can be constructed inductively in the following way. Let us consider that we use *AP* to denote the set of atomic propositions in the formula ϕ and υ represents the set of propositions true in the current state.

$$new[p] = (p \in \upsilon) \; when \; p \in AP$$

$$new[\phi_1 \wedge \phi_2] = new[\phi_1] \wedge new[\phi_2]$$

$$new[\neg\phi] = \neg new[\phi]$$

$$new[\ominus\phi] = old[\phi]$$

$$new[\boxminus\phi] = old[\boxminus\phi] \wedge new[\phi]$$

$$new[\diamondminus\phi] = old[\diamondminus\phi] \vee new[\phi]$$

$$new[\phi_1 \mathcal{S} \phi_2] = new[\phi_2] \vee (old[\phi_1 \mathcal{S} \phi_2] \wedge new[\phi_1])$$

There are other automata based runtime monitoring techniques (*e.g.*, security automata [14,48, 66,92,115], edit automata [88], *etc.*) but they have a large overhead.

2.5 Overview of HIPAA

We now briefly overview the Health Insurance Portability and Accountability Act (HIPAA) of 1996 also referred to as Public Law 104-191. The Department of Health and Human Services (HHS) is the responsible organization for enforcing HIPAA. According to the Department of Health and Human Services (HHS) the goal of the HIPAA privacy regulation is to ensure that consumers can access their health information and also to protect their information from unauthorized disclosure.

More specifically, Part 164 of HIPAA deals with the security and privacy aspect of the regulation. In this work, we primarily analyze subpart E of Part 164, which deals with protecting individually identifiable health information, covering §164.502 to §164.528. These rules precisely specify the security and privacy requirements that is applicable to "*covered entities*" with respect to protected health information (*PHI*). As defined by HIPAA and the HHS, covered entities include health plans, health care clearinghouses, such as billing services and community health information systems, and health care providers that transmit health care data in a way that is regulated by HIPAA. *Protected health information* (*PHI*) refers to individually identifiable health information

except a few cases where such information falls under the jurisdiction of other federal regulations such as the Family Educational Rights and Privacy Act (FERPA).

The privacy rules regulate the following kinds of actions: (1) Usage of the *PHI* within the covered entity itself. (2) Disclosure of *PHI* to some other entity.

Furthermore, the purposes for which the covered entity (or, any other entity) is using or disclosing the *PHI* is also referred in the privacy rules. The following is an incomplete list of purposes that are used: treatment; payment; health care operations; creating de-identified *PHI*; communicate; marketing; reporting to public health authority; health oversight.

When disclosing *PHI*, the role of the entity to which the disclosure is made is also important. An incomplete list of the different roles that are referred in the HIPAA rules include: individual (i.e., the person whose *PHI* is about); representative of an individual; business associates of a covered entity; healthcare provider; an attorney representing whistleblower; group health plan; Health Maintenance Organization (HMO); public health care authority; public health or government authority authorized by law to receive child abuse report; a person who may have been exposed to a communicable disease; employer of an individual; a family member, other relative, or a close personal friend of an individual, or any other person identified by the individual; a person subject to Food and Drug Administration (FDA) regulated activity.

The HIPAA rules also often refer to other documents and contracts among the entities involved with the applicable disclosure. We briefly summarize these documents and contracts in the following discussion.

Privacy notice. According to the privacy rules, when a privacy notice is required, an access must be consistent with the privacy notice, in addition to following privacy rule (§164.502(i)). "*A covered entity that is required by §164.520 to have a notice may not use or disclose protected health information in a manner inconsistent with such notice. A covered entity that is required by §164.520(b)(1)(iii) to include a specific statement in its notice if it intends to engage in an activity listed in §164.520(b)(1)(iii)(A)-(C), may not use or disclose protected health information*

for such activities, unless the required statement is included in the notice." According to the above clause, an organization must check each access against the privacy notice. Therefore, privacy notices should be encoded as policies that must also authorize a request for it to be allowed. When an organization has multiple privacy notices (for example, Google had over 70 different privacy policies before consolidating them), then it is necessary to remember for each patient which policy encodes the privacy notice for that patient and checks with that policy.

Authorizations. Accesses (use or disclose) to *PHI* not explicitly authorized by the privacy rules can still be allowed when a valid authorization from the individual is obtained for the specified purpose. This is explicitly mentioned in the privacy rule §164.508 which specifies that: "*Except as otherwise permitted or required by this subchapter, a covered entity may not use or disclose protected health information without an authorization that is valid under this section. When a covered entity obtains or receives a valid authorization for its use or disclosure of protected health information, such use or disclosure must be consistent with such authorization*." To enforce this clause, authorizations signed by individuals also need to be encoded as policies and checked against.

Contracts and Restrictions. The HIPAA privacy rules in §164.522 requires that a covered entity must permit an individual to request that the covered entity restrict: (A) Uses or disclosures of protected health information about the individual to carry out treatment, payment, or health care operations; and (B) Disclosures permitted under §164.510(b). A covered entity is not required to agree to a restriction, but if it agrees to it, it must respect the restriction.

Furthermore, the HIPAA privacy rules in §164.510 allows a covered entity to use or disclose protected health information without the written consent or authorization of the individual as described by §164.506 and §164.508, respectively, provided that the individual is informed in advance of the use or disclosure and has the opportunity to agree to or prohibit or restrict the disclosure in accordance with the applicable requirements of this section. The covered entity may orally inform the individual and obtain the individual's oral agreement or objection to a use or disclosure permitted by this section.

2.6 XACML

Architecture. The main components of the XACML architecture include a Policy Enforcement Point (PEP), a Policy Decision Point (PDP), a Policy Information Point (PIP), a Policy Administration Point (PAP), and obligations service. The **PEP** performs access control, by receiving decision requests, consulting the PDP for authorization decision, and enforcing the decisions. The **PDP** evaluates applicable policies and yields authorization decisions, together with obligations and advice, if any. The **PIP** acts as a source of attribute values, such as subject, resource, action, environment attributes. The **PAP** administrates policies and policy sets and makes them available to the PDP. The **obligations service** handles obligations forwarded by the PEP. However, XACML does not specify how PIP, PAP and obligations service should behave and how they should be implemented.

Rules, Policies, and Policy-sets. Both XACML 2.0 and 3.0 [127] define three levels of policy elements: rules, policies, and policy-sets. A *rule* is the most basic policy element; it has three main components: a *target*, a *condition*, and an *effect*. The target defines a set of subjects, resources, and actions that the rule applies to; the condition specifies restrictions on the attributes in the target and refines the applicability of the rule; the effect is either `Permit`, in which case we call the rule a *permit rule*, or `Deny`, in which case we call it a *deny rule*. If a request satisfies both the rule target and rule condition, the rule *is applicable* to the request and yields the decision specified by the effect element; otherwise, the rule *is not applicable* to the request and yields the decision `NotApplicable`. Note that when an error occurs while evaluating a rule, then the decision `Indeterminate` is returned as a decision. There are other situations when `Indeterminate` is returned as a decision. They are discussed below.

Unlike XACML 2.0, XACML 3.0 allows one to specify obligations and advice in a rule, so the rule would return a decision together with a set of (possibly empty) obligations and advice if it is applicable to a request. Each obligation represents functions to be executed in conjunction with the enforcement of an authorization decision. *Advice* is newly added in XACML 3.0, which is a

```
<PolicySet> := <Target><Policy>+[Obligations]
 Attributes: PolicySetId, PolicyCombiningAlgId
<Policy> := <Target><Rule>+[Obligations]
 Attributes: PolicyId, RuleCombiningAlgId
<Rule> := [Target][Condition]
 Attributes: RuleId, Effect
```

Figure 2.1: XACML schema of policy set, policy and rule in BNF form

supplementary piece of information provided together with a decision, and it is like an optional obligation, which can be safely ignored by the PEP.

A *policy* consists of four main components: a *target*, a *rule-combining algorithm (RCA)*, a set of *rules*, and *obligations/advice*. The policy target decides whether a request is applicable to the policy and it has a similar structure as the rule target. The RCA specifies how the decisions from the rules are combined to yield one decision. A *policy-set* also has four main components: a *target*, a *policy-combining algorithm (PCA)*, a set of *sub-policies*, and *obligations/advice*. A sub-policy can be either be a policy or a policy-set. The PCA specifies how the results of evaluating the sub-policies are combined to yield a decision. Figure 2.1 shows the schema of policy set, policy and rule in BNF form of the base XACML specification language.

Policy Combining Algorithms. XACML 2.0 and 3.0 have a number of standard RCAs and PCAs. They are "*Deny-overrides*", "*Ordered-deny-overrides*", "*Permit-overrides*", "*Ordered-permit-overrides*", "*First-applicable*", and "*Only-one-applicable*" ("Only-one-applicable" is only defined as a PCA. Ordered-deny-overrides and ordered-permit-overrides are the same as deny-overrides and permit-overrides, respectively, except that rules and policies have to be evaluated in the order they appear. XACML 3.0 redefines "*Deny-overrides*", "*Permit-overrides*" and their ordered versions, considering uncertainty when handling errors. Also, XACML 3.0 introduces two

new PCAs: "*deny-unless- permit*" and "*Permit-unless-deny*". As a result, XACML 3.0 has eleven RCAs and twelve PCAs. Among the RCAs and PCAs, "Permit-overrides", "Deny-overrides", and "First-applicable" are helpful and sufficient for combining HIPAA policies/rules.

XACML "Permit-overrides" PCA has the preference `Permit` > `Deny` > `Indeterminate` > `NotApplicable`. That is, when any sub-policy permits the request, the policy as a whole permits it. When no sub-policy permits the request, and at least one denies it, the policy as a whole denies it. Otherwise, when there is an error somewhere, the policy reports error on the request. Otherwise, the policy is non-applicable. The "Deny-overrides" PCA uses the preference `Deny` > `Permit` > `NotApplicable`; in addition, it treats `Indeterminate` as always equivalent to `Deny`. That is, whenever a sub-policy returns `Indeterminate`, the policy would return `Deny`. The "First-applicable" PCA returns the effect of the first applicable sub-policy as the result if no errors occur. Whenever an error occurs, the policy returns `Indeterminate`. The "Only-one-applicable" PCA returns the effect of the unique policy in the policy-set that applies to the request. If there are more than one applicable policies, the PCA reports the conflict by returning `Indeterminate`. Furthermore, if an error occurs during evaluation of any policy, the PCA also returns `Indeterminate`.

Chapter 3: PRIVACY POLICY SPECIFICATION LANGUAGE

Privacy regulations like HIPAA [62], GLBA [2], and SOX [116] are specified in natural language. To develop an efficient compliance checking algorithm that can automatically check whether a certain action is compliant with the applicable privacy policies or to give a formal guarantee that a policy permits any incurred obligation, it is first necessary to express the privacy requirements imposed by the regulation in some policy specification language. Several frameworks have been proposed for specifying and analyzing privacy policies [3, 11–13, 20, 31, 35, 37, 55, 73, 79, 94]. To this end, we consider two possible policy specification languages. One candidate privacy policy specification language we consider is a generic, off-the-shelf access control policy specification language, XACML [127]. *OASIS's eXtensible Access Control Markup Language* (*XACML*) [127] is one of the most popular access control specification languages. Along with the rich specification language, XACML [127] also has a robust enforcement engine that can enforce policies specified in the language. Although XACML is an expressive specification language, it lacks features needed to specify HIPAA-like privacy policies. This is natural as XACML is designed for specifying access control policies instead of privacy policies like HIPAA. We thus assess XACML's adequacy for expressing HIPAA. More precisely, we investigate what features a specification language requires to sufficiently specify HIPAA. We discuss which of these necessary features XACML possesses and also propose extensions of XACML to support the missing features.

One of the apparent advantages of extending XACML to support privacy policies like HIPAA is that one uniform specification language and enforcement mechanism can be used to specify and enforce the access control policies and the privacy policies of the system. Managing the access control policies and the privacy policies differently is cumbersome as an action can be mandated by both policies. However, if we use XACML for expressing both policies, then XACML's enforcement mechanism will combine the separate permissibility decisions of an action by using

The content of this chapter is based on the joint work with Ninghui Li, Haining Chen, Elisa Bertino, Andreas Gampe, Jianwei Niu, Jeffery von Ronne, Jared Bennatt, Anupam Datta, Limin Jia, and William H. Winsborough [26], [27].

policy combination algorithms (PCAs). Furthermore, organizations can have their own business privacy policy on top of the federal privacy regulations. In this case, the organization's privacy policy would be the composition of their business privacy policy and the federal privacy regulations. This composition of privacy policies can be very easily achieved by XACML.

Our evaluation of XACML as a candidate specification language is based on a set of features required for expressing HIPAA, proposed by DeYoung *et al.* [35]. In our evaluation of XACML, we found out that XACML has some of the features (*e.g.*, attributes, policy/policy rule combination) needed to support HIPAA. However, it lacks some other necessary features (*e.g.*, event history, obligations, subjective belief, reference to other rules) to adequately capture the HIPAA privacy rules. We thus propose necessary extensions for XACML to specify HIPAA.

In our analysis of XACML's candidacy as a possible specification language, we found that features like temporal conditions, quantification, which are necessary to specify HIPAA, cannot be expressed in XACML in a flexible and extensible way. To mitigate this, *the next candidate specification language* we consider is specialized for expressing privacy policies like HIPAA and is a hybrid of the specification languages proposed by Barth *et al.* [11] and DeYoung *et al.* [35]. We represent this privacy policy specification language with ***FOPSL***. ***FOPSL*** is based on a restricted fragment of first-order temporal logic (FOTL). We have shown the efficacy of ***FOPSL*** by expressing all 84 disclosure-related HIPAA clauses in it (see Appendix A). Due to its well-defined formal semantics and its expressive power, we use ***FOPSL*** as the language of our choice for our static policy analysis technique which gives formal assurance that incurred obligations can be met in a privacy policy conforming way. As we shall show, in general the problem of policy analysis is undecidable. Thus, we impose some restrictions, some of which are on the specification language, to make the problem decidable. The language ***FOPSL*** was thus developed taking into account the restrictions required for policy analysis.

3.1 Features for HIPAA Specification

In this section, we inventory and briefly summarize the features proposed by DeYoung *et al.* [35,36] that a policy specification language requires to sufficiently capture the HIPAA privacy rules.

Attributes. Each HIPAA privacy rule mandating a disclosure or usage can restrict the sender's, receiver's, the subject's, and the message's current attributes. For instance, the HIPAA privacy rule §164.502(a)(1)(i) specifies that: "*A covered entity is permitted to use or disclose protected health information as follows: To the individual*". According to this regulation (when considering a disclosure action), the sender's role attribute must be covered entity, the receiver's and the subject's role attribute must be individual, and the information in question is the subject's *PHI* attribute.

Attribute Inference Policy. *Attribute inference policies* specify whether a certain individual has a specific attribute based on conditions on his current attributes. Consider the HIPAA privacy policy rule in §164.502(a)(1) that allows a covered entity to send a patient's *PHI* to the patient's personal representative. While evaluating this policy rule, one might need to check whether a certain individual p_1 is the personal representative of the patient p_2. Attribute inference policies can specify under what circumstances p_1 can be considered a personal representative of p_2. An example of such an attribute inference policy can be found in §164.502(g)(2) of HIPAA. It specifies that p_1 can be considered the personal representative of p_2 when p_1 has the authority to make health care decisions for p_2 where p_1 is either an adult or an emancipated minor.

Past Events. The HIPAA privacy rule restricts a request for disclosure or usage of a patient's protected health information (*PHI*) based on some events on the past. Consider the regulation §164.502(e)(1)(i) which mentions that: "*A covered entity may disclose protected health information to a business associate and may allow a business associate to create or receive protected health information on its behalf, if the covered entity obtains satisfactory assurance that the business associate will appropriately safeguard the information.*" According to this regulation, the covered entity can disclose a patient's *PHI* to its business associate when it has already received satisfactory assurance from its business associate regarding the safeguarding of the *PHI*.

Obligations with Deadlines. The HIPAA privacy rules also impose obligatory restrictions on the covered entity. Furthermore, the obligations have specific deadlines within which the obligation needs to be carried out. Consider the §164.524(b)(2)(i) of HIPAA, which mentions: "*the covered entity must act on a request for access no later than 30 days after the receipt of the request*". Here, the covered entity is obligated to act within 30 days after it has received a request for access from an individual.

Purpose of Usage or Disclosure. The HIPAA privacy rules restrict certain usage or disclosure requests based on the purpose of that action. For instance, the HIPAA privacy rule in §164.506(c)(1) specifies that: "*A covered entity may use or disclose protected health information for its own treatment, payment, or health care operations.*". This HIPAA privacy rule allows a covered entity to use or disclose *PHI* of patient for the purpose of either its own treatment, payment, or health care.

Subjective Beliefs. The HIPAA privacy rules allows a certain disclosure or usage of a patient's *PHI* based on the covered entity's subjective belief or professional judgement. An example of such a HIPAA privacy rule can be found in §164.512(f)(5) of the regulation. It states that: "*A covered entity may disclose to a law enforcement official protected health information that the covered entity believes in good faith constitutes evidence of criminal conduct that occurred on the premises of the covered entity*". This rule allows a covered entity to disclose *PHI* of a patient to the police for reporting a crime on premise and additional believes the *PHI* can be used as evidence.

Reference to Other Laws/Rules. The HIPAA privacy rules also restrict a certain disclosure or usage of a patient's *PHI* based on other laws and also others rules (sections and paragraphs) of HIPAA. One example of the HIPAA privacy rule referring to another law can be found in §164.512(a)(1) which specifies that: " *A covered entity may use or disclose protected health information to the extent that such use or disclosure is required by law and the use or disclosure complies with and is limited to the relevant requirements of such law*". The HIPAA privacy rule §164.502(a)(1)(ii) can serve as an example where one rule of HIPAA refers to another HIPAA privacy rule. It specifies that: "*A covered entity is permitted to use or disclose protected health information as follows: For treatment, payment, or health care operations, as permitted by and in*

compliance with §164.506;".

Policy/Policy Rule Combination. HIPAA has different types of privacy rules based on the restrictions they impose. More precisely, (i) some of the HIPAA privacy rules allow certain disclosure, (ii) some of the HIPAA privacy rules prohibit certain disclosure to take place, (iii) some of the rules allow certain disclosure when certain condition is satisfied, and (iv) some of the rules require certain disclosure to take place. To check whether certain disclosure or usage is in compliant with the HIPAA privacy rules, one should be able to consult all the above types of privacy rules to get decisions and combine them to get one consistent decision. An example of type (i) rules can be found in §164.506(c)(2) of HIPAA, which specifies that: "*A covered entity **may** disclose protected health information for treatment activities of a health care provider*". The HIPAA privacy rule §164.502(g)(3)(ii)(B) demonstrates the type (ii) rules. It specifies that: "*If, and to the extent, prohibited by an applicable provision of State or other law, including applicable case law, a covered entity **may not** disclose, or provide access in accordance with §164.524 to, protected health information about an unemancipated minor to a parent, guardian, or other person acting in loco parentis; and...*". An example of type (iii) privacy rules can be found in §164.508(a)(1) of HIPAA which specifies that: "*Notwithstanding any provision of this subpart, other than the transition provisions in §164.532, a covered entity **must** obtain an authorization for any use or disclosure of psychotherapy notes,...*". Example of type (iv) rule can be found in the HIPAA privacy rule §164.502(a)(2)(ii) which specifies that, "*A covered entity **is required to** disclose protected health information: When required by the Secretary under subpart C of part 160 of this subchapter to investigate or determine the covered entity's compliance with this subpart.*".

3.2 Evaluating XACML for HIPAA

We identify the mismatches that occur when expressing HIPAA in XACML. These serve as the motivation for future research and development of access control specification languages. We use XACML as an example of state-of-the-art access control language with enforcement support.

3.2.1 Stateful Policies vs. Stateless Mechanism

XACML policies are largely stateless. Essentially XACML provides a component that takes a request as input, and returns a decision. The XACML architecture puts forward keywords such as PEP (Policy Enforcement Point), PDP (Policy Decision Point), PAP (Policy Adminstration Point), and PIP (Policy Information Point). However, it does not suggest how to implement each of PAP and PIP, let alone modeling their interactions.

When using XACML to encode a set of complicated policies, all one can do is to create a stateless policy that takes requests as inputs and give decisions as outputs. Anything else is beyond the actual XACML standard. The HIPAA privacy rules, however, goes beyond a simple policy providing answers to requests.

Obligations. Although XACML seems to integrate obligations as part of it, it treats obligations largely as black boxes, without specifying what an obligation should include and how to handle them. In short, XACML does not assign any semantics to obligations, which we believe is necessary.

Event History. XACML is stateless and assumes any stateful information (e.g, history) is kept outside the policy engine. One possibility is to use Condition semantics of XACML 3.0 to handle history. However, one has to assume that there exists a sophisticated component outside the policy engine that maintains relevant history information and knows exactly which part of the history information is needed for a given request to put such information in the request context. In a sense, one has to assume a policy engine beyond XACML to handle these things. We believe that the decisions about how history information are maintained and used are largely policy driven, and should be handled together with access control policies, inside the XACML framework.

Policy-Directed Attribute Retrieval. In HIPAA, different attributes need to be provided for different requests. Deciding which attributes to retrieve, is often dictated by the policy itself. XACML currently does not contain support for this operation.

Policy-Directed Policy Retrieval. As we have discussed before, the HIPAA privacy rules can refer

to other documents or contracts (*e.g.*, privacy notices, authorizations) between the covered entity and the subject in question. If such a document or contract exists, it can dominate the response from the regular privacy rules. In that sense, we can consider these documents or contracts as separate policies and retrieve them when necessary. Currently, XACML does not support such interactions.

3.2.2 Interactive vs. Non-interactive Policy Evaluation

Reading HIPAA, one gets the sense that policy evaluation needs to be more interactive. For a disclosure request, depending on which justification one plans to use for the disclosure, a different set of conditions need to be checked. One cannot simply send a request and get back a decision.

Purpose of Disclosure or Usage and Subject Beliefs. The HIPAA regulation sometimes permits a disclosure or usage of a patient's *PHI* based on the purpose or based on the subject's belief. However, deciding whether certain disclosure or usage is requested for certain purposes, is difficult. It is impossible to decide from the static context of the request arguments. The same is true for subjective beliefs. It is often difficult to decide subjective beliefs without interacting with the requester.

Reference to Other Laws/Rules. As we have seen before, HIPAA privacy rules can refer to other HIPAA privacy rules and also other laws. As a result of which, while evaluating a privacy rule we might have to evaluate a different rule (referred in the original privacy rule) first before making decision about the first rule. Currently, XACML does not support such interactions between policy/policy rules.

3.2.3 Attribute Inference vs. Authorization Decisions

In XACML, all rules assign some kind of truth value to a particular request, and one cannot write a rule/policy assigning truth value to a query that is not an access request. For example, in HIPAA one condition for accessing *PHI* is that the requester is a personal representative of the patient. However, HIPAA has guidelines that dictate whether someone should be considered to be a per-

sonal representative. Ideally these conditions for deciding personal representative should also be specified as XACML policies and rules. However, these policies and rules are not about deciding the request, but about the inference of some attribute relevant to the current decision. Therefore, they cannot be expressed in current XACML.

3.2.4 Quantification Over Infinite Domains

As pointed out by existing work [11, 35, 36, 55], concise specification of the HIPAA privacy rules require quantifications over the infinite domains of the involved principals, message attributes, messages, *etc.* XACML supports implicit universal quantification (outer-most) of the sender, receiver, subject, message, message attributes, *etc.*, of the use or disclosure action which the rule mandates. However, while specifying HIPAA privacy rules, the condition associated with a privacy rule can also have quantifications. Currently, XACML does not support the specification of the explicit quantifications appearing in the condition of a rule.

3.3 Extensions of XACML to Support HIPAA Policies

Inspired by Barth *et al.* [11], we divide the HIPAA privacy regulation regarding disclosure or usage of a patient's *PHI* into two types of privacy rules, *allowing policy rules* and *prohibitive policy rules*. An allowing policy rule (*e.g.*, §164.502(a)(1)(i)) enables a disclosure or usage whereas a prohibitive policy rule (*e.g.*, §164.508(a)(2)) permits a disclosure only when its associate condition is satisfied. We describe how these rules are combined in section 3.3.6. Each HIPAA policy rule regulating a disclosure or usage contains the following restrictions: (1) sender's attributes, (2) recipient's attribute, (3) subject's attribute (the individual whose *PHI* is considered), (4) purpose of the disclosure, (5) the information that is being disclosed (*e.g.*, age, name, ssn), (6) obligations, (7) history, and (8) other conditions. In this section, we summarize how each of these are specified in the extended XACML. Our proposed extensions are based on the investigation of the HIPAA privacy rules in §164.502-§164.514, §164.522, and §164.524. Note that, the goal of this work is not completely specifying and enforcing the HIPAA privacy rules rather evaluating XACML as a

candidate for specifying and enforcing HIPAA privacy rules.

Assumptions. We now discuss the assumptions we make while considering XACML as a specification language for HIPAA. In our system, the actions of the system that are regulated by the privacy policy are disclose (*e.g.*, send a message, send postal mail), use (*e.g.*, read, write), request (*e.g.*, request from patient), and access (*e.g.*, patient accessing her own *PHI*). More precisely, we only regulate disclosure or usage regarding to the *PHI* of a patient. We also assume that it is the responsibility of the sending user to tag the message with the appropriate attributes (*e.g.*, address, ssn, age) based on the content of the message. Additionally, current work makes the assumption that when an obligation is incurred, the user incurring the obligation (*obligatee*) will be permitted according to the privacy and access control policy. However, this might not be the case. Relaxing this assumption and designing a static analysis of the policy to check whether it has this desired property is a subject of future work.

In the current work, we have abstracted away some portions of HIPAA. Consider the regulation in §164.502(g)(3)(ii)(A), that allows a covered entity to disclose the *PHI* to the guardian provided that other laws allow it. It is not feasible to encode all possible applicable laws in our language. As a result, we use an oracle (possibly a company attorney) to decide whether the disclosure is allowed by other laws. We additionally assume that the patient policies that the covered entity agrees to comply with, is consistent with the HIPAA privacy rules.

3.3.1 Obligations

Sometimes HIPAA policies specify obligations required to be performed by covered entities. For example, the HIPAA regulation §164.524(b)(2)(i) says that the covered entity is obligated to act within 30 days after it has received a request from an individual. As mentioned before, XACML's support for obligations is not rich enough to capture the obligatory requirements of HIPAA. As a result, in the current work, we adopt the obligation model by Li *et al.* [86] to support specification and enforcement of obligations in XACML. Note that, there are other approaches to manage obligations [13, 55, 109], but the model by Li *et al.* [86] is a natural fit as it can readily be used with

XACML without any significant modifications. The key ideas of the state-machine-based approach proposed in [86] are as follows. An obligation is modeled as a state machine that communicates with the PEP using events. The PEP manages the life-cycle of obligations. An obligation includes rulesets to specify its responses to input events. These responses include changing its state in response to events, which informs the PEP about what course of actions it should take regarding the request, and generating events, which inform the environment about what actions must be taken to fulfill the obligation. Some of these actions are deployment specific. These deployment specific actions are implemented by obligation modules. Multiple obligation modules can be attached to the PEP, each implementing some actions. These obligation modules communicate with the PEP and the obligations through an event interface. The details of this approach can be referred to [86].

3.3.2 History Management

The HIPAA privacy regulation sometimes allows a certain disclosure or usage of a patient's *PHI* when certain condition/event in the past (temporal condition) is true. To facilitate this, we propose a *history manager* that keeps track of important past events that might influence the permissibility of a certain disclosure or usage. A history manager is a relational database that saves important events and can be queried efficiently. In the example above, whenever we receive a court order requesting the *PHI* of a certain patient, we would save this event on the history manager. Now, in response to the court order, if the covered entity attempts to send the required information to the court, we check whether the covered entity actually received a court order. We achieve this by checking history table for an entry which is a court order that the covered entity received. Note that, design of such history manager has been proposed in the literature [55, 73] but we design it specifically for XACML and discuss its interactions with the PDP to make an access decision.

Recall that, the history condition of a policy rule can contain quantifications over the domains of principal, message, and message attribute. XACML cannot express such quantifications. We overcome this by expressing the history conditions as stored database procedures which take arguments. In the rule specification, we refer to the appropriate database procedure. We use events

to pass the proper arguments of the database procedure. We follow the same approach for quantification in other elements of the conditions (*e.g.*, attribute inference, *etc.*). Now, to support event history in the condition of the privacy rules, we have to extend the <*Policy*> element of XACML which we discuss just below.

3.3.3 Interactions with Users

The HIPAA regulation sometimes permits a disclosure or usage of a patient's *PHI* based on the purpose or based on the subject's belief. However, deciding whether certain disclosure or usage is requested for certain purposes, is difficult. It is impossible to decide from the static context of the request arguments. We follow the approach of Lam *et al.* [79] and require that the user provides the purpose as an argument of the request. We present a list of possible purposes to the user and she chooses the appropriate one. Automatically determining whether a certain action is for some certain purpose [121] is out of the scope.

It is also not trivial how to model subjective belief of a principal in a computer information system. Thus, whenever we try to evaluate a policy rule that allows a usage or disclosure based on a subject's belief, we require additional information from the principal requesting the action. The additional information in this case is the information about the subjective belief.

Extension of the <*Policy*> element. In order to support interactions with users during policy evaluations, the <*Policy*> element in XACML needs to be extended. As aforementioned, some attribute values like subjective beliefs might be missing when checking whether a condition is satisfied, and thus user inputs might be required. In this case, the policy evaluation has to be stopped, and events should be sent out to inform users to provide the missing information. And then user inputs, if provided, will be sent back also by events. One possible extension is that attributes that will be required during the policy evaluation are specified in an optional <*RequiredAttributeList*> element. Hence we extend the <*Policy*> element in XACML, as shown below.

The *Source* of a <*RequiredAttributeSelector*> element can be *User*, *Database*, or *Oracle*, which indicates where the required attribute comes from. If the attribute is from the database,

<Policy> := [RequiredAttributeList]<Target><Rule>$^{+}$[Obligations]

Attributes: PolicyId, RuleCombiningAlgId

<RequiredAttributeList> := <RequiredAttributeSelector>$^{+}$

<RequiredAttributeSelector> := [Keys]

Attributes: AttributeId, DataType, Source, DatabaseId (optional), TableId (optional)

<Source> := "*User*"|"*Database*"|"*Oracle*"

<Keys> := <Key>$^{+}$

<Key> := <KeyValue>

Attributes: KeyId

the*DatabaseId*, the *TableId*, and the *Keys* of the table should be specified for the query.

According to the <*RequiredAttributeList*> element, the system will send events to the user (i.e., requester) informing what missing attribute values are required, query the database to retrieve the attribute values, or query the oracle for additional information (e.g., whether a disclosure is allowed by other laws). (Note that, for the condition of rules containing history restrictions, we have to query the history database.) Once responses are obtained from the user or the database, events carrying information about the required attributes will be sent back to the policy enforcement mechanism. Therefore, there should exist a way to get the attribute values from the incoming events when checking <*Condition*> in policy rules. One possibility would be to encode the attribute values as arguments to the event. Once an event is received, it can then be parsed to obtain the attribute values. Hence an <*EventSelector*> element is added into the <*Expression*> element substitution group so that the return values of the query can be obtained from the event.

<Condition> := <Expression>

The <*Expression*> element substitution group includes:

<AttributeSelector>, <AttributeValue>,
<VariableReference>, <ActionAttributeDesignator>,
<ResourceAttributeDesignator>, <Function>,
<SubjectAttributeDesignator>, <Apply>,
<EnvironmentAttributeDesignator>, <***EventSelector***>

<EventSelector> :=
Attributes: EventType, EventField, DataType

3.3.4 Attribute Inference Policies

Attribute inference policies specify whether a certain individual has a specific attribute based on conditions on his current attributes. Thus, an attribute inference policy can be viewed as an *Oracle* which responds with *True* or *False* to queries like "Does user p_1 has attribute a_1 based on p_1's current attributes?". Note that, in the context of distributed authorization Li *et al.* [87] proposed the RT language which achieves something similar to what we propose. However, we propose attribute inference policies in context of HIPAA and XACML.

To facilitate attribute inference policies, <*Condition*> needs to be further extended in a way that <*AttributeInferencePolicyReference*> is added into the substitution group of <*Expression*> to support references to attribute inference policies.

The same schema for HIPAA privacy policies provided by the extended XACML can be reused to specify this type of attribute inference policies. The only small difference is that the evaluation results of these policies are *True* or *False*, instead of *Permit* or *Deny*.

3.3.5 Additional Policies

An organization that is interested in enforcing HIPAA will have some additional policies (*i.e.*, organizational access control policies and patient policies). We present these here and show how they fit the big picture of enforcing HIPAA. In section 3.3.6, we present how these policies are

<Condition> := <Expression>

The <*Expression*> element substitution group includes:

<AttributeSelector>, <AttributeValue>,
<VariableReference>, <ActionAttributeDesignator>,
<ResourceAttributeDesignator>, <Function>,
<SubjectAttributeDesignator>, <Apply>,
<EnvironmentAttributeDesignator>, <***EventSelector***>,
<***AttributeInferencePolicyReference***>

<AttributeInferencePolicyReference> := <Input>$^{+}$

Attributes: AttributeInferencePolicyId

combined with the HIPAA policies.

Organizational Access Control Policies. The HIPAA privacy regulation mandates what information about a patient that the covered entity can disclose or use and under what circumstances. However, the covered entity (*e.g.*, hospital, clinic, doctor's office) might have some additional access control requirements that further restricts which employees of the covered entity can access the *PHI* of a certain patient. For instance, a covered entity might only allow the assigned doctors and the assigned nurses to access the *PHI* of a certain patient. Thus, even in the case where HIPAA allows doctors/nurses to use or disclose the *PHI* of a patient for allowed purposes, a doctor or nurse might be still denied access if they are not the patient's assigned doctor/nurse.

Patient Policies. *Patient policies* are those policies specified by the patients themselves. According to §164.522 of HIPAA, a covered entity can agree or disagree to comply with the patient policy. If the covered entity agrees to do so, the covered entity must comply with the patient policy along with the HIPAA policies.

3.3.6 Policy Combination

HIPAA policy combination. The HIPAA policies can be organized in the following way. In the top level, permit-overrides PCA is used to combine two types of policies: (1) required policies that

specify disclosures that are required and must be permitted, such as §164.502(a)(2)(i); and (2) permitted policies that specify uses or disclosures that might be permitted, such as §164.502(a)(1)(i). All required policies are combined using permit-overrides PCA, while the permitted policies are combined using deny-overrides PCA. The permitted policies are further divided into allowed policies, which are combined with permit-overrides PCA, and prohibitive policies, which are combined using deny-overrides PCA. Permit-overrides PCA and deny-overrides PCA are used in most cases, and sometimes their ordered versions are utilized. Thus the existing PCAs in XACML are sufficient to combine results of policy evaluations.

Combining additional policies. A disclosure or usage request of a covered entity is permitted, when it is allowed by all the policies: the organizational access control policy, the patient policy (if there is one), and the HIPAA policy. These policies are combined using the ordered-deny-overrides PCA. Recall that, ordered-deny-overrides PCA is the same as deny-overrides PCA, except that policies have to be evaluated exactly in the order they appear. Policies can be arranged in the following order: the access control policy, the patient policy, and the HIPAA policy. For performance sake, once a policy denies a request we do not evaluate the policies ordered after it and simply deny the request.

3.3.7 Architecture Design for Checking Compliance With XACML

Based on XACML's architecture and the architecture presented in [86], we propose an architecture which supports the enforcement of policies specified in our extended XACML language. Figure 3.1 shows our proposed architecture.

The overall system is divided into two parts: a HIPAA Compliance Checking Component (HCCC), and an External Environment (EE). The EE is where applications are executed (e.g., a web-based HIPAA information system), and the HCCC helps the EE decide whether a usage or disclosure of *PHI* is permitted.

The main components in HCCC include a PEP, a PDP, a PIP, databases storing policies, attributes, histories, and logs, respectively, an Oracle, and a Timer. The **PEP** receives requests,

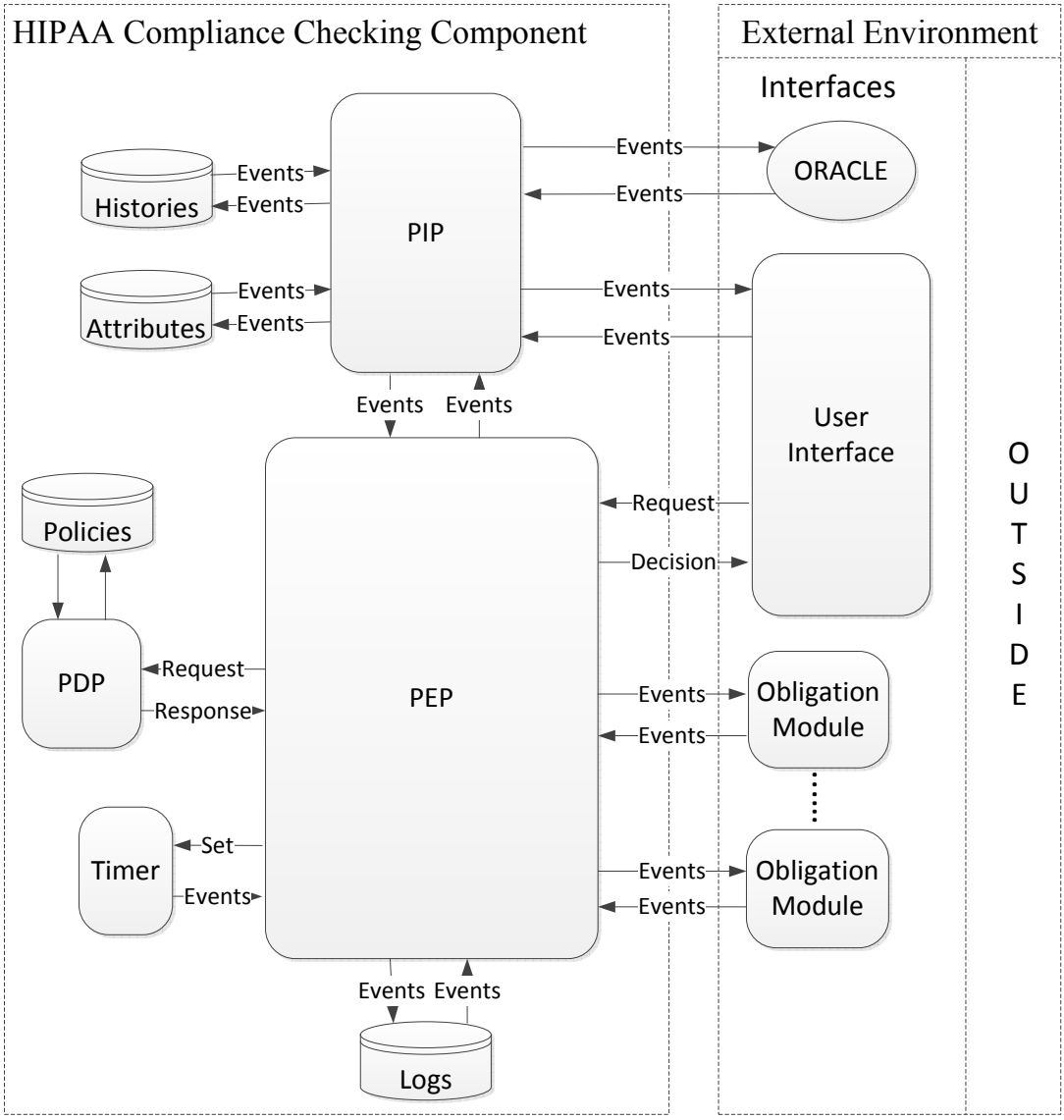

Figure 3.1: Proposed architecture

consults the PDP for a decision, handles any associated obligations, and makes the final decision about the request. The **PDP** evaluates the attribute-based policy provided by the policy database, and returns, to the PEP, a PDP decision, together with obligations, if any. The **PIP** serves as an attribute query point for subjects, resource and environmental attributes, and an information query point which obtains inputs from users, histories from the database and additional information from the Oracle. The **Attribute database** acts as a storage of subject and object attributes needed in policy evaluations. The **History database** stores history records such as authorizations or court

orders for covered entities. The **Policy database** stores HIPAA privacy policies, access control policies, and patient policies. The **Log database** can be used to keep logs, which leaves a room for auditing in the future work. The **Oracle** interacts with the PIP in a way that the PIP queries the Oracle and gets back a "Yes/No" (boolean) response. This is necessary for capturing whether a use or disclosure is allowed by other laws. The **Timer** informs the PEP that either a specific time point (e.g., 11:59P.M. on January 27th, 2012; it is pre-set by the PEP) arrives or a time duration (e.g., 5 minutes; it is pre-set by the PEP) is up.

The EE interacts with the HCCC through an interface, which include a user interface and zero or more Obligation Modules. The **User interface** is the component through which users can submit their requests and learn whether the requests are allowed or denied according to HIPAA privacy policies and other policies. In addition, it will interact with the PIP via events if user inputs are required during policy evaluations. The **Obligation modules** implement obligation-handling functionalities (such as notifying users, and writing to logs). More details can be referred to [86]. For XACML to support attribute inference policies, extensions are necessary for the policy engine so that the policy engine supports the following feature: while evaluating the original access request, the policy generates another request (representing a question about some attribute) and selects and evaluates other relevant policies for this new request (for attribute inference), and then the attribute inference result will be integrated with other policy to make a decision about the original request.

3.4 *FOPSL*

We now introduce our privacy policy specification language ***FOPSL***. ***FOPSL*** is a restricted fragment of first-order linear temporal logic (FOTL). It is inspired by the following specification languages: Contextual Integrity (CI) by Barth *et al.* [11] and PrivacyLFP by DeYoung *et al.* [35]. We demonstrate the adequacy of ***FOPSL*** by expressing all disclosure-related clauses of the HIPAA Privacy Rule [62] in it (see Appendix A). Note that we cannot express obligation deadlines in ***FOPSL***. Although enhancing ***FOPSL*** to express obligation deadlines [7, 72] is plausible, we do

$$\wp ::= \Box\,(\forall p_1, p_2, q : P. \forall m : M. \forall t : T. \forall u : U.$$
$$\text{send}(p_1, p_2, m) \wedge \text{contains}(m, q, t) \wedge \text{for-purpose}(m, u) \longrightarrow (\,(\bigvee_i \phi_i^+) \quad \bigwedge \quad (\bigwedge_j \phi_j^-)\,)\,)$$

Figure 3.2: Forms of our privacy policy ($\wp$) specified in ***FOPSL***

not take obligation deadlines into account in our policy analysis. This is further discussed in Chapter 8.

3.4.1 Top-level Policy

The form of our privacy policies is shown in Figure 3.2. We use $\wp$ to denote policies which has the form shown in Figure 3.2. ***FOPSL*** uses the sorts P, T, M, R, and U (denoting agents, attributes, messages, roles, and purposes) with associated carriers $\mathcal{P}$, $\mathcal{T}$, $\mathcal{M}$, $\mathcal{R}$, and $\mathcal{U}$, respectively. We use the convention that the variables p_1, p_2, and q are of sort P, t is of sort T, m is of sort M, and u is of sort U.

The privacy policies we consider (*e.g.*, HIPAA) mandate transmission of messages between different parties. A communication action is denoted by $\text{send}(p_1, p_2, m)$, in which p_1 is the sender, p_2 is the receiver, and m is the message being sent. Each message contains a set of agent, attribute pairs, $content(m) \subseteq \mathcal{P} \times \mathcal{T}$. The predicate $\text{contains}(m, q, t)$ holds if message m contains attribute t of subject q. A *knowledge state* κ is a subset of $\mathcal{P} \times \mathcal{P} \times \mathcal{T}$. If $(p, q, t) \in \kappa$, this means p knows the value of attribute t of agent q. For example, Alice knows Bob's height. A transition between knowledge states occurs when a message is transmitted, as the attributes contained in the message become known to the recipient. We use $\text{inrole}(p, \hat{r})$ to specify that the principal p is in role $\hat{r}$, in which $\hat{r}$ is a constant of sort R. For instance, $\text{inrole}(p, psychiatrist)$ holds when the principal p is in the role psychiatrist. We also allow role hierarchies and consider them as input to the system. For instance, the role *psychiatrist* is a specialization of the role *doctor*. The predicate $\text{for-purpose}(m, u)$ holds true when the message m is sent for the purpose u (*e.g.*, payment). We use the predicate $\text{in}(t, \hat{t})$ to specify that the attribute t can be calculated from the attribute $\hat{t}$, in which $\hat{t}$

is a constant (*e.g.*, procedure) of sort T. For instance, the zip code can be calculated from a postal address. Finally, the predicate purpose$(u, \hat{u})$ holds when the purpose u has the value $\hat{u}$, in which $\hat{u}$ is a constant (*e.g.*, payment) of sort U.

Our policies consist of two kinds of norms of transmission, *positive norms* and *negative norms*. Positive norms can be thought of *allowing* policy rules whereas negative norms can be thought of *denying* policy rules. A positive norm (ϕ_i^+) allows a message transmission *if* the condition associated with it holds. On the contrary, a negative norm (ϕ_j^-) allows a message transmission *only if* the condition associated with it is satisfied. An action is thus allowed by the policy if it satisfies at least one of the positive norms and all the negative norms. Finally, the policy of Figure 3.2 has the following intuitive meaning. For all senders p_1, for all receivers p_2, for all subjects of the information q, for all messages m, for all message attributes t, for all purposes u, p_1 can send a message to p_2 about q's attribute t for purpose u if it satisfies at least one of the positive norms and all the negative norms.

3.4.2 Syntax of Norms in *FOPSL*

The form of the policy norms are shown in Figure 3.3. The formula meta-variables in the norms (*i.e.*, ψ, β, and χ) correspond to syntactic categories introduced below in Figure 3.4. Exception formulas $\psi_{exception}$ have the same form as ψ. In the norms (see Figure 3.3), the non-temporal formulas $\mathbb{C}_{\text{sender}}$, $\mathbb{C}_{\text{receiver}}$, and $\mathbb{C}_{\text{subject}}$ impose constraints on the role of the sender, receiver, and subject, respectively. Formulas $\mathbb{C}_{\text{sender}}$, $\mathbb{C}_{\text{receiver}}$, and $\mathbb{C}_{\text{subject}}$ are boolean combinations of atomic formulas of the form inrole$(p, \hat{r})$ with p being the variable used in the send/contains predicates for the sender/receiver/subject, respectively. In the same vein, the non-temporal formulas $\mathbb{C}_{\text{attribute}}$ and $\mathbb{C}_{\text{purpose}}$ intuitively impose restrictions on the message attributes and the purposes of the message transmission. Formulas $\mathbb{C}_{\text{attribute}}$ and $\mathbb{C}_{\text{purpose}}$ are boolean combinations of atomic formulas of the form in$(t, \hat{t})$ and purpose$(u, \hat{u})$, respectively, in which t and u are both constrained. The formula $\mathbb{C} = \mathbb{C}_{\text{sender}} \wedge \mathbb{C}_{\text{receiver}} \wedge \mathbb{C}_{\text{subject}} \wedge \mathbb{C}_{\text{attribute}} \wedge \mathbb{C}_{\text{purpose}}$ can be viewed as specifying the target send event to which this norm applies to.

$$
\begin{array}{ll}
\text{Positive Norm, } \phi_i^+ : & (\mathbb{C} \wedge \psi \wedge \beta) \vee \psi_{exception} \\
\text{Negative Norm, } \phi_j^- : & \mathbb{C} \wedge \psi \rightarrow (\chi \vee \psi_{exception}) \\
\multicolumn{2}{l}{\text{where } \mathbb{C} = \mathbb{C}_{\text{sender}} \wedge \mathbb{C}_{\text{receiver}} \wedge \mathbb{C}_{\text{subject}} \wedge \mathbb{C}_{\text{attribute}} \wedge \mathbb{C}_{\text{purpose}}}
\end{array}
$$

Figure 3.3: Norms of transmission in ***FOPSL***

$$
\begin{array}{rl}
\text{(Atomic Formulas) } \gamma ::= & R(\vec{x}) \mid \mathit{true} \\
\text{(Non-temporal Formulas) } \mu ::= & \gamma \mid \mu \wedge \mu \mid \mu \vee \mu \mid \exists \vec{x} : \tau . \mu \mid \\
& \forall \vec{x} : \tau . (\mu_1(\vec{x}) \rightarrow \mu_2(\vec{x})) \\
\text{(Pure Past Formulas) } \psi ::= & \mu \mid \psi \wedge \psi \mid \neg \psi \mid \psi \mathcal{S} \psi \mid \exists \vec{x} : \tau . \psi \\
& \mid \forall \vec{x} : \tau . (\mu_1(\vec{x}) \rightarrow \mu_2(\vec{x})) \\
\text{(Obligation Formulas) } \beta ::= & \Diamond \mu \mid \beta \wedge \beta \\
\text{(Mixed Formulas) } \chi ::= & \beta \mid \psi \mid \psi \wedge \beta \mid \psi \rightarrow \beta
\end{array}
$$

Figure 3.4: Meta-variables and syntactic categories of the ***FOPSL***

We have already discussed some pre-defined predicates of ***FOPSL*** (*e.g.*, inrole). We allow additional predicates denoted by $R(\vec{x})$ (see Figure 3.4) in which $\vec{x}$ denotes its arguments. Each element of $\vec{x}$ is a constant or a variable. We envision these predicates to be regulation-specific.

3.4.3 Restrictions

We now discuss the different constraints we impose on ***FOPSL*** and their implications. Note, in particular, the limited way in which future temporal operators are used. Aside from the $\Box$ at the outer-most level, the only future sub-formulas are of the form given by β and $\Diamond$ can be applied only to positive, non-temporal formulas. This is the key to our ability to syntactically extract the past and future requirements from the policy formula. It also enables us to define weak compliance (*WC*) gracefully in Chapter 4. More precisely, we do not allow formulas expressing *general liveness properties* ($\Box\Diamond q$). Instead we allow formulas expressing *response properties* [89]. Response properties have the general form $\Box(p \rightarrow \Diamond q)$, in which p is a pure-past formula and q is a non-temporal formula. The formula $\Box(p \rightarrow \Diamond q)$ intuitively requires every p to be followed by a q. Among the past temporal operators, we do not allow the $\ominus$ operator. As we shall show in Chapter 5, a policy containing the $\ominus$ operator can fail to satisfy the Δ-property. We also do not allow function symbols in our specification language.

3.4.4 Differences between CI and *FOPSL*

We now discuss the differences between our specification language and that of Barth *et al.* [11] (CI). Recall that, we allow future operators only in specific places. However, this is not the case for CI. It allows arbitrary nesting of future and past temporal operators. In such a case, separating past and future requirements from a FOTL formula is not trivial [52]. Although Gabbay [52] provides a syntactic way of achieving it for pLTL, it is not trivial to extend the approach for FOTL due to predicates sharing variables among each other.

Additionally, in CI, one cannot express the purpose of the transmission and other conditions in HIPAA as they only have a fixed set of pre-defined predicates. CI also has a sort and predicate to refer to the context of the policy (*e.g.*, financial, health-care). In our case, we have only one context: health-care. As a result, we do not refer to context in our policy.

3.4.5 Example norms from HIPAA Expressed in *FOPSL*

A positive norm (shown below) can be found in §164.502(d)(1) of HIPAA. It states that a covered entity can send an individual's protected health information (*PHI*) to its business associate for creating de-identified (or, anonymized) information.

$$\begin{aligned}&\text{inrole}(p_1, \textit{covered-entity}) \wedge \text{inrole}(p_2, \textit{business-associate}) \wedge \\ &\quad \text{inrole}(q, \textit{individual}) \wedge \text{in}(t, \textit{PHI}) \wedge \\ &\quad \text{purpose}(u, \textit{creating-deidentified-in}) \wedge \text{businessAssociateOf}(p_2, p_1)\end{aligned}$$

A negative norm (shown below) can be found in §164.508(a)(2) of HIPAA. It specifies that a covered entity can disclose an individual's *psych-notes* only if the covered entity received a valid authorization from the individual regarding this disclosure.

$$\text{inrole}(p_1, \textit{covered-entity}) \wedge \text{inrole}(q, \textit{individual}) \wedge \text{in}(t, \textit{psych-notes}) \longrightarrow$$
$$\exists m_2 : M. \diamondsuit(\text{send}(q, p_1, m_2)) \wedge \text{satisfiesAllValidAuthReqs}(m_2, p_1, p_2, q, t, u)$$
$$\wedge \neg \text{violatesValidAuthReqs}(m_2, p_1, p_2, q, t, u))$$

We use ***FOPSL*** as our choice of privacy policy specification language in the rest of the work in regards to giving static assurance about permissibility of obligations in privacy policies.

Chapter 4: PRIVACY POLICY COMPLIANCE

We now formally specify what it means for an action to be compliant with a privacy policy. Privacy policies $\wp$ can impose *present requirements* (which includes past requirements) and also *obligatory (future) requirements*. An example of a present requirement can be found in clause §164.502(e)(1)(i) of HIPAA. It states that a covered entity can disclose an individual's *PHI* to a business associate if the covered entity receives satisfactory assurance that the business associate will safeguard the *PHI*. Obtaining the satisfactory assurance from the business associate is a present requirement of that clause. An obligatory requirement can be found in §160.310 of HIPAA, which requires the covered entity to provide *PHI* of an individual to the Secretary of Health and Human Services for compliance investigation, if she has requested for the information. The covered entity's action of providing access to the individual's *PHI* to the Secretary of Health and Human Services for compliance investigation is an obligatory requirement imposed by the clause §160.310 of the HIPAA privacy rule.

To this end, for checking compliance with policies $\wp$ it is helpful to separate the concerns of checking compliance with present and obligatory requirements. The syntactic restrictions in our policy language allow us to extract a formula that expresses the present requirements imposed by the policy. We can determine whether a contemplated action is in compliance with the present requirements of a policy by looking only at the current history. We call a contemplated action *weakly compliant* with respect to a policy when it is consistent with the present requirements of that policy. However, the present requirements do not give any assurance about whether the obligatory requirements can be met and can restrict an entity from performing its pending obligations. To this end, we use *strong compliance* [11], which formalizes the notion that a contemplated action will neither prevent pending obligatory requirements to be met nor incur any unsatisfiable obligatory requirements. An action is compliant with a privacy policy if it is both weakly compliant and strongly compliant. We will show that for our privacy policy language **FOPSL**, checking whether

Some of the contents of this chapter is based on the joint work with Andreas Gampe, Jianwei Niu, Jeffery von Ronne, Jared Bennatt, Anupam Datta, Limin Jia, and William H. Winsborough [27].

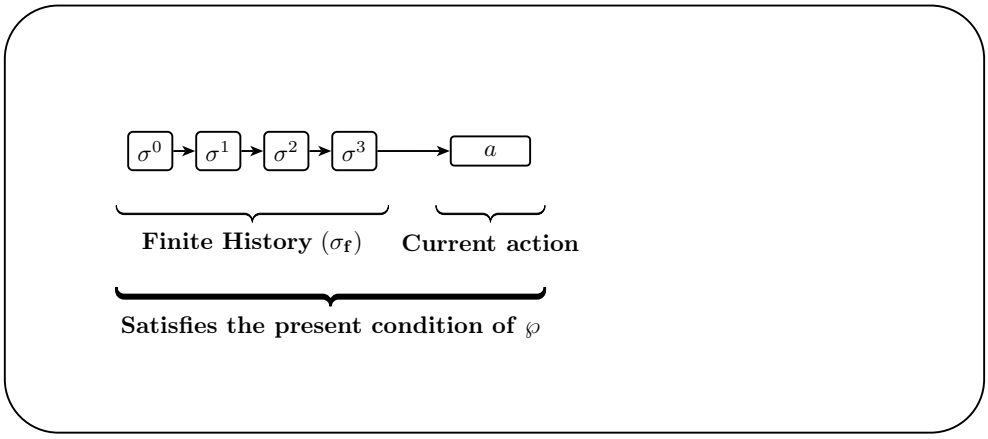

Figure 4.1: Weak Compliance

an action is weakly compliant with a policy is feasible whereas checking whether that action is strong compliant with the policy is undecidable.

4.1 Weak Compliance (WC)

For formally specifying what it means for an action to be weakly compliant with $\wp$, we use the formula $weak(\wp)$.

The formula $weak(\wp)$**.** $weak(\wp)$ denotes the formula derived from $\wp$ by replacing the future sub-formulas (sub-formulas of the form $\Diamond\mu$) with logical *true* and removing the outermost $\Box$. Due to the syntactic manipulation, the formula obtained only contains past temporal operators and expresses the present requirements of $\wp$.

Definition 14 (Weak Compliance (WC))**.** *Given a policy* $\wp$*, a finite trace* σ*, and a contemplated action a, a is* weakly compliant *with respect to* σ *and* $\wp$ *if for all environments* η*, the following holds* $\sigma \cdot s, |\sigma|, \eta \models \boxminus weak(\wp)$ *where state* $s \models a$*.*

Although we have formally defined $\models$ only in terms of infinite traces (see Chapter 2), the usage of $\models$ for finite σ here is well defined because $weak(\wp)$ is a pure-past formula: $\sigma \cdot s, |\sigma|, \eta \models \boxminus weak(\wp)$ depends only on the states in σ and the state s.

Consider the HIPAA privacy rule in §164.508(a)(2) which states that a covered entity can disclose an individual's psychotherapy notes if he has received the authorization from the individual. Now, if the covered entity discloses an individual's *psych-notes* without the authorization from the individual, then the action will not be weakly compliant with respect to the policy rule in §164.508(a)(2) as it violates the present requirement of obtaining an authorization.

Complexity of WC. For checking WC with respect to a policy $\wp$, we have to check whether a finite trace (including the current contemplated action) satisfies the formula $weak(\wp)$ in every point in that trace. Note that $weak(\wp)$ is a pure-past FOTL formula with quantifiers and carriers of which can be potentially infinite. To achieve termination, Garg *et al.* [55, 56] present *mode restriction* [9, 34, 96], which ensures that quantifiers can be expressed as finite conjunctions or disjunctions. This enables an algorithm with the complexity PSPACE of the policy size that can check whether a finite trace satisfies a first-order logic policy. Garg *et al.* [55] also show that mode restriction is still practical as the HIPAA privacy rule satisfies it. Note that their language is a proper superset of our language fragment used to express $weak(\wp)$. We can thus translate $weak(\wp)$ into their language, and if it passes mode checking, use their algorithm to check WC for $\wp$. Basin *et al.* [13] also present an algorithm for checking WC for a language similar to ours. Moreover, we also present an algorithm in Section 4.3 that check weak compliance of a privacy policy in PSPACE of the policy size.

Theorem 15 (Complexity of Checking WC)**.** *Given a policy $\wp$, a finite execution history σ_f, and a contemplate action a, whether a is weakly compliant with respect to $\wp$ and σ_f can be checked in PSPACE in the size of $\wp$ if the formula $weak(\wp)$ satisfies the mode restriction.*

Proof. Please see the algorithm in Section 4.3. □

4.2 Strong Compliance (SC)

When we check weak compliance, we ensure that the present requirements of the policy are met. However, there can be a situation where the obligatory requirements are not consistent with the

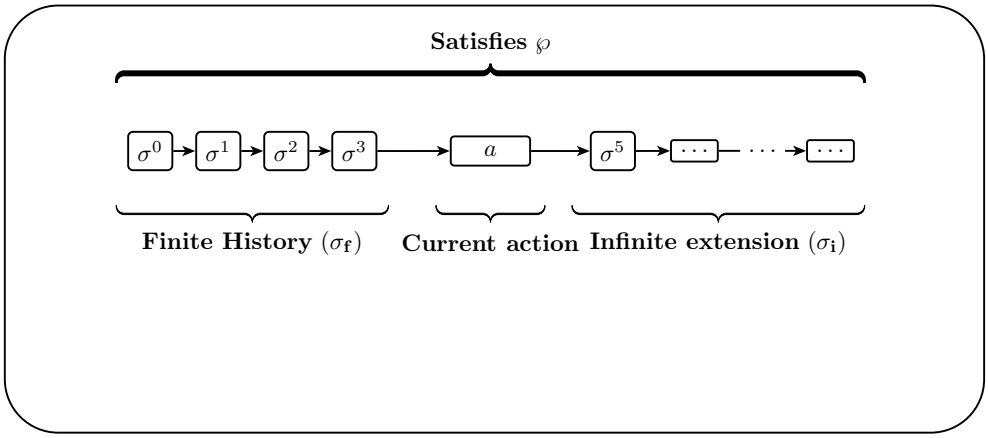

Figure 4.2: Strong Compliance

present requirements of the policy [33]. More precisely, it could be the case that an action that respects the present requirements of the policy incurs an obligation. Performing that obligation might not be allowed by the policy due to not being consistent with the present requirements. Strong compliance (SC) [11] ensures that this is not the case.

More precisely, a contemplated action is strongly compliant with a policy $\wp$ if the current history (including the current action) can be extended to an infinite trace such that the concatenation of the finite trace and the infinite extension satisfies $\wp$. Intuitively, a strongly compliant action neither incurs an obligation that cannot be met nor prevents any pending obligations to be met. We can formally define strong compliance in the following way.

Definition 16 (Strong Compliance (SC))**.** *Given a finite history σ'_f and a contemplated action a where state $s \models a$ and $\sigma_f = \sigma'_f \cdot s$, the action a is strongly compliant with the privacy policy $\wp$ if there is an infinite extension σ_i of the current history σ_f such that $\sigma_f \cdot \sigma_i \models \wp$.*

Complexity of Checking SC. Checking SC requires deciding whether the incurred future requirements (non-monadic FOTL formula) of an action are satisfiable. A monadic FOTL formula is a formula in which every formula that starts with a $\mathcal{S}$ or $\mathbb{U}$ temporal operator has at most one free variable [64]. Policies written in ***FOPSL*** do not satisfy this restriction and thus is not a fragment of monadic FOTL. Given a privacy policy $\wp$, a finite execution history σ_f, and a send event a (with

some positions ground), to check whether a is strongly compliant with $\wp$ with respect to σ_f, we have to check whether $(\varphi_{\sigma_f} \wedge \boxminus \wp)$ is satisfiable in which φ_{σ_f} is a formula that encodes the current execution history up to the current contemplated action into a FOTL formula. We first show how to encode the history into a FOTL formula of our form. Consider the finite execution history to be of form $\sigma_f = s_0 s_1 s_2 \ldots s_k$ in which s_k contains the current contemplated action a. Each state s_i, where $0 \leq i \leq k$, of the execution history maps a set of predicates to relations. Let us consider for state s_i, each relation that is true in that state is represented by $p_j^i(\vec{t})$ where $j \leq TR(s_i)$ in which the function TR takes as input a state and returns all the relations that are true in that state. Thus, one can encode the execution history as a FOTL formula as follows:

$$\varphi_{\sigma_f} = \bigwedge_{j \leq TR(s_k)} p_j^k(\vec{t}) \wedge \ominus (\bigwedge_{j \leq TR(s_{(k-1)})} p_j^{(k-1)}(\vec{t}) \wedge \ominus (\ldots))$$

We then have to find out whether the FOTL $(\varphi_{\sigma_f} \wedge \boxminus \wp)$ is satisfiable. However, policies $\wp$ written in ***FOPSL*** is the non-monadic fragment of FOTL, satisfiability of which is shown to be undecidable in general [64]. If we follow this approach it is undecidable to check given a policy $\wp$ and finite execution history σ_f whether a contemplated action is strongly compliant. To show that generally to check whether an action is strongly compliant with policies written in ***FOPSL*** is undecidable, we reduce the Turing machine halting problem [47] to checking strong compliance of a policy written in ***FOPSL***

Theorem 17 (Complexity of Checking SC)**.** *Given a policy $\wp$, a finite history σ, and a contemplated action a, to check whether action a is* strongly compliant *with respect to the policy $\wp$ is undecidable.*

Proof. We present here our reduction sketch and a similar detailed reduction is presented in Section 5 (Theorem 53). We reduce the Turing machine halting problem [47] to checking strong compliance of an action for policies written in ***FOPSL***.

We use principals/agents in our system to represent Turing machine tape cell. States and symbols of the Turing machine are represented as attributes in our system. We also use barred version of the states and symbols as attributes of our system. If a symbol x is written in a Turing machine

cell, it is represented in our system by the agent, denoting the cell, sending a message with attribute x. However, whenever we want to delete a symbol x from a cell, it is represented in our system as the agent, denoting the cell, sending a message with an attribute $\bar{x}$. Now, one can check whether a symbol x appears in a cell y by checking whether the agent representing the cell y send a message with attribute x before and since then the agent representing cell y did not send a message with attribute $\bar{x}$. The same goes for states of the Turing machine.

Once we have modeled the execution of a Turing machine in our system, we add two additional negative norms : (1) The first negative norm incurs an obligation when the Turing machine is in the initial state (the setting up the initial state is complete, details in the proof of Theorem 53). (2) The second negative norm allows the obligation incurred according to the previous negative norm only if the Turing machine has reached the designated halting state. Thus, if we can check whether the initial action is strongly compliant (there exists an execution in which the incurred obligation is fulfilled), then we can figure out whether the Turing machine will reach the designated halting state. □

4.3 Mode Driven Mechanism for Checking Weak Compliance

Recall that Cignet Health Center was fined a staggering $4.3 million for violating the HIPAA privacy rule [112]. It is thus incumbent upon the organization, that collects, stores, and discloses personal information, collected from individuals, for providing the individuals with some services, to have the means to efficiently check compliance with applicable privacy regulations. Note that we have two notions of compliance: weak and strong. We have already shown that checking strong compliance with respect to policies written in ***FOPSL*** is undecidable in general. In this section, we will present an algorithm for checking weak compliance with respect to a policy $\wp$ and a given finite execution history. In the next section, we will discuss how to overcome the undecidability result of checking strong compliance by presenting techniques to check a property of the policy, Δ-property. We will show that checking weak compliance is sufficient in the case the policy in question possesses the desired Δ-property.

Note that our policy language, ***FOPSL***, is non-monadic fragment [64] of the first order temporal logic (FOTL). There are techniques [60, 61, 85, 115] in the literature which can decide whether an action is weakly compliant with respect to a given finite execution history and the privacy policy where the privacy policy is specified in propositional LTL (pLTL). These techniques are not applicable to policies written in ***FOPSL*** or in general FOTL due to presence of quantifiers, variables, and predicates in the language. However, there is one naive way to get around this. One can just instantiate each variable with all the elements of the appropriate domain after which one can rewrite universal quantifiers as finite conjunctions and existential quantifiers as finite disjunctions. Finally, all the ground predicates can be replaced by propositional variables. The result is a pLTL policy for which we can use the techniques available in the literature [60, 61, 85, 115] for checking weak compliance. Note that domains of our policies can be infinite and consequently the naive approach will not terminate.

To achieve termination, researchers [13,15,16,55,73] have suggested different restrictions with which it is sufficient to instantiate the variables with selective elements of the appropriate domains rather than blindly instantiating the variables with all possible values in the domain. One such restriction proposed by the researchers [16,73] is to only allow "*event predicates*" in the policy. As the number of events that can happen at each point of time is finite, it is sufficient to consider only finite number of substitutions[1] for each of the predicates. Basin *et al.* [13,15] proposed a restriction called "*safe-range check*". One can check in static time whether a policy satisfies this safe-range check. Once a policy passes this safe-range check, it ensures that the number of substitutions that makes each predicate true is finite. Then they propose to build summary structures for all temporal formulas which summarize the execution history and saves all substitutions for which the formulas become true. Their approach has several advantages: (1) the whole execution history does not need to be saved, (2) their approach can be used as a runtime-monitor. Note that their policy language is however Metric first order temporal logic (MFOTL) which extends LTL by introducing time intervals associated with each temporal operator. For instance consider the following formula:

[1] A substitution can be viewed as a list of variable, value pairs.

$\diamondsuit_{[c,d]}p$, where $c,d \in \mathbb{N}$, which is a metric temporal logic (MTL) formula. The above formula is true in the current state with time stamp τ_i when the propositional variable p held true in some prior state with time stamp τ_j and additionally $\tau_i - \tau_j \in [c,d]$.

Garg *et al.* [55] proposed a technique which they borrow from logic programming called "*mode checking*" [9,34,96]. Given a n-ary relation symbol p, the *mode* of p is function m_p that maps each argument position of p to either '+' or '-' where '+' represents input position and '-' represents output position. The implication of the mode of a predicate is that when all the arguments in the input position are ground (concrete values), then the number of concrete values for the output argument positions of the predicate that satisfy the relation will be finite. For instance, let us consider a predicate $mul(x,y,z)$, where $x,y,z \in \mathbb{N}$, which holds for given x, y, and z if $x \times y = z$. If we are asked the number of satisfiable substitutions for variables x, y, and z such that $mul(x,y,z)$ holds, then the answer would be infinite. However, if we are given concrete values (*e.g.*, 1, 2) for variables x and z then the number of satisfiable substitutions for variable y will be finite. To this end, one possible moding of this predicate *mul* is $mul(x^+,y^-,z^+)$. It suggests that provided that we have concrete values for x and z, the number of concrete values for y which will make the condition $x \times y = z$ true is finite. Their algorithm uses this technique to instantiate the variables with selective values of the domain and achieve termination. Note that mode checking generalizes safe range checking in the view that safe range only checks to see whether all the predicates in the policy satisfies the following: all the arguments of the predicate is in the output mode. In this sense, the algorithm proposed by Garg *et al.* can support more expressive policies than what was known before. Note that their policy language is first order logic (FOL) and they can additionally handle incompleteness in the finite history. However, they require that the whole execution history (or, a log) to be around while they check for compliance.

To this end, our goal is to develop an algorithm which is a hybrid of the algorithm proposed by Basin *et al.* [13] and by Garg *et al.* [55]. More precisely, we want to build summary-structures for temporal sub-formulas which would summarize the execution history by keeping track of satisfiable substitutions for the temporal sub-formulas. As a result, we would not require the whole

$$\varphi ::= p(t_1,\ldots,t_n) \mid \top \mid \bot \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \mathcal{S} \varphi_2 \mid \varphi_1 \widehat{\mathcal{S}} \varphi_2 \mid \forall \vec{x}.(\varphi_1(\vec{x}) \rightarrow \varphi_2(\vec{x})) \mid \exists \vec{x}.\varphi(\vec{x})$$

Figure 4.3: Policy language for checking weak compliance ($\widehat{\texttt{FOPSL}}$)

execution history to be around when not needed. Additionally, we want to support policies as expressive as the policies supported by the algorithm presented by Garg *et al.* [55]. In this sense, our algorithm has the advantages of both algorithms [13, 55]. Moreover, we use a restricted fragment of FOTL instead of FOL. However, we assume the execution history to be past-complete and cannot handle incompleteness in the execution history as supported by Garg *et al.* [55] and Basin *et al.* [15]. Techniques to extend our algorithm with the support of incompleteness is an open problem.

We now briefly enumerate our technical contributions. We first extend the mode checking approach proposed by Garg *et al.* [55] for a restricted fragment of FOTL. Note that the extension is not trivial due to the fact that our mode checking takes into consideration temporal availability of substitutions which Garg *et al.* did not consider. We also show that summary structures can be build for a richer fragment of FOTL than previously considered [13, 16, 73]. We also prove that when the policy satisfies the mode checking, while evaluating that policy, our algorithm is sound, complete, and it terminates. We also propose a labeling algorithm which labels sub-formulas, for which a summary structure can be build to keep track of all possible satisfiable substitutions. We prove that when the formula representing the policy is labeled as **B**, our algorithm do not need to access any other states of the execution history except the current state and the summary structures. Given a finite history $\mathcal{L}$ and a policy $\wp$, the runtime complexity of our algorithm is $O(|\mathcal{L}|^{(|\wp|)})$ and the space requirement is polynomial to the $\wp$ size.

4.3.1 Policy Language

In this section, we introduce the readers with our policy specification language, which we call $\widehat{\texttt{FOPSL}}$. It is a fragment of first order temporal logic with restricted quantifiers and additionally

a more expressive fragment than the past fragment of ***FOPSL***, hence the name $\widehat{\textbf{\textit{FOPSL}}}$. Note that, unlike ***FOPSL***, $\widehat{\textbf{\textit{FOPSL}}}$ does not allow any future temporal operators due to the fact that we just want $weak(\wp)$ (which is pure past FOTL formula) to be expressive in $\widehat{\textbf{\textit{FOPSL}}}$. As we have mentioned, the restrictions in ***FOPSL*** is necessary for the decidability of our policy analysis discussed in the next section. We will use φ, α, and β (possibly with subscripts) to represent policies written in $\widehat{\textbf{\textit{FOPSL}}}$ to differentiate policies written in ***FOPSL***, for which we used $\wp$.

In our policy language, we use $p(t_1,\ldots,t_n)$ to represent relation between *terms* $t_1,\ldots,t_n$ where terms are constants, variables, or terms applied to uninterpreted function symbols. We use $\top$ and $\bot$ to represent logical true and false, respectively. Note that we do not allow negation in our policy language directly. For instance consider a formula $\neg p(x)$ where x ranges over the set of natural numbers. If we have the moding restriction that $p(x^-)$, it signifies that at each state the numbers of substitutions for x which makes $p(x)$ is finite. However, due to negation, the number of satisfiable substitution for x which makes $\neg p(x)$ true is infinite. To mitigate this complication, we follow Garg *et al.* [55] to assume every predicate $p(t_1,\ldots,t_n)$ has a dual which we represent by $\overline{p}(t_1,\ldots,t_n)$. Thus, when for some given $t_1,\ldots,t_n$, $p(t_1,\ldots,t_n)$ is true then $\bar{p}(t_1,\ldots,t_n)$ is false, and vice versa. In the same vein, we can lift the duals to formulas. Thus, for a given formula φ, the dual is represented as $\bar{\varphi}$. The dual works in the same fashion as $\neg\varphi$. Thus, we have the following equivalences:

$$\overline{p(t_1,\ldots,t_n)} \equiv \overline{p}(t_1,\ldots,t_n)$$

$$\overline{\top} \equiv \bot$$

$$\overline{\bot} \equiv \top$$

$$\overline{\varphi_1 \wedge \varphi_2} \equiv \overline{\varphi_1} \vee \overline{\varphi_2}$$

$$\overline{\varphi_1 \vee \varphi_2} \equiv \overline{\varphi_1} \wedge \overline{\varphi_2}$$

$$\overline{\forall \vec{x}.(\varphi_1(\vec{x}) \rightarrow \varphi_2(\vec{x}))} \equiv \exists \vec{x}.(\varphi_1(\vec{x}) \wedge \overline{\varphi_2(\vec{x})})$$

$$\overline{\exists \vec{x}.\varphi(\vec{x})} \equiv \forall \vec{x}(\top \rightarrow \overline{\varphi(\vec{x})})$$

$$\overline{\varphi_1 \,\mathcal{S}\, \varphi_2} \equiv \overline{\varphi_1}\, \widehat{\mathcal{S}}\, \overline{\varphi_2}$$

$$\overline{\varphi_1\, \widehat{\mathcal{S}}\, \varphi_2} \equiv \overline{\varphi_1}\, \mathcal{S}\, \overline{\varphi_2}$$

$$\overline{\overline{\varphi}} \equiv \varphi$$

We can also derive other logical connectives and temporal operators from the ones we have provided using the following equivalences.

$$\diamondminus \varphi \equiv \top \,\mathcal{S}\, \varphi$$

$$\boxminus \varphi \equiv \bot\, \widehat{\mathcal{S}}\, \overline{\varphi}$$

$$\varphi_1 \rightarrow \varphi_2 \equiv \overline{\varphi_1} \vee \varphi_2$$

$$\varphi_1 \leftrightarrow \varphi_2 \equiv (\varphi_1 \rightarrow \varphi_2) \wedge (\varphi_2 \rightarrow \varphi_1)$$

Note that even though our language includes $\widehat{\mathcal{S}}$ (dual of the $\mathcal{S}$ operator), for brevity, we do not show the cases corresponding to this operator in our algorithm and mode checking. The operator $\widehat{\mathcal{S}}$ can be handled in the similar way $\mathcal{S}$ is handled in our algorithm and mode checking.

Semantics. Given an execution history $\mathcal{L}$, a position in the history $i \in \mathbb{N}$, an environment η, and a formula φ, we use $\mathcal{L}, i, \eta \models \varphi$ to represent that φ is satisfied in the i^{th} position of the execution history $\mathcal{L}$ with respect to the environment η. This is defined inductively as below. The environment η maps free variables to values in the appropriate carrier. We use $\mathcal{L}_i$ to denote the i^{th} position of the history.

- $\mathcal{L}, i, \eta \models \top$
- $\mathcal{L}, i, \eta \not\models \bot$
- $\mathcal{L}, i, \eta \models p(t_1, \ldots, t_n)$ if and only if $p(t_1, \ldots, t_n)\eta \in \mathcal{L}_i$.
- $\mathcal{L}, i, \eta \models \varphi_1 \wedge \varphi_2$ if and only if $\mathcal{L}, i, \eta \models \varphi_1$ and $\mathcal{L}, i, \eta \models \varphi_2$.
- $\mathcal{L}, i, \eta \models \varphi_1 \vee \varphi_2$ if and only if $\mathcal{L}, i, \eta \models \varphi_1$ or $\mathcal{L}, i, \eta \models \varphi_2$.
- $\mathcal{L}, i, \eta \models \exists \vec{x}.\varphi(\vec{x})$ if and only if there exists $\vec{t}$ such that $\mathcal{L}, i, \eta[\vec{x} \mapsto \vec{t}] \models \varphi(\vec{x})$.
- $\mathcal{L}, i, \eta \models \forall \vec{x}.(\varphi_1(\vec{x}) \rightarrow \varphi_2(\vec{x}))$ if and only if for all $\vec{t}$ if $\mathcal{L}, i, \eta[\vec{x} \mapsto \vec{t}] \models \varphi_1(\vec{x})$ holds then $\mathcal{L}, i, \eta[\vec{x} \mapsto \vec{t}] \models \varphi_2(\vec{x})$ holds.
- $\mathcal{L}, i, \eta \models \varphi_1 \, \mathcal{S} \, \varphi_2$ if and only if there exists $k \leq i$, where $k \in \mathbb{N}$, such that $\mathcal{L}, k, \eta \models \varphi_2$ and for all j, where $j \in \mathbb{N}$ and $k < j \leq i$, it implies that $\mathcal{L}, j, \eta \models \varphi_1$ holds.
- $\mathcal{L}, i, \eta \models \varphi_1 \, \widehat{\mathcal{S}} \, \varphi_2$ if and only if for all $k \leq i$, where $k \in \mathbb{N}$, such that $\mathcal{L}, k, \eta \models \overline{\varphi_2}$ or there exists j, where $j \in \mathbb{N}$ and $k < j \leq i$, and $\mathcal{L}, j, \eta \models \overline{\varphi_1}$ holds.

A variable x is free in a formula φ if it is not bound by any quantifiers in the formula φ. For instance, given a formula $\forall x, y.(p(x, y, z) \rightarrow q(x, y, z))$, the variable z appears free in this formula as it is not bound by any quantifiers. Given a formula φ, we assume the function $fv(\varphi)$ returns the set of all free variables in φ.

4.3.2 Substitution

The next notion necessary to understand our weak compliance checking algorithm is the notion of *substitution* which is something similar to environments. A substitution can be viewed as a partial mapping that maps free variables to concrete values in the appropriate domain. The only way substitutions differ from environments is the way when a substitution is applied to a formula with free variables. When a substitution is applied to a formula, then the formula is changed in a way that all the free variables are replaced by their respective concrete value mappings provided by the substitution. However, in the case of the environments, when applied to a formula, they do not alter the formula itself. We use σ to denote substitutions. A substitution is defined in the following way.

Definition 18 (Substitution)**.** *We define a substitution (denoted by σ, possibly with subscript or superscript) to be a (partial) mapping from variables to values, where $\sigma(v)$ is in the domain of the variable v. Given a substitution σ, **domain**(σ) is defined as follows: **domain**$(\sigma) = \{x \mid \sigma(x) \neq x\}$.*

If a substitution is finite, we may consider it as a list of pairs of form $\langle var, val \rangle$ where var denotes a variable whereas val denotes the associated value which var is mapped to.

We use Σ (possibly with subscript or superscript) to denote a set of substitutions. We use $\{\bullet\}$ to represent the identity substitution and ***empty*** to represent an invalid substitution. Note that, given a substitution σ and a variable x, σ *concretely maps* x, if and only if $\sigma(x) \neq x$.

The next notion we introduce is extension of substitutions. Informally, given two substitutions σ_1 and σ_2, σ_2 extends σ_1 if σ_2 agrees with all the variable, value mappings with σ_1 for all variables which σ_1 concretely maps. For instance, let σ_1 be $([x \mapsto Alice], [y \mapsto Bob])$ and σ_2 be $([x \mapsto Alice], [y \mapsto Bob], [z \mapsto Carol])$. Then σ_2 extends σ_1 (denoted by $\sigma_2 \geq \sigma_1$) as σ_2 agrees with the variable, value mappings for variables x and y (which σ_1 concretely maps) with σ_1. Let σ_1 be $([x \mapsto Alice], [y \mapsto John])$ and σ_2 be $([x \mapsto Alice], [y \mapsto Bob], [z \mapsto Carol])$. However, in this case, $\sigma_2 \not\geq \sigma_1$ as σ_2 and σ_1 does not agree with the variable, value mapping for variable y. We also assume given any substitution σ, $\sigma \geq \sigma$. Moreover, for a given σ_1 and σ_2, if $\sigma_2 \geq \sigma_1$, we say σ_1 to

be the *abbreviation* of σ_2. Extension of a substitution is defined formally in the following way.

Definition 19 (Extension of Substitution)**.** *Given two substitutions* σ *and* σ'*, we say* σ' *extends* σ*, denoted by* $\sigma' \geq \sigma$*, if the following holds:* $\boldsymbol{domain}(\sigma') \supseteq \boldsymbol{domain}(\sigma)$ *and* $\forall x \in \boldsymbol{domain}(\sigma).(\sigma(x) = \sigma'(x))$*. if* $\sigma' \geq \sigma$*, we call* σ *the abbreviation of* σ'*.*

Recall that, substitutions and environments only differ when they are applied to a formula. In case a substitution is applied to a formula, it changes the formula by replacing free variables with the concrete values of the appropriate domain, it maps to. We formally define what it means for a substitution to be applied to a formula in the following way.

Definition 20 (Substitution Application)**.** *The application of a substitution* σ *to a formula* φ*, denoted* $\varphi\sigma$*, is recursively defined by*

$$\varphi\sigma = \begin{cases} \top & \varphi = \top \\ \bot & \varphi = \bot \\ p(\sigma(t_1), \ldots, \sigma(t_n)) & \varphi = p(t_1, \ldots, t_n) \\ (\varphi_1\sigma) \vee (\varphi_2\sigma) & \varphi = \varphi_1 \vee \varphi_2 \\ (\varphi_1\sigma) \wedge (\varphi_2\sigma) & \varphi = \varphi_1 \wedge \varphi_2 \\ \exists\vec{x}.\varphi_1(\vec{x})[\sigma \setminus \{\vec{x}\}] & \varphi = \exists\vec{x}.\varphi_1(\vec{x}) \\ \forall\vec{x}.(\varphi_1(\vec{x})[\sigma \setminus \{\vec{x}\}] \rightarrow \varphi_2(\vec{x})[\sigma \setminus \{\vec{x}\}]) & \varphi = \forall\vec{x}.(\varphi_1(\vec{x}) \rightarrow \varphi_2(\vec{x})) \\ (\varphi_1\sigma)\, \mathcal{S}\, (\varphi_2\sigma) & \varphi = \varphi_1\, \mathcal{S}\, \varphi_2 \end{cases}$$

where σ *is extended such that* $\sigma(e) = e$ *for any e not a variable or in the domain of* σ*.*

Notations. We now introduce the readers with some notations we use. Given a substitution σ, we use $\sigma \downarrow S$ to denote a new substitution which is same as σ except all the variable, value mappings for variables *not* in set S are removed in the new substitution. Let $\sigma' = \sigma \downarrow S$, then the following holds: $\boldsymbol{domain}(\sigma') \subseteq \boldsymbol{domain}(\sigma)$, $\forall x \in S.(\sigma(x) = \sigma'(x))$, and $\forall x \in \boldsymbol{domain}(\sigma).(x \notin S \rightarrow (\sigma'(x) = x))$. We

now generalize the above operation for a set of substitutions. Consider Σ' is a set of substitutions and $\Sigma = \Sigma' \downarrow S$. We define $\Sigma = \Sigma' \downarrow S$ in the following way, $\forall \sigma \in \Sigma'.(\Sigma \leftarrow \Sigma \cup \{\sigma \downarrow S\})$.

We use $\sigma \setminus S$ to denote a new substitution which is same as σ except the variable, value mappings for variables in set S are removed. More precisely, consider $\sigma' = \sigma \setminus S$, then $\textbf{\textit{domain}}(\sigma') \subseteq \textbf{\textit{domain}}(\sigma)$, $\forall x \in \textbf{\textit{domain}}(\sigma).(x \notin S \rightarrow (\sigma(x) = \sigma'(x)))$, and $\forall x \in \textbf{\textit{domain}}(\sigma).(x \in S \rightarrow (\sigma'(x) = x))$ holds. We now generalize the above operation for a set of substitutions. Consider Σ' is a set of substitutions and $\Sigma = \Sigma' \setminus S$. We define $\Sigma = \Sigma' \setminus S$ in the following way, $\forall \sigma \in \Sigma'.(\Sigma \leftarrow \Sigma \cup \{\sigma \setminus S\})$.

Given a substitution σ, we use $\sigma[x \mapsto t]$ to denote a new substitution which is same as σ except the variable x is now mapped to the new value t according to the new substitution. We now generalize the above operation for a set of substitutions. Consider Σ' is a set of substitutions and $\Sigma = \Sigma'[x \mapsto t]$. We define $\Sigma = \Sigma'[x \mapsto t]$ in the following way, $\forall \sigma \in \Sigma'.(\Sigma \leftarrow \Sigma \cup \{\sigma[x \mapsto t]\})$.

Given two substitutions σ_1 and σ_2 such that $\textbf{\textit{domain}}(\sigma_1) \cap \textbf{\textit{domain}}(\sigma_2) = \emptyset$, we use $\sigma_1 + \sigma_2$ to denote the concatenation of the variable, value mappings of both σ_1 and σ_2. Consider $\sigma = \sigma_1 + \sigma_2$, then the following holds: $\forall x \in \textbf{\textit{domain}}(\sigma).((x \in \textbf{\textit{domain}}(\sigma_1) \rightarrow \sigma(x) = \sigma_1(x))) \wedge (x \in \textbf{\textit{domain}}(\sigma_2) \rightarrow \sigma(x) = \sigma_2(x))))$. We also have: $\sigma + \{\bullet\} = \sigma$ and $\sigma + \textbf{\textit{empty}} = \textbf{\textit{empty}}$.

Given two substitutions σ_1 and σ_2, we use $\sigma_1 \bowtie \sigma_2$ to denote a new substitution which is the natural join of the two substitutions σ_1 and σ_2. Let $\sigma = \sigma_1 \bowtie \sigma_2$, σ is ***empty*** when the following holds: $\exists x \in (\textbf{\textit{domain}}(\sigma_1) \cap \textbf{\textit{domain}}(\sigma_2)).(\sigma_1(x) \neq \sigma_2(x))$. When $\sigma \neq \emptyset$ then the following holds: $\textbf{\textit{domain}}(\sigma) = \textbf{\textit{domain}}(\sigma_1) \cup \textbf{\textit{domain}}(\sigma_2)$ and $\forall x \in \textbf{\textit{domain}}(\sigma).(((x \in \textbf{\textit{domain}}(\sigma_1) \wedge x \notin S) \rightarrow \sigma(x) = \sigma_1(x)) \wedge ((x \in \textbf{\textit{domain}}(\sigma_2) \wedge x \notin S) \rightarrow \sigma(x) = \sigma_2(x)) \wedge (x \in S \rightarrow (\sigma(x) = \sigma_1(x) = \sigma_2(x))))$ where $S = \textbf{\textit{domain}}(\sigma_1) \cap \textbf{\textit{domain}}(\sigma_2)$. We consider the $\bowtie$ operation to be symmetric, that is $\sigma_1 \bowtie \sigma_2 = \sigma_2 \bowtie \sigma_1$. We also assume it is possible to calculate the join operation of two finite substitutions in some finite amount of time. We also have the following: $\sigma \bowtie \{\bullet\} = \sigma$ and $\sigma \bowtie \textbf{\textit{empty}} = \textbf{\textit{empty}}$.

We now generalize the above operation for two sets of substitutions. Consider Σ_1, Σ_2 are sets of substitutions and $\Sigma = \Sigma_1 \bowtie \Sigma_2$. We define $\Sigma = \Sigma_1 \bowtie \Sigma_2$ in the following way, $\Sigma = \{\sigma \mid \exists \sigma_1 \in \Sigma_1.\exists \sigma_2 \in \Sigma_2.(\sigma = \sigma_1 \bowtie \sigma_2 \wedge \sigma \neq \textbf{\textit{empty}})\}$.

We use $\bowtie_{0\leq k\leq j} \sigma_k$ to represent $\sigma_0 \bowtie \sigma_1 \bowtie \ldots \sigma_{j-1} \bowtie \sigma_j$. We also generalize this for sets of substitutions. Thus, $\bowtie_{0\leq k\leq j} \Sigma_k = \Sigma_0 \bowtie \Sigma_1 \bowtie \ldots \Sigma_{j-1} \bowtie \Sigma_j$. We sometimes use $\bowtie \sigma_\bullet$ where $\bullet$ represents a sequence when the sequence is understood from the context.

As the necessary notations have been introduced, we now discuss some obvious properties of substitutions which we use in proving the soundness and completeness of our algorithm.

Lemma 21 (Basic Substitution Properties)**.** *Let* σ *and* σ' *be arbitrary substitutions such that* $\boldsymbol{domain}(\sigma) \cap \boldsymbol{domain}(\sigma') = \emptyset$*, and let* φ *be any formula. Then*

1. *if* $\boldsymbol{domain}(\sigma) \cap fv(\varphi) = \emptyset$*, then* $\varphi\sigma = \varphi$*,*
2. $fv(\varphi\sigma) = fv(\varphi) \setminus \boldsymbol{domain}(\sigma)$*,*
3. $\varphi(\sigma + \sigma') = (\varphi\sigma)\sigma' = (\varphi\sigma')\sigma$*,*
4. *if* $\boldsymbol{domain}(\sigma) \supseteq fv(\sigma)$*, then* $fv(\varphi\sigma) = \emptyset$*,*
5. *if* $\boldsymbol{domain}(\sigma) \supseteq fv(\sigma)$*, then* $\varphi\sigma = \varphi(\sigma + \sigma')$
6. $\varphi\sigma = \varphi(\sigma \downarrow fv(\varphi))$.

Proof. The first three are by induction on the structure of φ. 4 follows from 2. 5 follows from 3, 4 and 1. 6 follows from 3 and 1. □

In the above Lemma, (1) specifies that when a given substitution σ does not concretely map any of the free variables of a formula φ then applying σ to φ has no effect. (2) formalizes the notion that, given a substitution σ and a formula φ, the set of free variables of the formula $\varphi\sigma$ (σ applied to φ) is the same as removing the set of concretely mapped variables of σ from the free variables of φ. (3) specifies that given two substitutions σ and σ' such that the sets of the concretely mapped variables are disjoint, then applying the two substitutions to any formula φ is commutative. (4) specifies that given a substitution σ and a formula φ such that σ concrete maps all the free variables of φ, then the resulting formula $\varphi\sigma$ (where σ is applied to φ) has no free variables. (5) specifies that given two substitutions σ, σ' and a formula φ such that σ concretely

maps all the free variables of φ and the sets of the concretely mapped variables of σ and σ' are disjoint, $\varphi\sigma = \varphi(\sigma + \sigma')$. (6) specifies that given any substitution σ and a formula φ, if we create another substitution σ' in a way that we remove from the set of concretely mapped variables of σ, all the variables which is not a free variable in φ then $\varphi\sigma = \varphi\sigma'$.

The following Lemma and Corollary, depict the effect of a substitution on the satisfaction of a formula with respect to a given execution history and a position on the history.

Lemma 22 (Substitution - fv Restriction and Extension)**.** *Let σ and σ' be substitutions and φ a formula such that **domain**$(\sigma) \cap$ **domain**$(\sigma') = \emptyset$ and **domain**$(\sigma') \cap fv(\varphi) = \emptyset$. Then for all $\mathcal{L}$, j, η it holds that $\mathcal{L}, j, \eta \models \varphi(\sigma \downarrow fv(\varphi)) \iff \mathcal{L}, j, \eta \models \varphi\sigma \iff \mathcal{L}, j, \eta \models \varphi(\sigma + \sigma')$.*

Proof. By Lemma 21, we have $\varphi(\sigma \downarrow fv(\varphi)) = \varphi\sigma = \varphi(\sigma + \sigma')$. Thus the statement is trivially true. □

Corollary 23 (fv Substitution)**.** *Let σ be a substitution and φ a formula, with **domain**$(\sigma) \supseteq fv(\varphi)$. Then for all $\mathcal{L}$, j, η and $\sigma' \geq \sigma$ it holds that $\mathcal{L}, j, \eta \models \varphi\sigma \iff \mathcal{L}, j, \eta \models \varphi\sigma'$.*

Proof. Let $\sigma'' = \sigma \downarrow fv(\varphi)$. Then $\mathcal{L}, j, \eta \models \varphi\sigma'' \iff \mathcal{L}, j, \eta \models \varphi\sigma$ and $\mathcal{L}, j, \eta \models \varphi\sigma'' \iff \mathcal{L}, j, \eta \models \varphi\sigma'$ by previous lemma. Thus, $\mathcal{L}, j, \eta \models \varphi\sigma \iff \mathcal{L}, j, \eta \models \varphi\sigma'$. □

4.3.3 Modes to the Rescue

Consider the following policy written in $\widehat{\texttt{FOPSL}}$: $\forall x.((\exists y.(p(x,y))) \rightarrow (q(x)))$ where x and y ranges over the set of natural numbers $\mathbb{N}$. In this case, if we want to check whether this policy is satisfied in a naive way, we will check whether for all natural numbers x, the following holds: $(\exists y.(p(x,y))) \rightarrow (q(x))$. Now the set of natural numbers is countably infinite, as a result, this approach will not terminate. One possibility to get around this is to impose semantic restrictions on the policy so that while evaluating the policy it is sufficient to instantiate the variables with selective (finite) individuals of the appropriate domain. The majority of the existing work on compliance checking [13, 15, 16, 73] achieves this by restricting the policy to contain only predicates, which has a finite number of substitutions at each state. These algorithms [13, 15, 16, 73] use FOTL

or some variant of it (*e.g.*, Metric FOTL, one sort FOTL) as their choice of policy language. These algorithms build summary structures for each temporal formula, where the summary structure for a temporal formula keeps track of all the substitutions that make the formula in question true. As a result, while evaluating whether the policy is satisfied if the satisfiable substitutions for a temporal sub-formula is needed, it can be looked up from the summary structure rather than going back to the execution history. It implies that the whole execution history does not need to be saved and it consequently decreases the space requirements of the algorithm. Moreover, these summary structures can be updated incrementally so keeping track of the summary structures from the previous state is sufficient. As one can update the summary structures of the previous state with the help of the current state to get the new updated summary structure.

Garg *et al.* [55] relaxed this restriction of requiring each predicates of the policy to have finite substitutions in each state by using "modes" [9, 34, 96] to achieve finite substitutions of the quantifiers, which in turn gives their algorithm termination and completeness. We now define modes and their implications.

Definition 24 (Definition of Modes by Apt and Marchiori [9])**.** *Given a n-ary predicate symbol p,* mode *of p is a function, denoted by m_p, from $\{1,\ldots,n\}$ to $\{+,-\}$. If $m_p(i) =$ "+", then i an input position of p with respect to m_p. In the same vein, when $m_p(i) =$ "-", then i an output position of p with respect to m_p.*

We assume each predicate symbol used in the policy has a unique mode. In case, one need one predicate to have multiple modes, then one can easily rename the predicate symbol to achieve this. The implication of modes in the context of compliance checking is that, for a given predicate symbol, if we are given concrete (or, ground) values for arguments in the input position, then the number of substitutions for the variables in the output position that satisfy the relation is finite. For instance, consider a predicate $addLessEq(x,y,a)$ where x, y, and a are variables with iterate over the set of natural numbers. Let us also assume that the predicate $addLessEq(x,y,a)$ holds for given x, y, and a if and only if $x+y \leq a$ holds. Now without giving any concrete values for any of the variables x, y, and a, if we are asked what is the number of satisfiable valuations for this predicate

addLessEq, then the answer is obviously infinite. However, if we are given concrete values (*e.g.*, 3, 5) for x and a, then the number of satisfiable substitutions for y for which *addLessEq* holds will be finite. In the vein, one possible moding of the predicate *addLessEq* can be the following: $addLessEq(x^+, y^-, a^+)$.

Note that, the policy specification language Garg *et al.* [55] is first order logic and their mode checking techniques are not trivially extendable to our policy language, $\widehat{\textbf{FOPSL}}$. Moreover, they require that the whole execution history to be around and they do not take advantage of building summary structures to keep track of satisfiable valuations of the temporal sub-formulas of the formula representing the policy. However, the policies their algorithm can handle is more expressive than ones known before.

To this end, we first present a labeling algorithm, using modes of the predicates, that takes as input a policy φ and tries to label all the temporal sub-formula of φ with either the label **B** or empty. When a formula has the label **B**, it signifies that it is possible to incrementally build a summary structure that can be used to keep track of all the substitutions which make the formula true. Thus, while evaluating the formula, we can just get the satisfiable valuations directly by accessing the structure. The advantage of the summary structures is that the summary structures only keep track of necessary satisfiable substitutions and the memory necessary to store the summary structures will likely be less than the memory needed for saving the whole execution trace. Our first contribution is that, we show that it is possible to build summary structures for formulas, which are much more expressive than any other fragments of FOTL known before.

We then extend the mode checking algorithm provided by Garg *et al.* [55] for $\widehat{\textbf{FOPSL}}$. We finally provide an algorithm which for formulas labeled as **B** get the satisfiable substitutions from the summary structure and calculates the satisfiable substitutions for formulas which are not labeled as **B**. In this sense, our algorithm extends the algorithm of Garg *et al.* with the support of building temporal structures. Note that our algorithm assumes the execution history to be complete, however, the algorithm by Garg *et al.* does not make this assumption. We now introduce the readers with our labeling algorithm which takes as input a formula/policy φ and labels each

Table 4.1: Labeling rules for **B** labels

$$\boxed{\chi_C \vdash_{\mathbf{B}} \varphi : \chi_O}$$

$$\frac{}{\chi_C \vdash_{\mathbf{B}} \top : \emptyset}\ [\textit{B-BASE-1}] \qquad \frac{}{\chi_C \vdash_{\mathbf{B}} \bot : \emptyset}\ [\textit{B-BASE-2}]$$

$$\frac{\forall k \in I(p).fv(t_k) \subseteq \chi_C \quad \chi_O = \bigcup_{j \in O(p)} fv(t_j)}{\chi_C \vdash_{\mathbf{B}} p(t_1,\ldots,t_n) : \chi_O}\ [\textit{B-BASE-3}]$$

$$\frac{\emptyset \vdash_{\mathbf{B}} \varphi_2 : \chi_1 \qquad \chi_1 \vdash_{\mathbf{B}} \varphi_1 : \chi_2 \qquad \chi_O = \chi_1}{\chi_C \vdash_{\mathbf{B}} \varphi_1 \mathcal{S} \varphi_2 : \chi_O}\ [\textit{B-SINCE}]$$

$$\frac{\chi_C \vdash_{\mathbf{B}} \varphi_1 : \chi_1 \qquad \chi_C \cup \chi_1 \vdash_{\mathbf{B}} \varphi_2 : \chi_2 \qquad \chi_O = \chi_1 \cup \chi_2}{\chi_C \vdash_{\mathbf{B}} \varphi_1 \wedge \varphi_2 : \chi_O}\ [\textit{B-CONJUNCTION}]$$

$$\frac{\chi_C \vdash_{\mathbf{B}} \varphi_1 : \chi_1 \qquad \chi_C \vdash_{\mathbf{B}} \varphi_2 : \chi_2 \qquad \chi_O = \chi_1 \cap \chi_2}{\chi_C \vdash_{\mathbf{B}} \varphi_1 \vee \varphi_2 : \chi_O}\ [\textit{B-DISJUNCTION}]$$

$$\frac{\chi_C \vdash_{\mathbf{B}} \varphi_1(\vec{x}) : \chi_1 \quad \chi_O = \chi_1 \setminus \{\vec{x}\}}{\chi_C \vdash_{\mathbf{B}} \exists \vec{x}.\varphi_1(\vec{x}) : \chi_O}\ [\textit{B-EXISTENTIAL}]$$

sub-formula of φ as either **B** or empty.

4.3.4 Labeling formulas for which summary structures can be built

We now present our labeling algorithm that labels a given formula φ as **B** or empty. Note that the heart of the labeling algorithm is the modes of the predicates used in the formula φ. We present our labeling algorithm as judgements where $\chi_C \vdash_{\mathbf{B}} \varphi : \chi_O$ signifies that given the substitutions for variables in the set χ_C when we evaluate the formula φ, we can get finite substitutions for variables in set χ_O and additionally label the formula with the label **B**. In the judgment $\chi_C \vdash_{\mathbf{B}} \varphi : \chi_O$, χ_C contains the substitutions which are available in the current point of time, this will be made more precise just below. Please note that our labeling algorithm is sound but not complete. This signifies that it does not recognize all formulas for which it is possible to build summary structures.

However, when a formula is labeled with **B**, it is then possible to build a summary structure for that formula to keep track of all satisfiable valuations of that formula.

The judgements [B-BASE-1] and [B-BASE-2] basically says it is possible to built summary structures for the following formula: $\top$ and $\bot$. This is trivial due to the fact that $\top$ (logical true) and $\bot$ (logical false), does not have any free variables in them and thus there are no substitutions to save in the summary structures. The judgement [B-BASE-3] specifies that a predicate can be labeled as **B** if we have ground substitutions (concrete values) for all the variables in the input positions ($\forall k \in I(p).fv(t_k) \subseteq \chi_C$) and after evaluating the predicate we will get ground substitutions for all the free variables in the output positions of the predicate.

The judgement [B-CONJUNCTION] specifies that it is possible to build a structure for a formula of form $\varphi_1 \wedge \varphi_2$ with respect to a given set of ground variables χ_C and additionally obtain ground variables in set χ_O after evaluating the formula: (1) if it is possible to build a structure for the formula φ_1 with respect to the given set of ground variables χ_C and additionally obtain ground variables χ_1 after evaluating φ_1, (2) if it is possible to build a structure for the formula φ_2 with respect to the given set of ground variables $\chi_C \cup \chi_1$ and additionally obtain ground variables χ_2 after evaluating φ_2, and (3) $\chi_O = \chi_1 \cup \chi_2$. For instance, consider a formula $p(x^+,y^-,z^-) \wedge q(y^+,z^+,w^-)$ and we are given the set of ground variables to be $\chi_C = \{x\}$. We want to know whether for the given χ_C, it is possible to make a summary structure for this formula. From premise (1), we have to first check whether $\{x\} \vdash_{\mathbf{B}} p(p(x^+,y^-,z^-) : \chi_1$ holds for some χ_1. This actually falls under the judgement [B-BASE-3]. As required by the judgement [B-BASE-3], $\{x\} \vdash_{\mathbf{B}} p(p(x^+,y^-,z^-) : \chi_1$ satisfy the first premise, that all variables in the input position (x) is ground. The next premise gives us $\chi_1 = \{y,z\}$. Thus, premise (1) of judgement [B-CONJUNCTION] holds. Now we have to check whether for the given $\chi_C \cup \chi_1 = \{x\} \cup \{y,z\}$, $\{x,y,z\} \vdash_{\mathbf{B}} q(y^+,z^+,w^-) : \chi_2$ holds, for some χ_2. Again this fall under the judgement [B-BASE-3]. According to the first premise of [B-BASE-3], $\{x,y,z\}$ covers all the variables in the input argument position (y,z), we see this is the case. Now according to the second premise of [B-BASE-3], we have $\chi_2 = \{w\}$. This satisfies the second premise of [B-CONJUNCTION] and according to the third premise of [B-CONJUNCTION],

we have $\chi_O = \chi_1 \cup \chi_2 = \{y,z\} \cup \{w\} = \{w,y,z\}$. Thus, we can safely conclude that for the given $\chi_C = \{x\}$, it is possible to build a structure for the formula $p(x^+,y^-,z^-) \wedge q(y^+,z^+,w^-)$ and evaluating the formula gives us ground values for variables $\{w,y,z\}$. We want to emphasize that according to the prior work [13, 15, 16, 73], it is not possible to build structure for the above formula. Prior work can build summary structures for the above formula if it had the following moding restriction: $p(x^-,y^-,z^-) \wedge q(y^-,z^-,w^-)$, which is more restricted than what we consider. The same goes for judgements [B-DISJUNCTION] and [B-EXISTENTIAL]. However, note that according to our judgements, a formula of form $\forall \vec{x}.(\varphi_1(\vec{x}) \rightarrow \varphi_2(\vec{x}))$ is never labeled with **B**. The reason behind it is that the number of satisfiable valuations for this formula is never finite and thus it is not possible to build a summary structure for it. There are two ways the formula containing the universal quantifier can be satisfied, (1) all substitutions for $\vec{x}$ for which the antecedent is false, (2) all substitutions for $\vec{x}$ for which the antecedent is true, it is also the case the consequent is also true. A summary structure should keep track of both kinds of substitutions. Now the number of valuations for $\vec{x}$ which does not satisfy the antecedent can be potentially infinite and cannot be saved. Thus, we never label a formula with the universal quantifier with **B**.

As we have mentioned before, χ_C represents the set of variables for which substitutions are available in the current point of time, we will now make it more precise. Consider the formula, $p(x^-) \wedge (r(x^+) \mathcal{S} q(x^+))$ and we are given $\chi_C = \emptyset$. According to the judgement rule for conjunction [B-CONJUNCTION], $\emptyset \vdash_{\mathbf{B}} p(x^-) : \{x\}$. Then, according to the second premise of [B-CONJUNCTION], we require $\{x\} \vdash_{\mathbf{B}} (r(x^+) \mathcal{S} q(x^+)) : \chi_2$ for some χ_2. If we analyze the semantics of $\mathcal{S}$ we know that, $r(x^+) \mathcal{S} q(x^+))$ is true in the current state if and only $q(x^+)$ holds in a previous state (possibly in the current state) and from that state till the current state $r(x^+)$ holds. Thus, $q(x^+)$ might hold in a state previous than the current state where $p(x^-)$ holds. If we were to build a structure keeping track of all satisfiable valuations for the formula $r(x^+) \mathcal{S} q(x^+))$, we cannot depend on the ground valuations of x that p provides. In that sense, the grounded substitutions for x can come from the future time point with respect to when $q(x^+)$ holds. The same is applicable to when we are building summary structure for $r(x^+)$. Thus, in the judgement [B-SINCE], the first

premise require that it is possible to build a structure for φ_2 without using any ground substitutions coming from χ_C, as the substitutions in the set χ_C comes from the future time point with respect to φ_2, thus we have $\emptyset \vdash_{\mathbf{B}} \varphi_2 : \chi_1$. Then, the second premise require that it is possible to build summary structure for φ_1 by using only the substitutions coming from φ_2 which is supposed to occur before the time point when φ_1 has to hold. Moreover, note that $\varphi_1 \mathcal{S} \varphi_2$ also holds in the current state, if φ_2 holds in the current state, so we are guaranteed to get ground substitutions from φ_2. Thus, we have $\chi_O = \chi_1$.

As we have described our labeling judgements, we now present the following Lemma which states that for a given χ_C, a given formula φ, if the following judgement can be derived: $\chi_C \vdash_{\mathbf{B}} \varphi : \chi_O$, then $\chi_O \subseteq fv(\varphi)$. Intuitive, it basically says that the set of additional ground variables χ_O is a subset of the free variables of the formula in question. This Lemma is used to prove correctness of our algorithm, described later.

Lemma 25 (Upper Bound of $\vdash_{\mathbf{B}}$)**.** *For all φ, χ_C and χ_O, if $\chi_C \vdash_{\mathbf{B}} \varphi : \chi_O$, then $\chi_O \subseteq fv(\varphi)$.*

Proof. Induction on the derivation of $\chi_C \vdash_{\mathbf{B}} \varphi : \chi_O$.

Cases [B-BASE-1], [B-BASE-2].

Then $\varphi = [\top]$ or $\varphi = [\bot]$, $\chi_O = \emptyset$ and $fv(\varphi) = \emptyset$. Trivially $\chi_O \subseteq fv(\varphi)$.

Case [B-BASE-3].

Then $\varphi = [p(t_1, \ldots, t_n)]$, $\chi_O = \bigcup_{j \in O(p)} fv(t_j)$ by [B-BASE-3] and $fv([p(t_1, \ldots, t_n)]) = \bigcup_{j \in \{1,\ldots,n\}} fv(t_j)$ by definition. Thus trivially $\chi_O \subseteq fv(\varphi)$.

Case [B-SINCE].

Then $\varphi = [\varphi_1 \mathcal{S} \varphi_2]$ and $\left[\dfrac{\emptyset \vdash_{\mathbf{B}} \varphi_2 : \chi_1 \qquad \chi_1 \vdash_{\mathbf{B}} \varphi_1 : \chi_2 \qquad \chi_O = \chi_1}{\chi_C \vdash_{\mathbf{B}} \varphi_1 \mathcal{S} \varphi_2 : \chi_O}\right]$. By inductive hypothesis, $\chi_1 \subseteq fv(\varphi_2)$. Thus $\chi_O = \chi_1 \subseteq fv(\varphi_2) \subseteq fv(\varphi_1) \cup fv(\varphi_2) = fv(\varphi)$.

Case [B-CONJUNCTION].

Then $\varphi = [\varphi_1 \wedge \varphi_2]$ and $\left[\dfrac{\chi_C \vdash_{\mathbf{B}} \varphi_1 : \chi_1 \qquad \chi_C \cup \chi_1 \vdash_{\mathbf{B}} \varphi_2 : \chi_2 \qquad \chi_O = \chi_1 \cup \chi_2}{\chi_C \vdash_{\mathbf{B}} \varphi_1 \wedge \varphi_2 : \chi_O}\right]$. By inductive hypothesis, $\chi_1 \subseteq fv(\varphi_1)$ and $\chi_2 \subseteq fv(\varphi_2)$. Thus $\chi_O = \chi_1 \cup \chi_2 \subseteq fv(\varphi_1) \cup fv(\varphi_2) = fv(\varphi)$.

Case [B-DISJUNCTION].

Then $\varphi = [\varphi_1 \vee \varphi_2]$ and $[\dfrac{\chi_C \vdash_{\mathbf{B}} \varphi_1 : \chi_1 \qquad \chi_C \vdash_{\mathbf{B}} \varphi_2 : \chi_2 \qquad \chi_O = \chi_1 \cap \chi_2}{\chi_C \vdash_{\mathbf{B}} \varphi_1 \vee \varphi_2 : \chi_O}]$. By inductive hypothesis, $\chi_1 \subseteq fv(\varphi_1)$ and $\chi_2 \subseteq fv(\varphi_2)$. Thus $\chi_O = \chi_1 \cap \chi_2 \subseteq fv(\varphi_1) \cap fv(\varphi_2) \subseteq fv(\varphi_1) \cup fv(\varphi_2) = fv(\varphi)$.

Case [B-EXISTENTIAL].

Then $\varphi = [\exists \vec{x}.\varphi_1(\vec{x})]$ and $[\dfrac{\chi_C \vdash_{\mathbf{B}} \varphi_1(\vec{x}) : \chi_1 \quad \chi_O = \chi_1 \setminus \{\vec{x}\}}{\chi_C \vdash_{\mathbf{B}} \exists \vec{x}.\varphi_1(\vec{x}) : \chi_O}]$. By inductive hypothesis, $\chi_1 \subseteq fv(\varphi_1(\vec{x}))$. By set properties, $\chi_O = \chi_1 \setminus \{\vec{x}\} \subseteq fv(\varphi_1(\vec{x})) \setminus \{\vec{x}\} = fv([\exists \vec{x}.\varphi_1(\vec{x})])$.

□

We now define what it means for a formula φ to have the label **B** with respect to a given χ_C.

Definition 26 (Labeling)**.** *Given substitutions for a set of ground variables χ_C, for all formulas φ if the following judgement can be derived for φ: $\chi_C \vdash_{\boldsymbol{B}} \varphi : \chi_O'$ where $\chi_O' \subseteq fv(\varphi)$. In the same vein, if a formula φ does not satisfy the above, we write $\boldsymbol{B} \notin label(\varphi)$.*

The following Lemma specifies a property of the labeling judgements. It basically specifies that if a formula φ can be labeled **B**, then it implies that all the sub-formula of φ can also be labeled **B**.

Lemma 27 (Subformulas of a temporal formula with label **B**)**.** *For a given χ_C and a formula φ, if $\chi_C \vdash_{\boldsymbol{B}} \varphi : \chi_O$ can be derived, where $\chi_O \subseteq fv(\varphi)$, then for all sub-formula $\hat{\varphi}$ of φ, there exists a χ_C' for which the judgment $\chi_C' \vdash_{\boldsymbol{B}} \hat{\varphi} : \chi_O'$ can be derived where $\chi_O' \subseteq fv(\hat{\varphi})$.*

Proof. The proof proceeds by doing an induction on the derivation of the $\vdash_{\mathbf{B}}$ judgements. □

The following Lemma specifies that whenever for a given formula φ and a given χ_C, the following judgement can be derived: $\chi_C \vdash_{\mathbf{B}} \varphi : \chi_O$ where $\chi_O \subseteq fv(\varphi)$, then for any χ_C' where $\chi_C' \supseteq \chi_C$, the following judgement can be derived: $\chi_C' \vdash_{\mathbf{B}} \varphi : \chi_O$. Intuitively, it specify that if we provide more ground variables as input, the set of additional ground variables will be same. This Lemma is used in proving the soundness and completeness of our algorithm described later.

Lemma 28 (Monotonicity of $\vdash_{\mathbf{B}}$ judgement)**.** *For a given χ_C and a formula φ, if $\chi_C \vdash_{\mathbf{B}} \varphi : \chi_O$, where $\chi_O \subseteq fv(\varphi)$, can be derived, then for any χ'_C such that $\chi'_C \supseteq \chi_C$, $\chi'_C \vdash_{\mathbf{B}} \varphi : \chi_O$ can be derived.*

Proof. We do induction on the derivation of the $\vdash_{\mathbf{B}}$ judgements.

Cases [B-BASE-1], [B-BASE-2].

We can see from the derivation of $[\overline{\chi_C \vdash_{\mathbf{B}} \top : \emptyset}]$ and $[\overline{\chi_C \vdash_{\mathbf{B}} \bot : \emptyset}]$ that the premise of the judgements do not use χ_C, thus we can trivially write $\chi'_C \vdash_{\mathbf{B}} \top : \emptyset$ and $\chi'_C \vdash_{\mathbf{B}} \bot : \emptyset$, without changing the derivation.

Case [B-BASE-3].

From the first premise of the judgement, it is required that $\forall k \in I(p).fv(t_k) \subseteq \chi_C$. We know $\chi'_C \supseteq \chi_C$. Thus, we can write $\forall k \in I(p).fv(t_k) \subseteq \chi'_C$. Then we get the judgement $\chi'_C \vdash_{\mathbf{B}} p(t_1,\ldots,t_n) : \chi_O$.

Case [B-SINCE].

Then $\left[\dfrac{\emptyset \vdash_{\mathbf{B}} \varphi_2 : \chi_1 \qquad \chi_1 \vdash_{\mathbf{B}} \varphi_1 : \chi_2 \qquad \chi_O = \chi_1}{\chi_C \vdash_{\mathbf{B}} \varphi_1 \mathcal{S} \varphi_2 : \chi_O}\right]$. We can see that the premises do not use χ_C. Thus, we can replace χ_C with χ'_C and can derive the judgement $\chi'_C \vdash_{\mathbf{B}} \varphi_1 \mathcal{S} \varphi_2 : \chi_O$.

Case [B-CONJUNCTION].

Then $\left[\dfrac{\chi_C \vdash_{\mathbf{B}} \varphi_1 : \chi_1 \qquad \chi_C \cup \chi_1 \vdash_{\mathbf{B}} \varphi_2 : \chi_2 \qquad \chi_O = \chi_1 \cup \chi_2}{\chi_C \vdash_{\mathbf{B}} \varphi_1 \wedge \varphi_2 : \chi_O}\right]$. We see that it is required that $\chi_C \vdash_{\mathbf{B}} \varphi_1 : \chi_1$ and $\chi_C \cup \chi_1 \vdash_{\mathbf{B}} \varphi_2 : \chi_2$. From I.H., we can write $\chi'_C \vdash_{\mathbf{B}} \varphi_1 : \chi_1$ and $\chi'_C \cup \chi_1 \vdash_{\mathbf{B}} \varphi_2 : \chi_2$ as $(\chi'_C \cup \chi_1) \supseteq (\chi_C \cup \chi_1)$. Thus, enabling us to derive the judgement $\chi'_C \vdash_{\mathbf{B}} \varphi_1 \wedge \varphi_2 : \chi_O$.

Case [B-DISJUNCTION].

Then $\left[\dfrac{\chi_C \vdash_{\mathbf{B}} \varphi_1 : \chi_1 \qquad \chi_C \vdash_{\mathbf{B}} \varphi_2 : \chi_2 \qquad \chi_O = \chi_1 \cap \chi_2}{\chi_C \vdash_{\mathbf{B}} \varphi_1 \vee \varphi_2 : \chi_O}\right]$. We see that it is required that $\chi_C \vdash_{\mathbf{B}} \varphi_1 : \chi_1$ and $\chi_C \vdash_{\mathbf{B}} \varphi_2 : \chi_2$. From I.H., we can write $\chi'_C \vdash_{\mathbf{B}} \varphi_1 : \chi_1$ and $\chi'_C \vdash_{\mathbf{B}} \varphi_2 : \chi_2$. Thus, enabling us to derive the judgement $\chi'_C \vdash_{\mathbf{B}} \varphi_1 \vee \varphi_2 : \chi_O$.

Case [B-EXISTENTIAL].

Then $\left[\dfrac{\chi_C \vdash_{\mathbf{B}} \varphi_1(\vec{x}) : \chi_1 \quad \chi_O = \chi_1 \setminus \{\vec{x}\}}{\chi_C \vdash_{\mathbf{B}} \exists \vec{x}.\varphi_1(\vec{x}) : \chi_O}\right]$. We see that it is required that $\chi_C \vdash_{\mathbf{B}} \varphi_1(\vec{x}) : \chi_1$. From I.H., we can write $\chi'_C \vdash_{\mathbf{B}} \varphi_1(\vec{x}) : \chi_1$. Thus, enabling us to derive the judgment $\chi'_C \vdash_{\mathbf{B}}$: $\exists \vec{x}.\varphi_1(\vec{x})\chi_O$.

□

The following Lemma states a property of the labeling judgements, specifically for the formulas of form $\varphi_1 \mathcal{S} \varphi_2$. It specifies that if a formula $\varphi_1 \mathcal{S} \varphi_2$ has the **B** label with respect to a given χ_C, then it will have the **B** label with respect to a given χ'_C where $\chi'_C = \emptyset$. This signifies that one can only build summary structures for a formula of form $\varphi_1 \mathcal{S} \varphi_2$, if it does not require any substitution that comes from the future time point.

Lemma 29 (Invariance of $\vdash_{\mathbf{B}}$)**.** *Given χ_C and χ_F, for all formulas of form $\varphi_1 \mathcal{S} \varphi_2$ such that $\chi_C, \chi_F \vdash \varphi_1 \mathcal{S} \varphi_2 : \chi_O$, if $\chi_C \vdash_{\mathbf{B}} \varphi_1 \mathcal{S} \varphi_2 : \chi'_O$ then $\emptyset \vdash_{\mathbf{B}} \varphi_1 \mathcal{S} \varphi_2 : \chi'_O$ where $\chi'_O = fv(\varphi_2)$.*

Proof. The proof follows from the judgement [B-SINCE] in Table 4.1. None of the premises of the judgement [B-SINCE] use χ_C. We can thus replace χ_C with $\emptyset$ without changing the judgement result. □

4.3.5 Mode Checking

Recall that, to achieve termination, we have to ensure that when we instantiate variables with finite number of individuals from appropriate domain. To achieve this, we extend the mode checking approach introduced by Garg *et al.* [55] in context of privacy policy compliance checking. Given substitutions for a set of variables V_g, a predicate $p(t_1, \ldots, t_n)$ will have finite number of satisfiable valuations if all the free variable in the input argument positions are in the set V_g, then the number of satisfiable substitutions for the free variables of the output argument position is finite and obtainable. We present this using the following judgement: $\chi_C, \chi_F \vdash p(t_1, \ldots, t_n) : \chi_O$ where $V_g = \chi_C \cup \chi_F$ and χ_O represents the set of the free variables of the output argument position of p for which it

Table 4.2: Mode checking judgements for $\varphi \equiv \top \mid \bot \mid p(t_1,\ldots,t_n)$

$$\boxed{\chi_C, \chi_F \vdash \varphi : \chi_O}$$

$$\frac{}{\chi_C, \chi_F \vdash \top : \emptyset}\ [\textit{BASE-1}] \qquad \frac{}{\chi_C, \chi_F \vdash \bot : \emptyset}\ [\textit{BASE-2}] \qquad \frac{\chi_C \vdash_{\mathbf{B}} p(t_1,\ldots,t_n) : \chi_O}{\chi_C, \chi_F \vdash p(t_1,\ldots,t_n) : \chi_O}\ [\textit{BASE-3}]$$

$$\frac{\bigcup_{k \in I(p).} fv(t_k) \subseteq (\chi_C \cup \chi_F) \qquad \chi_O = \bigcup_{j \in O(p)} fv(t_j)}{\chi_C, \chi_F \vdash p(t_1,\ldots,t_n) : \chi_O}\ [\textit{BASE-4}]$$

is possible to get finite number of satisfiable substitutions. In Table 4.2 judgement [BASE-4], the first premise requires that all the free variables in the input argument position should be an element of the set $\chi_C \cup \chi_F$. The second premise of that judgement specifies that after evaluating the predicate the set of additional ground variables will be the free variables of the output position position and will be the element of set of χ_O. The next obvious question is why do we have two different sets (χ_C and χ_F) to represent the set of variables for which substitutions are available. The set χ_C represents variables for which the substitutions come from past or present point of time whereas χ_F represents the set of variables for which the substitutions are available from some future point of time. This is particularly useful for judgements such as [BASE-3] in Table 4.2. The premise of the judgement requires that the predicate has the label **B** which is defined with respect to a set of variables, for which substitutions are available currently.

According to the judgement of form $\chi_C \vdash_{\mathbf{B}} \varphi : \chi_O$, we know that χ_C represents the set of variables for which substitutions are available currently (present or past point in time). If we did not have the separation between χ_C and χ_F, we would not be able to check satisfiability of judgement derivations like [BASE-3]. We would now describe how to lift the judgement $\chi_C, \chi_F \vdash p(t_1,\ldots,t_n) : \chi_O$ to arbitrary formulas φ. Thus, for given χ_C and χ_F, $\chi_C, \chi_F \vdash \varphi : \chi_O$ holds, when evaluating φ with respect to χ_C and χ_F, enables one to get finite substitutions for variables in set χ_O. We will briefly explain select cases of the derivations of the judgements.

Table 4.3: Mode checking judgements for $\varphi \equiv \varphi_1 \vee \varphi_2$

$$\boxed{\chi_C, \chi_F \vdash \varphi : \chi_O}$$

$$\frac{\chi_C \vdash_{\mathbf{B}} \varphi_1 : \chi_1 \quad \chi_C, \chi_F \vdash \varphi_2 : \chi_2 \quad \chi_O = \chi_1 \cap \chi_2}{\chi_C, \chi_F \vdash \varphi_1 \vee \varphi_2 : \chi_O} \; [\textit{DISJUNCTION-1}]$$

$$\frac{\chi_C, \chi_F \vdash \varphi_1 : \chi_1 \quad \chi_C, \chi_F \vdash \varphi_2 : \chi_2 \quad \chi_O = \chi_1 \cap \chi_2}{\chi_C, \chi_F \vdash \varphi_1 \vee \varphi_2 : \chi_O} \; [\textit{DISJUNCTION-2}]$$

$$\frac{\chi_C, \chi_F \vdash \varphi_1 : \chi_1 \quad \chi_C \vdash_{\mathbf{B}} \varphi_2 : \chi_2 \quad \chi_O = \chi_1 \cap \chi_2}{\chi_C, \chi_F \vdash \varphi_1 \vee \varphi_2 : \chi_O} \; [\textit{DISJUNCTION-3}]$$

$$\frac{\chi_C \vdash_{\mathbf{B}} \varphi_1 : \chi_1 \quad \chi_C \vdash_{\mathbf{B}} \varphi_2 : \chi_2 \quad \chi_O = \chi_1 \cap \chi_2}{\chi_C, \chi_F \vdash \varphi_1 \vee \varphi_2 : \chi_O} \; [\textit{DISJUNCTION-4}]$$

Table 4.4: Mode checking judgements for $\varphi \equiv \varphi_1 \wedge \varphi_2$

$$\boxed{\chi_C, \chi_F \vdash \varphi : \chi_O}$$

$$\frac{\chi_C \vdash_{\mathbf{B}} \varphi_1 : \chi_1 \quad \chi_C \cup \chi_1, \chi_F \setminus \chi_1 \vdash \varphi_2 : \chi_2 \quad \chi_O = \chi_1 \cup \chi_2}{\chi_C, \chi_F \vdash \varphi_1 \wedge \varphi_2 : \chi_O} \; [\textit{CONJUNCTION-1}]$$

$$\frac{\chi_C, \chi_F \vdash \varphi_1 : \chi_1 \quad \chi_C, \chi_F \cup \chi_1 \vdash \varphi_2 : \chi_2 \quad \chi_O = \chi_1 \cup \chi_2}{\chi_C, \chi_F \vdash \varphi_1 \wedge \varphi_2 : \chi_O} \; [\textit{CONJUNCTION-2}]$$

$$\frac{\chi_C, \chi_F \vdash \varphi_1 : \chi_1 \quad \chi_C \vdash_{\mathbf{B}} \varphi_2 : \chi_2 \quad \chi_O = \chi_1 \cup \chi_2}{\chi_C, \chi_F \vdash \varphi_1 \wedge \varphi_2 : \chi_O} \; [\textit{CONJUNCTION-3}]$$

$$\frac{\chi_C \vdash_{\mathbf{B}} \varphi_1 : \chi_1 \quad \chi_C \cup \chi_1 \vdash_{\mathbf{B}} \varphi_2 : \chi_2 \quad \chi_O = \chi_1 \cup \chi_2}{\chi_C, \chi_F \vdash \varphi_1 \wedge \varphi_2 : \chi_O} \; [\textit{CONJUNCTION-4}]$$

Table 4.5: Mode checking judgements for $\varphi \equiv \varphi_1 \mathcal{S} \varphi_2$

$$\boxed{\chi_C, \chi_F \vdash \varphi : \chi_O}$$

$$\frac{\emptyset \vdash_{\mathbf{B}} \varphi_2 : \chi_1 \quad \chi_1, (\chi_C \cup \chi_F) \setminus \chi_1 \vdash \varphi_1 : \chi_2 \quad \chi_O = \chi_1}{\chi_C, \chi_F \vdash \varphi_1 \mathcal{S} \varphi_2 : \chi_O} \; [\textit{SINCE-1}]$$

$$\frac{\emptyset, \chi_C \cup \chi_F \vdash \varphi_2 : \chi_1 \quad \emptyset, \chi_C \cup \chi_F \cup \chi_1 \vdash \varphi_1 : \chi_2 \quad \chi_O = \chi_1}{\chi_C, \chi_F \vdash \varphi_1 \mathcal{S} \varphi_2 : \chi_O} \; [\textit{SINCE-2}]$$

$$\frac{\emptyset, \chi_C \cup \chi_F \vdash \varphi_2 : \chi_1 \quad \emptyset \vdash_{\mathbf{B}} \varphi_1 : \chi_2 \quad \chi_O = \chi_1}{\chi_C, \chi_F \vdash \varphi_1 \mathcal{S} \varphi_2 : \chi_O} \; [\textit{SINCE-3}]$$

$$\frac{\emptyset \vdash_{\mathbf{B}} \varphi_2 : \chi_1 \quad \chi_1 \vdash_{\mathbf{B}} \varphi_1 : \chi_2 \quad \chi_O = \chi_1}{\chi_C, \chi_F \vdash \varphi_1 \mathcal{S} \varphi_2 : \chi_O} \; [\textit{SINCE-4}]$$

Table 4.6: Mode checking judgements for $\varphi \equiv \exists \vec{x}.\varphi_1(\vec{x})$

$$\boxed{\chi_C, \chi_F \vdash \varphi : \chi_O}$$

$$\frac{\chi_C, \chi_F \vdash \varphi_1(\vec{x}) : \chi_1 \quad \chi_O = \chi_1 \setminus \{\vec{x}\}}{\chi_C, \chi_F \vdash \exists \vec{x}.\varphi_1(\vec{x}) : \chi_O} \; [\textit{EXISTENTIAL-1}]$$

$$\frac{\chi_C \vdash_{\mathbf{B}} \varphi_1(\vec{x}) : \chi_1 \quad \chi_O = \chi_1 \setminus \{\vec{x}\}}{\chi_C, \chi_F \vdash \exists \vec{x}.\varphi_1(\vec{x}) : \chi_O} \; [\textit{EXISTENTIAL-2}]$$

Table 4.7: Mode checking judgements for $\varphi \equiv \forall \vec{x}.(\varphi_1(\vec{x}) \rightarrow \varphi_2(\vec{x}))$

$$\boxed{\chi_C, \chi_F \vdash \varphi : \chi_O}$$

$$\frac{\chi_C \vdash_{\mathbf{B}} \varphi_1(\vec{x}) : \chi_1 \quad \{\vec{x}\} \subseteq \chi_1 \quad fv(\varphi_1(\vec{x})) \subseteq \chi_C \cup \{\vec{x}\} \quad fv(\varphi_2(\vec{x})) \subseteq (\chi_C \cup \chi_1) \quad \chi_C \cup \chi_1 \vdash_{\mathbf{B}} \varphi_2(\vec{x}) : \chi_2}{\chi_C, \chi_F \vdash \forall \vec{x}.(\varphi_1(\vec{x}) \rightarrow \varphi_2(\vec{x})) : \emptyset} \; [\textit{UNIVERSAL-1}]$$

$$\frac{\chi_C \vdash_{\mathbf{B}} \varphi_1(\vec{x}) : \chi_1 \quad \{\vec{x}\} \subseteq \chi_1 \quad fv(\varphi_1(\vec{x})) \subseteq \chi_C \cup \{\vec{x}\} \quad fv(\varphi_2(\vec{x})) \subseteq (\chi_C \cup \chi_1 \cup \chi_F) \quad \chi_C \cup \chi_1, \chi_F \setminus \chi_1 \vdash \varphi_2(\vec{x}) : \chi_2}{\chi_C, \chi_F \vdash \forall \vec{x}.(\varphi_1(\vec{x}) \rightarrow \varphi_2(\vec{x})) : \emptyset} \; [\textit{UNIVERSAL-2}]$$

$$\frac{\chi_C, \chi_F \vdash \varphi_1(\vec{x}) : \chi_1 \quad \{\vec{x}\} \subseteq \chi_1 \quad fv(\varphi_1(\vec{x})) \subseteq \chi_C \cup \chi_F \cup \{\vec{x}\} \quad fv(\varphi_2(\vec{x})) \subseteq (\chi_C \cup \chi_1 \cup \chi_F) \quad \chi_C, \chi_F \cup \chi_1 \vdash \varphi_2(\vec{x}) : \chi_2}{\chi_C, \chi_F \vdash \forall \vec{x}.(\varphi_1(\vec{x}) \rightarrow \varphi_2(\vec{x})) : \emptyset} \; [\textit{UNIVERSAL-3}]$$

$$\frac{\chi_C, \chi_F \vdash \varphi_1(\vec{x}) : \chi_1 \quad \{\vec{x}\} \subseteq \chi_1 \quad fv(\varphi_1(\vec{x})) \subseteq \chi_C \cup \chi_F \cup \{\vec{x}\} \quad fv(\varphi_2(\vec{x})) \subseteq \chi_C \quad \chi_C \vdash_{\mathbf{B}} \varphi_2(\vec{x}) : \chi_2}{\chi_C, \chi_F \vdash \forall \vec{x}.(\varphi_1(\vec{x}) \rightarrow \varphi_2(\vec{x})) : \emptyset} \; [\textit{UNIVERSAL-4}]$$

Let us consider a formula of form $\varphi_1 \vee \varphi_2$. For formulas of this form, the mode checking judgements are specified in Table 4.3. Let us specifically consider judgement derivation [DISJUNCTION-2]. The mode checking judgement for above formula is of form $\chi_C, \chi_F \vdash \varphi_1 \vee \varphi_2 : \chi_O$ for some given χ_C and χ_F. The first premise of the judgement derivation require that if we evaluate φ_1 with respect to the given χ_C and χ_F, we will additionally get finite substitutions for the variables in the set χ_1. The second premise in the same vein require that when we evaluate φ_2 with respect to χ_C and χ_F, we will additionally get finite substitutions for variables in χ_2. Then finally the third premise require that if we evaluate the whole formula with respect to χ_C and χ_F, we will additionally get finite substitutions for variables in the set $\chi_1 \cap \chi_2$. Let us now consider a concrete formula $p(x^+, y^-, z^-) \vee q(x^+, y^-, w^-)$ and additionally χ_C be $\{x\}$ and χ_F be $\emptyset$. The first premise requires that $\{x\}, \emptyset \vdash p(x^+, y^-, z^-) : \chi_1$. We see that the judgement derivation [BASE-4] on Table 4.2 is applicable and according to it we have $\chi_1 = \{y, z\}$. In the same vein, from the second premise, we have $\{x\}, \emptyset \vdash q(x^+, y^-, w^-) : \{y, w\}$. Then finally we have $\chi_O = \{y, z\} \cap \{y, w\} = \{y\}$. Now, we justify why do we take the intersection of the χ_1 and χ_2. The formula $p(x^+, y^-, z^-) \vee q(x^+, y^-, w^-)$ is true when either (1) $p(x^+, y^-, z^-)$ is true or (2) $q(x^+, y^-, w^-)$ is true. In the case, (1) is true we will get substitutions for variables $\{y, z\}$ and similarly when (2) is true we will get substitutions for variables $\{y, w\}$. Thus, after the evaluation of the whole formula we are guaranteed to have substitutions for y and thus the intersection.

Now consider a formula of form $\forall \vec{x}.(\varphi_1(\vec{x}) \rightarrow \varphi_2(\vec{x}))$. For formulas of this form, the mode checking judgements are specified in Table 4.7. Let us specifically consider the mode checking judgement derivation [UNIVERSAL-2]. According to this mode checking rule, the first premise require that it is possible to build a structure keeping track of all satisfiable substitutions for the formula $\varphi_1(\vec{x})$ where evaluating the formula additionally ground the variables in set χ_1. The second premise require that there are finite substitutions for all the variables $\vec{x}$ after evaluating the formula $\varphi_1(\vec{x})$. The third premise require that there are finite substitutions for all the free variables of $\varphi_1(\vec{x})$. The fourth premise require that all the free variables there all finite substitutions for all variables in $\varphi_2(\vec{x})$ according to $\chi_C \cup \chi_F \cup \chi_1$. The final premise require that there are finite substitutions when

the formula φ_2 is evaluated. Note that, when the formula $\forall\vec{x}.(\varphi_1(\vec{x}) \rightarrow \varphi_2(\vec{x}))$ is evaluated, we assume no additional variables are ground due to the fact that the number of satisfiable valuations for a formula with universal quantifier can possibly be infinite as discussed before. All the mode checking judgement derivations are presented in Table 4.2, 4.3, 4.4, 4.5, 4.6, and 4.7.

Now that we have introduced the readers with our mode checking procedure, we now put forward some properties that can be derived from the mode checking judgements and which will be later used to prove the soundness and completeness of our algorithm. The following Lemma states that for any formula φ and for any given χ_C, χ_F, if the following judgement can be derived: $\chi_C, \chi_F \vdash \varphi : \chi_O$, then χ_O will be a subset of the set of free variables of φ.

Lemma 30 (Upper Bound of $\vdash$)**.** *For all* φ*,* χ_C*,* χ_F *and* χ_O*, if* $\chi_C, \chi_F \vdash \varphi : \chi_O$*, then* $\chi_O \subseteq fv(\varphi)$.

Proof. Induction on the derivation of $\chi_C, \chi_F \vdash \varphi : \chi_O$. Most cases are equivalent to Lemma 25. We show select other cases.

Case [UNIVERSAL-1], [UNIVERSAL-2], [UNIVERSAL-3], [UNIVERSAL-4].

Then $\chi_O = \emptyset$, which is trivially a subset of the free variables of any formula.

□

As we have introduced the necessary notions, we now introduce the readers with what it means for a formula φ to be *well-moded* with respect to some χ_C and χ_F.

Definition 31 (Well-moded formulas)**.** *A formula* φ *is* well-moded *with respect to a given* χ_C *and* χ_F *if we can derive the following judgement for* φ*:* $\chi_C, \chi_F \vdash \varphi : \chi_O$ *where* $\chi_O \subseteq fv(\varphi)$.

The following two Lemmas depict the relationship between our two judgements ($\vdash_{\mathbf{B}}$ and $\vdash$). The first Lemma (Lemma 32) specify that for a given χ_C and a given formula φ if one can derive the judgement $\chi_C \vdash_{\mathbf{B}} \varphi : \chi_O$ where $\chi_O \subseteq fv(\varphi)$ then it is possible to derive the judgement $\chi_C, \emptyset \vdash \varphi : \chi_O$. Then the next Lemma (Lemma 33) states that for the above case, we can actually derive $\chi_C, \chi_F \vdash \varphi : \chi_O$ for any given χ_F.

Lemma 32 (Invariance of $\vdash$)**.** *For all formula* φ *and for some given* χ_C*, if* $\chi_C \vdash_{\mathbf{B}} \varphi : \chi_O$ *holds then* $\chi_C, \emptyset \vdash \varphi : \chi_O$ *holds.*

Proof. Induction on the derivation of the $\vdash_{\mathbf{B}}$ and $\vdash$ judgements. □

Lemma 33 (Switching to $\vdash$ judgements from $\vdash_{\mathbf{B}}$ judgements)**.** *For a given* χ_C *and a formula* φ*, if* $\chi_C \vdash_{\mathbf{B}} \varphi : \chi_O$*, where* $\chi_O \subseteq fv(\varphi)$*, can be derived then for any* χ_F*,* $\chi_C, \chi_F \vdash \varphi : \chi_O$ *can be derived.*

Proof. The proof follows from Lemma 32 and 34. According to Lemma 32, if we have $\chi_C \vdash_{\mathbf{B}} \varphi : \chi_O$, we can write $\chi_C, \emptyset \vdash \varphi : \chi_O$. By Lemma 34, if we have $\chi_C, \emptyset \vdash \varphi : \chi_O$, we can write $\chi_C, \chi_F \vdash \varphi : \chi_O$ for any χ_F as $\chi_F \supseteq \emptyset$. □

Finally, we have the following Lemma which depicts the monotonicity property of the $\vdash$ judgement. It states that for a given χ_C, χ_F and a given formula φ, if one can derive the judgement: $\chi_C, \chi_F \vdash \varphi : \chi_O$, then it is possible to derive the judgement $\chi'_C, \chi'_F \vdash \varphi : \chi_O$ for any χ'_C, χ'_F such that $\chi_C \subseteq \chi'_C$ and $\chi_F \subseteq \chi'_F$. As mentioned above, this is an helping Lemma for proving the correctness of our algorithm.

Lemma 34 (Monotonicity of $\vdash$)**.** *Given* χ_C*,* χ_F*, and a formula* φ*, if* $\chi_C, \chi_F \vdash \varphi : \chi_O$*, where* $\chi_O \subseteq fv(\varphi)$*, can be derived, then for any* χ'_C, χ'_F *such that* $\chi_C \subseteq \chi'_C$ *and* $\chi_F \subseteq \chi'_F$*, the judgement* $\chi'_C, \chi'_F \vdash \varphi : \chi_O$ *can be derived.*

Proof. By induction on the derivation of the $\vdash$ judgements. We show select cases.

Case [BASE-1]/[BASE-2].

The premise of the judgement neither uses χ_C nor χ_F. Thus, we can write $\chi'_C, \chi'_F \vdash \top : \emptyset$ and $\chi'_C, \chi'_F \vdash \bot : \emptyset$.

Case [BASE-3].

From the derivation of the judgement, we see that it is required that $\chi_C \vdash_{\mathbf{B}} p(t_1, \ldots, t_n) : \chi_O$. In this premise of the judgement, χ_F is not used. So, we can easily replace χ_F with χ'_F. By Lemma 28, if we have $\chi_C \vdash_{\mathbf{B}} p(t_1, \ldots, t_n) : \chi_O$, we can write $\chi'_C \vdash_{\mathbf{B}} p(t_1, \ldots, t_n) : \chi_O$. Thus, we can derive the judgement $\chi'_C, \chi'_F \vdash p(t_1, \ldots, t_n) : \chi_O$.

Case [BASE-4].

From the derivation of the judgement, we see that it is required to satisfy $\bigcup_{k \in I(p)} .fv(t_k) \subseteq (\chi_C \cup \chi_F)$. As $\chi_C \subseteq \chi'_C$ and $\chi_F \subseteq \chi'_F$, we have $(\chi_C \cup \chi_F) \subseteq (\chi'_C \cup \chi'_F)$. Thus, we have $\chi'_C, \chi'_F \vdash p(t_1, \ldots, t_n) : \chi_O$.

Case [SINCE-1]

From the derivation of the judgement, we see that it is required to satisfy $\emptyset \vdash_{\mathbf{B}} \varphi : \chi_1$ and $\chi_1, (\chi_C \cup \chi_F) \setminus \chi_1 \vdash \varphi_1 : \chi_2$. The first premise of the derivation of the judgement neither uses χ_C nor χ_F. However, this is not the case for the second premise. We can write $((\chi_C \cup \chi_F) \setminus \chi_1) \subseteq ((\chi'_C \cup \chi'_F) \setminus \chi_1)$ as $\chi_C \subseteq \chi'_C$ and $\chi_F \subseteq \chi'_F$. From I.H., we have $\chi_1, (\chi'_C \cup \chi'_F) \setminus \chi_1 \vdash \varphi_1 : \chi_2$. We can thus derive the judgement $\chi'_C, \chi'_F \vdash \varphi_1 \, \mathcal{S} \, \varphi_2 : \chi_O$.

The other cases are similar. □

4.3.6 Weak Compliance Checking Algorithm

The building block of our algorithm is a function *sat* that takes as input an execution history $\mathcal{L}$, a position in the history i, a predicate p, and an input substitution σ_{in} (which contains substitutions for free variables in the input positions of p), and returns all the substitutions for the variables in the output positions of p for which p holds. The number of substitutions that *sat* returns is also finite. This is formalized in the following Axiom. We assume that we have a function I that takes as input a predicate p and returns all the input argument positions of the predicate. Additionally, we also assume that we have another function O that takes as input a predicate p and returns all the output argument positions of the predicate. Note that *sat* is not well-defined when the input substitution does not contain values for all the variables in the input position.

Axiom 35 (*sat* function). *Given a history $\mathcal{L}$, a position $j \in \mathbb{N}$, an input substitution σ_{in}, and a predicate $p(t_1, \ldots, t_n)$ such that for all $k \in I(p)$ the following holds: $\forall x \in fv(t_k).x \in \textbf{\textit{domain}}(\sigma_{\text{in}})$, then $\textbf{\textit{sat}}(\mathcal{L}, j, p(t_1, \ldots, t_n), \sigma_{\text{in}})$* terminates *and returns the* finite *set of all substitutions Σ_{out} for*

variables in $\bigcup_{k\in I(p)} fv(t_k) \cup \bigcup_{i\in O(p)} fv(t_i) \cup \textbf{\textit{domain}}(\sigma_{in})$*, where for all* $\sigma \in \Sigma_{out}$, $\sigma \geq \sigma_{\text{in}}$ *and* $\mathcal{L}, j \models p(t_1,\ldots,t_n)\sigma$ *hold.*

We now show how to extend ***sat*** function for arbitrary formulas allowed by our language $\widehat{\textbf{\textit{FOPSL}}}$. To achieve this, we have a function ***ips*** which takes as input an execution history $\mathcal{L}$, a position in the execution history $i \in \mathbb{N}$, a well-moded formula φ, a state π (see Section 4.3.6.1), and an input substitution for free variables in φ, and it returns a set of substitutions for the subset of the free variables of φ for which the formula φ holds true. For all well-moded formulas, the function ***ips*** inspects the execution history and tries to obtains all possible substitutions for which the formula holds true. Recall that, from Lemma 29, when a formula is labeled **B**, we can build summary structure that keeps track of all satisfiable substitutions for a formula φ of form $\alpha \mathcal{S} \beta$ without using any substitution which comes from the future point of time. Using this result, we build summary structures for all temporal formulas of form $\alpha \mathcal{S} \beta$ which has the label **B**. We will show that the summary structures can be updated incrementally by inspecting the current state and the current position of the execution history only. Note that we do not build any summary structures for formula whose top level connective is not a $\mathcal{S}$. To incrementally update the summary structure we use the function ***bts*** which recursively calls the function ***ips***. Whenever ***ips*** encounters a formula of form $\alpha \mathcal{S} \beta$ with label **B**, it actually accesses the corresponding summary structure and obtains all the satisfiable substitutions for the formula from which it calculates the appropriate substitutions based on the input substitution σ_{in}.

We now explain how to build a summary structure for temporal formulas of form $\alpha \mathcal{S} \beta$ which has the label **B**. In this explanation, we will use the function ***ips*** as a black box and will assume that whenever ***ips*** is called for a formula with respect to an input substitution σ_{in} then it returns all the satisfiable substitutions which is an extension of σ_{in} and satisfies the formula. For each temporal formula of form $\alpha \mathcal{S} \beta$ with label **B**, we have three persistent structures which we denote respectively by S_β, S_α, and S_R. The structure S_β contains a set of pairs of form $\langle \sigma, k \rangle$ in which σ represents a substitution and $k \in \mathbb{N}$ is a position in the execution history. Each pair of form $\langle \sigma, k \rangle \in S_\beta$ represents that the formula β was true with substitution, most recently in the execution

history position k. In the same vein, the structure S_α contains a set of pairs of form $\langle \sigma, k \rangle$, each of which represents that the formula α has been true with substitution σ from execution history position k till the current position in the execution history. The structure S_R contains a set of substitutions each of which, σ, represents that the formula $\alpha \mathcal{S} \beta$ is true with the substitution σ in the current position of the execution history. We will now show (just below) how to calculate the structures S^i_β, S^i_α, S^i_R for execution history position i, if we are given the structures $S^{(i-1)}_\beta$, $S^{(i-1)}_\alpha$, $S^{(i-1)}_R$ of the execution history position $i-1$.

$$\Sigma_\beta \leftarrow \textbf{\textit{ips}}(\mathcal{L}, i, \pi, \{\bullet\}, \beta)$$
$$S^\beta_{\text{update}} \leftarrow \{\langle \sigma \bowtie \sigma_1, i \rangle \mid \exists k. \langle \sigma, k \rangle \in S^{(i-1)}_\beta \wedge \exists \sigma_1 \in \Sigma_\beta . \sigma \bowtie \sigma_1 \neq \textbf{\textit{empty}}\}$$
$$S^\beta_{\text{remove}} \leftarrow \{\langle \sigma, k \rangle \mid \langle \sigma, k \rangle \in S^{(i-1)}_\beta \wedge \exists \sigma_1 \in \Sigma_\beta . \sigma \bowtie \sigma_1 \neq \textbf{\textit{empty}}\}$$
$$S^\beta_{\text{new}} \leftarrow \{\langle \sigma, i \rangle \mid \sigma \in \Sigma_\beta \wedge \nexists \langle \sigma_1, k \rangle \in S^{(i-1)}_\beta . \sigma \bowtie \sigma_1 \neq \textbf{\textit{empty}}\}$$
$$S^i_\beta \leftarrow (S^{(i-1)}_\beta \setminus S^\beta_{\text{remove}}) \bigcup S^\beta_{\text{new}} \bigcup S^\beta_{\text{update}}$$

We first describe how to update the structure S_β at execution history position i if we are given the structure S_β at execution position $i-1$. We first calculate the set of substitutions Σ_β by calling ***ips*** for β in the current execution history position (i). This set contains all the substitutions with which β holds true in the current execution history position. Recall that each pair of $\langle \sigma, k \rangle \in S_\beta$ requires that β be true with substitution σ most recently in the k^{th} position of the execution history. We first calculate whether we can update this position k with the current execution history position i. The set S^β_{update} contains all the substitutions for which we can update this position k with the current execution history position i. The next thing we have to is remove all the old pairs of form $\langle \sigma, k \rangle$ for which we have an updated position. All such pairs are stored in the set S^β_{remove}. The next thing we check is whether there is any new substitution with which β holds true in the current execution history position. These substitutions along with the current position are saved in the set S^β_{new}. Finally, from the old structure $S^{(i-1)}_\beta$ we remove all the pairs in the set S^β_{remove}, then add the

updated pairs in the set $S^{\beta}_{\text{update}}$, and finally add the new pairs in the set S^{β}_{new}.

$$\Sigma_\alpha \leftarrow \bigcup_{\langle\sigma,k\rangle \in S^i_\beta \wedge k \neq i} \textbf{\textit{ips}}(\mathcal{L}, i, \pi, \sigma, \alpha)$$

$$S^{\alpha}_{\text{new}} \leftarrow \{\langle\sigma, i\rangle \mid \sigma \in \Sigma_\alpha \wedge \nexists \langle\sigma_1, k\rangle \in S^{(i-1)}_\alpha . \sigma \bowtie \sigma_1 \neq \textbf{\textit{empty}}\}$$

$$S^{\alpha}_{\text{update}} \leftarrow \{\langle\sigma \bowtie \sigma_1, k\rangle \mid \langle\sigma, k\rangle \in S^{(i-1)}_\alpha \wedge \exists \sigma_1 \in \Sigma_\alpha . \sigma \bowtie \sigma_1 \neq \textbf{\textit{empty}}\}$$

$$S^{\alpha}_{\text{remove}_1} \leftarrow \{\langle\sigma, k\rangle \mid \langle\sigma, k\rangle \in S^{(i-1)}_\alpha \wedge \exists \sigma_1 \in \Sigma_\alpha . \sigma \bowtie \sigma_1 \neq \textbf{\textit{empty}}\}$$

$$S^{\alpha}_{\text{remove}_2} \leftarrow \{\langle\sigma, k\rangle \mid \langle\sigma, k\rangle \in S^{(i-1)}_\alpha \wedge \nexists \sigma \in \Sigma_\alpha . \sigma \bowtie \sigma_1 \neq \textbf{\textit{empty}}\}$$

$$S^{i}_{\alpha} \leftarrow ((S^{(i-1)}_\alpha \setminus S^{\alpha}_{\text{remove}_1}) \setminus S^{\alpha}_{\text{remove}_2}) \bigcup S^{\alpha}_{\text{update}} \bigcup S^{\alpha}_{\text{new}}$$

We now describe how to update the structure S_α at the current execution history position i if we are given the structure S_α at execution position $i-1$. We first calculate the set of substitutions Σ_α by calling ***ips*** for α with each substitution σ as the input substitution where $\langle\sigma, k\rangle \in S^i_\beta$ such that $k \neq i$, in the current execution history position (i). We use the substitution σ due to the fact that we are interested only on those substitutions of α with which we have seen a β hold true. This is due to the fact that $\alpha \mathcal{S} \beta$ does not hold if β never holds. The set Σ_α contains all the substitutions with which α holds true in the current execution history position i. Recall that, each pair of form $\langle\sigma, k\rangle \in S_\alpha$ requires that α holds true with substitution σ from the execution history position k to the current execution history position i. The first thing we do is to get all substitutions for which α starts holding true from the current execution history position i. Thus, these pairs of form $\langle\sigma, i\rangle$ are stored in the set S^{α}_{new}, denoting that we have to add these new pairs. The next thing we see is whether we can extend some of the existing substitution, position pairs with updated substitutions with which α holds true in the current state, which are in turn stored in the set $S^{\alpha}_{\text{update}}$. We then have to throw out pairs whose substitution component has been updated with new substitutions and they are in turn stored in the set $S^{\alpha}_{\text{remove}_1}$. Then we have to throw out all pairs of form $\langle\sigma, k\rangle$ if α does not hold true in the current state with substitution σ. This pairs are stored in the set $S^{\alpha}_{\text{remove}_2}$.

Finally, to obtain S^i_α, we remove the pairs in the sets $S^\alpha_{\text{remove}_1}$ and $S^\alpha_{\text{remove}_2}$ from $S^{(i-1)}_\alpha$, and add the pairs in the sets S^α_{new} and S^α_{update}.

$$S_{R_1} \leftarrow \{\sigma \mid \langle \sigma, i \rangle \in S^i_\beta\}$$
$$S_{R_2} \leftarrow \{\sigma \bowtie \sigma_1 \mid \exists k, j. \langle \sigma, k \rangle \in S^i_\beta \wedge \langle \sigma_1, j \rangle \in S^i_\alpha \wedge (j \leq (k+1)) \wedge \sigma \bowtie \sigma_1 \neq \textbf{\textit{empty}}\}$$
$$S^i_R \leftarrow S_{R_1} \bigcup S_{R_2}$$

Finally, we show how to obtain the structure S_R at the current execution history position i when we have already calculated the structure S_β and S_α for the current execution history position i. Recall that S_R is a set of substitutions, each element σ of which represents that the formula $\alpha \mathcal{S} \beta$ is true with the substitution σ in the current execution history position. Note that from the semantics of $\mathcal{S}$ we know that, $\alpha \mathcal{S} \beta$ holds in the current execution history position i if the formula β holds true in the current execution history position i. Thus, we calculate the set S_{R_1} which contains all the substitutions with which the formula β holds true in the current state. We then calculate the set of substitutions for some abbreviations of which β held true in some prior execution history position k and α has held from execution history position $k+1$ till the current execution history position with some abbreviation of that substitution. We finally union the two sets to obtain the structure S_R for the current execution history position.

Algorithm 4.1 contains the pseudo-code for updating the structures appropriately. Note that the structures are stored in persistent store as part of our state. Next we discuss the form of our state and how to access the different structures of different execution history point.

4.3.6.1 States Storing Summary Structures

Recall that, we build summary structures for all temporal formulas of form $\alpha \mathcal{S} \beta$ which have the label **B**. The substitutions are stored in the state of the system. The state contains for each position of the execution history, a list of structures corresponding to each temporal formula of form $\alpha \mathcal{S} \beta$

Algorithm 4.1 The definition of the ***bts*** function.

Input: A prefix of a history $\angle$, a position in the history i, a state $\pi = (A, i)$, and a formula φ of form $\alpha \mathcal{S} \beta$.
Output: Returns a state π' that is strongly consistent at i with respect to $\angle$ and φ.

1: Allocate space for structures in $\pi.A(i)(\varphi)$
2: **if** $(i = 0)$ **then** /* **Initial state** */
3: $\quad \pi.A(i)(\varphi).S_\beta \leftarrow \emptyset$; $\quad \pi.A(i)(\varphi).S_\alpha \leftarrow \emptyset$; $\quad \pi.A(i)(\varphi).S_R \leftarrow \emptyset$;
4: **else**
5: $\quad \pi.A(i)(\varphi).S_\beta \leftarrow \pi.A(i-1)(\varphi).S_\beta$; $\quad \pi.A(i)(\varphi).S_\alpha \leftarrow \pi.A(i-1)(\varphi).S_\beta$; $\quad \pi.A(i)(\varphi).S_R \leftarrow \emptyset$
6: $\Sigma_\beta \leftarrow \textbf{\textit{ips}}(\angle, i, \pi, \{\bullet\}, \beta)$
7: **for all** $(\sigma \in \Sigma_\beta)$ **do**
8: $\quad found \leftarrow false$
9: $\quad$ **for all** $(\langle \sigma_1, k \rangle \in \pi.A(i)(\varphi).S_\beta)$ **do**
10: $\quad\quad$ **if** $(\sigma \bowtie \sigma_1 \neq \textbf{\textit{empty}})$ **then**
11: $\quad\quad\quad \pi.A(i)(\varphi).S_\beta \leftarrow \pi.A(i)(\varphi).S_\beta \setminus \langle \sigma_1, k \rangle$
12: $\quad\quad\quad \pi.A(i)(\varphi).S_\beta \leftarrow \pi.A(i)(\varphi).S_\beta \cup \langle \sigma \bowtie \sigma_1, k \rangle$
13: $\quad\quad\quad found \leftarrow true$
14: $\quad$ **if** $(found = false)$ **then**
15: $\quad\quad \pi.A(i)(\varphi).S_\beta \leftarrow \pi.A(i)(\varphi).S_\beta \cup \langle \sigma, i \rangle$
16: $\Sigma_\alpha \leftarrow \emptyset$
17: **for all** $(\langle \sigma, k \rangle \in \pi.A(i)(\varphi).S_\beta)$ **do**
18: $\quad$ **if** $(k \neq i)$ **then**
19: $\quad\quad \Sigma_t \leftarrow \textbf{\textit{ips}}(\angle, i, \pi, \sigma, \alpha)$; $\quad \Sigma_\alpha \leftarrow \Sigma_\alpha \cup \Sigma_t$
20: $marked \leftarrow \emptyset$
21: **for all** $(\sigma \in \Sigma_\alpha)$ **do**
22: $\quad found \leftarrow false$
23: $\quad$ **for all** $(\langle \sigma_1, k \rangle \in \pi.A(i)(\varphi).S_\alpha)$ **do**
24: $\quad\quad$ **if** $(\sigma \bowtie \sigma_1 \neq \textbf{\textit{empty}})$ **then**
25: $\quad\quad\quad \pi.A(i)(\varphi).S_\alpha \leftarrow \pi.A(i)(\varphi).S_\alpha \setminus \langle \sigma_1, k \rangle$
26: $\quad\quad\quad \pi.A(i)(\varphi).S_\alpha \leftarrow \pi.A(i)(\varphi).S_\alpha \cup \langle \sigma \bowtie \sigma_1, k \rangle$
27: $\quad\quad\quad found \leftarrow true$
28: $\quad\quad\quad marked \leftarrow marked \cup \{\sigma \bowtie \sigma_1\}$
29: $\quad$ **if** $(found = false)$ **then**
30: $\quad\quad \pi.A(i)(\varphi).S_\alpha \leftarrow \pi.A(i)(\varphi).S_\alpha \cup \langle \sigma, i \rangle$
31: $\quad\quad marked \leftarrow marked \cup \{\sigma\}$
32: **for all** $(\langle \sigma, k \rangle \in \pi.A(i)(\varphi).S_\alpha)$ **do**
33: $\quad$ **if** $(\sigma \notin marked)$ **then**
34: $\quad\quad \pi.A(i)(\varphi).S_\alpha \leftarrow \pi.A(i)(\varphi).S_\alpha \setminus \langle \sigma, k \rangle$
35: $S_{R_1} \leftarrow \emptyset$; $\quad S_{R_2} \leftarrow \emptyset$
36: **for all** $(\langle \sigma, k \rangle \in \pi.A(i)(\varphi).S_\beta)$ **do**
37: $\quad$ **if** $(k = i)$ **then**
38: $\quad\quad S_{R_1} \leftarrow S_{R_1} \cup \{\sigma\}$
39: $\quad$ **else**
40: $\quad\quad$ **for all** $(\langle \sigma_1, j \rangle \in \pi.A(i)(\varphi).S_\alpha)$ **do**
41: $\quad\quad\quad$ **if** $(\sigma \bowtie \sigma_1 \neq \textbf{\textit{empty}} \wedge j \leq (k+1))$ **then**
42: $\quad\quad\quad\quad S_{R_2} \leftarrow S_{R_2} \cup \{\sigma \bowtie \sigma_1\}$
43: $\pi.A(i)(\varphi).S_R \leftarrow S_{R_1} \cup S_{R_2}$
44: **return** π

$$\text{b-tsub}(\varphi) = \begin{cases} \emptyset & \text{if } \varphi \equiv \top | \bot | p(t_1, \ldots, p_n) \\ \text{b-tsub}(\varphi) \cup \text{b-tsub}(\varphi_2) & \text{if } \varphi \equiv \varphi_1 \vee \varphi_2 | \varphi_1 \wedge \varphi_2 \\ \{\varphi\} \cup \text{b-tsub}(\varphi) \cup \text{b-tsub}(\varphi_2) & \text{if } \varphi \equiv \alpha \mathcal{S} \beta \text{ and } \mathbf{B} \in \mathit{label}(\varphi) \\ \text{b-tsub}(\varphi) \cup \text{b-tsub}(\varphi_2) & \text{if } \varphi \equiv \alpha \mathcal{S} \beta \text{ and } \mathbf{B} \notin \mathit{label}(\varphi) \\ \text{b-tsub}(\varphi_1(\vec{x})) \cup \text{b-tsub}(\varphi_2(\vec{x})) & \text{if } \varphi \equiv \forall \vec{x}.(\varphi_1(\vec{x}) \rightarrow \varphi_2(\vec{x})) \\ \text{b-tsub}(\varphi(\vec{x})) & \text{if } \varphi \equiv \exists \vec{x}.\varphi(\vec{x}) \end{cases}$$

Figure 4.4: Function definition: b-tsub(φ)

with label **B**. If we want to access the satisfiable substitutions for a specific temporal formula in a specific position j in the execution history, we just access the corresponding structure specific to that position and formula. The state also keeps track of the maximum position of the execution history we have seen so far. A state that store all the summary structures is formally defined as follows.

Definition 36 (State)**.** *A state, denoted by* π*, has the type* $(\mathbb{N} \rightarrow (\mathit{Formula} \rightarrow \mathit{Structure})) \times \mathbb{N}$*. Each element of Structure has the type* $(S_\alpha \times S_\beta \times S_R)$*. Both* S_α *and* S_β *have the type* $(\mathit{Substitution} \rightarrow \mathbb{N})$*.* S_R *has the type* $(\mathit{List}\ \textbf{\textit{of}}\ \mathit{Substitution})$*.*

Given a state $\pi = \langle A, i \rangle$ (where $i \in \mathbb{N}$) and a specific formula φ, and a $j \leq i$ and $j \in \mathbb{N}$, we can access the S_α element of the structure of formula φ at position j as $\pi.A(j)(\varphi).S_\alpha$. In the same vein, for a given state $\pi = \langle A, i \rangle$ (where $i \in \mathbb{N}$) and a specific formula φ, and a $j \leq i$ and $j \in \mathbb{N}$, we can access the structure of formula φ at position j as $\pi.A(j)(\varphi)$. Thus, $\pi.A(j)(\varphi)$ will return a triple of the structures of form $\langle \pi.A(j)(\varphi).S_\alpha, \pi.A(j)(\varphi).S_\beta, \pi.A(j)(\varphi).S_R \rangle$. For concise representation, when we write $\langle \sigma, k \rangle \in \pi.A(j)(\varphi).S_\beta$, we mean $\sigma \in \textbf{\textit{domain}}(\pi.A(j)(\varphi).S_\beta)$ and $k = \pi.A(j)(\varphi).S_\beta(\sigma)$. Similarly, we use $\forall \langle \sigma, k \rangle \in \pi.A(j)(\varphi).S_\beta$ to iterate over all $\langle \sigma, k \rangle$ pairs such that $\sigma \in \textbf{\textit{domain}}(\pi.A(j)(\varphi).S_\beta)$ and $k = \pi.A(j)(\varphi).S_\beta(\sigma)$.

In Figure 4.4, we present a function b-tsub which takes as input a formula φ and it returns all the sub-formula of φ which is of form $\alpha \mathcal{S} \beta$ and additionally has the label **B**. Using the definition of b-tsub, we define another function b-s-tsub which takes as input a formula φ and returns all the strict sub-formula (not including φ) which is of form $\alpha \mathcal{S} \beta$ and additionally has the label **B**.

Note that, $\text{b-s-tsub}(\varphi) = \text{b-tsub}(\varphi) \setminus \{\varphi\}$. We use the definitions of b-tsub and b-s-tsub to define "*strongly*" and "*weakly*" consistent states.

Definition 37 (Buildable Strict Temporal Sub-formula)**.** *Given a formula* φ*, the set of buildable strict temporal sub-formulas of* φ *is denoted by* $\text{b-s-tsub}(\varphi)$ *and is defined as* $\text{b-s-tsub}(\varphi) = \text{b-tsub}(\varphi) \setminus \{\varphi\}$.

We now define what it means for a given state π to be *well-formed* with respect to a given execution history $\mathcal{L}$, a position in the history j, and a formula of form $\alpha \mathcal{S} \beta$.

Definition 38 (Well-formed State, Ψ)**.** *Given a state* $\pi = \langle A, i \rangle$ *(where* $i \in \mathbb{N}$*), we say* π *is well-formed at* $j \in \mathbb{N}$ *(where* $j \leq i$*) with respect to a history* $\mathcal{L}$ *and a formula* φ *of form* $\alpha \mathcal{S} \beta$*, where* $\emptyset \vdash_{\boldsymbol{B}} \varphi : \chi_O$*,* $\chi_O \subseteq fv(\varphi)$ *and consequently* $\emptyset \vdash_{\boldsymbol{B}} \beta : \chi_1$ *and* $\chi_1 \vdash_{\boldsymbol{B}} \alpha : \chi_2$*, denoted by* $\Psi(\mathcal{L}, \pi, \varphi, j)$*, if all of the following hold:*

1. ***(SOUNDNESS**-$\pi.A(j)(\varphi).S_\alpha$**)***

   $\forall \langle \sigma, k \rangle \in \pi.A(j)(\varphi).S_\alpha.(\textbf{\textit{domain}}(\sigma) = (\chi_1 \cup \chi_2) \wedge \forall l \in \mathbb{N}.((k \leq l \leq j) \wedge \forall \sigma'.(\sigma' \geq \sigma \rightarrow \mathcal{L}, l \models \alpha\sigma'))$

2. ***(COMPLETENESS**-$\pi.A(j)(\varphi).S_\alpha$**)***

   $\forall \sigma. \forall l \in \mathbb{N}.(\ ((l < j) \ \wedge \ \exists \sigma_\beta.((\textbf{\textit{domain}}(\sigma) = \chi_1 \cup \chi_2) \ \wedge \ (\sigma \geq \sigma_\beta) \wedge$

   $(\exists \sigma'.\sigma' \geq \sigma \wedge \forall k \in [l+1, \ldots, j].\mathcal{L}, k \models \alpha\sigma'))) \rightarrow \quad \langle \sigma, l+1 \rangle \in \pi.A(j)(\varphi).S_\alpha\ )$

3. ***(SOUNDNESS**-$\pi.A(j)(\varphi).S_\beta$**)***

   $\forall \langle \sigma, k \rangle \in \pi.A(j)(\varphi).S_\beta.$

   $(\textbf{\textit{domain}}(\sigma) = \chi_O \wedge \forall \sigma'.(\sigma' \geq \sigma \rightarrow \mathcal{L}, k \models \beta\sigma' \wedge \ \nexists l \in \mathbb{N}.\ \nexists \sigma''.(\sigma'' \geq \sigma \wedge (k < l \leq j) \wedge \mathcal{L}, l \models \beta\sigma'')))$

4. ***(COMPLETENESS**-$\pi.A(j)(\varphi).S_\beta$**)***

   $\forall \sigma. \forall l \in \mathbb{N}.(\textbf{\textit{domain}}(\sigma) = \chi_O \wedge \exists \sigma'.\sigma' \geq \sigma \wedge \mathcal{L}, l \models \beta\sigma' \wedge \ \nexists k.\ \nexists \sigma''.(\sigma'' \geq \sigma \wedge l < k \leq j \wedge$

   $\mathcal{L}, k \models \beta\sigma'') \rightarrow \langle \sigma, j \rangle \in \pi.A(j)(\varphi).S_\beta)$

5. ***(SOUNDNESS-$\pi.A(j)(\varphi).S_R$)***

   $\forall\sigma \in \pi.A(j)(\varphi).S_R.(\textbf{\textit{domain}}(\sigma) = \chi_O \wedge \forall\sigma'.(\sigma' \geq \sigma \rightarrow \mathcal{L}, j \models (\alpha \mathcal{S} \beta)\sigma'))$

6. ***(COMPLETENESS-$\pi.A(j)(\varphi).S_R$)***

   $\forall\sigma.((\textbf{\textit{domain}}(\sigma) = \chi_O \wedge \exists\sigma'.(\sigma' \geq \sigma \wedge \mathcal{L}, j \models (\alpha \mathcal{S} \beta)\sigma')) \rightarrow \sigma \in \pi.A(j)(\varphi).S_R)$

Based on the definition of well-formed states we now introduce the readers with the notion of what it means for a state to be "*consistent*" with respect to a given formula of form $\alpha \mathcal{S} \beta$, an execution history $\mathcal{L}$, and position in the history $j \in \mathbb{Z}$.

Definition 39 (Strong Consistency)**.** *A state $\pi = \langle A, i\rangle$ (where $i \in \mathbb{N}$) is* strongly consistent *at $j \in \mathbb{Z}$ (where $j \leq i$) with respect to a history $\mathcal{L}$ and a formula φ if: (1) $j < 0$ or (2) for all $\hat{\varphi} \in \mathsf{b\text{-}tsub}(\varphi)$ and for all $0 \leq k \leq j$, $\Psi(\mathcal{L}, \pi, \hat{\varphi}, k)$ holds.*

Definition 40 (Weak Consistency)**.** *A state $\pi = \langle A, i\rangle$ (where $i \in \mathbb{N}$) is* weakly consistent *at $j \in \mathbb{Z}$ (where $j \leq i$) with respect to a history $\mathcal{L}$ and a formula φ if: (1) $j < 0$ or (2) π is a strongly consistent state at $j-1$ with respect to $\mathcal{L}$ and φ, and additionally for all $\hat{\varphi} \in \mathsf{b\text{-}s\text{-}tsub}(\varphi)$, $\Psi(\mathcal{L}, \pi, \hat{\varphi}, j)$ holds.*

We now introduce how to calculate the size of a given state π with respect to a formula φ of form $\varphi_1 \mathcal{S} \varphi_2$ where $\mathbf{B} \in \mathit{label}(\varphi)$. According to this definition, we later show that when ***bts*** is called for a φ with the above constraints, the size of the state remains finite (Lemma 44).

Definition 41 (Size of a state with respect to a buildable temporal formula)**.** *Given a history $\mathcal{L}$, a formula φ of form $\varphi_1 \mathcal{S} \varphi_2$ such that $\emptyset \vdash_{\boldsymbol{B}} \varphi_1 \mathcal{S} \varphi_2 : \chi_O$ where $\chi_O \subseteq fv(\varphi_2)$, a state $\pi = \langle A, i\rangle$ where $i \in \mathbb{N}$, a position $j \in \mathbb{Z}$ such that $j \leq i$ and π is strongly consistent at j with respect to $\mathcal{L}$ and φ, then the size of π at j with respect to φ, denoted by $\Upsilon(\pi, j, \varphi)$, is defined as follows:*

- 0 *when* $j < 0$
- $\sum_{0 \leq k \leq j} (|\textbf{\textit{domain}}(\pi.A(k)(\varphi).S_\alpha)| + |\textbf{\textit{domain}}(\pi.A(k)(\varphi).S_\beta)| + |\pi.A(k)(\varphi).S_R|)$

4.3.7 The Algorithm

The top level function of our algorithm is the ***checkPolicyCompliance*** function. Initially, it calculates all the temporal sub-formula of the original policy φ for which it is necessary to build summary structures. The set of all such formulas are stored in the set denoted by S. It then runs an infinite loop which waits for states to be available. When a state is available, it analyzes it and when a violation of the policy is encountered it reports a violation.

Algorithm 4.2 The definition of the ***checkPolicyCompliance*** function.

Input: A history $\angle$ and a formula φ representing the policy.

```
 1: S ← b-tsub(φ);
 2: i ← 0;
 3: State π ← ∅;
 4: while (true) do
 5:     for (φ ∈ S in ascending formula size) do
 6:         π ← bts(∠, i, π, φ)
 7:     tVal ← cc(∠, i, π, φ)
 8:     if tVal = false then
 9:         REPORT VIOLATION
10:     i ← i + 1
```

$$\textbf{\textit{cc}}(\angle, i, \pi, \varphi) \quad = \quad \text{return} \quad \begin{cases} true & \text{if } \textbf{\textit{ips}}(\angle, i, \pi, \{\bullet\}, \varphi_2(\vec{x})) \neq \emptyset \\ false & \text{otherwise} \end{cases}$$

Figure 4.5: The definition of the ***cc*** function

In each iteration of the infinite loop, the ***checkCompliance*** function first updates all the necessary summary structures for all temporal formula with label **B** by calling the function ***bts***. The ***checkCompliance*** function loops through all the formulas in the set S in the ascending formula size guaranteeing that for two formulas φ_1 and φ_2 where φ_2 is a subformula of φ_1, summary structures for φ_2 be updated before summary structures for φ_1 as updating the summary structure for φ_1 the ***ips*** function might require to access the summary structure for formula φ_2. When the loop terminates, it means all the summary structures for all temporal formulas of form $\alpha \mathcal{S} \beta$ with label

B, has been updated. Then it calls the function ***cc*** which returns true when the current execution history position satisfies the policy and returns false, otherwise. The body of the ***cc*** function calls the ***ips*** function with the execution history $\mathcal{L}$, the current position in the execution history $i \in \mathbb{N}$, the policy φ, the current state containing all the updated summary structures, and the identity input substitution ($\{\bullet\}$). When ***ips*** returns $\emptyset$ it denotes that the policy is violated and the ***cc*** function in turn returns false.

We now briefly discuss select cases of the ***ips*** function based on the structure of the input formula φ. As mentioned before, when $\varphi \equiv p(t_1, \ldots, t_n)$, we just call the ***sat*** function and the mode checking guarantees that the precondition of ***sat*** is satisfied. When $\varphi \equiv \top$, the ***ips*** function return $\emptyset$ denoting there are no substitutions for which $\bot$ holds true. Consider $\varphi \equiv \varphi_1 \vee \varphi_2$, in that case, we call ***ips*** for φ_1 and φ_2 respectively and take union of both the results. In the case, $\varphi \equiv \alpha \mathcal{S} \beta$ and it has the label **B**, then ***ips*** access the associated summary structure and join it with the input substitution σ_{in}, to get the substitutions which is consistent with σ_{in}. Finally, let us consider $\varphi \equiv \forall \vec{x}.(\varphi_1(\vec{x}) \rightarrow \varphi_2(\vec{x}))$ for which ***ips*** is called first for $\varphi_1(\vec{x})$ and using each of substitution returned by ***ips*** as input substitution ***ips*** is called for $\varphi_2(\vec{x})$. If one of the ***ips*** calls return $\emptyset$, it signifies that there exists one substitution which satisfy the antecedent but does not satisfy the consequent, which is a violation according to the semantics and thus ***ips*** returns $\emptyset$. However, if this is not the case that means all substitutions that satisfy the antecedent also satisfy the consequent, then ***ips*** returns σ_{in}.

Now consider a very simple example policy $\varphi \equiv \text{send}(p_1^-, p_2^-, m^-) \wedge \text{contains}(m^+, q^-, t^-)$. Let us assume according to the current execution we know that the following relations are true: $\text{send}(H_1, Doc_1, m_1)$, $\text{send}(H_1, Doc_2, m_2)$, $\text{contains}(m_1, Alice, psych\text{-}notes)$, $\text{contains}(m_2, Alice, X\text{-}Ray\text{-}report)$, $\text{contains}(m_3, Bob, PHI)$, $\text{contains}(m_4, John, blood\text{-}test)$, $\text{contains}(m_5, Eve, PHI)$. Let us consider we make the following call to ***ips***: ***ips***$(\mathcal{L}, i, \pi, \{\bullet\}, \text{send}(p_1^-, p_2^-, m^-) \wedge \text{contains}(m^+, q^-, t^-))$. According to the definition of ***ips***, ***ips*** is first called for $\text{send}(p_1^-, p_2^-, m^-)$ in the following way: ***ips***$(\mathcal{L}, i, \pi, \{\bullet\}, \text{send}(p_1^-, p_2^-, m^-)$. This call to ***ips*** actually falls into the base case (for predicates) and ***sat*** is in turn called in the following way: ***sat***$(\mathcal{L}, i, \text{send}(p_1^-, p_2^-, m^-), \{\bullet\})$. As the precondition of ***sat*** is satisfied, it returns the following set of substitutions: $\Sigma = \{\{p_1 \mapsto H_1, p_2 \mapsto$

$$
\begin{array}{lcl}
\textbf{\textit{ips}}(\mathcal{L}, i, \pi, \sigma_{\text{in}}, \top) & = & \{\sigma_{\text{in}}\} \\
\textbf{\textit{ips}}(\mathcal{L}, i, \pi, \sigma_{\text{in}}, \bot) & = & \emptyset \\
\textbf{\textit{ips}}(\mathcal{L}, i, \pi, \sigma_{\text{in}}, p(t_1, \ldots, t_n)) & = & \textbf{\textit{sat}}(\mathcal{L}, i, p(t_1, \ldots, t_n), \sigma_{\text{in}}) \\
\textbf{\textit{ips}}(\mathcal{L}, i, \pi, \sigma_{\text{in}}, \varphi_1 \vee \varphi_2) & = & \textbf{\textit{ips}}(\mathcal{L}, i, \pi, \sigma_{\text{in}}, \varphi_1) \bigcup \textbf{\textit{ips}}(\mathcal{L}, i, \pi, \sigma_{\text{in}}, \varphi_2) \\
\textbf{\textit{ips}}(\mathcal{L}, i, \pi, \sigma_{\text{in}}, \varphi_1 \wedge \varphi_2) & = & \bigcup_{\sigma_c \in \textbf{\textit{ips}}(\mathcal{L}, i, \pi, \sigma_{\text{in}}, \varphi_1)} \textbf{\textit{ips}}(\mathcal{L}, i, \pi, \sigma_c, \varphi_2) \\
\textbf{\textit{ips}}(\mathcal{L}, i, \pi, \sigma_{\text{in}}, \exists \vec{x}.\varphi(\vec{x})) & = & \textbf{\textit{ips}}(\mathcal{L}, i, \pi, \sigma_{\text{in}} \setminus \{\vec{x}\}, \varphi(\vec{x}))[\vec{x} \mapsto \sigma_{in}(\vec{x})]
\end{array}
$$

$$
\textbf{\textit{ips}}(\mathcal{L}, i, \pi, \sigma_{\text{in}}, \forall \vec{x}.(\varphi_1(\vec{x}) \rightarrow \varphi_2(\vec{x}))) = \begin{array}{l} \text{let} \quad \Sigma_1 \leftarrow \textbf{\textit{ips}}(\mathcal{L}, i, \pi, \sigma_{\text{in}} \setminus \{\vec{x}\}, \varphi_1(\vec{x})) \\ \text{return} \begin{cases} \emptyset & \text{if } \exists \sigma_c \in \Sigma_1.(\textbf{\textit{ips}}(\mathcal{L}, i, \pi, \sigma_c, \varphi_2(\vec{x})) = \emptyset) \\ \{\sigma_{\text{in}}\} & \text{otherwise} \end{cases} \end{array}
$$

$$
\textbf{\textit{ips}}(\mathcal{L}, i, \pi, \sigma_{\text{in}}, \alpha \mathcal{S} \beta) = \sigma_{\text{in}} \bowtie \pi.A(i)(\alpha \mathcal{S} \beta).S_R \text{ if } \mathbf{B} \in \mathit{label}(\alpha \mathcal{S} \beta)
$$

$$
\textbf{\textit{ips}}(\mathcal{L}, i, \pi, \sigma_{\text{in}}, \alpha \mathcal{S} \beta) = \begin{array}{l} \text{If } \mathbf{B} \notin \mathit{label}(\alpha \mathcal{S} \beta) \\ \text{let} \\ S_\beta \leftarrow \{\langle \sigma, k \rangle | k = \max l.((0 \leq l \leq i) \wedge \sigma \in \textbf{\textit{ips}}(\mathcal{L}, \\ l, \pi, \sigma_{\text{in}}, \beta))\} \\ S_{R_1} \leftarrow \{\sigma | \langle \sigma, i \rangle \in S_\beta\} \\ S_{R_2} \leftarrow \{\bowtie \sigma_\bullet^\alpha \neq \textbf{\textit{empty}} | \exists \langle \sigma_\beta, k \rangle \in S_\beta.k < i \wedge \\ \qquad \forall l.(k < l \leq i \rightarrow \sigma_l^\alpha \in \textbf{\textit{ips}}(\mathcal{L}, l, \pi, \sigma_\beta, \varphi_1))\} \\ \text{return } S_{R_1} \cup S_{R_2} \end{array}
$$

Figure 4.6: The definition of the ***ips*** function.

$Doc_1, m \mapsto m_1\}, \{p_1 \mapsto H_2, p_2 \mapsto Doc_2, m \mapsto m_2\}\}$. According to definition of ***ips***, for each $\sigma \in \Sigma$, ***ips*** is called in the following way: $\textbf{\textit{ips}}(\mathcal{L}, i, \pi, \sigma, \text{contains}(m^+, q^-, t^-))$. We have two such calls: $\textbf{\textit{ips}}(\mathcal{L}, i, \pi, \{p_1 \mapsto H_1, p_2 \mapsto Doc_1, m \mapsto m_1\}, \text{contains}(m^+, q^-, t^-))$ and $\textbf{\textit{ips}}(\mathcal{L}, i, \pi, \{p_1 \mapsto H_2, p_2 \mapsto Doc_2, m \mapsto m_2\}, \text{contains}(m^+, q^-, t^-))$. For each of the calls, the base case is applicable. The first call to ***ips*** returns the following set of substitution: $\{\{p_1 \mapsto H_1, p_2 \mapsto Doc_1, m \mapsto m_1, q \mapsto Alice, t \mapsto psych\text{-}notes\}\}$. The second call to ***ips*** returns the following set of substitution: $\{\{p_1 \mapsto H_2, p_2 \mapsto Doc_2, m \mapsto m_2, q \mapsto Alice, t \mapsto X\text{-}Ray\text{-}report\}\}$. Thus, the set of substitutions returned by the original call to ***ips***, $\textbf{\textit{ips}}(\mathcal{L}, i, \pi, \{\bullet\}, \text{send}(p_1^-, p_2^-, m^-) \wedge \text{contains}(m^+, q^-, t^-))$, returns $\{\{p_1 \mapsto H_1, p_2 \mapsto Doc_1, m \mapsto m_1, q \mapsto Alice, t \mapsto psych\text{-}notes\}\} \bigcup \{\{p_1 \mapsto H_2, p_2 \mapsto Doc_2, m \mapsto m_2, q \mapsto Alice, t \mapsto X\text{-}Ray\text{-}report\}\}$.

Complexity of the algorithm. We now show the runtime complexity of each iteration of the ***checkPolicyCompliance*** function. In each iteration of the ***checkPolicyCompliance*** function, it first calls ***bts*** for each temporal sub-formula with label **B**. Then the function ***checkPolicyCompliance*** calls the ***cc*** function which in turn calls the ***ips*** function. Thus, the runtime complexity of one iteration of the ***checkPolicyCompliance*** for a given policy φ is $|\varphi| \times$ complexity of ***bts*** function $+$ complexity of ***ips*** function where $|\varphi|$ represents the size of the policy which is the number of sub-formulas. We first analyze the runtime complexity of the ***ips*** function. At each iteration of ***ips***, the number of satisfiable substitution for each predicate can be approximated by the number of satisfiable substitution that appear in the execution history which we denote as $|\mathcal{L}|$. Thus, the worst case runtime complexity of ***ips*** occurs when all of the formulas are of form $\alpha \mathcal{S} \beta$ in which case the complexity is $O(|\mathcal{L}|^{O(|\varphi|)})$. Similarly, the complexity of the ***bts*** function is also $O(|\mathcal{L}|^{O(|\varphi|)})$. Thus, the runtime complexity of each iteration of the ***checkPolicyCompliance*** function is $O(|\mathcal{L}|^{O(|\varphi|)})$.

We now discuss the space complexity of our algorithm. We keep 3 structures for each temporal subformula which has the label **B**. The number of such temporal subformula is bound by $|\varphi|$. We also keep such structure for each position of the execution history which can be viewed as a constant. For each structure the number of substitutions is a polynomial of $|\mathcal{L}|$. Thus, we need space polynomial to $|\varphi|$ and $|\mathcal{L}|$. Thus the complexity of the algorithm lies somewhere between $O(|\mathcal{L}|^{O(|\varphi|)})$ and PSPACE of $|\varphi|$ and $|\mathcal{L}|$.

4.3.8 Correctness and Properties of the Algorithm

Now that we have described our algorithm and all the necessary notions, in this Section we first show that our algorithm is sound and complete. Additionally, we also present other properties of our algorithm.

The following Lemma states that each of the substitutions σ returned by the ***ips*** function is an extension of the substitution σ_{in}, which ***ips*** takes as input. In short, $\sigma \geq \sigma_{in}$ for all σ returned by ***ips***. This Lemma is used by the next Lemma to show the soundness and correctness of both ***ips***

and ***bts*** function.

Lemma 42 (*ips* is Extension). *For all formulas φ, for all $j \in \mathbb{N}$, for all histories $\mathcal{L}$, for all states $\pi = (A, i)$ where $i \in \mathbb{N}$, for all substitutions σ_{in}, for any χ_C and χ_F, such that: (1) $\chi_C, \chi_F \vdash \varphi : \chi_O$ where $\chi_O \subseteq fv(\varphi)$, (2) $i \geq j$, (3) $\textbf{\textit{domain}}(\sigma_{in}) \supseteq \chi_C \cup \chi_F$, (4) π is strongly consistent at i with respect to φ and $\mathcal{L}$, if $\textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{in}, \varphi) = \Sigma_{out}$, then for all $\sigma \in \Sigma_{out}$ it holds that $\sigma \geq \sigma_{in}$.*

Proof. Induction on the structure of φ.

Case $\varphi \equiv \top$.

Then $\Sigma_{out} = \{\sigma_{in}\}$, and $\sigma_{in} \leq \sigma_{in}$.

Case $\varphi \equiv \bot$.

Since $\Sigma_{out} = \emptyset$, the statement is vacuously true.

Case $\varphi \equiv p(t_1, \ldots, t_n)$.

Then $\Sigma_{out} = \textbf{\textit{sat}}(\mathcal{L}, j, p(t_1, \ldots, t_n), \sigma_{\text{in}})$, and by Axiom 35, $\forall \sigma \in \textbf{\textit{sat}}(\mathcal{L}, j, p(t_1, \ldots, t_n), \sigma_{\text{in}}).\sigma \geq \sigma_{in}$.

Case $\varphi \equiv \varphi_1 \vee \varphi_2$.

Then $\Sigma_{out} = \textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{in}, \varphi_1) \cup \textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{in}, \varphi_2)$. W.l.o.g., $\sigma \in \textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{in}, \varphi_1)$. Inspection of the applicable mode checking judgements verifies that the inductive hypothesis is applicable, which yields that $\sigma \geq \sigma_{in}$.

Case $\varphi \equiv \varphi_1 \wedge \varphi_2$.

Then $\Sigma_{out} = \bigcup_{\sigma_c \in \textbf{\textit{ips}}(\mathcal{L}, i, \pi, \sigma_{\text{in}}, \varphi_1)} \textbf{\textit{ips}}(\mathcal{L}, i, \pi, \sigma_c, \varphi_2)$. If $\sigma \in \Sigma_{out}$, then there exists $\sigma_c \in \textbf{\textit{ips}}(\mathcal{L}, i, \pi, \sigma_{\text{in}}, \varphi_1)$ such that $\sigma \in \textbf{\textit{ips}}(\mathcal{L}, i, \pi, \sigma_c, \varphi_2)$ such inspection of the applicable mode checking judgements verifies that the inductive hypothesis is applicable, which first yields that $\sigma_c \geq \sigma_{in}$, and then $\sigma \geq \sigma_c$. By transitivity of $\geq$, $\sigma \geq \sigma_{in}$.

Case $\varphi \equiv \exists \vec{x}.\varphi_1(\vec{x})$.

Then $\Sigma_{out} = \textbf{\textit{ips}}(\mathcal{L}, i, \pi, \sigma_{in} \setminus \{\vec{x}\}, \varphi(\vec{x}))[\vec{x} \mapsto \sigma_{in}(\vec{x})]$. Then there exists $\sigma' \in \textbf{\textit{ips}}(\mathcal{L}, i, \pi, \sigma_{in} \setminus \{\vec{x}\}, \varphi(\vec{x}))$ such that $\sigma = \sigma'[\vec{x} \mapsto \sigma_{in}(\vec{x})]$. By inductive hypothesis, $\sigma' \geq \sigma_{in} \setminus \{\vec{x}\}$. Then $\sigma \geq \sigma_{in}$.

Case $\varphi \equiv \forall \vec{x}.(\varphi_1(\vec{x}) \rightarrow \varphi_2(\vec{x}))$.

Then $\Sigma_{out} = \{\sigma_{in}\}$, and $\sigma_{in} \leq \sigma_{in}$.

Case $\varphi \equiv \varphi_1 \mathcal{S} \varphi_2$.

Then $\mathbf{B} \in label(\varphi)$ or $\mathbf{B} \notin label(\varphi)$.

Sub-Case $\mathbf{B} \in label(\varphi)$.

Then $\Sigma_{out} = \sigma_{in} \bowtie \pi.A(i)(\varphi_1 \mathcal{S} \varphi_2).S_R$, so by $\bowtie$ properties $\forall \sigma \in \Sigma_{out}.\sigma \geq \sigma_{in}$.

Sub-Case $\mathbf{B} \notin label(\varphi)$.

Then $\sigma \in S_{R_1}$ or $\sigma \in S_{R_2}$.

Sub-Sub-Case $\sigma \in S_{R_1}$.

Then $\langle \sigma, j \rangle \in S_{\varphi_2}$, so $\sigma \in \textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{in}, \varphi_2)$. By inductive hypothesis, $\sigma \geq \sigma_{in}$.

Sub-Sub-Case $\sigma \in S_{R_2}$.

Then $\sigma \in S_{\varphi_1} = \{\bowtie\sigma'_\bullet \neq \textbf{\textit{empty}} | \exists \langle \sigma_\beta, k \rangle \in S_{\varphi_2}.k < j \wedge \forall l.(k < l \leq j \rightarrow \sigma'_l \in \textbf{\textit{ips}}(\mathcal{L}, l, \pi, \sigma_\beta, \varphi_1))\}$. Then $\sigma = \bowtie\sigma'_\bullet$ for some $\sigma'_\bullet$ with a certain k, so $\forall l.k < l \leq j \rightarrow \sigma \geq \sigma'_l$ by $\bowtie$ properties. By inductive hypothesis, $\forall l.k < l \leq j \rightarrow \sigma'_l \geq \sigma_\beta$, thus $\sigma \geq \sigma_\beta$. Since $\langle \sigma_\beta, k \rangle \in S_{\varphi_2}$, $\sigma_\beta \in \textbf{\textit{ips}}(\mathcal{L}, k, \pi, \sigma_{in}, \varphi_2)$. By inductive hypothesis, $\sigma_\beta \geq \sigma_{in}$. By transitivity, $\sigma \geq \sigma_{in}$.

□

The following Lemma specifies that both ***bts*** and ***ips*** is sound and complete for well-moded policies.

Lemma 43 (Correctness of ***bts*** and ***ips*** function)**.** *The functions* $\textbf{\textit{bts}} : Log \times \mathbb{N} \times State \times Formula \rightarrow State$ *and* $\textbf{\textit{ips}} : Log \times \mathbb{N} \times State \times Substitution \times Formula \rightarrow SubstitutionList$*, are correct if both the following hold:*

1. *For all formula* φ*, for all histories* $\mathcal{L}$*, for a specific* $i \in \mathbb{N}$*, for all state* $\pi = \langle A, i \rangle$ *where* φ *is of form* $\alpha \mathcal{S} \beta$*,* $\emptyset \vdash_{\textbf{\textit{B}}} \varphi : \chi_O$ *where* $\chi_O \subseteq fv(\varphi)$*, and* π *is weakly consistent at* i *with respect to*

*$\mathcal{L}$ and φ, if **bts**$(\mathcal{L}, i, \pi, \varphi) = \hat{\pi}$ then $\hat{\pi} = \langle A', i \rangle$ is strongly consistent at i with respect to φ and $\mathcal{L}$.*

2. *For all formula φ, for all $j \in \mathbb{N}$, for all histories $\mathcal{L}$, for all state $\pi = \langle A, i \rangle$ where $i \in \mathbb{N}$, for all substitution σ_{in}, for some given χ_C and χ_F, such that: (1) $\chi_C, \chi_F \vdash \varphi : \chi_O$ where $\chi_O \subseteq fv(\varphi)$, (2) $i \geq j$, (3) **domain**$(\sigma_{in}) \supseteq \chi_C \cup \chi_F$, (4) π is strongly consistent at i with respect to φ and $\mathcal{L}$, if **ips**$(\mathcal{L}, j, \pi, \sigma_{in}, \varphi) = \Sigma_{out}$ then the following holds:*

   (a) ***(SOUNDNESS)***

   $$\forall \sigma \in \Sigma_{out}.(\textbf{\textit{domain}}(\sigma) \supseteq (\chi_O \cup \chi_C \cup \chi_F) \wedge \forall \sigma'.(\sigma' \geq \sigma \rightarrow \mathcal{L}, j \models \varphi\sigma'))$$

   (b) ***(COMPLETENESS)***

   $$\forall \sigma.((\sigma \geq \sigma_{in} \wedge \textbf{\textit{domain}}(\sigma) \supseteq fv(\varphi) \wedge \mathcal{L}, j \models \varphi.\sigma) \rightarrow (\exists \sigma_o \in \Sigma_{out}.(\sigma \geq \sigma_o)))$$

Proof. Mutual induction on the structure of φ

Case $\varphi \equiv \top$.

(**Soundness**)

(I) From definition of ***ips***, **ips**$(\mathcal{L}, j, \pi, \sigma_{\text{in}}, \top) = \Sigma_{out} = \{\sigma_{\text{in}}\}$

(II) From premise 1 and [BASE-1], $\chi_C, \chi_F \vdash \top : \emptyset$

(III) From premise 3, **domain**$(\sigma) \supseteq \chi_C \cup \chi_F$

From I, II, and III, for all $\sigma \in \Sigma_{out}$, **domain**$(\sigma) \supseteq \chi_C \cup \chi_F \cup \chi_O$ $[\chi_O = \emptyset]$

TS: $\forall \sigma'.\sigma' \geq \sigma \wedge \mathcal{L}, j \models \top\sigma'$

From the semantics, any $\sigma' \geq \sigma$ trivially satisfy $\mathcal{L}, j \models \top\sigma'$.

(**Completeness**)

Let $\sigma_o = \sigma_{in}$. Then by premise $\sigma_o \leq \sigma$, and by definition of ***ips*** $\sigma_o \in \Sigma_{out}$.

Case $\varphi \equiv \bot$.

(**Soundness**)

From definition of ***ips***, **ips**$(\mathcal{L}, j, \pi, \sigma_{\text{in}}, \bot) = \Sigma_{out} = \emptyset$. Thus the statement is vacuously true.

(**Completeness**)

For any σ, $\bot\sigma = \bot$. Since there are no $\mathcal{L}$, j such that $\mathcal{L}, j \models \bot$, the statement is vacuously true.

Case $\varphi \equiv p(t_1, \ldots, t_n)$.

(**Soundness**)

From definition of ***ips***, $\textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{\text{in}}, p(t_1, \ldots, t_n)) = \Sigma_{out} = \textbf{\textit{sat}}(\mathcal{L}, j, p(t_1, \ldots, t_n), \sigma_{\text{in}})$. From premise 1 and 3, pre-condition of the ***sat*** function is satisfied (Axiom 35). From Axiom 35, for all $\sigma \in \textbf{\textit{sat}}(\mathcal{L}, j, p(t_1, \ldots, t_n), \sigma_{\text{in}})$, $\textbf{\textit{domain}}(\sigma) = \chi_C \cup \chi_F \cup \chi_O$. Thus, we can write, for all $\sigma \in \Sigma_{out}$, $\textbf{\textit{domain}}(\sigma) \supseteq \chi_C \cup \chi_F \cup \chi_O$. It remains to show $\forall \sigma'.\sigma' \geq \sigma \rightarrow \mathcal{L}, j \models p(t_1, \ldots, t_n)\sigma'$. From Axiom 35, for all $\sigma \in \textbf{\textit{sat}}(\mathcal{L}, j, p(t_1, \ldots, t_n), \sigma_{\text{in}})$, $\mathcal{L}, j \models p(t_1, \ldots, t_n)\sigma$. Note that, the function ***sat*** returns grounding substitutions[2] for $p(t_1, \ldots, t_n)$. Thus, by Corollary 23 for all $\sigma' \geq \sigma$ where $\sigma \in \textbf{\textit{sat}}(\mathcal{L}, j, p(t_1, \ldots, t_n), \sigma_{\text{in}})$, $\mathcal{L}, j \models p(t_1, \ldots, t_n)\sigma'$ holds.

(**Completeness**)

Let $V = fv(p(t_1, \ldots, t_n))$. By the semantics of predicates, it must be that $\textbf{\textit{domain}}(\sigma) \supseteq V$. Then by premise and Lemma 22, $\mathcal{L}, j \models p(t_1, \ldots, t_n)[\sigma \downarrow V]$. Let $\sigma_o = \sigma \downarrow (V \cup \textbf{\textit{domain}}(\sigma_{in}))$. Since $\sigma_{in} \leq \sigma$ by premise, $\sigma \downarrow V \leq \sigma_o \leq \sigma$. By Corollary 23, it follows that $\mathcal{L}, j \models p(t_1, \ldots, t_n)\sigma_o$.

Case $\varphi \equiv \varphi_1 \vee \varphi_2$.

(**Soundness**)

Let $\Sigma_1 \leftarrow \textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{\text{in}}, \varphi_1)$ and $\Sigma_2 \leftarrow \textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{\text{in}}, \varphi_2)$. From definition of ***ips***, $\textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{\text{in}}, \varphi_1 \vee \varphi_2) = \Sigma_{out} = \Sigma_1 \cup \Sigma_2$. Then $\sigma \in \Sigma_1$ or $\sigma \in \Sigma_2$. W.l.o.g., $\sigma \in \Sigma_1$. By inspection of disjunction judgment judgements (and Lemmas 28, 32, and 34), $\chi_C, \chi_F \vdash \varphi_1 : \chi_1$. By I.H., $\textbf{\textit{domain}}(\sigma) \supseteq (\chi_C \cup \chi_F \cup \chi_1)$ and $\forall \sigma'.\sigma' \geq \sigma \implies \mathcal{L}, j \models \varphi_1\sigma'$. Since $\chi_O = \chi_1 \cap \chi_2$, $\textbf{\textit{domain}}(\sigma) \supseteq (\chi_C \cup \chi_F \cup \chi_1) \supseteq (\chi_C \cup \chi_F \cup \chi_O)$. Further, by semantics of $\vee$, $\forall \sigma''.L, j \models$

[2]Substitutions for all free variables

$\varphi_1\sigma'' \implies L, j \models (\varphi_1 \vee \varphi_2)\sigma''$. Thus, $\forall\sigma'.\sigma' \geq \sigma \implies \mathcal{L}, j \models (\varphi_1 \vee \varphi_2)\sigma'$, which concludes soundness.

(**Completeness**)

If $\mathcal{L}, j \models (\varphi_1 \vee \varphi_2)\sigma$, then $\mathcal{L}, j \models \varphi_1\sigma$ or $\mathcal{L}, j \models \varphi_2\sigma$. W.l.o.g., $\mathcal{L}, j \models \varphi_1\sigma$. Since $fv(\varphi_1) \subseteq fv(\varphi_1 \vee \varphi_2)$, by I.H. there exists $\sigma_o \in \textbf{\textit{ips}}(\mathcal{L}, j, \eta, \sigma_{in}, \varphi_1)$ such that $\sigma_o \leq \sigma$. By definition of ***ips***, $\sigma_o \in \Sigma_{out}$.

Case $\varphi \equiv \varphi_1 \wedge \varphi_2$.

(**Soundness**)

From the definition of ***ips***, $\textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{\text{in}}, \varphi_1 \wedge \varphi_2) = \bigcup_{\sigma_c \in \textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{\text{in}}, \varphi_1)} \textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_c, \varphi_1)$. Take an arbitrary $\sigma \in \Sigma_{out}$. Then there exists $\sigma_c \in \textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{in}, \varphi_1)$ such that $\sigma \in \textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_c, \varphi_2)$. By inspection of the applicable mode checking judgements (and Lemmas 32, 28), the inductive hypothesis is applicable and yields $\textbf{\textit{domain}}(\sigma_c) \supseteq \chi_C \cup \chi_F \cup \chi_1$. Now, with the additional help of Lemma 34 the inductive hypothesis yields $\textbf{\textit{domain}}(\sigma) \supseteq \chi_C \cup \chi_F \cup \chi_1 \cup \chi_2$. Since $\chi_O = \chi_1 \cup \chi_2$, $\textbf{\textit{domain}}(\sigma) \supseteq \chi_C \cup \chi_F \cup \chi_O$.

To Show: $\forall\sigma'.(\sigma' \geq \sigma \rightarrow (\mathcal{L}, j \models (\varphi_1 \wedge \varphi_2)\sigma'))$. Take any arbitrary σ' such that $\sigma' \geq \sigma$. By inductive hypothesis on φ_2 we have $\mathcal{L}, j \models \varphi_2\sigma'$. Further $\sigma' \geq \sigma_c$, since $\sigma \geq \sigma_c$ by Lemma 42. Then we can apply the inductive hypothesis and get $\mathcal{L}, j \models \varphi_1\sigma'$. From the semantics of $\wedge$, we have $\mathcal{L}, j \models (\varphi_1 \wedge \varphi_2)\sigma'$.

(**Completeness**)

If $\mathcal{L}, j \models (\varphi_1 \wedge \varphi_2)\sigma$, then $\mathcal{L}, j \models \varphi_1\sigma$ and $\mathcal{L}, j \models \varphi_2\sigma$. Since $fv(\varphi_i) \subseteq fv(\varphi_1 \wedge \varphi_2)$, the I.H. is applicable and guarantees $\exists\sigma_i \in \textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{in}, \varphi_i).\sigma_i \leq \sigma$. By Lemma 42, also $\sigma_i \geq \sigma_{in}$, so $\textbf{\textit{domain}}(\sigma_i) \supseteq \chi_C \cup \chi_F$. Let $\sigma_2' = \sigma \downarrow \textbf{\textit{domain}}(\sigma_1) \cup \textbf{\textit{domain}}(\sigma_2) \cup fv(\varphi_2)$, which is a prefix of σ and $\textbf{\textit{domain}}(\sigma_2') = \textbf{\textit{domain}}(\sigma_1) \cup \textbf{\textit{domain}}(\sigma_2) \cup fv(\varphi_2)$, since $\textbf{\textit{domain}}(\sigma_i) \subseteq \textbf{\textit{domain}}(\sigma)$ and $\textbf{\textit{domain}}(\sigma) \supseteq fv(\varphi) \supseteq fv(\varphi_2)$. So $\sigma_i \leq \sigma_2' \leq \sigma$. By inductive hypothesis for soundness, since $\sigma_2' \geq \sigma_2$, $\mathcal{L}, j \models \varphi_2\sigma_2'$. Thus we can apply the inductive hypothesis

with $\sigma_{in} = \sigma_1$ and get $\exists \sigma_o \in \textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_1, \varphi_2).\sigma_o \leq \sigma_2'$. By definition of $\textbf{\textit{ips}}$ $\sigma_o \in \Sigma_{out}$, and by transitivity $\sigma_o \leq \sigma$.

Case $\varphi \equiv \exists \vec{x}.\varphi_1(\vec{x})$.

(**Soundness**)

Let $\sigma \in \Sigma_{out}$. By definition, $\textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{\text{in}}, \exists \vec{x}.\varphi(\vec{x})) = \textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{\text{in}} \setminus \{\vec{x}\}, \varphi(\vec{x}))[\vec{x} \mapsto \sigma_{in}(\vec{x})]$. Thus there exists $\sigma_1 \in \textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{\text{in}} \setminus \{\vec{x}\}, \varphi(\vec{x}))$ such that $\sigma = \sigma_1[\vec{x} \mapsto \sigma_{in}(\vec{x})]$. By inspection of the mode checking judgements, we can apply the inductive hypothesis, which yields $\textbf{\textit{domain}}(\sigma_1) \supseteq \chi_I \cup \chi_F \cup \chi_1$ and $\forall \sigma'' \geq \sigma_1.\mathcal{L}, j \models \varphi_1(\vec{x})\sigma''$.

Let $X = \textbf{\textit{domain}}(\sigma_{in}) \cap \{\vec{x}\}$ and $X' = \{\vec{x}\} \setminus X = \{\vec{x}\} \setminus \textbf{\textit{domain}}(\sigma_{in})$. Then we get $\textbf{\textit{domain}}(\sigma) = \textbf{\textit{domain}}(\sigma_1[\vec{x} \mapsto \sigma_{in}(\vec{x})]) = (\textbf{\textit{domain}}(\sigma_1) \cup X) \setminus X' \supseteq (\chi_C \cup \chi_F \cup \chi_1 \cup X) \setminus X'$. Since $\textbf{\textit{domain}}(\sigma_{in}) \supseteq \chi_C \cup \chi_F$, $\chi_C \setminus X' = \chi_C$ and $\chi_F \setminus X' = \chi_F$. Thus $\textbf{\textit{domain}}(\sigma) \supseteq \chi_C \cup \chi_F \cup X \cup (\chi_1 \setminus X') \supseteq \chi_C \cup \chi_F \cup (\chi_1 \setminus \{\vec{x}\}) = \chi_C \cup \chi_F \cup \chi_O$.

Now take an arbitrary $\sigma' \geq \sigma$. Then $\sigma' = \sigma + \sigma^+$ for some σ^+, so $\sigma' = \sigma_1[\vec{x} \mapsto \sigma_{in}(\vec{x})] + \sigma^+$. Then $\sigma'' = \sigma'[\vec{x} \mapsto \sigma_1(\vec{x})] = \sigma_1 + \sigma^+[\vec{x} \mapsto \sigma_1(\vec{x})] \geq \sigma_1$. Thus, $\mathcal{L}, j \models \varphi_1(\vec{x})\sigma''$, or equivalently $\mathcal{L}, j \models \varphi_1(\vec{x})\sigma'[\vec{x} \mapsto \vec{t}]$. Thus, $\mathcal{L}, j \models \exists \vec{x}.\varphi_1(\vec{x})\sigma'$.

(**Completeness**)

$\mathcal{L}, j \models (\exists \vec{x}.\varphi_1(\vec{x}))\sigma$ if and only if $\mathcal{L}, j \models \exists \vec{x}.(\varphi_1(\vec{x})(\sigma \setminus \vec{x}))$ if and only if there exists $\vec{t}$ such that $\mathcal{L}, j \models \varphi_1(\vec{t})(\sigma \setminus \vec{x})$, which is equivalent to $\mathcal{L}, j \models \varphi_1(\vec{x})(\sigma[\vec{x} \mapsto \vec{t}])$. By $\sigma_{in} \leq \sigma$ and judgment judgements $fv(\varphi_1(\vec{x})) \subseteq \textbf{\textit{domain}}(\sigma_{in}) \cup \{\vec{x}\} \subseteq \textbf{\textit{domain}}(\sigma[\vec{x} \mapsto \vec{t}])$. Also $\sigma_{in} \setminus \{\vec{x}\} \leq \sigma[\vec{x} \mapsto \vec{t}]$. Thus, by inductive hypothesis, $\exists \sigma' \in \textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{in} \setminus \{\vec{x}\}, \varphi_1(\vec{x})).\sigma' \leq \sigma[\vec{x} \mapsto \vec{t}]$. Let $\sigma_o = \sigma'[\vec{x} \mapsto \sigma_{in}(\vec{x})]$. This is a prefix of σ (if σ_{in} maps $\vec{x}$, then σ maps to the same values since $\sigma_{in} \leq \sigma$) and by definition $\sigma_o \in \Sigma_{out}$.

Case $\varphi \equiv \forall \vec{x}.(\varphi_1(\vec{x}) \rightarrow \varphi_2(\vec{x}))$.

(**Soundness**)

Consider, $\textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{\text{in}}, \forall \vec{x}.(\varphi_1(\vec{x}) \rightarrow \varphi_2(\vec{x}))) \neq \emptyset$. In which case, $\textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{\text{in}}, \forall \vec{x}.(\varphi_1(\vec{x}) \rightarrow$

$\varphi_2(\vec{x}))) = \{\sigma_{\text{in}}\}$. Consider any σ_1 such that $\sigma_1 \in \textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{\text{in}}, \varphi_1(\vec{x}))$ and σ_2 such that $\sigma_2 \in \textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_1, \varphi_1(\vec{x}))$. Here, $\sigma_1 \neq \emptyset$ and $\sigma_2 \neq \emptyset$. From premise 3, we know $\textbf{\textit{domain}}(\sigma_{\text{in}}) \supseteq \chi_C \cup \chi_F$. We also know $\sigma_{\text{in}} \in \textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{\text{in}}, \forall\vec{x}.(\varphi_1(\vec{x}) \rightarrow \varphi_2(\vec{x})))$. From mode checking judgements we know, $\chi_O = \emptyset$. Thus, for $\sigma \in \textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{\text{in}}, \forall\vec{x}.(\varphi_1(\vec{x}) \rightarrow \varphi_2(\vec{x})))$, $\textbf{\textit{domain}}(\sigma) \supseteq \chi_C \cup \chi_F \cup \chi_O$.

To Show: $\forall\sigma'.(\sigma' \geq \sigma \rightarrow (\mathcal{L}, j \models (\forall\vec{x}.(\varphi_1(\vec{x}) \rightarrow \varphi_2(\vec{x})))\sigma'))$. We know from the definition $\sigma = \sigma_{\text{in}}$ as $\textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{\text{in}}, \forall\vec{x}.(\varphi_1(\vec{x}) \rightarrow \varphi_2(\vec{x}))) \neq \emptyset$. Take any arbitrary σ' such that $\sigma' \geq \sigma = \sigma_{in}$.

Inspection of the applicable mode checking judgements reveals that in all cases $fv(\phi_2(\vec{x})) \subseteq \chi_C \cup \chi_F \cup \chi_1$ (with transitivity and additivity of $\subseteq$). From premise (1) and Lemma 25 or Lemma 30, $\chi_1 \subseteq fv(\phi_1(\vec{x}))$. Thus $fv(\phi_2(\vec{x})) \subseteq \chi_C \cup \chi_F \cup fv(\phi_1(\vec{x}))$. Then always $fv(\phi_1(\vec{x})) \subseteq \chi_C \cup \chi_F \cup \{\vec{x}\}$, so that $fv(\phi_2(\vec{x})) \subseteq \chi_C \cup \chi_F \cup \{\vec{x}\}$. Finally, by premise 3 we know $\textbf{\textit{domain}}(\sigma_{\text{in}}) \supseteq (\chi_C \cup \chi_F)$, which means: **(C-i)** $fv(\phi_2(\vec{x})) \subseteq \textbf{\textit{domain}}(\sigma_{in}) \cup \{\vec{x}\}$.

$\mathcal{L}, j \models (\forall\vec{x}.(\varphi_1(\vec{x}) \rightarrow \varphi_2(\vec{x})))\sigma'$ is equivalent to $\forall\vec{t}.(\mathcal{L}, j \models (\varphi_1(\vec{x}))(\sigma'[\vec{x} \mapsto \vec{t}]) \rightarrow \mathcal{L}, j \models (\varphi_2(\vec{x}))(\sigma'[\vec{x} \mapsto \vec{t}]))$. Take any arbitrary $\vec{t}$ such that $\mathcal{L}, j \models (\varphi_1(\vec{x}))(\sigma'[\vec{x} \mapsto \vec{t}])$.

[Z] We then have to show that: $\mathcal{L}, j \models (\varphi_2(\vec{x}))(\sigma'[\vec{x} \mapsto \vec{t}])$. By I.H. (Completeness), $\exists\sigma_1 \in \textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{\text{in}} \setminus \{\vec{x}\}, \varphi_1(\vec{x})).\sigma_1 \leq \sigma'[\vec{x} \mapsto \vec{t}]$. By inspection of the mode checking judgements and I.H. (Soundness), $\textbf{\textit{domain}}(\sigma_1) \supseteq \textbf{\textit{domain}}(\sigma_{in}) \cup \{\vec{x}\}$. From construction, $\textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_1, \varphi_2(\vec{x})) \neq \emptyset$. Take an arbitrary σ_2 from this set. By I.H. (Soundness), **(C-ii)** $\sigma_2 \geq \sigma_1 \wedge \mathcal{L}, j \models \varphi_2(\vec{x})\sigma_2$.

Now, we will show that: $\exists\sigma_2^m \in \textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_1, \varphi_2(\vec{x})).(\sigma_2 \geq \sigma_2^m \wedge \textbf{\textit{domain}}(\sigma_2^m) \subseteq \textbf{\textit{domain}}(\sigma_{\text{in}}) \cup \{\vec{x}\} \wedge \mathcal{L}, j \models \varphi_2(\vec{x})\sigma_2^m)$ [$\mathcal{T}\,\mathcal{S}$]. (A-II) $\mathcal{L}, j \models \varphi_2(\vec{x})\sigma_2$ [From **(C-ii)**]. (A-III) $\mathcal{L}, j \models \varphi_2(\vec{x})(\sigma_2 \downarrow fv(\varphi_2(\vec{x})))$ [From Lemma 22 + A-II]. (A-IV) $\forall\sigma, \sigma', \varphi.((\textbf{\textit{domain}}(\sigma) = fv(\varphi) \wedge \textbf{\textit{domain}}(\sigma) \cap \textbf{\textit{domain}}(\sigma') = \emptyset \wedge \mathcal{L}, j \models \varphi\sigma) \rightarrow (\mathcal{L}, j \models \varphi[\sigma + \sigma']))$ [From Lemma 22]. (A-V) $fv(\varphi_2(\vec{x})) \subseteq \textbf{\textit{domain}}(\sigma_{\text{in}}) \cup \{\vec{x}\}$ [From **(C-i)**]. It follows that: $\exists Y.(fv(\varphi_2(\vec{x})) \cup Y) = \textbf{\textit{domain}}(\sigma_{\text{in}}) \cup \{\vec{x}\}$. (A-VI) $\mathcal{L}, j \models \varphi_2(\vec{x})(\sigma_2 \downarrow (\textbf{\textit{domain}}(\sigma_{\text{in}}) \cup \{\vec{x}\}))$. [From (A-III), (A-IV), (A-V)] (X) By

I.H. (Completeness), $\exists \sigma_2^m \leq (\sigma_2 \downarrow (\textbf{\textit{domain}}(\sigma_{\text{in}}) \cup \{\vec{x}\})).\sigma_2^m \in \textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_1, \varphi_2(\vec{x}))$ By I.H. (Soundness), $\mathcal{L}, j \models \varphi_2(\vec{x})\sigma_2^m$ [As $\sigma_2^m \geq \sigma_2^m$] Thus, we have shown the third conjunct of $\mathcal{T}\,\mathcal{S}$ to be true. (Y) We know $\sigma_2^m \leq \sigma_2 \downarrow (\textbf{\textit{domain}}(\sigma_{\text{in}}) \cup \{\vec{x}\})$ [From (X)] It implies that $\sigma_2^m \leq \sigma_2$. Thus, we have shown the first conjunct of $\mathcal{T}\,\mathcal{S}$ to be true. From (Y), $\textbf{\textit{domain}}(\sigma_2^m) \subseteq \textbf{\textit{domain}}(\sigma_2 \downarrow (\textbf{\textit{domain}}(\sigma_{\text{in}}) \cup \{\vec{x}\})) = \textbf{\textit{domain}}(\sigma_{\text{in}}) \cup \{\vec{x}\}$ [From (A-V)]. Thus, we have shown the second conjunct of $\mathcal{T}\,\mathcal{S}$ to be true. This implies that we have shown $\mathcal{T}\,\mathcal{S}$ to be true.

Now, if we can show that $\sigma'[\vec{x} \mapsto \vec{t}] \geq \sigma_2^m$ then from the third conjunct of $\mathcal{T}\,\mathcal{S}$ and I.H. (Soundness), we can show $\mathcal{L}, j \models (\varphi_2(\vec{x}))(\sigma'[\vec{x} \mapsto \vec{t}])$ to hold. $\sigma' \geq \sigma_2^m[\vec{x} \mapsto \vec{t}]$ is equivalent to the following:

[U] $\forall v \in \textbf{\textit{domain}}(\sigma_2^m).\sigma_2^m(v) = \sigma'[\vec{x} \mapsto \vec{t}](v)$.

From second conjunct of $\mathcal{T}\,\mathcal{S}$, $\textbf{\textit{domain}}(\sigma_2^m) \subseteq \textbf{\textit{domain}}(\sigma_{\text{in}}) \cup \{\vec{x}\}$. From this and **U**, we can say that for all $v \in \textbf{\textit{domain}}(\sigma_2^m)$, either **(E-1)** $v \in ((\textbf{\textit{domain}}(\sigma_{\text{in}}) \setminus \{\vec{x}\}) \cap \textbf{\textit{domain}}(\sigma_2^m))$ or **(E-2)** $v \in (\{\vec{x}\} \cap \textbf{\textit{domain}}(\sigma_2^m))$ holds.

(E-1) $v \in ((\textbf{\textit{domain}}(\sigma_{\text{in}}) \setminus \{\vec{x}\}) \cap \textbf{\textit{domain}}(\sigma_2^m))$. We know $\sigma' \geq \sigma_{\text{in}}$. It implies that $\sigma'(v) = \sigma_{\text{in}}(v)$. We also have $\textbf{\textit{domain}}(\sigma') \supseteq \textbf{\textit{domain}}(\sigma_{\text{in}})$.

Consider $v \notin \{\vec{x}\}$, so **[R-1]** $\sigma'[\vec{x} \mapsto \vec{t}](v) = \sigma_{\text{in}}(v)$. We know $\sigma_2 \geq \sigma_1 \geq \sigma_{\text{in}} \setminus \{\vec{x}\}$. We can write $\sigma_2 \geq \sigma_{\text{in}} \setminus \{\vec{x}\}$. This is equivalent to $\forall v_1 \in \textbf{\textit{domain}}(\sigma_{\text{in}}) \setminus \{\vec{x}\}.\sigma_2(v_1) = \sigma_{\text{in}} \setminus \{\vec{x}\}(v_1)$. We also know from the first conjunct of $\mathcal{T}\,\mathcal{S}$ that $\sigma_2 \geq \sigma_2^m$. It is equivalent to $\forall v_2 \in \textbf{\textit{domain}}(\sigma_2^m).\sigma_2(v_2) = \sigma_2^m(v_2)$. As $v \in ((\textbf{\textit{domain}}(\sigma_{\text{in}}) \setminus \{\vec{x}\}) \cap \textbf{\textit{domain}}(\sigma_2^m))$, $v \in \textbf{\textit{domain}}(\sigma_{\text{in}}) \setminus \{\vec{x}\}$ and $v \in \textbf{\textit{domain}}(\sigma_2^m)$. It implies that $\sigma_2^m(v) = \sigma_2(v) = \sigma_{\text{in}} \setminus \{\vec{x}\}(v)$. As $v \notin \{\vec{x}\}$, it implies that $\sigma_2^m(v) = \sigma_{\text{in}}(v)$. From above and **R-1**, we have $\sigma_2^m(v) = \sigma'[\vec{x} \mapsto \vec{t}](v)$.

(E-2) $v \in (\{\vec{x}\} \cap \textbf{\textit{domain}}(\sigma_2^m))$ We have to show that $\sigma_2^m(v) = \sigma'[\vec{x} \mapsto \vec{t}](v)$. We know $\sigma_2 \geq \sigma_2^m$ which implies that $\forall v_1 \in \textbf{\textit{domain}}(\sigma_2^m).\sigma_2(v_1) = \sigma_2^m(v_1)$. As $\sigma_2^m \in \textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_1, \varphi_2(\vec{x}))$, we have $\sigma_2^m \geq \sigma_1$. It implies that $\forall v_2 \in \textbf{\textit{domain}}(\sigma_1).\sigma_2^m(v_2) = \sigma_1(v_2)$. We also know $\sigma'[\vec{x} \mapsto \vec{t}] \geq \sigma_1$ which implies that $\forall v_3 \in \textbf{\textit{domain}}(\sigma_1).\sigma_1(v_3) = \sigma'[\vec{x} \mapsto \vec{t}](v_3)$. By inspecting the

mode checking judgements we know $\{\vec{x}\} \subseteq \chi_1$. Thus, we know $\textbf{\textit{domain}}(\sigma_1) \subseteq \chi_C \cup \chi_F \cup \{\vec{x}\}$. As $v \in \{\vec{x}\}$, it implies that $v \in \textbf{\textit{domain}}(\sigma_1)$. Thus, we have $\forall v \in \{\vec{x}\}.\sigma_2^m(v) = \sigma_1(v) = \sigma'[\vec{x} \mapsto \vec{t}](v)$.

(**Completeness**)

Trivially $\sigma_o = \sigma_{in}$.

Case $\varphi \equiv \varphi_1 \mathcal{S} \varphi_2$.

To show (1):

To show that the current state $\hat{\pi}$ is strongly consistent with respect to i and φ, we have to show according to the Definition 38 that all the substitutions in the structures satisfies the corresponding formula and the structures contains all the possible substitutions. This can be shown similarly as the soundness and completeness argument of ***ips*** for formulas of form $\alpha \mathcal{S} \beta$ where $\mathbf{B} \notin \mathit{label}(\alpha \mathcal{S} \beta)$ as the structure updating operations are similar to the calculation of the substitutions by ***ips***.

(**Soundness**)

Sub-Case $\mathbf{B} \in \mathit{label}(\varphi)$

From the definition of ***ips***, we get $\textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{\text{in}}, \varphi_1 \mathcal{S} \varphi_2) = \sigma_{\text{in}} \bowtie \pi.A(j)(\alpha \mathcal{S} \beta).S_R$. We can say that for all $\sigma \in \textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{\text{in}}, \varphi_1 \mathcal{S} \varphi_2)$ where $\sigma \neq \emptyset$, there exists a $\sigma_1 \in \pi.A(j)(\alpha \mathcal{S} \beta).S_R$ such that $\sigma = \sigma_{\text{in}} \bowtie \sigma_1$. We have to first show that for all $\sigma \in \textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{\text{in}}, \varphi_1 \mathcal{S} \varphi_2)$, $\textbf{\textit{domain}}(\sigma) \supseteq \chi_C \cup \chi_F \cup \chi_O$. From premise 3, Lemma 43 (1), and Definition 39, $\textbf{\textit{domain}}(\sigma_{\text{in}}) \supseteq \chi_C \cup \chi_F$ and $\textbf{\textit{domain}}(\sigma_1) \supseteq \chi_O$. As $\sigma = \sigma_{\text{in}} \bowtie \sigma_1$ and $\sigma \neq \emptyset$, we can see that $\textbf{\textit{domain}}(\sigma) \supseteq \chi_C \cup \chi_F \cup \chi_O$.

To Show: $\forall \sigma'.(\sigma' \geq \sigma \rightarrow (\mathcal{L}, j \models (\varphi_1 \mathcal{S} \varphi_2)\sigma'))$

Take any arbitrary, σ' such that $\sigma' \geq \sigma$. As $\sigma = \sigma_{\text{in}} \bowtie \sigma_1$ and $\sigma \neq \emptyset$, we can write $\sigma' \geq \sigma_1$.

From Lemma 43 (1) and Definition 39, we know that $\forall \sigma'_1.\sigma'_1 \geq \sigma_1 \rightarrow \mathcal{L}, j \models (\varphi_1 \mathcal{S} \varphi_2)\sigma'_1$.

It follows that $\mathcal{L}, j \models (\varphi_1 \mathcal{S} \varphi_2)\sigma'$, completing the proof.

Sub-Case $\mathbf{B} \notin label(\varphi)$

Take an arbitrary $\sigma \in \Sigma_{out}$. By definition, $\Sigma_{out} = \textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{\text{in}}, \varphi_1 \mathcal{S} \varphi_2) = S_{R_1} \cup S_{R_2}$. Thus, $\sigma \in S_{R_1}$ or $\sigma \in S_{R_2}$.

Sub-Sub-Case $\sigma \in S_{R_1}$

From definition of S_{R_1}, we know $S_{R_1} = \textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{\text{in}}, \varphi_2)$, so $\sigma \in \textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{\text{in}}, \varphi_2)$. We first show that $\textbf{\textit{domain}}(\sigma) \supseteq (\chi_C \cup \chi_F \cup \chi_O)$. From premise 3, we know that $\textbf{\textit{domain}}(\sigma_{\text{in}}) \supseteq (\chi_C \cup \chi_F)$. From the mode checking judgements for $\mathcal{S}$ and I.H., $\textbf{\textit{domain}}(\sigma) \supseteq (\chi_C \cup \chi_F \cup \chi_1)$. Since $\chi_O = \chi_1$ by the applicable judgements, $\textbf{\textit{domain}}(\sigma) \supseteq (\chi_C \cup \chi_F \cup \chi_O)$.

To Show: $\forall \sigma'.(\sigma' \geq \sigma \rightarrow (\mathcal{L}, j \models (\varphi_1 \mathcal{S} \varphi_2)\sigma'))$. Take an arbitrary σ' such that $\sigma' \geq \sigma$. From the semantics of $\mathcal{S}$ we know that, $\mathcal{L}, j \models (\varphi_1 \mathcal{S} \varphi_2)\sigma'$ if and only if there exists $k \in \mathbb{N}$ and $k \leq j$ such that $\mathcal{L}, k \models \varphi_2\sigma'$ and for all $l \in \mathbb{N}$ such that $k < l \leq j$, it implies that $\mathcal{L}, l \models \varphi_1\sigma'$ holds. So if $\mathcal{L}, j \models \varphi_2\sigma'$ holds, then $\mathcal{L}, j \models (\varphi_1 \mathcal{S} \varphi_2)\sigma'$ holds. Now since $\sigma \in \textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{in}, \varphi_2)$ and $\sigma' \geq \sigma$, by inductive hypothesis it follows that $\mathcal{L}, j \models (\varphi_2)\sigma'$. From this, it follows that $\mathcal{L}, j \models (\varphi_1 \mathcal{S} \varphi_2)\sigma'$.

Sub-Sub-Case $\sigma \in S_{R_2}$

Then there exist $\langle \sigma_\beta, k \rangle \in S_{\varphi_2}$ and $\sigma^\alpha_\bullet$, such that $\bowtie \sigma^\alpha_\bullet = \sigma$ and $k < j$ and for all l with $k < l \leq j$ we have $\sigma^\alpha_l \in \textbf{\textit{ips}}(\mathcal{L}, l, \pi, \sigma_\beta, \varphi_1)$. For brevity, from here on we assume l is sufficiently restricted to the domain of $\sigma^\alpha_\bullet$. By construction, $\sigma_\beta \in \textbf{\textit{ips}}(\mathcal{L}, k, \pi, \sigma_{in}, \varphi_2)$. By inductive hypothesis, $\textbf{\textit{domain}}(\sigma_\beta) \supseteq \chi_C \cup \chi_F \cup \chi_1$, and since $\chi_O = \chi_1$, $\textbf{\textit{domain}}(\sigma_\beta) \supseteq \chi_C \cup \chi_F \cup \chi_O$. Now by Lemma 42, $\forall l.\sigma^\alpha_l \geq \sigma_\beta$. Thus, $\forall l.\textbf{\textit{domain}}(\sigma^\alpha_l) \supseteq \chi_C \cup \chi_F \cup \chi_O$, and so $\textbf{\textit{domain}}(\sigma) \supseteq \chi_C \cup \chi_F \cup \chi_O$.

To Show: $\forall \sigma'.(\sigma' \geq \sigma \rightarrow (\mathcal{L}, j \models (\varphi_1 \mathcal{S} \varphi_2)\sigma'))$. Take any arbitrary, σ' such that $\sigma' \geq \sigma$. Then $\sigma' \geq \sigma_\beta$, so by inductive hypothesis $\mathcal{L}, k \models \varphi_2\sigma'$. Also $\forall l.\sigma' \geq \sigma^\alpha_l$, so that again by inductive hypothesis $\mathcal{L}, l \models \varphi_1\sigma'$. The semantics of $\mathcal{S}$ is $\mathcal{L}, i \models (\alpha \mathcal{S} \beta)\sigma' \Leftrightarrow \exists m \in \mathbb{N}.(m \leq i \wedge \mathcal{L}, m \models \beta\sigma' \wedge \forall l \in \mathbb{N}.((m < l \leq i) \rightarrow \mathcal{L}, i \models \alpha\sigma'))$. Instantiation of m with k and i with j lets us conclude.

(**Completeness**)

$\mathcal{L}, j \models (\varphi_1 \mathcal{S} \varphi_2)\sigma$ if and only if $\mathcal{L}, j \models (\varphi_1\sigma)\mathcal{S}(\varphi_2\sigma)$ if and only if there exists $k \leq j$ such that $\mathcal{L}, k \models \varphi_2\sigma$ and for all l, where $k < l \leq j$, $\mathcal{L}, l \models \varphi_1\sigma$. Let k be maximal.

Sub-Case $\mathbf{B} \in label(\varphi_1 \mathcal{S} \varphi_2)$

Since $\mathbf{B} \in label(\varphi_1 \mathcal{S} \varphi_2)$, there exist $\chi_C^{\mathbf{B}}$, $\chi_O^{\mathbf{B}}$ with $\boldsymbol{domain}(\chi_O^{\mathbf{B}}) \subseteq fv(\varphi_1 \mathcal{S} \varphi_2)$, $\chi_1^{\mathbf{B}} = \chi_O^{\mathbf{B}}$ and $\chi_2^{\mathbf{B}}$, such that $\emptyset \vdash_{\mathbf{B}} \varphi_1 : \chi_1^{\mathbf{B}}$, $\chi_1^{\mathbf{B}} \vdash_{\mathbf{B}} \varphi_2 : \chi_2^{\mathbf{B}}$ and thus $\chi_C^{\mathbf{B}} \vdash_{\mathbf{B}} \varphi_1 \mathcal{S} \varphi_2 : \chi_O^{\mathbf{B}}$. Let $\sigma' = \sigma\downarrow \chi_O^{\mathbf{B}}$. Note that $dom(\sigma') = \chi_O^{\mathbf{B}}$, since $\boldsymbol{domain}(\sigma) \supseteq fv(\varphi_1 \mathcal{S} \varphi_2)$. Since π is strongly consistent at i and $j \leq i$, π is well-formed at j with respect to $\varphi_1 \mathcal{S} \varphi_2$ ($\Psi(\mathcal{L}, \pi, \varphi_1 \mathcal{S} \varphi_2, j)$). So, by Definition 38(6) $\sigma' \in \pi.A(j)(\varphi_1 \mathcal{S} \varphi_2).S_R$. Let $\sigma_o = \sigma_{in} \bowtie \sigma'$. Note that $\sigma_o \neq \emptyset$, because σ' is a prefix of σ, which itself is an extension of σ_{in}. Thus $\sigma_o \in \Sigma_{out}$. By the same arguments also $\sigma_o \leq \sigma$.

Sub-Case $\mathbf{B} \notin label(\varphi_1 \mathcal{S} \varphi_2)$

Sub-Sub-Case $k = j$.

Since $fv(\varphi_2) \subseteq fv(\varphi_1 \mathcal{S} \varphi_2)$, by inductive hypothesis $\exists \sigma_o \in \boldsymbol{ips}(\mathcal{L}, j, \pi, \sigma_{in}, \varphi_2).\sigma_o \leq \sigma$. By construction, $\langle \sigma_o, j \rangle$ in S_{φ_2}, thus $\sigma_o \in S_{R_1}$ and so $\sigma_o \in \Sigma_{out}$.

Sub-Sub-Case $k < j$.

Then analogous to the previous case $\exists \sigma_2 \in \boldsymbol{ips}(\mathcal{L}, k, \pi, \sigma_{in}, \varphi_2).\sigma_2 \leq \sigma$. Also, for all $l > k$ $\sigma_2 \notin \boldsymbol{ips}(\mathcal{L}, l, \pi, \sigma_{in}, \varphi_2)$, or k would not be maximal for σ. Thus, $\langle \sigma_2, k \rangle \in S_{\varphi_2}$.

By inspection of the mode checking judgements (and lemmas) and soundness, $\boldsymbol{domain}(\sigma_2) \supseteq \chi_C \cup \chi_O \cup \chi_1$. Thus, by I.H. for all l with $k < l \leq j$, $\exists \sigma_l^{\alpha} \in \boldsymbol{ips}(\mathcal{L}, l, \pi, \sigma_2, \varphi_1).\sigma_l^{\alpha} \leq \sigma$.

Since all σ_l^{α} are $\leq \sigma$, the join $\sigma_o = \bowtie \sigma_{\bullet}^{\alpha}$ exists and $\sigma_o \leq \sigma$. Furthermore, by construction $\sigma_o \in S_{R_2}$ and so $\sigma_o \in \Sigma_{out}$.

□

The part (1) of the following Lemma specifies that when we call the ***bts*** function for a well-moded formula φ where $\mathbf{B} \in label(\varphi)$, for a specific execution history $\mathcal{L}$, a position in the history

i, and for a state π which is weakly consistent at position i with respect to φ and $\mathcal{L}$, then the state size $\hat{pi}$ returned by ***bts*** is finite with respect to all temporal sub-formulas of φ. The part (2) of the following Lemma specifies that when we call the ***ips*** function for a well-moded formula φ, for a specific execution history $\mathcal{L}$, a position in the history i, and for a state π which is strongly consistent at position i with respect to φ and $\mathcal{L}$, then the number of substitutions returned by ***ips*** is finite. The following Lemma is used by the Lemma 45 to show that both call to ***ips*** and ***bts*** terminates.

Lemma 44 (Finite substitutions)**.** *1. For all $i \in \mathbb{N}$, for all formula φ of form $\varphi_1 \mathcal{S} \varphi_2$ such that $\boldsymbol{B} \in label(\varphi_1 \mathcal{S} \varphi_2)$, for all state $\pi = \langle A, i\rangle$ such that π is weakly consistent at i with respect to $\mathcal{L}$ and φ, if $\left(\sum_{\hat{\varphi} \in \text{b-s-tsub}(\varphi)} \Upsilon(\pi, i, \hat{\varphi})\right) + \Upsilon(\pi, i-1, \varphi)$ is finite then $\sum_{\hat{\varphi} \in \text{b-tsub}(\varphi)} \Upsilon(\hat{\pi}, i, \hat{\varphi})$ is finite where $\hat{\pi} = \boldsymbol{bts}(\mathcal{L}, i, \pi, \varphi_1 \mathcal{S} \varphi_2)$.*

2. For all formula φ, for all $j \in \mathbb{N}$, for all histories $\mathcal{L}$, for all state $\pi = (A, i)$ where $i \in \mathbb{N}$, for all substitution σ_{in}, for some given χ_C and χ_F, such that: (1) $\chi_C, \chi_F \vdash \varphi : \chi_O$ where $\chi_O \subseteq fv(\varphi)$, (2) $i \geq j$, (3) $\boldsymbol{domain}(\sigma_{in}) \supseteq \chi_C \cup \chi_F$, (4) π is strongly consistent at i with respect to φ and $\mathcal{L}$, if $\boldsymbol{ips}(\mathcal{L}, j, \pi, \sigma_{in}, \varphi) = \Sigma_{out}$ then $|\Sigma_{out}|$ is finite.

Proof. Mutual induction on the structure of φ.

Case $\varphi \equiv \top$.

According to the definition of ***ips***, $\boldsymbol{ips}(\mathcal{L}, j, \pi, \sigma_{in}, \top) = \Sigma_{out} = \{\sigma_{in}\}$. Thus, $|\Sigma_{out}| = 1$ and is thus finite.

Case $\varphi \equiv \bot$.

According to the definition of ***ips***, $\boldsymbol{ips}(\mathcal{L}, j, \pi, \sigma_{in}, \bot) = \Sigma_{out} = \emptyset$. Thus, $|\Sigma_{out}| = 0$ and is thus finite.

Case $\varphi \equiv p(t_1, \ldots, t_n)$.

According to the definition of ***ips***, $\boldsymbol{ips}(\mathcal{L}, j, \pi, \sigma_{in}, p(t_1, \ldots, t_n)) = \Sigma_{out} = \boldsymbol{sat}(\mathcal{L}, j, p(t_1, \ldots, t_n), \sigma_{in})$. From premise (1) we can say the pre-condition of the ***sat*** function (Axiom 35) is satisfied. From Axiom 35, we can say that $|\Sigma_{out}|$ is finite.

Case $\varphi \equiv \varphi_1 \vee \varphi_2$.

Let $\Sigma_1 \leftarrow \textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{in}, \varphi_1)$ and $\Sigma_2 \leftarrow \textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{in}, \varphi_2)$. According to the definition of ***ips***, $\textbf{\textit{ips}}(\mathcal{L}, j, \pi, \varphi_1 \vee \varphi_2) = \Sigma_{out} = \Sigma_1 \cup \Sigma_2$. From premise 1 and inspecting the Mode checking judgements, we can say the inductive hypothesis is applicable. By inductive hypothesis, $|\Sigma_1|$ and $|\Sigma_2|$ are both finite. Thus, $|\Sigma_{out}|$ is finite.

Case $\varphi \equiv \varphi_1 \wedge \varphi_2$.

From the definition of ***ips***, $\textbf{\textit{ips}}(\mathcal{L}, i, \pi, \sigma_{\text{in}}, \varphi_1 \wedge \varphi_2) = \Sigma_{out} = \bigcup_{\sigma_c \in \textbf{\textit{ips}}(\mathcal{L}, i, \pi, \sigma_{\text{in}}, \varphi_1)} \textbf{\textit{ips}}(\mathcal{L}, i, \pi, \sigma_c, \varphi_2)$. From premise 1 and inspecting the mode checking judgements, we see that the inductive hypothesis is applicable to $\textbf{\textit{ips}}(\mathcal{L}, i, \pi, \sigma_{\text{in}}, \varphi_1)$. Let $\Sigma_1 \leftarrow \textbf{\textit{ips}}(\mathcal{L}, i, \pi, \sigma_{\text{in}}, \varphi_1)$. By inductive hypothesis, $|\Sigma_1|$ is finite. For all $\sigma_c \in \Sigma_1$, $\textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_c, \varphi_2)$ is called. We also see that from premise 1 and inspecting the mode checking judgements, the inductive hypothesis is applicable to $\textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_c, \varphi_2) = \Sigma_2$ for some $\sigma_c \in \Sigma_1$. By inductive hypothesis, each such $|\Sigma_2|$ is finite from which it follows that $|\Sigma_{out}|$ is finite.

Case $\varphi \equiv \exists \vec{x}.\varphi(\vec{x})$.

Let $\Sigma_1 \leftarrow \textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{in} \setminus \{\vec{x}\}, \exists \vec{x}.\varphi(\vec{x}))$ According to the definition of ***ips***, $\textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{in}, \exists \vec{x}.\varphi(\vec{x})) = \Sigma_{out} = \Sigma_1[\vec{x} \mapsto \sigma_{in}(\vec{x})]$. From premise 1 and inspecting mode checking judgements, we see the induction hypothesis is applicable. By inductive hypothesis, $|\Sigma_1|$ is finite from which it follows that $|\Sigma_{out}|$ is finite.

Case $\varphi \equiv \forall \vec{x}.(\varphi_1(\vec{x}) \rightarrow \varphi_2(\vec{x}))$.

Let $\Sigma_{out} \leftarrow \textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{in}, \forall \vec{x}.(\varphi_1(\vec{x}) \rightarrow \varphi_2(\vec{x})))$. Σ_{out} can be either $\emptyset$ or $\{\sigma_{in}\}$ according to the definition of ***ips***. In both cases, $|\Sigma_{out}|$ is finite.

Case $\varphi \equiv \varphi_1 \, \mathcal{S} \, \varphi_2$ and $\mathbf{B} \in \mathit{label}(\varphi)$

Sub-Case To show (1):

Given $\left(\sum_{\hat{\varphi}\in \textsf{b-s-tsub}(\varphi)} \Upsilon(\pi,i,\hat{\varphi})\right) + \Upsilon(\pi,i-1,\varphi)$ is finite to show that $\sum_{\hat{\varphi}\in \textsf{b-tsub}(\varphi)} \Upsilon(\hat{\pi},i,\hat{\varphi})$ is finite, it is sufficient to show that $\sum_{\hat{\varphi}\in \textsf{b-tsub}(\varphi)} \Upsilon(\hat{\pi},i,\hat{\varphi}) - \left(\sum_{\hat{\varphi}\in \textsf{b-s-tsub}(\varphi)} \Upsilon(\pi,i,\hat{\varphi})\right) - \Upsilon(\pi,i-1,\varphi)$ is finite. From the definition of Υ (Definition 41), we know that: $\sum_{\hat{\varphi}\in \textsf{b-tsub}(\varphi)} \Upsilon(\hat{\pi},i,\hat{\varphi}) - \left(\sum_{\hat{\varphi}\in \textsf{b-s-tsub}(\varphi)} \Upsilon(\pi,i,\hat{\varphi})\right) - \Upsilon(\pi,i-1,\varphi) = (|\textbf{\textit{domain}}(\pi.A(i)(\varphi).S_\alpha)| + |\textbf{\textit{domain}}(\pi.A(i)(\varphi).S_\beta)| + |\pi.A(i)(\varphi).S_R|)$. Thus, it is sufficient to show that $(|\textbf{\textit{domain}}(\pi.A(i)(\varphi).S_\alpha)| + |\textbf{\textit{domain}}(\pi.A(i)(\varphi).S_\beta)| + |\pi.A(i)(\varphi).S_R|)$ is finite. We will show that the following are all finite: $|\textbf{\textit{domain}}(\pi.A(i)(\varphi).S_\alpha)|$, $|\textbf{\textit{domain}}(\pi.A(i)(\varphi).S_\beta)|$, and $|\pi.A(i)(\varphi).S_R|$.

By construction, $|\textbf{\textit{domain}}(\pi.A(i)(\varphi).S_\beta)| \leq |\textbf{\textit{domain}}(\pi.A(i-1)(\varphi).S_\beta)| \times |\Sigma_\beta|$. From definition of Σ_β, we know that $\Sigma_\beta \leftarrow \textbf{\textit{ips}}(\mathcal{L}, i, \pi, \{\bullet\}, \varphi_2)$. From premise and consulting the applicable mode checking judgements (and Lemma 28, 34), we see that the inductive hypothesis of (2) is applicable. By inductive hypothesis, $|\Sigma_\beta|$ is finite and let us assume it is m_1. From the premise, we know that $|\textbf{\textit{domain}}(\pi.A(i-1)(\varphi).S_\beta)|$ is finite and let us assume it is m_2. Thus, $|\textbf{\textit{domain}}(\pi.A(i)(\varphi).S_\beta)| \leq m_1 \times m_2$, which is also finite.

Again by construction, $|\textbf{\textit{domain}}(\pi.A(i)(\varphi).S_\alpha)| \leq |\Sigma_\alpha| \times |\textbf{\textit{domain}}(\pi.A(i-1)(\varphi).S_\alpha)|$. From premise, we know that $|\textbf{\textit{domain}}(\pi.A(i-1)(\varphi).S_\alpha)|$ is finite and thus to show $|\textbf{\textit{domain}}(\pi.A(i)(\varphi).S_\alpha)|$ is finite it is sufficient to show that $|\Sigma_\alpha|$ is finite.

We will now show that $|\Sigma_\alpha|$ is finite. To construct Σ_α, in the worst case, for all $\langle \sigma, k \rangle$ pairs in $\pi.A(i)(\varphi).S_\beta$, $\textbf{\textit{ips}}(\mathcal{L}, i, \pi, \sigma, \varphi_1)$ is called. From the premise and consulting the applicable mode checking judgements, we see that the inductive hypothesis of (2) is applicable. By inductive hypothesis of (2), each call to $\textbf{\textit{ips}}(\mathcal{L}, i, \pi, \sigma, \varphi_1)$ returns a finite set of substitutions. Let us assume the maximum cardinality of, all the sets of substitutions returned by the calls to ***ips***, is m_3. Thus by construction, $|\Sigma_\alpha| \leq m_1 \times m_2 \times m_3$, which is finite. It follows that $|\textbf{\textit{domain}}(\pi.A(i)(\varphi).S_\alpha)|$ is finite.

From construction of $\pi.A(i)(\varphi).S_R$, we know that $\pi.A(i)(\varphi).S_R = S_{R_1} \cup S_{R_2}$. Thus, $|\pi.A(i)(\varphi).S_R| \leq |S_{R_1}| + |S_{R_2}|$. We will show that $|S_{R_1}|$ and $|S_{R_2}|$ are both finite, concluding our

proof.

From construction, $|S_{R_1}| \leq |\textbf{\textit{domain}}(\pi.A(i)(\varphi).S_\beta)|$. As $|\textbf{\textit{domain}}(\pi.A(i)(\varphi).S_\beta)|$ is finite (shown above) and $|\textbf{\textit{domain}}(\pi.A(i)(\varphi).S_\beta)| \leq m_1 \times m_2$, we can write $|S_{R_1}| \leq m_1 \times m_2$, which is finite.

From construction, $|S_{R_2}| \leq |\textbf{\textit{domain}}(\pi.A(i)(\varphi).S_\beta)| \times |\textbf{\textit{domain}}(\pi.A(i)(\varphi).S_\alpha)|$. As shown above, $|\textbf{\textit{domain}}(\pi.A(i)(\varphi).S_\beta)|$ and $|\textbf{\textit{domain}}(\pi.A(i)(\varphi).S_\beta)|$ are both finite. This concludes our proof that $|S_{R_2}|$ is finite and in turn $|\pi.A(i)(\varphi).S_R|$ is finite.

Sub-Case To show (2):

Let $\Sigma_1 \leftarrow \pi.A(i)(\varphi_1 \mathcal{S} \varphi_2).S_R$. From the definition of ***ips***, $\textbf{\textit{ips}}(\mathcal{L}, i, \pi, \sigma_{\text{in}}, \varphi_1 \mathcal{S} \varphi_2) = \Sigma_{out} = \sigma_{\text{in}} \bowtie \pi.A(i)(\varphi_1 \mathcal{S} \varphi_2).S_R$. By inductive hypothesis of (1), $|\Sigma_1|$ is finite. It follows that $|\Sigma_{out}|$ is finite.

Case $\varphi \equiv \varphi_1 \mathcal{S} \varphi_2$ and $\mathbf{B} \notin label(\varphi)$

According to the definition of ***ips***, $\textbf{\textit{ips}}(\mathcal{L}, j, \pi, \varphi_1 \mathcal{S} \varphi_2) = \Sigma_{out} = S_{R_1} \cup S_{R_2}$. It is sufficient to show that $|S_{R_1}|$ and $|S_{R_2}|$ are both finite.

We will first show that by inductive hypothesis, $|S_\beta|$ is finite. Let us consider, for all l where $0 \leq l \leq j$, $m = max|\Sigma_1|.\Sigma_1 \leftarrow \textbf{\textit{ips}}(\mathcal{L}, l, \pi, \sigma_{in}, \varphi_2)$. The maximum size of $|S_\beta|$ can be $(j+1) \times m$ by construction. By the inductive hypothesis, m is finite. As $j \in \mathbb{N}$ is finite, it follows that $|S_\beta|$ is finite.

By construction, as $S_{R_1} \subseteq S_\beta$, $|S_{R_1}| \leq |S_\beta|$ and it follows that $|S_{R_1}|$ is finite.

By construction of S_{R_2}, for each $\langle \sigma_\beta, k \rangle \in S_\beta$ where $k \neq j$, $\textbf{\textit{ips}}(\mathcal{L}, q, \pi, \sigma_\beta, \varphi_1)$ is called for all q such that $k < q \leq j$. By inductive hypothesis, $|\textbf{\textit{ips}}(\mathcal{L}, q, \pi, \sigma_\beta, \varphi_1)|$ is finite and let us say for all q such that $k < q \leq j$, $m_1 = max|\Sigma_2|.\Sigma_2 \leftarrow \textbf{\textit{ips}}(\mathcal{L}, q, \pi, \sigma_\beta, \varphi_1)$. By construction ($\bowtie$ of all substitutions for all positions q), the maximum size of $|S_{R_2}|$ will be less than $(j+1) \times m \times m_1^{(j+1)}$, which is finite.

Thus, it follows that $|\Sigma_{out}|$ is finite.

□

The following Lemma specifies that calls to ***ips*** and ***bts*** terminates.

Lemma 45 (Termination)**.** *For all formula* φ*, the following holds:*

1. *For all* $\mathcal{L}$*, for all* $j \in \mathbb{N}$*, for all substitution* σ_{in}*, for all state* $\pi = \langle A, i \rangle$ *where* $i \in \mathbb{N}$*, for some given* χ_C, χ_F *such that: (1)* $\chi_C, \chi_F \vdash \varphi : \chi_O$ *where* $\chi_O \subseteq fv(\varphi)$*, (2)* $\textbf{\textit{domain}}(\sigma_{in}) \supseteq \chi_C \cup \chi_F$*, (3)* π *is strongly consistent at i with respect to* φ *and* $\mathcal{L}$*, then* $\textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{in}, \varphi)$ *terminates.*

2. *For all* $\mathcal{L}$*, for all* $i \in \mathbb{N}$*, for all state* $\pi = \langle A, i \rangle$ *such that* $\varphi \equiv \varphi_1 \, \mathcal{S} \, \varphi_2$*,* $\emptyset \vdash_{\textbf{\textit{B}}} \varphi : \chi_O$*, and* π *is weakly consistent at i with respect to* φ *and* $\mathcal{L}$*, then* $\textbf{\textit{bts}}(\mathcal{L}, i, \pi, \varphi)$ *terminates.*

Proof. Mutual induction on the structure of φ.

Case $\varphi \equiv \top$.

$\textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{in}, \top)$ terminates trivially.

Case $\varphi \equiv \bot$.

$\textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{in}, \bot)$ terminates trivially.

Case $\varphi \equiv p(t_1, \ldots, t_n)$.

According to the definition of ***ips***, $\textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{in}, p(t_1, \ldots, t_n)) = \Sigma_{out} = \textbf{\textit{sat}}(\mathcal{L}, j, p(t_1, \ldots, t_n), \sigma_{in})$. From premise (1) we can say the pre-condition of the ***sat*** function (Axiom 35) is satisfied. From Axiom 35, we can say that ***sat*** terminates and from it follows that $\textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{in}, p(t_1, \ldots, t_n))$ terminates.

Case $\varphi \equiv \varphi_1 \vee \varphi_2$.

According to the definition of, $\textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{in}, \varphi_1 \vee \varphi_2) = \textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{in}, \varphi_1) \cup \textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{in}, \varphi_2)$. From premise 1 and inspecting the mode checking judgements, we can say the inductive hypothesis is applicable. By inductive hypothesis, $\textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{in}, \varphi_1)$ and $\textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{in}, \varphi_2)$ both terminate. It follows that $\textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{in}, \varphi_1 \vee \varphi_2)$ terminates.

Case $\varphi \equiv \varphi_1 \wedge \varphi_2$.

From the definition of ***ips***, $\textbf{\textit{ips}}(\mathcal{L}, i, \pi, \sigma_{\text{in}}, \varphi_1 \wedge \varphi_2) = \Sigma_{out} = \bigcup_{\sigma_c \in \textbf{\textit{ips}}(\mathcal{L}, i, \pi, \sigma_{\text{in}}, \varphi_1)} \textbf{\textit{ips}}(\mathcal{L}, i, \pi, \sigma_c, \varphi_2)$. From premise 1 and inspecting the mode checking judgements, we see that the inductive hypothesis is applicable to $\textbf{\textit{ips}}(\mathcal{L}, i, \pi, \sigma_{\text{in}}, \varphi_1)$. By inductive hypothesis, $\textbf{\textit{ips}}(\mathcal{L}, i, \pi, \sigma_{\text{in}}, \varphi_1)$ terminates. Let $\Sigma_1 \leftarrow \textbf{\textit{ips}}(\mathcal{L}, i, \pi, \sigma_{\text{in}}, \varphi_1)$. From Lemma 44, we have $|\Sigma_1|$ is finite. For all $\sigma_c \in \Sigma_1$, $\textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_c, \varphi_2)$ is called. We also see that from premise 1 and inspecting the mode checking judgements, the inductive hypothesis is applicable to $\textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_c, \varphi_2)$. By inductive hypothesis, each call to $\textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_c, \varphi_2)$ terminates and there are finite number of such calls.

It follows that $\textbf{\textit{ips}}(\mathcal{L}, i, \pi, \sigma_{\text{in}}, \varphi_1 \wedge \varphi_2)$ terminates.

Case $\varphi \equiv \exists \vec{x}.\varphi(\vec{x})$.

According to the definition of ***ips***, $\textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{in}, \exists \vec{x}.\varphi(\vec{x})) = \textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{in} \setminus \{\vec{x}\}, \exists \vec{x}.\varphi(\vec{x}))[\vec{x} \mapsto \sigma_{in}(\vec{x})]$. From premise 1 and inspecting mode checking judgements, we see the induction hypothesis is applicable. By inductive hypothesis, $\textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{in} \setminus \{\vec{x}\}, \exists \vec{x}.\varphi(\vec{x}))$ terminates from which it follows that $\textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{in}, \exists \vec{x}.\varphi(\vec{x}))$ terminates.

Case $\varphi \equiv \forall \vec{x}.(\varphi_1(\vec{x}) \rightarrow \varphi_2(\vec{x}))$.

According to the definition of ***ips***, to calculate $\textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{in}, \forall \vec{x}.(\varphi_1(\vec{x}) \rightarrow \varphi_2(\vec{x})))$ we first make a call to $\textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{in}, \varphi_1(\vec{x}))$. Let $\Sigma_1 \leftarrow \textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{in}, \varphi_1(\vec{x}))$. From premise 1 and inspecting the mode checking judgements, we see that the inductive hypothesis is applicable. By inductive hypothesis, $\textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{in}, \varphi_1(\vec{x}))$ terminates. From Lemma 44, we know that $|\Sigma_1|$ is finite. For all $\sigma_c \in \Sigma_1$, a call to $\textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_c, \varphi_2(\vec{x}))$ is made. From premise 1 and inspecting the mode checking judgements, we can again see that the inductive hypothesis is applicable to $\textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_c, \varphi_2(\vec{x}))$. By inductive hypothesis, each such call to $\textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_c, \varphi_2(\vec{x}))$ terminates and there are finite number of such calls.

It follows that $\textbf{\textit{ips}}(\mathcal{L}, j, \pi, \sigma_{in}, \forall \vec{x}.(\varphi_1(\vec{x}) \rightarrow \varphi_2(\vec{x})))$ terminates.

Case $\varphi \equiv \varphi \mathcal{S} \varphi_2$ and $\mathbf{B} \in label(\varphi)$

Sub-Case To show (1):

Let $\Sigma_1 \leftarrow \pi.A(i)(\varphi_1 \mathcal{S} \varphi_2).S_R$. From the definition of ***ips***, ***ips***$(\mathcal{L}, i, \pi, \sigma_{\text{in}}, \varphi_1 \mathcal{S} \varphi_2) = \Sigma_{out} = \sigma_{\text{in}} \bowtie \pi.A(i)(\varphi_1 \mathcal{S} \varphi_2).S_R$. From premise (1) and inspecting the mode checking judgements, we see that the inductive hypothesis of Lemma 44 (1) is applicable. By inductive hypothesis, $|\Sigma_1|$ is finite. Thus, the join operation terminates and it follows that ***ips***$(\mathcal{L}, i, \pi, \sigma_{\text{in}}, \varphi_1 \mathcal{S} \varphi_2)$ terminates.

Sub-Case To show (2):

From definition of ***bts***, to calculate Σ_β, ***ips***$(\mathcal{L}, i, \pi, \{\bullet\}, \varphi_2)$ is called once. From premise and inspecting applicable mode checking judgements, we see that inductive hypothesis of (1) is applicable. From inductive hypothesis (1), we can say that ***ips***$(\mathcal{L}, i, \pi, \{\bullet\}, \varphi_2)$ terminates.

According to the proof of Lemma 44 (1), we know that $|\Sigma_\beta|$ is finite. In the worst case, each $\sigma \in \Sigma_\beta$ is joined with all substitutions of $\pi.A(i-1)(\varphi).S_\beta$. This join operation terminates as both $|\Sigma_\beta|$ and $|$***domain***$(\pi.A(i-1)(\varphi).S_\beta)|$ is finite. We assume it is possible to calculate the join operation of two finite substitutions in some finite amount of time.

Then for each $\langle \sigma, k \rangle$ pair in $\pi.A(i)(\varphi).S_\beta$, ***ips*** is called once. From premise and inspecting applicable mode checking judgements, we see that inductive hypothesis of (1) is applicable. By inductive hypothesis (1), each such call to ***ips*** terminates. There are finite such calls to ***ips*** as there are finite $\langle \sigma, k \rangle$ pairs in $\pi.A(i)(\varphi).S_\beta$.

Finally, in the worst case, while calculating $\pi.A(i)(\varphi).S_R$, each $\langle \sigma, k \rangle$ pairs in $\pi.A(i)(\varphi).S_\beta$ is joined with each $\langle \sigma_1, j \rangle$ pairs in $\pi.A(i)(\varphi).S_\alpha$. As $|$***domain***$(\pi.A(i)(\varphi).S_\beta)|$ and $|$***domain***$(\pi.A(i)(\varphi).S_\alpha)|$ are finite, the join operations terminate from which it follows that ***bts*** terminates concluding our proof.

Case $\varphi \equiv \varphi_1 \mathcal{S} \varphi_2$ and $\mathbf{B} \notin label(\varphi)$

According to the definition of ***ips***, ***ips***$(\mathcal{L}, j, \pi, \varphi_1 \mathcal{S} \varphi_2) = \Sigma_{out} = S_{R_1} \cup S_{R_2}$. It is sufficient to show that the construction of sets S_{R_1} and S_{R_2} terminates. We will then show that $|S_{R_1}|$ and

$|S_{R_2}|$ are both finite and thus the set union operation terminates.

We will first show that the construction of S_β terminates. By construction of S_β, a call to $\textbf{\textit{ips}}(\mathcal{L}, l, \pi, \sigma_{in}, \varphi_2)$ is made for all l where $0 \leq l \leq j$. From premise 1 and inspecting the mode checking judgements, we see that the inductive hypothesis is applicable to $\textbf{\textit{ips}}(\mathcal{L}, l, \pi, \sigma_{in}, \varphi_2)$. By inductive hypothesis, each call to $\textbf{\textit{ips}}(\mathcal{L}, l, \pi, \sigma_{in}, \varphi_2)$ terminates and there are finite $(j+1)$ number of such calls. It follows that the construction of S_β terminates.

By Lemma 44, $|S_\beta|$ is finite. By construction, each σ is added in S_{R_1} where $\langle \sigma, j \rangle \in S_\beta$. As there are finite number of such σ, the construction of S_{R_1} terminates. By construction, $|S_{R_1}| \leq |S_\beta|$ and it follows that $|S_{R_1}|$ is finite.

By construction of S_{R_2}, for each $\langle \sigma_\beta, k \rangle \in S_\beta$ where $k \neq j$, $\textbf{\textit{ips}}(\mathcal{L}, q, \pi, \sigma_\beta, \varphi_1)$ is called for all q such that $k < q \leq j$. By inductive hypothesis, each call to $\textbf{\textit{ips}}(\mathcal{L}, q, \pi, \sigma_\beta, \varphi_1)$ terminates and there are finite number of such calls. Thus, the construction of the set S_{R_2} terminates.

As both $|S_{R_1}|$ and $|S_{R_2}|$ are finite, the set union operation terminates.

Thus, it follows that $\textbf{\textit{ips}}(\mathcal{L}, j, \pi, \varphi_1 \mathcal{S} \varphi_2)$ terminates.

□

We now introduce the notion of history equivalence. Given two execution histories $\mathcal{L}_1$ and $\mathcal{L}_2$, $\mathcal{L}_1$ is equivalent to $\mathcal{L}_2$ with respect to a specific position in the history $i \in \mathbb{N}$ such that $i \leq |\mathcal{L}_1|, |\mathcal{L}_2|$, then the i^{th} entry of both the history are same.

Definition 46 (History Equivalence)**.** *For all histories $\mathcal{L}_1$ and $\mathcal{L}_2$, $\mathcal{L}_1$ is equivalent to $\mathcal{L}_2$ with respect to a specific $i \in \mathbb{N}$ such that $i \leq |\mathcal{L}_1|, |\mathcal{L}_2|$, denoted by $\mathcal{L}_1 \approx_i \mathcal{L}_2$, if and only if $\mathcal{L}_1 \downarrow i = \mathcal{L}_2 \downarrow i$ where $\downarrow$ represents the projection operator.*

The following Lemma specifies that when the ***ips*** or ***bts*** function is called for a formula φ, such that $\mathbf{B} \in \mathit{label}(\varphi)$, then both the functions do not need to access any other positions in the execution history except the current one. This signifies that it is possible to check weak compliance of a policy φ without needing to store all the whole execution history instead just by incrementally

updating the structures in the state appropriately. Moreover, it also shows that for updating the structures for a given formula, it just requires to access the structures of the current entry and structures of the previous entry. Thus, it possible to check compliance for policy φ where $\mathbf{B} \in label(\varphi)$ just by storing the current state of the history and previous entry of the structures.

Lemma 47 (Not Looking Back in the Execution History)**.** *For all formula* φ *such that* $\chi_C \vdash_{\mathcal{B}} \varphi : \chi_O$ *holds for some given* χ_C *where* $\chi_O \subseteq fv(\varphi)$ *then*

1. *For all histories* $\mathcal{L}_1$ *and* $\mathcal{L}_2$*, for all* $i \in \mathbb{N}$*, for all substitutions* σ_{in}*, for all state* $\pi = \langle A, i\rangle$ *such that* $\textbf{\textit{domain}}(\sigma_{in}) \supseteq \chi_C$ *and* $\mathcal{L}_1 \approx_i \mathcal{L}_2$*,* $\textbf{\textit{ips}}(\mathcal{L}_1, i, \pi, \sigma_{in}, \varphi) = \textbf{\textit{ips}}(\mathcal{L}_2, i, \pi, \sigma_{in}, \varphi)$.

2. *For all histories* $\mathcal{L}_1$ *and* $\mathcal{L}_2$*, for all* $i \in \mathbb{N}$*, for all state* $\pi_1 = \langle A_1, i\rangle$ *and* $\pi_2 = \langle A_2, i\rangle$ *such that* $\varphi \equiv \alpha \mathcal{S} \beta$*,* $\pi_1.A_1(i-1) = \pi_2.A_2(i-1)$*,* $\pi_1.A_1(i) = \pi_2.A_2(i)$*, and* $\mathcal{L}_1 \approx_i \mathcal{L}_2$*,* $\textbf{\textit{bts}}(\mathcal{L}_1, i, \pi, \varphi) = \textbf{\textit{bts}}(\mathcal{L}_2, i, \pi, \varphi)$.

Proof. We do an induction on the structure of φ. To show the above, it is enough to show that when the functions ***ips*** and ***bts*** are called with a specific i for a formula φ where $\mathbf{B} \in label(\varphi)$, it does not recursively call ***ips*** or ***bts*** with a j where $j < i$.

Case $\varphi \equiv \top | \bot | p(t_1, \ldots, t_n)$.

(1) is true as ***ips*** is not called at all. (2) is vacuously true as φ is not of form $\varphi_1 \mathcal{S} \varphi_2$.

Case $\varphi \equiv \varphi_1 \vee \varphi | \varphi_1 \wedge \varphi_2$.

We can see from the definition of ***ips*** that ***ips*** is called for φ_1 and φ_2 with the same i. From I.H., we get that (1) is true for both the calls. We can thus conclude (1) is true. (2) is vacuously true as φ is not of form $\varphi_1 \mathcal{S} \varphi_2$.

Case $\varphi \equiv \exists \vec{x}.\varphi(\vec{x})$.

We can see from the definition of ***ips*** that ***ips*** is called for $\varphi(\vec{x})$ with the same i. From I.H., we get that (1) is true for the call to ***ips*** for $\varphi(\vec{x})$. We can thus conclude (1) is true. (2) is vacuously true as φ is not of form $\varphi_1 \mathcal{S} \varphi_2$.

Case $\varphi \equiv \forall \vec{x}.(\varphi_1(\vec{x}) \rightarrow \varphi_2(\vec{x}))|\varphi_1 \mathbb{U}_{[c,d]} \varphi_2$.

From the derivation of $\vdash_{\mathbf{B}}$, if a formula φ is of form either $\forall \vec{x}.(\varphi_1(\vec{x}) \rightarrow \varphi_2(\vec{x}))$ or $\varphi_1 \mathbb{U}_{[c,d]} \varphi_2$, then $\mathbf{B} \notin label(\varphi)$. Thus, (1) is vacuously true. (2) is vacuously true as φ is not of form $\varphi_1 \mathcal{S} \varphi_2$.

Case $\varphi \equiv \varphi_1 \mathcal{S} \varphi_2$ and $\mathbf{B} \in label(\varphi)$

(1) is true as from the definition of ***ips*** we can see that for a φ of the above form, ***ips*** not called at all. From the definition of ***bts***, we can see that ***ips*** is called for φ_2 and φ_1 with the same i. From I.H., we can say that these call to ***ips*** satisfy (1). Moreover, during the initialization phase for history position i, the structures are initialized with the values of the structures from the immediate previous state (history position $i-1$). Thus, no other entries of the structures which are required to be accessed by ***bts***. Thus, we can safely conclude that (2) is satisfied.

□

The following Lemma specifies that the ***cc*** function is correct.

Lemma 48 (Correctness of ***cc*** function)**.** *The function* $\mathbf{cc} : Log \times \mathbb{N} \times State \times Formula \rightarrow Boolean$ *is correct if the following holds:*

For all formula φ*, for all* $j \in \mathbb{N}$*, for all histories* $\mathcal{L}$*, for all state* $\pi = (A,i)$*, for some* χ_C *and* χ_F*, such that: (1)* π *is strongly consistent at* $i \in \mathbb{N}$ *with respect to* $\mathcal{L}$ *and* φ*, (2)* $i \geq j$*, (3)* $\chi_C, \chi_F \vdash \varphi : \chi_O$ *where* $\chi_O \subseteq fv(\varphi)$*, if* $\mathbf{cc}(\mathcal{L}, j, \pi, \varphi) = truthValue$*, then*

$(truthValue = true) \leftrightarrow \exists \sigma.(\mathcal{L}, j \models \varphi\sigma)$.

Proof. The proof follows from the soundness argument of ***ips*** correctness, Lemma 43. □

The following Lemma specifies that each call to the ***cc*** function terminates.

Lemma 49 (Termination of ***cc*** function)**.** *For all formula* φ*, for all* $j \in \mathbb{N}$*, for all histories* $\mathcal{L}$*, for all state* $\pi = (A,i)$*, for some* χ_C *and* χ_F*, such that: (1)* π *is strongly consistent at* $i \in \mathbb{N}$ *with respect to* $\mathcal{L}$ *and* φ*, (2)* $i \geq j$*, (3)* $\chi_C, \chi_F \vdash \varphi : \chi_O$ *where* $\chi_O \subseteq fv(\varphi)$*, the function* $\mathbf{cc}(\mathcal{L}, j, \pi, \varphi)$ *terminates.*

Proof. The proof follows from the termination argument of the ***ips*** function (Lemma 45 (1)), as according to the definition of ***cc*** function, only the ***ips*** function is called from the ***cc*** function. □

Theorem 50 (Correctness of the Algorithm)**.** *Each iteration of the* **checkPolicyCompliance** *function terminates and it is sound and complete.*

Proof. The termination of the ***checkPolicyCompliance*** function follows from the termination argument of ***bts*** and the ***cc*** function. The soundness and completeness of the ***checkPolicyCompliance*** functions follows from the soundness and completeness argument of ***bts*** and ***cc*** function. □

Chapter 5: PRIVACY POLICY ANALYSIS

An action is compliant with a privacy policy if it is both weakly compliant and strongly compliant. Although checking weak compliance is feasible [13, 55], checking strong compliance with respect to policies written in ***FOPSL*** is undecidable. Thus, to facilitate efficient compliance checking we now formally specify the property *weak compliance entails strong compliance* (denoted by Δ) [11]. A ***FOPSL*** policy has the Δ-property if every weakly compliant action is also strongly compliant. To check whether an action is compliant with such a policy, it suffices to just check whether the action is weakly compliant with that policy. We believe well-written policies should have this property. We also show that when a policy $\wp$ has the Δ-property, the present conditions of $\wp$, denoted by $weak(\wp)$, express the safety property imposed by $\wp$. Such a policy guarantees that no obligation will need to be left unfulfilled because the policy is badly written and does not allow the obligation to be fulfilled in that situation.

For a given privacy policy $\wp$, we syntactically construct a first order CTL* with linear past (denoted by FO-CTL*$_{lp}$) [77] formula $\delta(\wp)$ from $\wp$. We prove that *the most permissible model* (denoted by $\mathbb{M}_{\wp}$) of a policy $\wp$ satisfies $\delta(\wp)$ if and only if $\wp$ has the Δ-property (section 5.1). The most permissive model $\mathbb{M}_{\wp}$ of a policy $\wp$ is the model in which at each step any action referred to by $\wp$ can be non-deterministically chosen to be performed. Considering $\mathbb{M}_{\wp}$ of a policy $\wp$ is reasonable: if $\wp$ can incur obligations that cannot be met even in the most permissive model, then it is not possible that those obligations can be met in other more restricted models. Also, more restrictive models cannot reach any policy "states" that are not reachable with the most permissible model, so other models would not be able to incur any obligations that would not be able to be incurred by the most permissive model. Other models exhibit a subset of behaviors which are of interest to the analysis of the Δ-property. Model checking an arbitrary FO-CTL*$_{lp}$ specification with respect to an arbitrary given model is undecidable. Thus, in section 5.3, based on some

The content of this chapter is based on the joint work with Andreas Gampe, Jianwei Niu, Jeffery von Ronne, Jared Bennatt, Anupam Datta, Limin Jia, and William H. Winsborough [27].

reasonable assumptions, we develop a sound, semi-automated technique that can feasibly decide in many practical cases whether a policy has the Δ-property.

5.1 The WC Entails SC Property (Δ-property)

As the Δ-property is a statically analyzable property of the policy, it enables offloading all complexity of checking this property to before the policy is actually deployed. We can then use more expensive decision methods that would not be feasible if we were to check it at runtime.

A policy satisfies the Δ-property if for any weakly compliant finite trace (history), there exists an infinite extension of the finite trace such that the concatenation of the finite trace and the infinite extension satisfies the policy. We call a finite trace a *weakly compliant finite trace* if each action of that finite trace is weakly compliant with the policy. Formally, for a given environment η, a finite trace (σ_f) is weakly compliant with respect to the policy $\wp$ if the following holds: $\sigma_f, |\sigma_f| - 1, \eta \models \boxminus \mathit{weak}(\wp)$. We now formally specify what it means for a privacy policy $\wp$ to have the Δ-property.

Definition 51 (Δ-property)**.** *A policy $\wp$ has the Δ-property if and only if for any environment η and for any history (finite trace) σ_f that satisfies $\sigma_f, |\sigma_f| - 1, \eta \models \boxminus \mathit{weak}(\wp)$, there exists an infinite trace (extension) σ_i such that $\sigma_f \cdot \sigma_i \models \wp$.*

We now construct from a policy $\wp$ a formula $\delta(\wp)$ in the logic FO-CTL*$_{lp}$ [77] that is satisfied by the *most permissible model* ($\mathbb{M}_\wp$) of the policy (denoted by $\mathbb{M}_\wp \models \delta(\wp)$) if and only if $\wp$ has the Δ-property. To the best of our knowledge, we are the first to give a specification of the Δ-property within a formal logic. This formalization is an important first step toward being able to identify policies which have the Δ-property. We first discuss why using the logic FO-CTL*$_{lp}$ [77] is natural. First, the logic is expressive enough to capture the Δ-property. Second, we chose the first-order variation of the CTL*$_{lp}$ logic, as our policy is written in FOTL and we want our policy formula to be a sub-formula of the logic. Finally, our policy allows past temporal operators, unlike typical branching-time logic CTL* [45]. We could rewrite the policy to contain only future temporal operators, but this might cause exponential blowup of the formula length [77, 83]. As shown by

Kupferman *et al.* [77], adding linear past to CTL* [45] does not increase the expressive power but is more succinct. They showed that CTL$^*_{lp}$ and CTL* have the same expressive power.

A policy $\wp$ has the Δ-property if and only if $\mathbb{M}_\wp$ satisfies the FO-CTL$^*_{lp}$ formula $\delta(\wp)$.The formula $\delta(\wp)$ is defined in Figure 5.1. The formula states that, given a finite weakly compliant history, it is possible to extend the finite history to an infinite one in which all the pending obligations are discharged while maintaining weak compliance.

$$\mathcal{A}\Box\ \big(\ (\boxminus \mathit{weak}(\wp)) \longrightarrow$$
$$E\Big(\bigwedge_{\langle\lambda,\gamma\rangle\in\alpha(\wp)} \forall p_1,p_2,q{:}P.\forall m{:}M.\forall t{:}T.\forall u{:}U.\big(\ (\neg\gamma\,\mathcal{S}\,\lambda) \rightarrow \Diamond\gamma\big) \wedge \Box \mathit{weak}(\wp)\big)\ \big)$$

Figure 5.1: FO-CTL$^*_{lp}$ formulation of the Δ-property

α function: The function α in the formula $\delta(\wp)$ takes as input a privacy policy $\wp$ and returns all possible $\langle\lambda,\gamma\rangle$ pairs in $\wp$. In a $\langle\lambda,\gamma\rangle$ pair, λ characterizes a condition, which, when true, incurs the obligation γ according to the policy $\wp$. The function α works on the norm level of the policy $\wp$ and syntactically extracts all the $\langle\lambda,\gamma\rangle$ pairs. The definition of α is as follows.

Our positive norms have the form: $(\mathbb{C}\wedge\psi\wedge\beta)\vee\psi_{exception}$. For such a positive norm where β is not trivially true, the α function would return the $\langle\lambda,\gamma\rangle$ pair $\langle(\mathbb{C}\wedge\psi\wedge\neg(\psi_{exception})\wedge\bigwedge_j\hat{\phi}_j^-,\beta\rangle$ in which $\bigwedge_j\hat{\phi}_j^-$ is the conjunction of all the modified negative norms. The modified negative norms are the same as the original negative norms except that they do not contain have any future temporal operators (modified ones).

The form of our negative norms is as follows: $\mathbb{C}\wedge\psi\rightarrow\chi\vee\psi_{exception}$. In the negative norms, χ can have one of the following forms: (1) β, (2) ψ_1, (3) $\psi_1\wedge\beta$, and (4) $\psi_1\rightarrow\beta$. When χ has form (2) then there are no obligations in that norm. In all other cases let us consider that β is not trivially true. For a negative norm ϕ_j^- whose χ is of form (1), α will return the $\langle\lambda,\gamma\rangle$ pair $\langle\mathbb{C}\wedge\psi\wedge\neg(\psi_{exception})\wedge\bigwedge_{k\neq j}\hat{\phi}_k^-\wedge\bigvee_i\hat{\phi}_i^+,\beta\rangle$ in which $\hat{\phi}_k^-$ and $\hat{\phi}_i^+$ respectively, represent modified negative and positive norms. When χ has the form (3) or (4), the function α will return the $\langle\lambda,\gamma\rangle$

pair $\langle \mathbb{C} \wedge \psi \wedge \neg(\psi_{exception}) \wedge \psi_1 \wedge \bigwedge_{k \neq j} \hat{\phi}_k^- \wedge \bigvee_i \hat{\phi}_i^+, \beta \rangle$.

The following theorem states that a policy $\wp$ has the Δ-property if and only if $\mathbb{M}_\wp \models \delta(\wp)$.

Theorem 52. *Given a policy $\wp$, $\mathbb{M}_\wp \models \delta(\wp)$ if and only if $\wp$ has the Δ-property.*

Proof. According to the definition of the Δ-property (Definition 51), a policy $\wp$ has the Δ-property if and only if the following holds:

$$\mathbf{L} : \forall \sigma_f . \forall \eta_1 . (\sigma_f, |\sigma_f| - 1, \eta_1 \models \boxminus weak(\wp) \rightarrow \exists \sigma_i . \sigma_f \cdot \sigma_i \models \wp)$$

We have to show that $\mathbb{M}_\wp \models \delta(\wp)$ if and only if the logical sentence $\mathbf{L}$ holds, or more precisely, $\mathbb{M}_\wp \models \delta(\wp) \leftrightarrow \mathbb{L}$. We will show the *only if* direction ($\rightarrow$); the *if* part ($\leftarrow$) is similar. So let us consider that $\mathbb{M}_\wp \models \delta(\wp)$ holds and we have to show that $\mathbf{L}$ holds. In the logical sentence $\mathbf{L}$, take an arbitrary σ_f, η_1 such that $\sigma_f, |\sigma_f| - 1, \eta_1 \models weak(\wp)$.

From the semantics of FO-CTL*$_{lp}$ and $\delta(\wp)$, $\mathbb{M}_\wp \models \delta(\wp)$ if and only if for all infinite paths ρ in $\mathbb{M}_\wp$, for all positions $i \in \mathbb{N}$ in ρ, and for all environments η, $\boxminus weak(\wp) \rightarrow E(\forall p_1, p_2, q : P . \forall m : M . \forall t : T . ((\neg \gamma \mathcal{S} \lambda) \rightarrow \Diamond \gamma) \wedge \Box weak(\wp))$ holds. Now consider an environment η, an infinite path ρ, a position $i \in \mathbb{N}$ in it such that $\eta = \eta_1$ and $\rho[..i] = \sigma_f$, where we use $\rho[..i]$ to denote a prefix of an infinite path with $i+1$ elements. As we consider the most permissive model of $\wp$, which contains all possible infinite traces in which all the behaviors of the policy are captured, such an infinite ρ and a position i such that $\rho[..i] = \sigma_f$ exists. As $\sigma_f, |\sigma_f| - 1, \eta_1 \models \boxminus weak(\wp)$, $\rho[..i] = \sigma_f$, $\eta = \eta_1$, and the semantics of FOTL formula are preserved in a FO-CTL*$_{lp}$ formula, it follows that $\rho[..i], i, \eta \models \boxminus weak(\wp)$. We know that $\mathbb{M}_\wp \models \delta(\wp)$ and $\rho[..i], i, \eta \models \boxminus weak(\wp)$ hold. It follows that $E(\forall p_1, p_2, q : P . \forall m : M . \forall t : T . ((\neg \gamma \mathcal{S} \lambda) \rightarrow \Diamond \gamma) \wedge \Box weak(\wp))$ holds.

According to the semantics of FO-CTL*$_{lp}$, $\rho[..i], i, \eta \models \boxminus weak(\wp) \rightarrow E(\forall p_1, p_2, q : P . \forall m : M . \forall t : T . ((\neg \gamma \mathcal{S} \lambda) \rightarrow \Diamond \gamma) \wedge \Box weak(\wp))$ holds, if there exists an infinite path $\widehat{\rho}$ and $\hat{\eta}$ such that $\rho[..i] = \widehat{\rho}[..i]$, $\hat{\eta} = \eta$ and $\widehat{\rho}, i, \hat{\eta} \models \forall p_1, p_2, q : P . \forall m : M . \forall t : T . ((\neg \gamma \mathcal{S} \lambda) \rightarrow \Diamond \gamma) \wedge \Box weak(\wp)$. From the semantics of FO-CTL*$_{lp}$, $\widehat{\rho}, i, \hat{\eta} \models \forall p_1, p_2, q : P . \forall m : M . \forall t : T . ((\neg \gamma \mathcal{S} \lambda) \rightarrow \Diamond \gamma) \wedge \Box weak(\wp)$ holds if and only if $\widehat{\rho}, i, \hat{\eta} \models \forall p_1, p_2, q : P . \forall m : M . \forall t : T . ((\neg \gamma \mathcal{S} \lambda) \rightarrow \Diamond \gamma)$ and $\widehat{\rho}, i, \hat{\eta} \models \Box weak(\wp)$

hold.

According to the semantics of FO-CTL*$_{lp}$, $\widehat{\rho}, i, \hat{\eta} \models \forall p_1, p_2, q : P. \forall m : M. \forall t : T. ((\neg \gamma \mathcal{S} \lambda) \rightarrow \Diamond \gamma)$ holds if and only if $\widehat{\rho}, i, \hat{\eta}' \models ((\neg \gamma \mathcal{S} \lambda) \rightarrow \Diamond \gamma)$[1], which states that all the pending obligations in $\rho[..i]$ are fulfilled in some finite position from i in $\widehat{\rho}$. On the other hand, according to the semantics of FO-CTL*$_{lp}$, $\widehat{\rho}, i, \hat{\eta} \models \Box \mathit{weak}(\wp)$ if and only if the infinite extension is weakly compliant at each position in the path $\widehat{\rho}$.

Due to the syntactic requirement that past and future can be separated syntactically and from the semantics of ***FOPSL***, by structural induction on the formulas it can be shown that a ***FOPSL*** formula (similarly FO-CTL*$_{lp}$ path formula) is satisfied if there is an infinite trace (similarly path) in which all the past requirements are maintained and all the future requirements are satisfied. Both the conjuncts above ensure this is the case, specifically $\Box$ $\mathit{weak}(\wp)$ ensuring satisfiability of the past requirements and $\forall p_1, p_2, q : P. \forall m : M. \forall t : T. ((\neg \gamma \mathcal{S} \lambda) \rightarrow \Diamond \gamma)$ ensuring that all the pending future requirements can be discharged. From this it follows that if an infinite extension that satisfies both the conjuncts can be found, that infinite extension will satisfy $\wp$. The infinite extension $\widehat{\rho}$ fulfills all the pending obligations in $\rho[..i]$ while maintaining weak compliance. Furthermore, by induction on the length of the finite path $i \in \mathbb{N}$, we can show that the formula $((\neg \gamma \mathcal{S} \lambda) \rightarrow \Diamond \gamma) \wedge \Box \mathit{weak}(\wp)$ also guarantees that all the obligations incurred in $\widehat{\rho}$ are fulfilled in some finite steps from the position they are incurred. Thus, it follows that $\rho[..i] \cdot \widehat{\rho}[(i+1)..] \models \wp$.[2] In the logical sentence **L**, the consequent require an infinite extension such that $\sigma_f \cdot \sigma_i \models \wp$. Note that, by construction, $\sigma_f = \rho[..i]$. Now let $\sigma_i = \widehat{\rho}[i+1..]$. From $\rho[..i] \cdot \widehat{\rho}[(i+1)..] \models \wp$ it follows that $\sigma_f \cdot \sigma_i \models \wp$, which completes our proof. □

Theorem 53. *Given a privacy policy $\wp$ specified in* ***FOPSL****, to check whether $\wp$ has the Δ-property is undecidable in general.*

Proof. We reduce the Turing machine halting problem [47] to checking whether a policy $\wp$ written in ***FOPSL*** does *not* have the Δ-property. To be more precise, according to our reduction, we will

[1] for all $\hat{p}_1, \hat{p}_2, \hat{q} \in P$, for all $\hat{m} \in M$ and for all $\hat{t} \in \mathcal{T}$, $\hat{\eta}' = \hat{\eta}[p_1 \mapsto \hat{p}_1][p_2 \mapsto \hat{p}_2][q \mapsto \hat{q}][m \mapsto \hat{m}][t \mapsto \hat{t}]$

[2] We use $\rho[i..]$ to denote the suffix starting at the position i of the infinite sequence ρ.

show that a given Turning machine will halt after reaching the pre-defined final state *if and only if* the encoded policy can incur an unsatisfiable obligation. Our reduction is inspired by the reduction of the Turing machine halting problem to the HRU safety problem [59].

Let us consider a Turing machine $\mathbb{T} = (\mathbb{A}, \mathbb{S}, q_0, \delta)$. Associated with a Turing machine is an infinite tape which is divided into cells[3], and a tape head which resides on one of the cells of the tape, can read from and write to the current tape cell, and can move the left-adjacent or right-adjacent cell. $\mathbb{A}$ denotes the machine's alphabet of symbols with a distinguished "blank" symbol represented by $\mathfrak{b}$. Each cell of the tape contains one symbol. $\mathbb{S}$ denotes the set of states the Turing machine may be in. At the beginning, $\mathbb{T}$ is in state $q_0 \in \mathbb{S}$.

The operation of $\mathbb{T}$ is guided by the total transition function $\delta : \mathbb{A} \times \mathbb{S} :\rightarrow \mathbb{A} \times \mathbb{S} \times \{L, R\}$[4]. δ controls the movement of the tape head of $\mathbb{T}$ and accompanying state changes. Consider that the tape head of $\mathbb{T}$ is in a cell where the symbol is A and the state of $\mathbb{T}$ is p and we have the following $\delta(A, p) = (B, q, R)$. This denotes that the tape head will write the symbol B into the current cell, replacing A. The tape head will then move to the cell right of the current cell. Last but not least, $\mathbb{T}$ will change its state from p to q. Similarly, if the valuation of the transition function were $\delta(A, p) = (B, q, L)$, then tape head would move to the left instead of the right.

Given a Turing machine $\mathbb{T} = (\mathbb{A}, \mathbb{S}, q_0, \delta)$, the halting problem asks whether the machine $\mathbb{T}$ will, started on an empty tape[5], enter a predefined state q_f. The halting problem has been shown to be undecidable [47].

On a high level, we model the halting problem as follows. Execution of $\mathbb{T}$ for i steps corresponds to a finite trace σ_i such that (1) σ_i models the current state and tape of $\mathbb{T}$ after i steps, (2) σ_i is weakly compliant with respect to the policy $\wp_{\mathbb{T}}$ created from $\mathbb{T}$, (3) σ_i can only be extended with actions modeling the execution of step $i+1$ (and ultimately lead to σ_{i+1}, and (4) if $\mathbb{T}$ ever entered state q_f, then some action in σ_i incurred an obligation that is not fulfillable under $\wp_{\mathbb{T}}$. With

[3] Tapes can be formalized as total functions $\mathbb{Z} \mapsto \mathbb{A}$.

[4] Recall that we used δ to represent the FO-CTL*$_{lp}$ formulation of the Δ-property of a given policy $\wp$. For consistency with prior work, only here we use δ to represent the transition relation of $\mathbb{T}$. In all other contexts we use δ to mean the FO-CTL*$_{lp}$ formulation of the Δ-property for a given policy $\wp$.

[5] A tape is empty if all cells contain the blank symbol $\mathfrak{b}$.

this setup, for any weakly compliant finite trace σ_f, there exists an infinite extension σ_∞ such that $\sigma_f \cdot \sigma_\infty$ satisfies $\wp_{\mathbb{T}}$ if and only if $\mathbb{T}$ did not or will not enter q_f.

For this encoding we assume that the following predicates have a fixed interpretation: send, contains, in. We do not need any other predicates and thus leave interpretation of all other predicates open. We use send events to model whether a cell in the tape has been visited, whether a tape cell is the right most cell visited, whether a tape cell is the left most cell visited, and whether two cells are neighboring cells. We also use the most recent send event to model the current state and also the current head position of $\mathbb{T}$. We encode the transition function of $\mathbb{T}$ into negative norms of our policy. We model tape cells with principals in our domain. To model an infinite tape, we assume the carrier of principals to be infinite. States in $\mathbb{S}$ and presence or absence of symbols in $\mathbb{A}$ are modeled as attributes of our domain. The carrier of attributes is assumed to be large enough to contain all constants we require. We will also use attributes *endRight* and *endLeft* to respectively represent the right most cell and left most cell that have been visited by the $\mathbb{T}$'s head. We will also use attribute *neighbour* to check whether two cells are neighboring cells of the tape. Let us consider that we want to know whether two cells (represented as principals in the domain) p_1 and p_2 are neighboring cells where p_1 is the left cell and p_2 is the right cell. In that case, we just need to check whether we have seen a message from p_1 to p_2 containing the attribute *neighbour*. If we want to model that we want to write symbol A in the tape cell represented by p_1, we just need to send a message where the sender is p_1 and the attribute is A. Now, to delete a symbol A from a tape cell p_1 we just need to send a message whose sender is p_1 and the message contains the attribute $\bar{A}$.

Note that our reduction creates a policy that allows stuttering of send events. This means that the (deterministic) execution of $\mathbb{T}$ is potentially mapped to many traces that are weakly compliant. For instance, consider an execution $abcde\ldots$ of $\mathbb{T}$, which is equivalent to both traces $aabbbbcdeee\ldots$ and $aaaaabccccdddddddee\ldots$ for our policy. This does not have an impact on the result, though, since for any equivalent prefix the existence of an infinite extension is preserved. For simplicity we will only consider the canonical non-stuttering trace in this development.

$$
\begin{aligned}
\textbf{msg}(\hat{p_1},\hat{p_2},\hat{t}) &\equiv \exists q' : P.\exists m' : M.\text{send}(\hat{p_1},\hat{p_2},m') \wedge \text{contains}(m',q',\hat{t}) \\
\textbf{msg-s}(\hat{p_1},\hat{t}) &\equiv \exists p_2', q' : P.\exists m' : M.\text{send}(\hat{p_1},p_2',m') \wedge \text{contains}(m',q',\hat{t}) \\
\textbf{msg-t}(\hat{t}) &\equiv \exists p_1', p_2', q' : P.\exists m' : M.\text{send}(p_1',p_2',m') \wedge \text{contains}(m',q',\hat{t}) \\
\textbf{in-transition}_{(Z,a)} &\equiv \neg\textbf{msg-t}(\textit{end-transition})\, \mathcal{S}\, \textbf{msg-t}(\textit{start-transition}_{(Z,a)})
\end{aligned}
$$

Figure 5.2: Macros used in the reduction

Our reduction has the following four steps:

1. Write *intermediate negative policy norms* which will enable setting up the initial configuration of $\mathbb{T}$
2. Encode each transition relation element of $\mathbb{T}$ as a set of intermediate negative policy norms
3. Translate the intermediate negative norms to actual negative norms of the encoded policy $\wp$
4. Add an additional policy norm that incurs an unsatisfiable obligation if the $\mathbb{T}$ reaches the state q_f.

For the ease of presentation we use the macros $\textbf{msg}(p_1,p_2,t)$, $\textbf{msg-s}(p_1,t)$, and $\textbf{msg-t}(t)$ defined in Figure 5.2.

Step 1. $\mathbb{T}$ is initially in the state $q_0 \in \mathbb{S}$. We assume that $\mathbb{T}$'s tape head is in cell 0, which contains the `b` symbol. During execution, the amount of tape $\mathbb{T}$ has seen is finite. At each point in time there is a left-most visited and right-most visited cell. In the beginning, cell 0 is both the left-most and right-most visited cell.

Step 1 models the initial setup of the Halting problem. It ensures that the initialization phase is the only thing that is allowed in the beginning in a weakly compliant (finite) trace. Only when the initial setup has been performed can any transition of $\mathbb{T}$ take place. Templates for the intermediate negative norms guiding the initialization are given in Figure 5.3.

$$\mathrm{in}(t, \textit{setup-start}) \rightarrow \neg \diamondsuit \textbf{msg-t}(\textit{create-cell})$$

$$\begin{aligned}\mathrm{in}(t, \textit{create-cell}) \rightarrow & (\neg \textbf{msg-t}(q_0)\, \mathcal{S}\, \textbf{msg-t}(\textit{setup-start})) \\ & \wedge (\textbf{msg}(p_1, p_2, \textit{create-cell})\, \mathcal{S}\, \textbf{msg-t}(\textit{setup-start}))\end{aligned}$$

$$\begin{aligned}\mathrm{in}(t, q_0) \rightarrow & (\neg \textbf{msg-t}(\textit{endLeft})\, \mathcal{S}\, \textbf{msg-t}(\textit{create-cell})) \\ & \wedge (\textbf{msg}(p_1, p_2, q_0)\, \mathcal{S}\, \textbf{msg}(p_1, p_2, \textit{create-cell}))\end{aligned}$$

$$\begin{aligned}\mathrm{in}(t, \textit{endLeft}) \rightarrow & (\neg \textbf{msg-t}(\textit{endRight})\, \mathcal{S}\, \textbf{msg-t}(q_0)) \\ & \wedge (\textbf{msg}(p_1, p_2, \textit{endLeft})\, \mathcal{S}\, \textbf{msg}(p_1, p_2, q_0))\end{aligned}$$

$$\begin{aligned}\mathrm{in}(t, \textit{endRight}) \rightarrow & (\neg \textbf{msg-t}(\mathtt{b})\, \mathcal{S}\, \textbf{msg-t}(\textit{endLeft})) \\ & \wedge (\textbf{msg}(p_1, p_2, \textit{endRight})\, \mathcal{S}\, \textbf{msg}(p_1, p_2, \textit{endLeft}))\end{aligned}$$

$$\begin{aligned}\mathrm{in}(t, \mathtt{b}) \rightarrow & (\neg \textbf{msg-t}(\textit{setup-end})\, \mathcal{S}\, \textbf{msg-t}(\textit{endRight})) \\ & \wedge (\textbf{msg}(p_1, p_2, \mathtt{b})\, \mathcal{S}\, \textbf{msg}(p_1, p_2, \textit{endRight}))\end{aligned}$$

$$\begin{aligned}\mathrm{in}(t, \textit{setup-end}) \rightarrow & \diamondsuit \textbf{msg-t}(\textit{setup-start}) \wedge \diamondsuit \textbf{msg-t}(\textit{create-cell}) \wedge \diamondsuit \textbf{msg-t}(q_0) \\ & \wedge \diamondsuit \textbf{msg-t}(\textit{endLeft}) \wedge \diamondsuit \textbf{msg-t}(\textit{endRight}) \wedge \diamondsuit \textbf{msg-t}(\mathtt{b}) \wedge \\ & \bigwedge_{(W,u) \in \delta} \neg \diamondsuit \textbf{msg-t}(\textit{start-transition}_{W,u})\end{aligned}$$

Figure 5.3: Step 1: Intermediate negative norm template to setup the initial configuration of $\mathbb{T}$

Step 2. In this step, we encode the transition relation of $\mathbb{T}$ as intermediate negative norms of our policy. We consider each element of the transition relation and convert it to a set of intermediate negative norms. Without loss of generality we consider a left-moving element of the transition relation: $\delta(A, p) = (B, q, L)$; right-moving cases are symmetric. There are two separate sub-cases to consider. The first case is the one where the left cell of the current cell has been previously visited. Obviously the other case is when the left cell of the current cell has not been previously visited. In the first case, we will just delete the symbol A and state p from the current cell. To achieve this we have to allow sending two messages from the principal representing the current cell of $\mathbb{T}$. The two messages should contain the attributes $\bar{A}$ and $\bar{p}$, respectively, denoting the deletion. The next thing we have to do is to write the symbol B into the current cell. To achieve

$$\mathrm{in}(t, \textit{start-transition}_{(A,p)}) \rightarrow \neg \mathbf{msg}(p_1, p_2, \bar{p})\, \mathcal{S} \left(\mathbf{msg\text{-}t}(\textit{end-transition}) \vee \mathbf{msg\text{-}t}(\textit{setup-end}) \right)$$
$$\wedge \Big((\mathbf{msg}(p_1, p_2, \textit{start-transition}_{(A,p)})\, \mathcal{S}\, \mathbf{msg\text{-}t}(\textit{end-transition})) \vee$$
$$(\mathbf{msg}(p_1, p_2, \textit{start-transition}_{(A,p)})\, \mathcal{S}\, \mathbf{msg\text{-}t}(\textit{setup-end})) \Big) \wedge$$
$$\diamondminus \mathbf{msg}(p_2, p_1, \textit{neighbour}) \wedge (\neg \mathbf{msg\text{-}s}(p_1, \bar{p})\, \mathcal{S}\, \mathbf{msg\text{-}s}(p_1, p))$$
$$\wedge (\neg \mathbf{msg\text{-}s}(p_1, \bar{A})\, \mathcal{S}\, \mathbf{msg\text{-}s}(p_1, A))$$

$$\mathrm{in}(t, \bar{p}) \rightarrow (\mathbf{msg}(p_1, p_2, \bar{p})\, \mathcal{S}\, \mathbf{msg}(p_1, p_2, \textit{start-transition}_{(A,p)})) \bigwedge$$
$$(\neg \mathbf{msg}(p_1, p_2, \bar{A})\, \mathcal{S}\, \mathbf{msg}(p_1, p_2, \textit{start-transition}_{(A,p)}))$$

$$\mathrm{in}(t, \bar{A}) \rightarrow (\mathbf{msg}(p_1, p_2, \bar{A})\, \mathcal{S}\, \mathbf{msg}(p_1, p_2, \bar{p})) \wedge \mathbf{in\text{-}transition}_{(A,p)}$$
$$(\neg \mathbf{msg}(p_1, p_2, B)\, \mathcal{S}\, \mathbf{msg}(p_1, p_2, \bar{p}))$$

$$\mathrm{in}(t, B) \rightarrow (\mathbf{msg}(p_1, p_2, B)\, \mathcal{S}\, \mathbf{msg}(p_1, p_2, \bar{A})) \wedge \mathbf{in\text{-}transition}_{(A,p)}$$
$$(\neg \mathbf{msg}(p_2, p_1, q)\, \mathcal{S}\, \mathbf{msg}(p_1, p_2, \bar{A}))$$

$$\mathrm{in}(t, q) \rightarrow (\mathbf{msg}(p_1, p_2, q)\, \mathcal{S}\, \mathbf{msg}(p_2, p_1, B)) \wedge \mathbf{in\text{-}transition}_{(A,p)}$$
$$(\neg \mathbf{msg}(p_2, p_1, \textit{end-transition})\, \mathcal{S}\, \mathbf{msg}(p_2, p_1, B))$$

$$\mathrm{in}(t, \textit{end-transition}) \rightarrow \left(\neg \bigvee_{(Z,a) \in \delta} \mathbf{msg\text{-}t}(\textit{start-transition}_{(Z,a)}) \right) \mathcal{S}\, \mathbf{msg\text{-}t}(q)$$

Figure 5.4: Step 2: Intermediate negative norm template for transition of form $\delta(A, p) = (B, q, L)$ where $\mathbb{T}$'s tape head is not on the left-most seen cell

this, the principal representing the current tape cell should send a message containing the attribute B. The next thing would be to change the state of $\mathbb{T}$ to q. To achieve this and to show that the tape head head has moved left, we would send a message from the principal representing the left cell of the current cell containing the attribute q. Note that to check whether the cell is actually the left cell of the current cell, we see whether the cell representing the left cell has actually sent a message to the principal representing the current cell containing the attribute *neighbour*. Once this is done, we allow a message containing the attribute *end-transition* representing that the transition has been

completed. Note that, to ensure consistency and atomicity of the changes required for a transition to happen, we use a locking mechanism that ensures that until all operations of the current transition have been completed, no other transitions are enabled. The template for generating the negative norms for this case is shown in Figure 5.4.

The next case is the one where we are required to go left according to the transition function and we have not previously visited the left cell of the current cell in the tape. The first thing we do is to send a message from the principal, representing the current cell, containing the attribute *end$\bar{L}$eft*. It signifies that the current cell is no longer the left-most cell $\mathbb{T}$ has visited. We then send a message from the principal representing the left cell that contains the attribute *create-cell*, which binds that principal to this cell. A principal can only represent one cell, so we can only allow this if that principal has not sent such a message before. We then allow the principal representing the left cell to send a message to the the principal representing the current cell containing the attribute *neighbour*. This sets the neighboring relation and allows us in the future to check whether the cells are connected. Next we mark the new cell as the new left-most cell by allowing a send from the left principal containing the attribute *endLeft*. Once this has been established, we allow sending the same messages discussed above, simulating the normal operation of $\mathbb{T}$. Note that we also use the locking mechanism to enable the consistency and atomicity of the transition relation. The template of the intermediate negative norms for this case is shown in Figure 5.5.

The templates of intermediate negative norms for the right-moving cases are shown in Figures 5.6 and 5.7.

Step 3. According to the previous step (Step 2), once we have intermediate negative norms, we have to convert these intermediate negative norms into actual negative norms of the encoded policy. These negative norms will force the Turing machine simulation to only take enabled transitions. The intermediate negative norms obtained from the previous step have the form $Ant \rightarrow Con$. For each unique antecedent, for instance A_0, we collect all the intermediate negative norms C_i which have the same antecedent A_0, and create a negative norm $A_0 \rightarrow \bigvee_i C_i$. For example, consider that

$$
\begin{aligned}
\mathrm{in}(t, \textit{start-transition}_{(A,p)}) \rightarrow \Big(& \neg \mathbf{msg}(p_1, p_2, \overline{\textit{endLeft}}) \, \mathcal{S} \, (\mathbf{msg\text{-}t}(\textit{end-transition}) \vee \\
& \mathbf{msg\text{-}t}(\textit{setup-end})) \Big) \wedge \Big((\mathbf{msg}(p_1, p_2, \textit{start-transition}_{(A,p)}) \, \mathcal{S} \, \mathbf{msg\text{-}t}(\textit{end-transition})) \\
& \vee (\mathbf{msg}(p_1, p_2, \textit{start-transition}_{(A,p)}) \, \mathcal{S} \, \mathbf{msg\text{-}t}(\textit{setup-end})) \Big) \\
& \wedge (\neg \mathbf{msg\text{-}s}(p_1, \overline{\textit{endLeft}}) \, \mathcal{S} \, \mathbf{msg\text{-}s}(p_1, \textit{endLeft})) \\
& \wedge (\neg \mathbf{msg\text{-}s}(p_1, \bar{p}) \, \mathcal{S} \, \mathbf{msg\text{-}s}(p_1, p)) \wedge (\neg \diamondsuit \mathbf{msg\text{-}s}(p_2, \textit{create-cell})) \\
& \wedge (\neg \mathbf{msg\text{-}s}(p_1, \bar{A}) \, \mathcal{S} \, \mathbf{msg\text{-}s}(p_1, A)) \\
\mathrm{in}(t, \overline{\textit{endLeft}}) \rightarrow & (\mathbf{msg}(p_1, p_2, \overline{\textit{endLeft}}) \, \mathcal{S} \, \mathbf{msg}(p_1, p_2, \textit{start-transition}_{(A,p)})) \\
& \wedge (\neg \mathbf{msg}(p_2, p_1, \textit{create-cell}) \, \mathcal{S} \, \mathbf{msg}(p_1, p_2, \textit{start-transition}_{(A,p)})) \\
\mathrm{in}(t, \textit{create-cell}) \rightarrow & (\mathbf{msg}(p_1, p_2, \textit{create-cell}) \, \mathcal{S} \, \mathbf{msg}(p_2, p_1, \overline{\textit{endLeft}})) \\
& \wedge (\neg \mathbf{msg}(p_1, p_2, \textit{neighbour}) \, \mathcal{S} \, \mathbf{msg}(p_2, p_1, \overline{\textit{endLeft}})) \\
\mathrm{in}(t, \textit{neighbour}) \rightarrow & (\mathbf{msg}(p_1, p_2, \textit{neighbour}) \, \mathcal{S} \, \mathbf{msg}(p_2, p_1, \textit{create-cell})) \wedge \\
& \mathbf{in\text{-}transition}_{(A,p)} \wedge (\neg \mathbf{msg}(p_2, p_1, \textit{endRight}) \, \mathcal{S} \, \mathbf{msg}(p_1, p_2, \textit{create-cell})) \\
\mathrm{in}(t, \textit{endLeft}) \rightarrow & (\mathbf{msg}(p_1, p_2, \textit{endLeft}) \, \mathcal{S} \, \mathbf{msg}(p_1, p_2, \textit{neighbour})) \wedge \\
& \mathbf{in\text{-}transition}_{(A,p)} \wedge (\neg \mathbf{msg}(p_2, p_1, \bar{p}) \, \mathcal{S} \, \mathbf{msg}(p_1, p_2, \textit{neighbour})) \\
\mathrm{in}(t, \bar{p}) \rightarrow & (\mathbf{msg}(p_1, p_2, \bar{p}) \, \mathcal{S} \, \mathbf{msg}(p_2, p_1, \textit{endLeft})) \wedge \mathbf{in\text{-}transition}_{(A,p)} \\
& \wedge (\neg \mathbf{msg}(p_1, p_2, \bar{A}) \, \mathcal{S} \, \mathbf{msg}(p_2, p_1, \textit{endLeft})) \\
\mathrm{in}(t, \bar{A}) \rightarrow & (\mathbf{msg}(p_1, p_2, \bar{A}) \, \mathcal{S} \, \mathbf{msg}(p_1, p_2, \bar{p})) \wedge \mathbf{in\text{-}transition}_{(A,p)} \wedge \\
& (\neg \mathbf{msg}(p_1, p_2, B) \, \mathcal{S} \, \mathbf{msg}(p_1, p_2, \bar{p})) \\
\mathrm{in}(t, B) \rightarrow & (\mathbf{msg}(p_1, p_2, B) \, \mathcal{S} \, \mathbf{msg}(p_1, p_2, \bar{A})) \wedge \mathbf{in\text{-}transition}_{(A,p)} \wedge \\
& (\neg \mathbf{msg}(p_2, p_1, q) \, \mathcal{S} \, \mathbf{msg}(p_1, p_2, \bar{A})) \\
\mathrm{in}(t, q) \rightarrow & (\mathbf{msg}(p_1, p_2, q) \, \mathcal{S} \, \mathbf{msg}(p_2, p_1, B)) \wedge \mathbf{in\text{-}transition}_{(A,p)} \wedge \\
& (\neg \mathbf{msg\text{-}t}(\textit{end-transition}) \, \mathcal{S} \, \mathbf{msg}(p_2, p_1, B)) \\
\mathrm{in}(t, \textit{end-transition}) \rightarrow & \Big(\neg \bigvee_{(Z,a) \in \delta} \mathbf{msg\text{-}t}(\textit{start-transition}_{(Z,a)}) \Big) \, \mathcal{S} \, \mathbf{msg\text{-}t}(q)
\end{aligned}
$$

Figure 5.5: Step 2: Intermediate negative norm template for transition of form $\delta(A,p) = (B,q,L)$ where $\mathbb{T}$'s tape head is in the left-most seen cell

$$\text{in}(t, \textit{start-transition}_{(A,p)}) \rightarrow \Big(\neg \mathbf{msg}(p_1, p_2, \bar{p})\, \mathcal{S}\, (\mathbf{msg\text{-}t}(\textit{end-transition}) \vee \mathbf{msg\text{-}t}(\textit{setup-end})) \Big)$$
$$\wedge \Big((\mathbf{msg}(p_1, p_2, \textit{start-transition}_{(A,p)})\, \mathcal{S}\, \mathbf{msg\text{-}t}(\textit{end-transition}))$$
$$\vee (\mathbf{msg}(p_1, p_2, \textit{start-transition}_{(A,p)})\, \mathcal{S}\, \mathbf{msg\text{-}t}(\textit{setup-end})) \Big)$$
$$\wedge \mathbf{msg}(p_1, p_2, \textit{neighbour}) \wedge (\neg \mathbf{msg\text{-}s}(p_1, \bar{p})\, \mathcal{S}\, \mathbf{msg\text{-}s}(p_1, p)) \wedge (\neg \mathbf{msg\text{-}s}(p_1, \bar{A})\, \mathcal{S}\, \mathbf{msg\text{-}s}(p_1, A))$$

$$\text{in}(t, \bar{p}) \rightarrow (\mathbf{msg}(p_1, p_2, \bar{p})\, \mathcal{S}\, \mathbf{msg}(p_1, p_2, \textit{start-transition}_{(A,p)})) \bigwedge$$
$$(\neg \mathbf{msg}(p_1, p_2, \bar{A})\, \mathcal{S}\, \mathbf{msg}(p_1, p_2, \textit{start-transition}_{(A,p)}))$$

$$\text{in}(t, \bar{A}) \rightarrow (\mathbf{msg}(p_1, p_2, \bar{A})\, \mathcal{S}\, \mathbf{msg}(p_1, p_2, \bar{p})) \wedge \mathbf{in\text{-}transition}_{(A,p)}$$
$$(\neg \mathbf{msg}(p_1, p_2, B)\, \mathcal{S}\, \mathbf{msg}(p_1, p_2, \bar{p}))$$

$$\text{in}(t, B) \rightarrow (\mathbf{msg}(p_1, p_2, B)\, \mathcal{S}\, \mathbf{msg}(p_1, p_2, \bar{A})) \wedge \mathbf{in\text{-}transition}_{(A,p)}$$
$$(\neg \mathbf{msg}(p_2, p_1, q)\, \mathcal{S}\, \mathbf{msg}(p_1, p_2, \bar{A}))$$

$$\text{in}(t, q) \rightarrow (\mathbf{msg}(p_1, p_2, q)\, \mathcal{S}\, \mathbf{msg}(p_2, p_1, B)) \wedge \mathbf{in\text{-}transition}_{(A,p)}$$
$$(\neg \mathbf{msg}(p_2, p_1, \textit{end-transition})\, \mathcal{S}\, \mathbf{msg}(p_2, p_1, B))$$

$$\text{in}(t, \textit{end-transition}) \rightarrow \left(\neg \bigvee_{(Z,a) \in \delta} \mathbf{msg\text{-}t}(\textit{start-transition}_{(Z,a)}) \right) \mathcal{S}\, \mathbf{msg\text{-}t}(q)$$

Figure 5.6: Step 2: Intermediate negative norm template for transition of form $\delta(A, p) = (B, q, R)$ where $\mathbb{T}$'s tape head is not on the right most seen cell

these intermediate norms are the following: $A_0 \rightarrow C_1$, $A_0 \rightarrow C_2$, ..., $A_0 \rightarrow C_n$. Then we create a new negative norm $A_0 \rightarrow C_1 \vee C_2 \vee \ldots \vee C_n$. As mentioned before, we do this process for each unique antecedent in the set of intermediate negative norms.

Step 4. Let us consider that $\mathbb{T}$ reaches the state q_f after i steps. We want to ensure that a corresponding σ_i is weakly compliant, but that there exists no infinite extension such that the concatenation satisfies $\wp_{\mathbb{T}}$. For this we add two negative norms as shown in Figure 5.8. The first one incurs an unsatisfiable obligation when $\mathbb{T}$ reaches q_f. The second negative norm disallows the obligation.

$$\begin{aligned}
\text{in}(t, \textit{start-transition}_{(A,p)}) \rightarrow & \Big(\neg \mathbf{msg}(p_1, p_2, \overline{\textit{endRight}}) \, \mathcal{S} \, (\mathbf{msg\text{-}t}(\textit{end-transition}) \vee \\
& \mathbf{msg\text{-}t}(\textit{setup-end})) \Big) \wedge \Big((\mathbf{msg}(p_1, p_2, \textit{start-transition}_{(A,p)}) \, \mathcal{S} \, \mathbf{msg\text{-}t}(\textit{end-transition})) \\
& \vee (\mathbf{msg}(p_1, p_2, \textit{start-transition}_{(A,p)}) \, \mathcal{S} \, \mathbf{msg\text{-}t}(\textit{setup-end})) \Big) \\
& \wedge (\neg \mathbf{msg\text{-}s}(p_1, \overline{\textit{endRight}}) \, \mathcal{S} \, \mathbf{msg\text{-}s}(p_1, \textit{endRight})) \\
& \wedge (\neg \mathbf{msg\text{-}s}(p_1, \bar{p}) \, \mathcal{S} \, \mathbf{msg\text{-}s}(p_1, p)) \wedge (\neg \diamondsuit \mathbf{msg\text{-}s}(p_2, \textit{create-cell})) \\
& \wedge (\neg \mathbf{msg\text{-}s}(p_1, \bar{A}) \, \mathcal{S} \, \mathbf{msg\text{-}s}(p_1, A)) \\
\text{in}(t, \overline{\textit{endRight}}) \rightarrow & (\mathbf{msg}(p_1, p_2, \overline{\textit{endRight}}) \, \mathcal{S} \, \mathbf{msg}(p_1, p_2, \textit{start-transition}_{(A,p)})) \\
& \wedge (\neg \mathbf{msg}(p_2, p_1, \textit{create-cell}) \, \mathcal{S} \, \mathbf{msg}(p_1, p_2, \textit{start-transition}_{(A,p)})) \\
\text{in}(t, \textit{create-cell}) \rightarrow & (\mathbf{msg}(p_1, p_2, \textit{create-cell}) \, \mathcal{S} \, \mathbf{msg}(p_2, p_1, \overline{\textit{endRight}})) \\
& \wedge (\neg \mathbf{msg}(p_2, p_1, \textit{neighbour}) \, \mathcal{S} \, \mathbf{msg}(p_2, p_1, \overline{\textit{endRight}})) \\
\text{in}(t, \textit{neighbour}) \rightarrow & (\mathbf{msg}(p_1, p_2, \textit{neighbour}) \, \mathcal{S} \, \mathbf{msg}(p_2, p_1, \textit{create-cell})) \wedge \\
& \mathbf{in\text{-}transition}_{(A,p)} \wedge (\neg \mathbf{msg}(p_2, p_1, \textit{endRight}) \, \mathcal{S} \, \mathbf{msg}(p_2, p_1, \textit{create-cell})) \\
\text{in}(t, \textit{endRight}) \rightarrow & (\mathbf{msg}(p_1, p_2, \textit{endRight}) \, \mathcal{S} \, \mathbf{msg}(p_2, p_1, \textit{neighbour})) \wedge \\
& \mathbf{in\text{-}transition}_{(A,p)} \wedge (\neg \mathbf{msg}(p_2, p_1, \bar{p}) \, \mathcal{S} \, \mathbf{msg}(p_2, p_1, \textit{neighbour})) \\
\text{in}(t, \bar{p}) \rightarrow & (\mathbf{msg}(p_1, p_2, \bar{p}) \, \mathcal{S} \, \mathbf{msg}(p_2, p_1, \textit{endRight})) \wedge \mathbf{in\text{-}transition}_{(A,p)} \\
& \wedge (\neg \mathbf{msg}(p_1, p_2, \bar{A}) \, \mathcal{S} \, \mathbf{msg}(p_2, p_1, \textit{endRight})) \\
\text{in}(t, \bar{A}) \rightarrow & (\mathbf{msg}(p_1, p_2, \bar{A}) \, \mathcal{S} \, \mathbf{msg}(p_1, p_2, \bar{p})) \wedge \mathbf{in\text{-}transition}_{(A,p)} \wedge \\
& (\neg \mathbf{msg}(p_1, p_2, B) \, \mathcal{S} \, \mathbf{msg}(p_1, p_2, \bar{p})) \\
\text{in}(t, B) \rightarrow & (\mathbf{msg}(p_1, p_2, B) \, \mathcal{S} \, \mathbf{msg}(p_1, p_2, \bar{A})) \wedge \mathbf{in\text{-}transition}_{(A,p)} \wedge \\
& (\neg \mathbf{msg}(p_2, p_1, q) \, \mathcal{S} \, \mathbf{msg}(p_1, p_2, \bar{A})) \\
\text{in}(t, q) \rightarrow & (\mathbf{msg}(p_1, p_2, q) \, \mathcal{S} \, \mathbf{msg}(p_2, p_1, B)) \wedge \mathbf{in\text{-}transition}_{(A,p)} \wedge \\
& (\neg \mathbf{msg\text{-}t}(\textit{end-transition}) \, \mathcal{S} \, \mathbf{msg}(p_2, p_1, B)) \\
\text{in}(t, \textit{end-transition}) \rightarrow & \Big(\neg \bigvee_{(Z,a) \in \delta} \mathbf{msg\text{-}t}(\textit{start-transition}_{(Z,a)}) \Big) \, \mathcal{S} \, \mathbf{msg\text{-}t}(q)
\end{aligned}$$

Figure 5.7: Step 2: Intermediate negative norm template for transition of form $\delta(A, p) = (B, q, R)$ where $\mathbb{T}$'s tape head is in the right most seen cell

$$\begin{array}{rcl} \text{in}(t, q_f) & \rightarrow & \Diamond(\exists p'_1, p'_2, q' : P.\exists m' : M.\text{send}(p'_1, p'_2, m') \wedge \text{contains}(m', q', \mathit{arbitrary})) \\ \text{in}(t, \mathit{arbitrary}) & \rightarrow & \bot \end{array}$$

Figure 5.8: Step 4: Template of two additional negative norms, one of which incurs an unsatisfiable obligation when $\mathbb{T}$ reaches state q_f and the other one disallows the obligation.

From the setup of steps 1 through 3, it follows that a finite trace is weakly compliant if and only if it corresponds to some steps of execution of the Turing machine $\mathbb{T}$ (or is in the middle of the initialization phase or a transition), and furthermore that such traces exist for the execution. Thus, if $\mathbb{T}$ can reach q_f, there exists a weakly compliant finite trace σ that corresponds to the execution. By the fourth step, this trace incurs an unsatisfiable obligation an can thus not be extended to be compliant with the policy. Thus, $\mathbb{T}$ will reach state q_f if and only if the resulting policy does not have the Δ-property. □

5.2 Sufficient and Necessary Condition for Δ-property

There are two cases in which an action that is weakly compliant with a policy $\wp$ is not strongly compliant with $\wp$. The first case occurs when taking a weakly compliant action can lead to a state from which it is not possible to take an unbounded number of weakly-compliant valid transitions (no infinite weakly compliant extension). We call a policy which does ***not*** allow this case an*incrementally satisfiable policy*.

Definition 54 (Incrementally Satisfiable)**.** *A pure past FOTL formula ϕ is* incrementally satisfiable *iff for any given finite trace σ and for any logical environment η, $\sigma, |\sigma| - 1, \eta \models \boxminus\phi$ implies that there exists an infinite trace $\hat{\sigma}$ such that $\sigma \cdot \hat{\sigma} \models \Box\phi$.*

The second case in which a weakly compliant action for $\wp$ is not strongly compliant for $\wp$ is when that action incurs a future obligation which cannot be met. Theorem 55 states that these two cases are necessary and sufficient.

Theorem 55. *A policy $\wp$ has the Δ-property if and only if weak($\wp$) is incrementally satisfiable and*

no weakly compliant action of $\wp$ incurs any unsatisfiable future obligations.

Proof. The proof follows from Theorem 52 which shows that $\mathbb{M}_{\wp} \models \delta(\wp)$ if and only if $\wp$ has the Δ-property. If we inspect the consequent of the formula $\delta(\wp)$, we have for all finite weak compliant histories that $E(\forall p_1, p_2, q : P.\forall m : M.\forall t : T.((\neg\gamma \mathcal{S}\ \lambda) \rightarrow \Diamond\gamma) \wedge \Box weak(\wp))$ holds. The above formula holds if and only if there exists an infinite path where $\forall p_1, p_2, q : P.\forall m : M.\forall t : T.((\neg\gamma \mathcal{S}\ \lambda) \rightarrow \Diamond\gamma)$ and $\Box weak(\wp)$ holds. Now, for all finite weakly compliant histories and for each such finite history the existence of an infinite trace that satisfies the conjunct $\Box weak(\wp)$ of the $\delta(\wp)$ consequent gives us the requirements of $weak(\wp)$ to be incrementally satisfiable. Moreover, for all finite weakly compliant histories and for each such finite history the existence of an infinite trace that satisfies the conjunct $\forall p_1, p_2, q : P.\forall m : M.\forall t : T.((\neg\gamma \mathcal{S}\ \lambda) \rightarrow \Diamond\gamma)$ ensures that all obligations incurred by any weakly compliant action can be discharged properly. Thus, for a policy $\wp$ to have the Δ-property, for all weakly compliant finite traces, we need an infinite weakly compliant extension in which all the pending obligations must be discharged. Now, the only ways to violate the Δ-property of a policy $\wp$ is to fail to satisfy $\Box weak(\wp)$, $\forall p_1, p_2, q : P.\forall m : M.\forall t : T.((\neg\gamma \mathcal{S}\ \lambda) \rightarrow \Diamond\gamma)$, or both. From this it follows that if we satisfy both the conjuncts we satisfy the Δ-property. □

Negative Norms :
$a \rightarrow \neg(\diamondsuit h)$
$b \rightarrow \neg(\diamondsuit h)$
$h \rightarrow \ominus(\neg(\diamondsuit h))$
Positive Norms :
$a \wedge \diamondsuit b$
b
h

Figure 5.9: Violation (1)

Positive Norms :
$a \wedge \diamondsuit b \wedge \diamondsuit c$
$b \wedge \diamondsuit c \wedge \diamondsuit d$
$c \wedge \diamondsuit d$
d
$f \wedge (\neg(\diamondsuit c))$
Negative Norms :
$a \rightarrow \Diamond f$

Figure 5.10: Violation (2)

We will now give an example for each violation case. First, consider the simple example policy in Figure 5.9, denoted by $\wp_1$. For brevity, we consider a pLTL policy in which a, b, and h are LTL formulas that are consistent with dynamic send events (actions) A, B, and H, respectively.

We also assume that at each step only one dynamic send event (action) can happen. A dynamic send event (action) is allowed by $\wp_1$ if it satisfies one of the positive norms and all the negative norms. Consider the finite trace BAH. Here, B is allowed as it satisfies the second positive norm and also satisfies all the negative norms (satisfies second negative norm as we have not seen any dynamic send event before that is consistent with h, and satisfies the rest of the negative norms by falsifying the antecedents). The same is true for A as it satisfies the first positive norm requiring that there is a dynamic send event (B) that is consistent with formula b before an A and additionally it satisfies all the negative norms. Finally, the same applies to action H. However, after H no more actions are allowed to take place. A cannot take place as it would violate the first negative norm requiring no dynamic events (H) consistent with h have not happened before. Similarly for B, the second negative norm would be violated. Finally, H cannot happen as an H has happened already which is consistent with the formula h, which would in turn violate the third negative norm. The action H leads to a bad state from which it is not possible to take an unbounded number of weakly compliant transitions. Thus, $\wp_1$ is not incrementally satisfiable. The action that leads to a bad state is H, which, although weakly compliant for $\wp_1$, is not strongly compliant with respect to $\wp_1$.

We use the policy in Figure 5.10 (denoted by $\wp_2$) to demonstrate the second violation case. We assume that at each step only one action can happen. For brevity, we consider a pLTL policy in which a, b, c, d, and f are LTL formulas that are respectively consistent with dynamic send events (actions) A, B, C, D, and F. Let us consider the finite trace where the following actions are taken in order: $DCBA$. Each action of the trace is weakly compliant with respect to $\wp_2$. The action A, however, incurs the obligation (future requirement) to perform an action (F) that is consistent with formula f. Note that the last positive norm allows F (F is consistent with formula f) under the condition that there is no action (C) consistent with c that has happened before. However, for the above finite trace this is not the case. As a result, the obligation of taking action F cannot be discharged in a compliant fashion. Thus, the action A, although weakly compliant for $\wp_2$, is not strongly compliant with respect to $\wp_2$.

Note that there are policies without future temporal operators that allow violation case 1 (not

incrementally verifiable). However, for violation case 2 (unsatisfiable future obligation) to occur, the policy needs to have future temporal operators.

We now prove that the first violation cannot happen for our forms of policies $\wp$ when they are satisfiable. A satisfiable policy $\wp$ that does not have the $\ominus$ operator is incrementally verifiable. This is stated in the following Theorem 57. Before showing the theorem we have the following auxliary lemma which we use to proof Theorem 57.

Lemma 56 (Stuttering)**.** *For all closed, pure-past, and satisfiable policy $\wp$ without the $\ominus$ temporal operator, for all environment η, for all finite trace σ_f, and for all state s, the following holds:* $\sigma_f \cdot s, |\sigma_f|, \eta \models \boxminus \wp \leftrightarrow \sigma_f \cdot s \cdot s, |\sigma_f| + 1, \eta \models \boxminus \wp$.

Proof. We know for all $0 \leq i \leq |\sigma_f|$, $\sigma_f \cdot s, i, \eta \models \wp$. To complete the proof, we have to show that $\sigma_f \cdot s \cdot s, |\sigma_f| + 1, \eta \models \wp$. It can be shown by induction on the structure of $\wp$ that the truth value assumed by each subformula of $\wp$ is identical in two adjacent states if the action parts of the labels of those two states are identical. The base cases are trivial. In the step, the logical connectives and quantifier cases are straightforward. Recall that $\wp$ is pure-past and does not have the $\ominus$ temporal operator and thus the only temporal operator we have to consider is the *non-strict* version of $\mathcal{S}$. The semantics of the strict version of $\mathcal{S}$ does not take the current state into consideration while evaluating whether a formula of form $\wp_1 \mathcal{S} \wp_2$ holds in the current state. According to the semantics of non-strict $\mathcal{S}$, the formula $\wp_1 \mathcal{S} \wp_2$ is true in state k when $\wp_2$ is true in the current state k. Otherwise it is false when $\wp_1$ is false in the current state k. In the remaining case, in both of the two adjacent states (k and $(k-1)$) under consideration it takes on the same value as it did in the state immediately prior to them. This can be written as: $[\![\wp_1 \mathcal{S} \wp_2]\!]^k = [\![\wp_2]\!]^k \vee \left([\![\wp_1 \mathcal{S} \wp_2]\!]^{(k-1)} \wedge [\![\wp_1]\!]^k \right)$ where $[\![\psi]\!]^k$ denotes the truth value of the formula ψ at state k. □

Theorem 57. *All closed, pure-past, and satisfiable policy $\wp$ without the $\ominus$ temporal operator, is incrementally satisfiable.*

Proof. To complete the proof, we have to show that (according the definition of incremental satisfiability), for all σ_f, for all η, for all closed, pure-past, and satisfiable policy $\wp$ without the $\ominus$

temporal operator, $\sigma_f, |\sigma_f| - 1, \eta \models \boxminus \wp \rightarrow \exists \sigma_i . \sigma_f \cdot \sigma_i \models \Box \wp$. Let us consider any σ_f and any η such that $\sigma_f = \sigma_p \cdot s$ and $\sigma_p \cdot s, |\sigma_p|, \eta \models \boxminus \wp$. Now we have to show that there exists an infinite extension σ_i such that $\sigma_f \cdot \sigma_i \models \Box \wp$. For this it is sufficient to show, for all k, $\sigma_f \cdot \sigma_i, k, \eta \models \wp$. We can construct the σ_i to be s^ω. An induction on k, using Lemma Lemma 56 shows, we can now show that $\sigma_f \cdot \sigma_i, k, \eta \models \wp$, where $\sigma_i = s^\omega$. □

Privacy policies of our form ($\wp$) do not allow the $\ominus$ temporal operator, which yields the following two corollaries.

Corollary 58. *For a given privacy policy $\wp$, weak($\wp$) is incrementally satisfiable if weak($\wp$) is satisfiable.*

Proof. This follows by Theorem 57, as *weak*($\wp$) is closed, has no future operators, and does not allow the $\ominus$ temporal operator. □

Corollary 59. *A satisfiable privacy policy $\wp$ satisfies Δ-property if and only if no weakly compliant action of $\wp$ incurs any unsatisfiable future obligations.*

Proof. Follows from Theorem 55 and Corollary 58. □

Given a policy $\wp$, *weak*($\wp$) denotes the present conditions imposed by $\wp$. Note that when $\wp$ has the Δ-property, *weak*($\wp$) denotes the safety property imposed by $\wp$. The proof uses the definition of the safety property. Weak compliance ensures that $\boxminus$*weak*($\wp$) is true in the current state, which includes the current contemplated action. This signifies that we have a finite prefix (history) which has satisfied the safety property of $\wp$. The Δ property ensures that we have an infinite extension of this finite prefix (that satisfies $\boxminus$*weak*($\wp$)) such that the concatenation of the finite prefix and the infinite extension satisfies $\wp$.

Theorem 60. *For a given privacy policy $\wp$ with the Δ-property, weak($\wp$) expresses the strongest safety property that contains the property expressed by $\wp$.*

Proof. The proof uses the definition of the safety property by Alpern and Schneider [6] (cf. Section 2.2). Weak compliance ensures that $\boxminus weak(\wp)$ is true in the current state, which includes the current contemplated action. This signifies that we have a finite prefix (history) which has satisfied the safety property of $\wp$. The definition of the safety property requires that for any finite prefix of a trace which is in the set of traces accepted by the safety property there exists an infinite extension of the finite prefix such that the concatenation of finite prefix and the infinite extension is also in the set of the safety property in question. Note that weak compliance (containing only past temporal operators) just ensures that the finite prefix does not violate the safety property imposed by $\wp$. However, it does not provide any guarantee about the existence of the infinite extension. The Δ property on the other hand ensures we have an infinite extension of this finite prefix (that satisfies $\boxminus weak(\wp)$) such that the concatenation of the finite prefix and the infinite extension satisfies $\wp$. Thus, from this we can say that when a policy $\wp$ has the Δ-property, $weak(\wp)$ expresses the strongest safety property that contains the property expressed by $\wp$. □

The satisfiability of the privacy policy is weaker than the Δ-property. The satisfiability of the privacy policy can neither capture the incrementally unsatisfiable policies nor the unsatisfiable future obligations. For a privacy policy to be satisfiable, it requires the *existence of just one trace* which satisfies the policy. Consider the privacy policy in Figure 5.9. Although the policy does not have the Δ-property, it has a satisfiable trace where the event H never happens. Such a trace would look like: $BA\cdots$. The same argument applies to the privacy policy in Figure 5.10. It also has a satisfiable trace where the event A never occurs and in turn does not incur the unsatisfiable obligation F. Such a trace could look like $DCB\cdots$, for example. Note that when a policy has the Δ-property, then it is satisfiable. However, the converse is not true.

We have proved that a privacy policy $\wp$ has the Δ-property if and only if $\mathbb{M}_{\wp} \models \delta(\wp)$. However, model checking an arbitrary specification written in FO-CTL$^*_{lp}$ with respect to a model is undecidable in general. The complexity of model checking a propositional CTL$^*_{lp}$ formula with respect to a model is in EXPSPACE [77] in the formula length. Thus, for a pLTL policy we can check whether the policy has the Δ-property in exponential space in the policy size. As our privacy

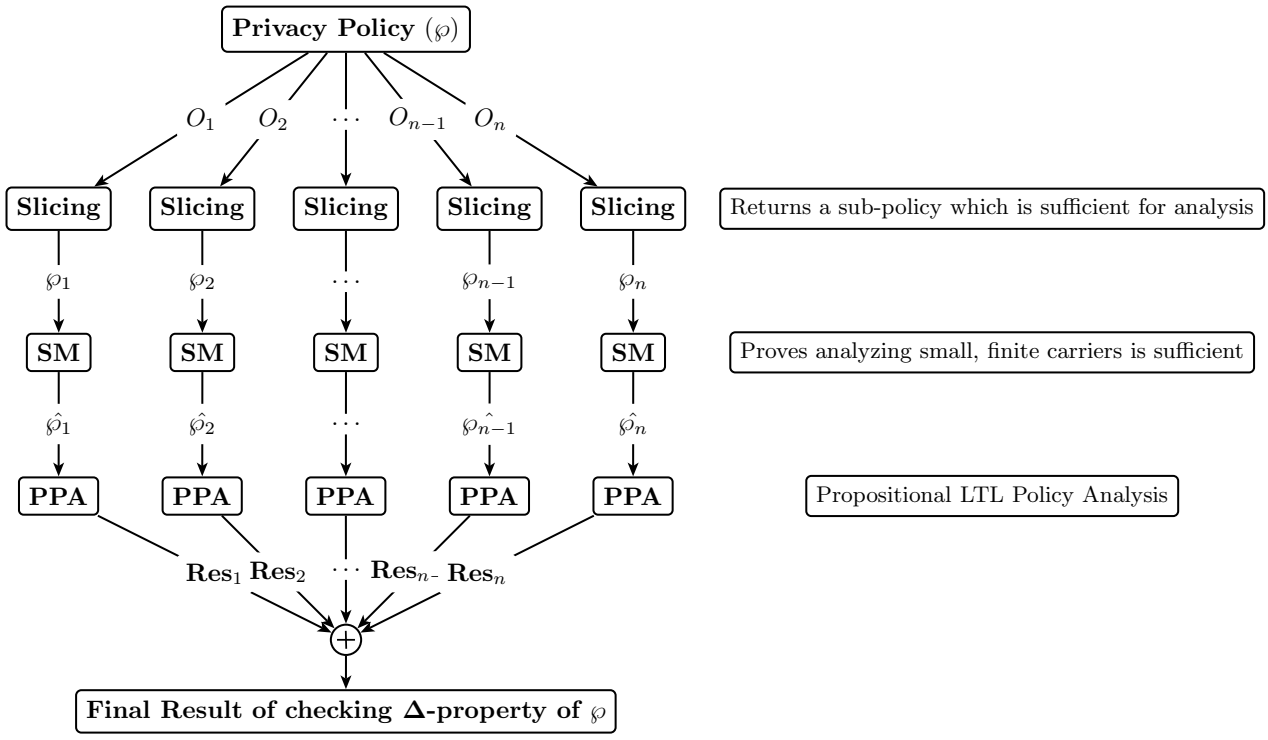

Figure 5.11: Overview of the policy analysis technique

policy is in FOTL, we cannot directly use this technique to check the Δ-property. The following subsection details our sound, semi-automated approach to the problem.

5.3 Analysis Technique for Checking the Δ-property

Recall that deciding whether a policy $\wp$ has the Δ-property is in general undecidable. However, this result is not discouraging as we can develop a sound, semi-automated technique which can decide whether $\wp$ has the Δ-property in many practical cases. We now present our sound, semi-automated analysis technique to check whether a policy $\wp$ has the Δ-property. Our analysis technique consists of the following three steps (see Figure 5.11).

1. Privacy policy slicing.

2. Developing a small model theorem [46].

3. pLTL policy analysis.

By Corollary 59, a policy $\wp$ can violate the Δ-property only if $\wp$ allows a weakly compliant action to incur an unsatisfiable obligation. When obligations do not interact with each other, we can analyze permissibility of each of the obligations independently. To this end, we introduce *privacy policy slicing* which decomposes the policy to a sub-policy which only contains the norms that can potentially influence the permissibility of the obligation in question (step 1). In practice, the sliced policy is significantly smaller but deciding whether the sliced policy has the Δ-property might still be undecidable due to the fact that the policy language of the slice is still a restricted non-monadic fragment of FOTL which has been shown to be highly undecidable [64].

The next step (step 2) of our analysis addresses the undecidability of the sub-policies obtained from the previous step. Step 2 requires developing a small model theorem [46]. It proves that finite elements of the carriers of the policy are sufficient to simulate all possible behaviors necessary to prove the Δ-property of the policy. Then we can rewrite the universal and the existential quantifiers as finite conjunctions and disjunctions, obtaining a pLTL policy which can be analyzed. We show a template of the small model theorem which must be instantiated for a specific policy analysis problem instance and whose proof is necessary to check whether a policy has the Δ-property. We want to emphasize that step 2 will be specific for each policy analysis instance and relies heavily on human assistance and domain knowledge of the assisting human.

Finally, we analyze the pLTL policy $\wp$ (step 3). We obtain a CTL$^*_{lp}$ formula from $\wp$ (see Figure 5.1). We then model check $\mathbb{M}_\wp$ with respect to the CTL$^*_{lp}$ specification. If $\mathbb{M}_\wp$ satisfies the CTL$^*_{lp}$ specification, then we can say that $\wp$ has the Δ-property. Note that while there are known algorithms for CTL$^*_{lp}$ model checking, *e.g.*, Kupferman *et al.* [77], there exists no tool support for this. Currently, we rely on the approach proposed by Barth *et al.* [11]. Their algorithm begins by building a *tableau* [91] (with Büchi accepting condition [22]) from the pLTL formula representing the privacy policy. Then it checks to see whether all the reachable states from the *initial states* can reach a strongly connected component containing at least an *accepting state*. If this is the case, then the privacy policy specified in pLTL has the Δ-property. Note that the size of the tableau is exponential in the pLTL policy formula length. Thus, their algorithm has the complexity of

EXPSPACE in the policy formula length.

5.3.1 Assumptions and Limitations

Recall that to check whether a policy $\wp$ specified in ***FOPSL*** has the Δ-property is undecidable in general. To mitigate this undecidability, we make some assumptions based on which we develop a sound, semi-automatic technique to check whether $\wp$ has the Δ-property. We discuss the assumptions and limitations of our approach just below.

The first assumption we make is that obligations specified in the policy $\wp$ cannot interact with each other. To be more precise, we assume that one obligation cannot incur another obligation (no cascading obligations). Moreover, we also assume that obligations are not mutually exclusive. Consider a policy according to which two obligations o_1 and o_2 are incurred. However, the policy has the restriction that if one performs obligation o_1, then o_2 is not permitted and vice versa. We assume such case does not occur in the context of our policy analysis. We have verified that our interpretation of the HIPAA privacy rule satisfies this constraint. The implication of this constraint is that to check the permissibility of obligations, we can consider each obligation separately. Moreover, this enables us to reduce the original problem of checking whether a policy has the Δ-property into multiple smaller problems each of which can be analyzed in isolation and in parallel.

We only assume positive obligations and also assume that formulas under a $\Diamond$ temporal operator are non-temporal positive formulas. This enables us to syntactically replace all the sub-formulas whose outer most operator is $\Diamond$ with logical true to extract the present requirements of the policy. Moreover, negative obligations can be expressed with past temporal formulas. For instance, consider the policy rule which requires that whenever the action a is performed then in the future the action b cannot be performed. This can be intuitively written as $a \rightarrow \Box(\neg b))$. We can then rewrite this policy rule to $b \rightarrow \diamondminus(\neg a))$ specifying the same requirement. This rewritten policy states that action b can be performed if action a has not been performed in the past. As we can see both policies accept the same sets of traces. We also do not allow the $\mathbb{U}$ temporal operator in ***FOPSL***.

We also disallow the $\ominus$ temporal operators in ***FOPSL***. Recall that we have two cases in which

the policy $\wp$ might not possess the Δ-property. The first case occurs when taking a weakly compliant action can lead to a state from which it is not possible to take an unbounded number of weakly-compliant valid transitions (no infinite weakly compliant extension). The second case in which a weakly compliant action for $\wp$ is not strongly compliant for $\wp$ is when that action incurs a future obligation which cannot be met. The implication of the restriction of disallowing the $\ominus$ operator is that for policies of our form violation case 1 cannot happen. Thus, we can just consider violation case 2, which occurs when a weakly compliant action incurs an unsatisfiable obligation. We also consider the non-strict version of the temporal operators. The semantics of the strict version of the temporal operators do not take the current state into account when reasoning about whether a trace satisfies a formula. If we allow the strict version of the temporal operators our policies can demonstrate the first violation case.

Obligations in privacy policies may have a deadline associated with them. For instance, §164.524 requires that when an individual requests to access her own *PHI*, the covered entity is obligated to provide the individual with access within 30 days (but not more than 60 days). One cannot express obligation deadlines in ***FOPSL***. However, extending ***FOPSL*** to express obligation deadlines is feasible [13]. However, even then we cannot take obligation deadlines into account, as this would require to extend FO-CTL$^*_{lp}$ with support of deadlines. No such logic and associated model checking algorithms exist in the literature. That being said, it is possible to extend our policy analysis result for a policy without obligation deadlines to the policy analysis result for a policy which has only deadlines in obligations. Consider a policy with the following two rules. (1) When a patient's parents request to access the patient's *PHI*, then the doctor is obligated to give the patient's parents access to the *PHI*. (2) The doctor can disclose a patient's *PHI* to anybody if the patient gave an authorization to do so. Now according to our analysis this policy has the Δ-property as the incurred obligation of the doctor (from rule 1) can be fulfilled if the doctor received a patient's authorization (rule 2). Now let us add a deadline of 10 days for the doctor's obligation. Even with the deadline, it is possible for the doctor to discharge the obligation in 10 days if the patient sends the authorization in 10 days.

In our model, we assume the role state of the users is an input to the model. In our model, we assume that roles of individuals do not change over time. Consider a situation where an obligation o_1 is incurred by a user u_1, and to fulfill that obligation, u_1 requires the role r_1 which he currently possesses. Now, if the role r_1 is revoked from u_1 (*e.g.*, u_1 is fired from the job), he cannot fulfill the obligation. From an operational point view of the system, one possibility of handling the revocation of privileges is that the obligation can be reassigned to a new user with appropriate privileges. However, modeling this in a formal verification setting is not trivial and this might cause the small model theorem to not exist as considering a finite number of users might not suffice. Moreover, we assume all the different carrier sets can be unbounded but fixed over time. If we allow the carrier to change over time and also allow the roles of the users to change then we can have a situation where users keep getting fired and new users are added to replace them. In that case, we cannot show the existence of a small model theorem in a straightforward fashion. Note that if we have the situation that whenever a user's role is revoked, there always exists another user who has the appropriate authorization to whom the obligation can be reassigned, then our analysis result considering roles to be fixed can be generalized for a system where roles can change over time. We also assume an individual cannot have conflicting roles. For instance, a user cannot be both patient and doctor at the same time. We also assume that the role hierarchy is an input to the model.

We also assume in each point of time, there is only one transmission event or send event. In a distributed system setting, there could be a situation where multiple send events have the same time stamp but we assume the clock granularity is fine enough to totally order them based on the time stamp.

We do not have any general result for developing small model theorems for policies written in ***FOPSL***. To be more precise, it is not clear whether all policies written in ***FOPSL*** have a small model theorem. Currently we have an ad-hoc approach for which we use domain specific knowledge and abstractions.

We disallow the $\neq$ operator or any predicate simulating it. More precisely, the policy in ques-

tion might have the $\neq$ operator as long as the policy slices we have to consider do not have it. In our case study, we see that the policy slices of the HIPAA privacy policy satisfy this constraint. The implication of this constraint is that when a policy does not have the $\neq$ operator, then it cannot differentiate between two elements of the carrier. As a result of this, we are able to develop a small model theorem. This is explained in more details later.

As mentioned before, there are two approaches of checking whether a policy specified in pLTL has the Δ-property. The first approach is based on infinite word automata called "tableaus" and is proposed by Barth *et al.* [11]. The other approach is proposed by us and it is based on CTL*$_{lp}$ model checking [77]. We do not have access to a CTL*$_{lp}$ model checker and to show efficacy we currently use the tableau-based approach.

We assume that users are diligent and they will attempt to fulfill their obligations when incurred. We also assume that the environmental agents will collaborate with the users in the system to fulfill the obligation of the users. To relax this assumption, we have to strengthen the definition of strong compliance. We discuss this further in Chapter 8.

We also assume messages that do not contain any individually identifiable information are not mandated by the privacy policy. We also assume that privacy notices do not contain any individually identifiable information and thus sending privacy notices is not mandated by the privacy regulations in question.

We also assume in ***FOPSL*** that the following predefined predicates inrole, send, contains, for-purpose, purpose, and in have a fixed interpretation. Other regulation-specific predicates do not have a fixed interpretation. We also assume that the policy is cycle-free. More precisely, whether a send event is allowed does not depend on whether the same event is allowed. Moreover, we also assume that we do not have any predicates, excluding the predefined predicate send, whose semantics depend on the permissibility of a send event containing *PHI* of an individual. Without these assumptions it is difficult to develop a sound slicing algorithm.

5.3.2 Privacy Policy Slicing

For our policy analysis, the privacy policy size is a bottleneck. As it turns out, ***FOPSL*** allows us to use a divide-and-conquer approach for verification. The benefit of decomposition is that for a single obligation, potentially not all norms of the policy are necessary for analysis, reducing the policy size to be analyzed.

Based on this intuition, we now introduce the notion of *adequate policy*. A policy $\wp_A$ is an adequate policy with respect to a policy $\wp$ and an obligation O_k in $\wp$, where the norm set of $\wp_A$ is a subset of the norm set of $\wp$, if it is sufficient to check whether all pending occurrences of obligation O_k can be fulfilled in $\wp_A$ to make a decision about whether all pending occurrences of obligation O_k can be fulfilled according to $\wp$. Thus, it is sound to decide that all pending occurrences of obligation O_k can be fulfilled by conforming to $\wp$ if we know that all pending occurrences of obligation O_k can be fulfilled in a conforming fashion with respect to $\wp_A$. Moreover, the advantage is that the size of an adequate policy $\wp_A$ with respect to a policy $\wp$ and an obligation O_k in $\wp$ is expected to be smaller than the original policy $\wp$.

Definition 61 (Adequate Policy)**.** *Given a privacy policy $\wp$ and an obligation O_k mentioned in it, we call another policy $\wp_A$ an adequate policy with respect to $\wp$ and O_k, if the set of norms of $\wp_A$ is a subset of the set of norms of policy $\wp$, and the following property holds:*

For any possible finite trace σ_f, any state s, environment η, such that $s, \eta \models T$, where T is an action which incurs the obligation O_k with respect to both $\wp$ and $\wp_A$, if $\sigma_f \cdot s, |\sigma_f|, \eta \models \boxminus weak(\wp)$ and $\sigma_f \cdot s, |\sigma_f|, \eta \models \boxminus weak(\wp_s)$, then there exists an infinite extension σ_i such that $\sigma_f \cdot s \cdot \sigma_i \models \wp_A$, only if there exists an infinite extension σ_j such that $\sigma_f \cdot s \cdot \sigma_j \models \wp$.

The following theorem (Theorem 62) precisely states that the policy analysis results of checking the Δ-property of the adequate policies with respect to a privacy policy $\wp$ and obligations in $\wp$ can be composed together to get the policy analysis results of checking the Δ-property of the original policy.

Theorem 62. *For a policy $\wp$, if for all obligations O_k in $\wp$, $\mathbb{M}_{\wp_{s(O_k)}} \models \delta(\wp_{s(O_k)})$ where $\wp_{s(O_k)}$ is an adequate policy with respect to O_k and $\wp$, then $\mathbb{M}_{\wp} \models \delta(\wp)$.*

Proof. From the definition of Δ-property, a policy $\wp$ has the Δ-property, if the following holds:

$$\forall \sigma_f . \forall \eta . \sigma_f, |\sigma_f| - 1, \eta \models \boxminus weak(\wp) \rightarrow \exists \sigma_i . \sigma_f \cdot \sigma_i \models \wp$$

Using this definition of Δ-property, it is sufficient to show that the following holds:

$$\forall k. \left(\forall \sigma_f^k . \forall \eta' . \sigma_f^k, |\sigma_f^k| - 1, \eta' \models \boxminus weak(\wp_{s(O_k)}) \rightarrow \exists \sigma_i^k . \sigma_f^k \cdot \sigma_i^k \models \wp_{s(O_k)} \right) \Longrightarrow$$
$$\forall \sigma_f . \forall \eta . \sigma_f, |\sigma_f| - 1, \eta \models \boxminus weak(\wp) \rightarrow \exists \sigma_i . \sigma \cdot \sigma_i \models \wp$$

Now let us consider that for each obligation O_k we have a specific send event (non-temporal formula) T_k which incurs the obligation O_k according to the policy $\wp$. Now take any arbitrary σ_f and η such that $\sigma_f, |\sigma_f| - 1, \eta \models \boxminus weak(\wp)$ and without loss of generality additionally consider that σ_f contains the triggering action T_k for all obligations O_k. From σ_f we will construct each σ_f^k and η' such that $\sigma_f^k \cdot s_{t_k}, |\sigma_f^k|, \eta' \models \boxminus weak(\wp_{s(O_k)})$ where $s_{t_k} \models T_k$. We will show how to construct this for a specific k. Find the state in σ_f in which T_k happens. We discard all the remaining states including that state where T_K happens (but save all the predicate valuations of the state in which T_k occurred as it will be necessary later to construct η' for the state s_{t_k} just below). The resulting trace σ_f^k is a strict prefix of σ_f. We then replace all the send events in σ_f^k that are not consistent with any send events in $\wp_{s(O_k)}$ by a send event that does not have any attribute in it and thus trivially being allowed by $\wp_{s(O_k)}$. We then change η to η' by removing all predicates and their associated valuations that do not appear in $\wp_{s(O_k)}$ and copying the consistent predicate valuations for the original state of σ_f in which T_k occurred. The resulting finite trace $\sigma_f^k \cdot s_{t_k}$ and η' satisfy the following: $\sigma_f^k \cdot s_{t_k}, |\sigma_f^k|, \eta' \models \boxminus weak(\wp_{s(O_k)})$. This is intuitively due to the fact that we retain all the necessary predicate valuations, we retain all the actions that can influence the satisfiability of $\boxminus weak(\wp_{s(O_k)})$, and the sends not containing any attribute of an individual are trivially allowed

by the policy (see the proof of Theorem 70 for a similar argument).

Now from the fact that for all obligations O_k in $\wp$, $\mathbb{M}_{\wp_{s(O_k)}} \models \delta(\wp_{s(O_k)})$ holds, we get for each $\sigma_f^k \cdot s_{t_k}$ an infinite extension σ_i^k such that $\sigma_f^k \cdot s_{t_k} \cdot \sigma_i^k \models \wp_{s(O_k)}$. Now by Theorem 70, we have that $\sigma_f^k \cdot s_{t_k} \cdot \sigma_i^k \models \wp$. We will now show how to construct the σ_i from all the σ_i^ks such that $\sigma_f \cdot \sigma_i \models \wp$. To construct σ_i from all the σ_i^k by interleaving the states of each of the σ_i^k and replicating the predicate valuations in a way that in each state a predicate valuation and its negation are not true at the same time. Based on the constraint that obligations do not interact with each other, the semantics of predicates (excluding the predefined predicate send) do not depend on the permissibility of a send event containing an attribute of an individual, and predicate valuations are maintained similarly, gives us that the interleaved σ_i satisfies $\sigma_f \cdot \sigma_i \models \wp$ completing our proof. □

The notion of adequate policy enables us to run our policy analysis of checking the Δ-property on smaller policies, the results of which can be composed to get the policy analysis results of checking the Δ-property of the original policy. Due to the syntactic restrictions of ***FOPSL***, we can syntactically over-approximate adequate policies with respect to a privacy policy $\wp$ and obligations O_k in $\wp$. To this end, we introduce *privacy policy slicing* analogous to *program slicing* [125]. Slicing decomposes the privacy policy with respect to an obligation. We show that the resulting sliced policy is also an adequate policy.

The requirements of privacy policy slicing, which make it interesting for our analysis, are the following.

(I) The slicing preserves the Δ-property of the original policy with respect to the slicing criterion.

(II) The analysis results on the sliced policies can be composed to verify that the Δ-property holds for the original policy.

Slicing a privacy policy with respect to a slicing criterion collects all the norms of a policy which the said criterion depends on. The *slicing criterion* (P) is a non-temporal formula and it represents a set of send events. P has the following form: $\text{send}(p_1, p_2, m) \wedge \text{contains}(m, q, t) \wedge \text{for-purpose}(m, u)$

$$
\begin{aligned}
\wp ::= \Box\, (\forall p_1, p_2, q : P. \forall m : M. \forall t : T. \forall u : U.\\
\quad \text{send}(p_1, p_2, m) \wedge \text{contains}(m, q, t) \wedge \text{for-purpose}(m, u) \longrightarrow\\
\qquad \bigvee_i \left((\mathbb{C}_{\text{sender}} \wedge \mathbb{C}_{\text{receiver}} \wedge \mathbb{C}_{\text{subject}} \wedge \mathbb{C}_{\text{attribute}} \wedge \mathbb{C}_{\text{purpose}} \wedge \psi \wedge \beta) \vee \psi_{exception} \right)\\
\qquad \bigwedge_j \left(\mathbb{C}_{\text{sender}} \wedge \mathbb{C}_{\text{receiver}} \wedge \mathbb{C}_{\text{subject}} \wedge \mathbb{C}_{\text{attribute}} \wedge \mathbb{C}_{\text{purpose}} \wedge \psi \rightarrow \left(\boxed{\chi} \vee \psi_{exception} \right) \right)\\
\quad)
\end{aligned}
$$

Figure 5.12: Different kinds of send events and their position in the policy, *Blue*=regulatory, *Green*=conditional, *Red*=obligatory

$\wedge \mathbb{C}_{\text{sender}} \wedge \mathbb{C}_{\text{receiver}} \wedge \mathbb{C}_{\text{subject}} \wedge \mathbb{C}_{\text{attribute}} \wedge \mathbb{C}_{\text{purpose}}$. Note that one or more conjuncts can be missing in P when they are trivially true. Before precisely defining privacy policy slicing, we introduce some key notions first.

5.3.2.1 Types of send Events

The first notion necessary for specifying policy slicing are the *types of send events*. We distinguish between three types of send actions: *regulatory*, *conditional*, and *obligatory*. We base our distinction on where they appear in the policy.

Definition 63 (Conditional Send)**.** *A send event, which has the form of a slicing criterion P, is called* conditional *with respect to a norm if it appears as a sub-formula in one of the following places: (1) In* ψ *or* $\psi_{exception}$ *portion of the positive norms. (2) In* ψ *portion of the negative norms' antecedent. (3) In* ψ *portion of* χ *in the negative norms' consequent. (4) In* $\psi_{exception}$ *portion of* χ *in the negative norms' consequent.*

Definition 64 (Obligatory Send)**.** *A send event is called* obligatory *with respect to a norm if it appears as a sub-formula in one of the following places: (1) In* β *portion of the positive norms. (2) In* β *portion of* χ *in the negative norm's consequent.*

Definition 65 (Regulatory Send)**.** *A send event is called* regulatory *or the* target *send event with respect to a norm if it is the target send event that the norms refers to (or, applies to).*

An example of the regulatory/target send event is given just below. Consider the following very simple policy with only one negative norm:

$$\Box\,(\forall p_1, p_2, q : P.\forall m : M.\forall t : T.\forall u : U.$$
$$\text{send}(p_1, p_2, m) \wedge \text{contains}(m, q, t) \wedge \text{for-purpose}(m, u) \longrightarrow$$
$$(\text{inrole}(p_1, \textit{covered-entity}) \wedge \text{inrole}(q, \textit{individual}) \wedge \text{in}(t, \textit{PHI}) \rightarrow$$
$$\text{believesMinimumNecessaryForPurpose}(p_1, p_2, q, t, u))\ )$$

The only regulated/target send of the above policy is as following.

$$\text{send}(p_1, p_2, m) \wedge \text{contains}(m, q, t) \wedge \text{for-purpose}(m, u) \wedge \text{inrole}(p_1, \textit{covered-entity})$$
$$\wedge\, \text{inrole}(q, \textit{individual}) \wedge \text{in}(t, \textit{PHI})$$

In Figure 5.12, we show the different kinds of send events we consider and their position in the policy. Any send event that appears in the formula marked green is conditional send event. On the contrary, any send event that appears in the formula marked red is obligatory send event. Any send event that appears in the boxed formula (χ) can be either conditional or obligatory send event depending on the outer most temporal connective. If the temporal connective is $\Diamond$, then the send event in question is an obligatory send event. If the top level temporal connective is a past temporal operator or a logical operator, then the send event that appears in χ is a conditional send event. The send event in head of the rule, marked in blue, is the regulatory or target send event.

5.3.2.2 Consistency

The goal of the notion of consistency is to statically decide given a send, which of the norms will be applicable (restricts or allows) for that send. Note that as we define consistency statically and to preserve soundness, we actually over-approximate the most precise consistency relation. According to our definition of consistency and in turn using it to check whether a norm applies

(restricts or allows) to the send in question, it can be the case that, our definition decide that a norm is applicable to a send event but at runtime (with more information) it might not actually be applicable. However, the converse case where the norm is applicable to a send in runtime and our definition decides it is not applicable, cannot happen for our formalization of consistency. This is what we mean by over-approximating the most precise consistency relation. We say a send Q_1 is *consistent* with another send Q_2 if all constraints (*e.g.*, $\mathbb{C}_{\text{sender}}$, *etc.*) in Q_1 are consistent with the constraints in Q_2. Q_1 and Q_2 have the same form as P and can contain free variables. One way to differentiate between the different sends are the constraints (*i.e.*, constraints on the sender role, *etc.*) on their free variables. Consistency is necessary for two reasons: first, Q_1 and Q_2 can contain free variables, which might not match the naming convention of each other, and second, it is admissible that one constraint subsumes another. For addressing the first issue, we rename the constraints to follow the same naming convention. We now formalize consistency (denoted by $\leftrightsquigarrow$) in the following way.

- $\text{inrole}(p,\hat{r}) \leftrightsquigarrow \text{inrole}(p,\bar{r})$ if and only if $\hat{r} = \bar{r}$, $\hat{r}$ is a specialization of $\bar{r}$, or $\bar{r}$ is a specialization of $\hat{r}$. For instance, $\text{inrole}(p,\text{doctor}) \leftrightsquigarrow \text{inrole}(p,\text{psychiatrist})$.

- $\neg\text{inrole}(p,\hat{r}) \leftrightsquigarrow \text{inrole}(p,\bar{r})$ if and only if $\text{inrole}(p,\hat{r}) \not\leftrightsquigarrow \text{inrole}(p,\bar{r})$.

- $\text{in}(t,\hat{t}) \leftrightsquigarrow \text{in}(t,\bar{t})$ if and only if $\hat{t} = \bar{t}$ or there exists an attribute t_1 such that t_1 can be calculated from both $\hat{t}$ and $\bar{t}$. For instance, $\text{in}(t,PHI) \leftrightsquigarrow \text{in}(t,psych\text{-}notes)$ as the attribute "*diagnosis*" can be calculated from both *PHI* (protected health information) and *psych-notes*.

- $\neg\text{in}(t,\hat{t}) \leftrightsquigarrow \text{in}(t,\bar{t})$ if and only if $\text{in}(t,\hat{t}) \not\leftrightsquigarrow \text{in}(t,\bar{t})$.

- $\text{for-purpose}(m,u_1) \leftrightsquigarrow \text{for-purpose}(m,u_2)$ if and only if (1) u_1 and u_2 are both constants then $u_1 = u_2$ and (2) either u_1 or u_2 are variables. We handle negation in the similar way similarly as other predicates above.

- $\text{contains}(m,q,t_1) \leftrightsquigarrow \text{contains}(m,q,t_2)$ if and only if (1) t_1 and t_2 are both constants and one of the following holds: (a) $t_1 = t_2$ or (b) there exists an attribute $\hat{t}$ such that $\hat{t}$ can be calculated

from both t_1 and t_2, and (2) either t_1 or t_2 are variables. We handle negation in the similar way similarly as other predicates above.

- Two send predicates (negated or not) are always consistent with each other.
- $\text{purpose}(u, \hat{u}) \leftrightsquigarrow \text{purpose}(u, \bar{u})$ if and only if $\hat{u} = \bar{u}$.
- $\neg\text{purpose}(u, \hat{u}) \leftrightsquigarrow \text{purpose}(u, \bar{u})$ if and only if $\text{purpose}(u, \hat{u}) \not\leftrightsquigarrow \text{purpose}(u, \bar{u})$.
- $\mathbb{C}_i^x \leftrightsquigarrow \mathbb{C}_i^y$ if and only if there exists an atomic formula a_x of $\mathbb{C}_i^x$ that is consistent with any atomic formula a_y of $\mathbb{C}_i^y$, where $i \in \{\text{sender}, \text{receiver}, \text{subject}, \text{attribute}, \text{purpose}\}$. For instance, $(\text{inrole}(p_1, \text{doctor}) \wedge \text{inrole}(p_1, \text{resident}))$ is consistent with $(\text{inrole}(p_1, \text{psychiatrist}) \vee \text{inrole}(p_1, \text{secretary}))$ as $\text{inrole}(p_1, \text{doctor}) \leftrightsquigarrow \text{inrole}(p_1, \text{psychiatrist})$. Note that we are conservatively over approximating consistency by only looking at atomic predicates individually while ignoring their logical connections.

We assume that an empty constraint is consistent with any constraint. We can now inductively check whether two send events are consistent.

5.3.2.3 Dependency

The next notion we need for defining privacy policy slicing is called the norm *dependencies*.

Definition 66 (Positive Dependency)**.** *A norm* ϕ_1 positively depends *on norm* ϕ_2*, if one of the conditional sends of* ϕ_1 *is* consistent *with the regulatory send of* ϕ_2*. Intuitively, this represents that the permissibility of a send regulated by* ϕ_1 *can be influenced by the occurrence of sends influenced by sends regulated by* ϕ_2*.*

Definition 67 (Negative Dependency)**.** *A norm* ϕ_1 negatively depends *on norm* ϕ_2*, if one of the obligatory sends of* ϕ_1 *is consistent with the regulatory send of* ϕ_2*. Roughly, this represents that the permissibility of sends that might be obligated by* ϕ_1 *might be regulated by* ϕ_2*.*

Definition 68 (Prospect-Dependency)**.** *A norm* ϕ_1 *has an* prospect-dependency *on norm* ϕ_2*, if the regulating send of* ϕ_1 *is consistent with the regulatory send of* ϕ_2*. This signifies that both norms* ϕ_1 *and* ϕ_2 *allow the same send based on possibly different conditions.*

We say norm ϕ_1 *depends* on norm ϕ_2 if ϕ_1 has a positive-, a negative-, or a prospect-dependency on ϕ_2. Note that as the definition of dependency uses the notion of consistency, our definition of norm dependency, to preserve soundness, over-approximates the most precise dependence relation between norms.

Recall that we use the notion of *privacy policy slice* to syntactically over-approximate the adequate policy of a given privacy policy $\wp$ and an obligation O_k. This is possible due to restrictions on ***FOPSL***. We use the obligation O_k as a slicing criterion of our privacy policy slicing. As we have introduced all the necessary notions, we can now precisely define privacy policy slicing.

Definition 69 (Slice of a Privacy Policy)**.** *Given a privacy policy* $\wp$ *with the norm set* ϕ*, a slicing criterion P,* $\wp_{s(P)}$ *is a slice of* $\wp$ *with respect to P if it satisfies the following.*

1. $\phi_{s(P)} \subseteq \phi$ *where* $\phi_{s(P)}$ *denotes the norm set of* $\wp_{s(P)}$*.*
2. $\phi_P \subseteq \phi_{s(P)}$ *where* ϕ_P *represents the set of all norms where P appears as an obligatory send.*
3. $\phi^* \subseteq \phi_{s(P)}$ *where* ϕ^* *is the transitive closure of the dependence relation on* ϕ_P*.*

These definitions of slicing, dependency, and consistency were carefully chosen so that the privacy policy slicing satisfies the requirements (I) and (II) mentioned above. To show that the slicing procedure satisfies both the requirements (I) and (II), we have the following theorems. The first (Theorem 70) formalizes the requirement that the slicing procedure preserves the Δ-property of the original policy with respect to the slicing criterion. Note that from the following theorem it follows that a policy slice $\wp_s$ with respect to a policy $\wp$ and obligation O_k (slicing criterion), is also an adequate policy with respect to $\wp$ and obligation O_k.

Theorem 70. *For a policy* $\wp$ *and a slicing criterion P, the resulting sliced policy* $\wp_{s(P)}$ *satisfies the following. For any possible finite trace* σ_f*, any state s, environment* η*, such that* $s, \eta \models T$*, where*

T is an action which incurs the obligation P with respect to both $\wp$ and $\wp_{s(P)}$, if $\sigma_f \cdot s, |\sigma_f|, \eta \models \boxminus weak(\wp)$ and $\sigma_f \cdot s, |\sigma_f|, \eta \models \boxminus weak(\wp_s)$, then there exists an infinite extension σ_i such that $\sigma_f \cdot s \cdot \sigma_i \models \wp_{s(P)}$, if and only if there exists an infinite extension σ_j such that $\sigma_f \cdot s \cdot \sigma_j \models \wp$.

Proof. Assume $\wp$, P, σ_f, s and η such that the requirements above are satisfied. We start with the forward direction $\Rightarrow$.

By assumption there exists an infinite extension σ_i such that $\sigma_f \cdot s \cdot \sigma_i \models \wp_{s(P)}$. By definition, this means that $\forall k.\sigma_f \cdot s \cdot \sigma_i, k, \eta \models \forall p_1, p_2, q : P.\forall m : M.\forall t : T.\forall u : U.\text{send}(p_1, p_2, m) \wedge \text{contains}(m, q, t) \wedge \text{for-purpose}(m, u) \rightarrow (\bigvee_i \phi_i^{s+} \wedge \bigwedge_j \phi_j^{s-})$. Now let $\sigma_j = \sigma_i$. We have to show that $\forall k.\sigma_f \cdot s \cdot \sigma_i, k, \eta \models \forall p_1, p_2, q : P.\forall m : M.\forall t : T.\forall u : U.\text{send}(p_1, p_2, m) \wedge \text{contains}(m, q, t) \wedge \text{for-purpose}(m, u) \rightarrow (\bigvee_i \phi_i^{+} \wedge \bigwedge_j \phi_j^{-})$. Take an arbitrary k. Assume p_1, p_2, q, m, t and u such that the antecedent is satisfied. We have to show that at least one positive norm ϕ^+ and all negative norms ϕ^- are satisfied. If the set of principals, attribute and purpose satisfy the antecedent for the original policy, trace and index k, the same holds for the sliced policy. Thus, there is at least one positive norm ϕ^{s+} that is satisfied. Since all sliced norms are in the original policy, this positive norm is part of the positive norms of the original policy. Thus with $\phi^+ = \phi^{s+}$ we have found our allowing norm. Similarly, all norms ϕ^{s-} are negative norms of the original policy and satisfied. It remains to show that the remaining negative norms are trivially satisfied. We proceed with a proof by contradiction.

Assume a negative norm $\mathbb{C} \wedge \psi \rightarrow (\chi \vee \psi_{exc})$, not part of the sliced policy, that is false, that is, its antecedent is satisfied, but the consequent is not. If the antecedent is satisfied, then the restriction $\mathbb{C}$ applies to the send event at index k. Now compare $\mathbb{C}$ to $\mathbb{C}^+$, the restriction of the regulatory send of the positive norm ϕ^{s+}. For each restriction category c, either $\mathbb{C}_c$ is empty, $\mathbb{C}_c^+$ is empty, or both are satisfied by the send. In the first two cases the restrictions are immediately consistent. In the last case we inspect the category. The following explains the simplified case of single predicates. More complex formulas follow immediately by inspection of the simple conjunctive and disjunctive nature of restrictions. For restrictions of role, for both to apply they either have to be equivalent or one has to be a descendant of the other in the role hierarchy. Then

they are consistent. For restrictions of attributes, for both to apply they either have to be equivalent or computationally related. Then they are consistent. For restrictions of purposes, they must be the same purpose. Then they are consistent.

In summary we have shown that the restrictions of the negative norm and the positive norm are consistent. Thus, the norms are dependent. By Definition 69(3) the negative norm must be part of the slice, which is a contradiction.

We now turn to the backward direction $\Leftarrow$. Let $\sigma_i = \lceil \sigma_j \rceil_{\wp_s}$ be the erasure of σ_j. The erasure is defined as follows:

$$\lceil \sigma \rceil_{\wp_s}(i) = \begin{cases} \sigma(i) & \text{if the send at } \sigma(i) \text{ is applicable to any restriction in } \wp_s \\ \text{send}_{def} & \text{else} \end{cases}$$

In this definition, send_{def} is a send event that is trivially allowed, e.g., one without attributes.

First we prove an auxiliary result. For all indices k and all pure-past and non-temporal formulas φ that contain at most sends contained in $\wp_s$, it holds that for all free-variable valuations ρ, $\sigma_f \cdot s \cdot \sigma_j, k, \eta \models \varphi\, \rho \iff \sigma_f \cdot s \cdot \sigma_i, k, \eta \models \varphi\, \rho$. The proof proceeds by induction on k and the structure of φ. We start with non-temporal formulas. The result holds since φ cannot contain send predicates that apply to erased send events. Conjunction and disjunction follow immediately by inductive hypothesis. Existential quantification is satisfied iff there is a valuation for the parameters such that the contained sub-formula with those parameters substituted is true. By inductive hypothesis, this is the case if and only if the contained sub-formula with the same parameters is satisfied in the erased trace, if and only if the existential holds in the erased trace. A similar argument applies to universal quantification.

For pure-past formulas, proving an iff statement allows us to invoke the inductive hypothesis on a non-satisfied subformula, as needed by the case of negation. The quantifier cases are analogous to the non-temporal case, and conjunction, disjunction and Since follow by inspection of the semantics and inductive hypothesis.

We now return to showing that the erasure satisfies the sliced policy. Analogous to the forward direction, let k be arbitrary and p_1, p_2, q, m, t and u such that the antecedent is satisfied. We can immediately discharge all cases of k where $\sigma_i(k) = \text{send}_{def}$, because it is trivially satisfied. So $\sigma_i(k) = \sigma_j(k) = \text{send}(p_1, p_2, m)$ and some restriction of $\wp_s$ applies to this send. This send is allowed by a positive norm ϕ^+ in $\wp$. Since this norm applies, its restriction must be consistent with the restriction in $\wp_s$ (see the forward direction). Thus, $\phi^{s+} = \phi^+$ is also a positive norm of the sliced policy. We to show that the norm is satisfied in the erased trace.

A positive norm has the form $(\mathbb{C} \wedge \psi \wedge \beta) \vee \psi_{exc}$. First assume the norm was satisfied in σ_j by its first disjunct. Then trivially $\mathbb{C}$ is satisfied at index k in σ_i. By our auxiliary result, ψ is satisfied in the erased trace. Last, β is a conjunction of $\Diamond\mu$. By inspection of the semantics, this holds if for each μ there exists $k' \geq k$ such that μ holds at k' in σ_j. Then by the auxiliary result, μ holds at k' in σ_i. The case that the norm was satisfied by ψ_{exc} follows similarly by the auxiliary result. In summary, the positive norm is satisfied in the erased trace σ_i.

Similarly we can establish that all negative norms of the sliced policy hold. Assume a negative norm $\mathbb{C} \wedge \psi \rightarrow (\chi \vee \psi_{exc})$, where the antecedent is true in σ_i. Then trivially $\mathbb{C}$ is true in σ_j. Furthermore, ψ holds in σ_j by the auxiliary. Thus, the antecedent is true in σ_j. Since every negative norm of the sliced policy is a negative norm of the original policy, the consequent must be satisfied in σ_j. Thus either χ or ψ_{exc} in σ_j. The exception case immediately induces ψ_{exc} in σ_i by auxiliary. The other case follows by induction on the construction of mixed formulas, using the auxiliary after unfolding the definition of obligation formulas similar to the case of positive norms.

In summary, we have shown that at least one positive norm and all negative norms of the sliced policy are satisfied in the erased trace, for any index k. It follows that the extension satisfies the sliced policy. □

The next corollary states that if we are given a privacy policy $\wp$, an obligation O_k in $\wp$, and a privacy policy $\wp_s$ in which $\wp_s$ is a slice of $\wp$ with respect to obligation O_k, then $\wp_s$ is an adequate policy with respect to $\wp$ and O_k.

Corollary 71. *Given a privacy policy $\wp$, an obligation O_k in $\wp$, and a privacy policy $\wp_s$ in which $\wp_s$ is a slice of $\wp$ with respect to obligation O_k, then $\wp_s$ is an* adequate policy *with respect to $\wp$ and O_k.*

Proof. The proof follows from the Theorem 70 and the definition of adequate policy (Definition 61). □

The next corollary states that the Theorem 62 also holds if we replaced the adequate policy of a given policy $\wp$ and an obligation O_k, with privacy policy slice of $\wp$ w.r.t slicing criterion O_k. It basically says all the result of the sliced policies can be merged together to get the policy analysis result of the original policy $\wp$. The proof follows from Theorem 62 and Corollary 71.

Corollary 72. *For a policy $\wp$, if for all obligations O_k in $\wp$, $\mathbb{M}_{\wp_{s(O_k)}} \models \delta(\wp_{s(O_k)})$ where $\wp_{s(O_k)}$ is a privacy policy slice with respect to O_k and $\wp$, then $\mathbb{M}_{\wp} \models \delta(\wp)$.*

The slicing procedure generates the transitive closure by computing dependency in a lazy, by-need fashion. This algorithm is trivially correct, since it follows our theoretical development above. The algorithm is presented just below.

5.3.2.4 Slicing Algorithm

We now present the algorithm for calculating the slice of a privacy policy $\wp$ based on a slicing criterion P. It additionally takes as input the role hierarchy and the attribute computation rules necessary for determining consistency. The slicing procedure calculates the transitive closure of the dependence relation in a lazy, by-need fashion.

Algorithm 5.1 is the main procedure. It takes as input a privacy policy (represented by the set of norms) and an obligatory send event which is used as the slicing criterion, the role hierarchy, the attribute computation rules and returns another sub-policy which influences the obligatory send P. Algorithm 5.2 is a utility procedure used by algorithm 5.1. The procedure AddSend takes as input a send event of form P and checks to see whether the send event has been processed before. If the

send has not been processed before, the procedure adds it to the queue UnderprocessingSends and also to the map ProcessedSends so that it is not processed again.

Algorithm 5.1 Slice(Φ, P)

Input: A privacy policy represented as a set of norms Φ and a slicing criterion P.
Output: returns a sub-policy of the input policy represented as a set of norms Φ_r.

```
1:  /* We assume the following variables to be global */
2:  Φ_r = empty
3:  Queue UnderprocessingSends = empty
4:  Map ProcessedSends = ProcessedNorms =empty
5:  Initialize(P)
6:  while UnderprocessingSends≠empty do
7:      Send Q = UnderprocessingSends.dequeue() ;
8:      for all ϕ ∈ Φ do
9:          if ϕ ∉ProcessedNorms then
10:             if ϕ is of form (R ∧ ψ ∧ β) ∨ ψ_exception and R ⇜⇝ Q then
11:                 Φ_r = Φ_r ∪ ϕ /* add ϕ to the result */
12:                 Insert ϕ to the map processedNorms
13:                 for all Send x ∈ ψ ∨ x ∈ ψ_exception ∨ x ∈ β do
14:                     AddSend(x)
15:             if ϕ is of form R ∧ ψ → χ ∨ ψ_exception and R ⇜⇝ Q then
16:                 Φ_r = Φ_r ∪ ϕ /* add ϕ to the result */
17:                 Insert ϕ to the map processedNorms
18:                 for all Send x ∈ ψ ∨ x ∈ ψ_exception ∨ x ∈ χ do
19:                     AddSend(x)
```

5.3.3 Small Model Theorem (SMT)

Our policies are specified in ***FOPSL***, which is a restricted fragment of FOTL. FOTL is not decidable and the additional restrictions imposed on FOPSL are not enough to make ***FOPSL*** decidable. Specifically, ***FOPSL*** is a non-monadic fragment of FOTL, satisfiability of which is known to be undecidable [64]. The fragment we consider can have more than one free variable for subformulas of the form $\psi_1 \mathcal{S} \psi_2$ (non-*monadic*) [64]. On the other hand, pLTL is decidable. A small model theorem (or, a finite model theorem) [46] will establish that any *behavior of interest* of a policy specified in ***FOPSL*** with infinite carriers can be captured with a small, finite amount of elements from each carrier. The behavior of interest in our case is the behavior necessary to prove the Δ-property of a policy. For example, it might be possible that the full infinite carrier $\mathcal{P}$ of principals

Algorithm 5.2 Initialize(Q)

Input: A send event Q (non-temporal formula)

1: **for all** $\phi \in \Phi$ **do**
2: **if** $\phi \notin$ ProcessedNorms **then**
3: **if** ϕ is of form $(R \wedge \psi \wedge \beta) \vee \psi_{exception}$ and Q appears as an obligatory send in β **then**
4: $\Phi_r = \Phi_r \cup \phi$, Insert ϕ to the map processedNorms
5: AddSend(R)
6: **for all** Send $x \in \psi \vee x \in \psi_{exception} \vee x \in \beta$ **do**
7: AddSend(x)
8: **if** ϕ is of form $R \wedge \psi \rightarrow \chi \vee \psi_{exception}$ and Q appears as an obligatory send in χ **then**
9: $\Phi_r = \Phi_r \cup \phi$, Insert ϕ to the map processedNorms
10: AddSend(R)
11: **for all** Send $x \in \psi \vee x \in \psi_{exception} \vee x \in \chi$ **do**
12: AddSend(x)

can be simulated with a finite amount of representatives, *e.g.*, just one person for each role. In that case, we can rewrite the FOTL policy to a pLTL policy by replacing universal quantifiers with finite conjunctions and existential quantifiers with finite disjunctions, where the quantified variables are instantiated with all carrier elements. Such a theorem would guarantee that the traces that model the resulting propositional policy are sufficiently related to those of the original first-order policy, such that the property we are interested in is preserved by the conversion to a propositional formula.

It is not clear whether there exists a small model theorem for all privacy policies specified in ***FOPSL***. This is what results in the incompleteness in our technique. In the absence of an affirmative finding in this regard, small model theorems must be derived for specific policy instances and specific properties of it (*e.g.*, consistency, Δ-property). Moreover, developing such a small model theorem requires domain specific knowledge, invariants, and abstractions. For instance in HIPAA, whether a covered entity (hospital) can share a patient's (p_a) *PHI* is not dependent on covered entity's interactions with another patient (p_b) where $p_a \neq p_b$. As we will show in section 5.4, it is possible to develop small model theorems for the sliced HIPAA policies we are interested in. The small model theorem necessary for proving the Δ-property of a policy $\wp$ written in ***FOPSL*** will have the following general form. The Theorem 74 in section 5.4 is a concrete example of the following small model theorem template.

Template of Small Model Theorem. A given policy $\wp$ has the Δ-property for every carrier set $\overrightarrow{\mathbb{C}} = \langle \mathbb{C}_1, \mathbb{C}_2, \ldots \rangle$ if and only if there exists a small, finite carrier set $\overrightarrow{\mathbb{C}_S} = \langle \mathbb{C}_{s1}, \mathbb{C}_{s2}, \ldots \rangle$ for which $\wp$ has the Δ-property.

5.4 HIPAA: A Case Study

In this section, we demonstrate the adequacy of our policy analysis techniques by using HIPAA as a case study.

5.4.1 Specification of HIPAA.

We have specified all 84 disclosure related clauses in ***FOPSL*** (see Appendix A). We considered the HIPAA privacy rule in Subpart E of CFR §164. We have 68 positive norms and 8 negative norms. We cannot fully express access related rules found in §164.524 which ensure that an individual gets access to its own *PHI*. However, this clause is not related to disclosure of individually identifiable information (*PHI*). We consider the following sections of HIPAA: §164.502, §164.506, §164.508, §164.510, §164.512, §164.514, and simplified versions of §164.524 and §160.310.

5.4.2 Satisfiability of HIPAA.

Any message that does not contain any individually identifiable information or is initiated by the patient or the environment (except business associates of the covered entity), is not regulated by the HIPAA privacy policy (denoted by $\wp_H$). Thus, messages not containing any *PHI* are trivially allowed by $\wp_H$. This kind of send events would falsify the contains predicate in the antecedent of $\wp_H$ making the implication trivially true. We can thus create an infinite trace, in each step of which, a message of the above kind is transmitted. Such a trace would trivially satisfy $\wp_H$.

5.4.3 Incremental satisfiability of HIPAA.

In ***FOPSL***, we do not allow the $\ominus$ operator. Moreover, $\wp_H$ is trivially satisfiable as discussed just above. By Corollary 58, it follows that $weak(\wp_H)$ is incrementally satisfiable.

5.4.4 Policy slicing algorithm implementation.

Note that obligations do not interact with each other in HIPAA and also HIPAA is trivially satisfiable. Thus, the only way $\wp_H$ can violate the Δ-property is through a weakly compliant action incurring unsatisfiable obligations. We have implemented our slicing algorithm using C++. The complexity of the algorithm is linear in the size of the policy norms and the number of send events that appear in the norms. We have sliced $\wp_H$ with respect to real obligations from HIPAA (§160.310, §164.524). The sliced policy contains 68 norms out of total 76 norms. The slicing of the policy on each of these obligations required less than 480 milliseconds on an Intel Core i7 1.73 GHz machine with 4GB of RAM running Ubuntu 12.04. There are two more obligations in HIPAA that require sending privacy notices. We assume that privacy notices do not contain any individually identifiable information (*PHI*) and thus is not regulated by HIPAA. In such case, we do not take them into account in our policy analysis.

5.4.5 Making the slicing procedure more precise.

To preserve soundness, the privacy policy slicing we define over-approximates the minimal adequate policy. This is due to the fact that our definition of dependence relation does not take into account the condition of the norms (when the condition is not a send event) or any other context information (*e.g.*, textual description of the regulations). However, it is not apparent how to incorporate conditions and other contextual information while defining our dependence relation. One way to get around it is human intervention while checking whether a norm is consistent to a send event. Note that for a send event to be allowed, we need that the send event satisfies one positive and all the applicable negative norms. Based on this intuition, we developed a heuristics in which a human assists our privacy policy slicing algorithm to select couple of applicable positive norms from a list of all applicable positive norms while checking which norms are applicable to a specific send in question. We have implemented our policy slicing algorithm with human intervention support and sliced $\wp_H$ with the obligations in §160.310 and §164.524 of HIPAA. The policy rule

in §164.524 states: when an individual requests for access to her own *PHI*, the covered entity is obligated to give access to the individual. Our sliced policies obtained by using this heuristic for obligation in §160.310 and obligation in §164.524 has 5 norms (2 positive and 3 negative norm) out of total 76 norms (see Figure 5.13 and Figure 5.14). This is a significant reduction in the size of the norms. We also used this heuristic to slice $\wp_H$ with respect to a synthetic obligation (*Synthetic-1*). To this end, we add an additional negative norm to $\wp_H$ that obligates a covered entity to provide access to an individual's parents to the individual's *PHI* when the parents request for it and also add an additional positive norms to $\wp_H$ that allows a parent to send a request to the covered entity to provide access to her child's *PHI*. The sliced policy has 2 positive norm and 6 negative norms out of total 78 norms (see Figure 5.15). However, to preserve soundness of our policy analysis technique, we have to prove that each of these sliced policies, obtained by using our heuristic, are also adequate policies with respect to the policy $\wp_H$ and the respective obligations. The following theorem states that $\wp_{H_{P_1}}$, obtained by using the heuristic to the slicing algorithm, is also an adequate policies with respect to $\wp_H$ and the obligation in §160.310. We show the theorem and its proof for $\wp_{H_{P_1}}$ and all the other two theorems and arguments for their proofs are similar.

Theorem 73 ($\wp_{H_{P_1}}$ is an adequate policy with respect to $\wp_H$ and obligation in §160.310)**.** *For any possible finite trace* σ_f*, any state s, environment* η*, such that* $s, \eta \models \text{send}(p_1, p_2, m) \wedge \text{contains}(m, q, t) \wedge \text{for-purpose}(m, u) \wedge \text{inrole}(p_1, \textit{secretary}) \wedge \text{inrole}(p_2, \textit{covered-entity}) \wedge \text{inrole}(q, \textit{individual}) \wedge \text{in}(t, \textit{PHI}) \wedge \text{purpose}(u, \textit{compliance-investigation})$*, where* $\text{send}(p_1, p_2, m) \wedge \text{contains}(m, q, t) \wedge \text{for-purpose}(m, u) \wedge \text{inrole}(p_1, \textit{secretary}) \wedge \text{inrole}(p_2, \textit{covered-entity}) \wedge \text{inrole}(q, \textit{individual}) \wedge \text{in}(t, \textit{PHI}) \wedge \text{purpose}(u, \textit{compliance-investigation})$ *incurs the obligation in §160.310 with respect to both* $\wp$ *and* $\wp_{s(P)}$*, if* $\sigma_f \cdot s, |\sigma_f|, \eta \models \boxminus \textit{weak}(\wp_H)$ *and* $\sigma_f \cdot s, |\sigma_f|, \eta \models \boxminus \textit{weak}(\wp_{H_{P_1}})$*, then there exists an infinite extension* σ_i *such that* $\sigma_f \cdot s \cdot \sigma_i \models \wp_{H_{P_1}}$*, only if there exists an infinite extension* σ_j *such that* $\sigma_f \cdot s \cdot \sigma_j \models \wp_H$.

Proof. For ease of presentation, let us denote the send that triggers the obligation in §160.310 with T where $T = \text{send}(p_1, p_2, m) \wedge \text{contains}(m, q, t) \wedge \text{for-purpose}(m, u) \wedge \text{inrole}(p_1, \textit{secretary}) \wedge \text{inrole}(p_2, \textit{covered-entity}) \wedge \text{inrole}(q, \textit{individual}) \wedge \text{in}(t, \textit{PHI}) \wedge \text{purpose}(u, \textit{compliance-investigation})$.

Let us take any arbitrary σ_f, η such that $\sigma_f \cdot s, |\sigma_f|, \eta \models \boxminus weak(\wp_H)$ where $s \models T$. From the premise we also know that $\sigma_f \cdot s, |\sigma_f|, \eta \models \boxminus weak(\wp_{H_{P_1}})$.

By assumption there exists an infinite extension σ_i such that $\sigma_f \cdot s \cdot \sigma_i \models \wp_{H_{P_1}}$. By definition, this means that $\forall k.\sigma_f \cdot s \cdot \sigma_i, k, \eta \models \forall p_1, p_2, q : P.\forall m : M.\forall t : T.\forall u : U.\text{send}(p_1, p_2, m) \wedge \text{contains}(m, q, t) \wedge \text{for-purpose}(m, u) \rightarrow (\bigvee_i \phi_i^{+\wp_{H_{P_1}}} \wedge \bigwedge_j \phi_j^{-\wp_{H_{P_1}}})$. Here we use $\phi_i^{+\wp_{H_{P_1}}}$ and $\phi_j^{-\wp_{H_{P_1}}}$ to respectively denote a positive norm and a negative norm of $\wp_{H_{P_1}}$. Now let $\sigma_j = \sigma_i$. We have to show that $\forall k.\sigma_f \cdot s \cdot \sigma_i, k, \eta \models \forall p_1, p_2, q : P.\forall m : M.\forall t : T.\forall u : U.\text{send}(p_1, p_2, m) \wedge \text{contains}(m, q, t) \wedge \text{for-purpose}(m, u) \rightarrow (\bigvee_i \phi_i^+ \wedge \bigwedge_j \phi_j^-)$. Here we use ϕ_i^+ and ϕ_j^- to respectively denote a positive norm and a negative norm of $\wp_H$. Take an arbitrary k. Assume p_1, p_2, q, m, t and u such that the antecedent is satisfied. We have to show that at least one positive norm ϕ^+ and all negative norms ϕ^- are satisfied. If the set of principals, attribute and purpose satisfy the antecedent for the original policy, trace and index k, the same holds for the $\wp_{H_{P_1}}$. Thus, there is at least one positive norm $\phi^{+\wp_{H_{P_1}}}$ that is satisfied. Since all the positive norms of $\wp_{H_{P_1}}$ are in the original policy $\wp_H$, this positive norm is part of the positive norms of the original policy $\wp_H$. Thus with $\phi^+ = \phi^{+\wp_{H_{P_1}}}$ we have found our allowing positive norm for $\wp_H$. Similarly, all norms $\phi^{\wp_{H_{P_1}}-}$ are negative norms of the original policy and satisfied. It remains to show that the remaining negative norms, that are included in $\wp_H$ but not in $\wp_{H_{P_1}}$, are trivially satisfied. For this, we use the manual inspection of the policy $\wp_H$. For all the different possible actions allowed by $\wp_{H_{P_1}}$, we will show that each of them trivially satisfies all the negative norms that are in $\wp_H$ but not in $\wp_{H_{P_1}}$. Consider the action where a principal in the role of the secretary sends a request to a principal in the role covered entity to access the *PHI* of a principal in role individual. By manual inspection of $\wp_H$ we see that this send event does not satisfy the antecedent of any of the negative norms and thus trivially satisfies all the remaining negative norms that are in $\wp_H$ but not in $\wp_{H_{P_1}}$. The remaining action allowed by $\wp_{H_{P_1}}$ is an action where a principal in the role of the covered entity responds to the request of a principal in the role secretary, by sending the *PHI* of a predefined principal in the role of individual. Again by manual inspection of $\wp_H$, we see that this action does not satisfy the antecedent of any negative norm that is included in $\wp_H$ but not included in $\wp_{H_{P_1}}$ and and thus trivially satisfies all the remain-

(**Synthetic Policy Rule**)
$\text{inrole}(p_1, \textit{secretary}) \wedge \text{inrole}(p_2, \textit{covered-entity}) \wedge \text{inrole}(q, \textit{individual}) \wedge \text{in}(t, \textit{PHI}) \wedge$
$\quad \text{purpose}(u, \textit{compliance-investigation}) \wedge \text{request}(p_1, p_2, q, \textit{PHI})$
(§**160**.**310**)
$\text{inrole}(p_1, \textit{secretary}) \wedge \text{inrole}(p_2, \textit{covered-entity}) \wedge \text{inrole}(q, \textit{individual}) \wedge$
$\quad \text{purpose}(u, \textit{compliance-investigation}) \wedge \text{in}(t, \textit{PHI}) \wedge \text{request}(p_1, p_2, q, \textit{PHI}) \longrightarrow$
$\quad\quad \Diamond(\exists m_1 : M.(\text{send}(p_2, p_1, m_1) \wedge \text{inrole}(p_1, \textit{secretary}) \wedge \text{inrole}(p_2, \textit{covered-entity}) \wedge$
$\quad\quad\quad \text{inrole}(q, \textit{individual}) \wedge \text{contains}(m_1, q, \textit{PHI})$
$\quad\quad\quad\quad \wedge \text{for-purpose}(m_1, \textit{compliance-investigation})))$
(§**164**.**502**(**a**)(**2**)(**ii**))
$\text{inrole}(p_1, \textit{covered-entity}) \wedge \text{inrole}(p_2, \textit{secretary}) \wedge$
$\quad\quad\quad\quad \text{inrole}(q, \textit{individual}) \wedge \text{in}(t, \textit{PHI}) \wedge \text{purpose}(u, \textit{compliance-investigation})$
(§**164**.**502**(**b**))
$\text{inrole}(p_1, \textit{covered-entity}) \wedge \text{inrole}(q, \textit{individual}) \wedge \text{in}(t, \textit{PHI}) \longrightarrow$
$\quad \text{believesMinimumNecessaryForPurpose}(p_1, p_2, q, t, u) \vee$
$\quad\quad (\text{inrole}(p_1, \textit{covered-entity}) \wedge \text{inrole}(p_2, \textit{secretary})$
$\quad\quad\quad\quad \wedge \text{inrole}(q, \textit{individual}) \wedge \text{in}(t, \textit{PHI}) \wedge \text{purpose}(u, \textit{compliance-investigation}))$
(§**164**.**508**(**a**)(**2**))
$\text{inrole}(p_1, \textit{covered-entity}) \wedge \text{inrole}(q, \textit{individual}) \wedge \text{in}(t, \textit{psych-notes}) \longrightarrow$
$\quad\quad \text{obtainedAuthorization}(p_1, p_2, q, t, u)$

Figure 5.13: Sliced HIPAA policy ($\wp_{H_{P_1}}$) norms w.r.t the obligation in §160.310 of HIPAA.

ing negative norms that are in $\wp_H$ but not in $\wp_{H_{P_1}}$. This completes our proof of showing that $\wp_{H_{P_1}}$ is an adequate policy with respect to $\wp_H$ and the obligation in §160.310. □

5.4.6 Small Model Theorem.

In the previous section, we have shown the general template of the small model theorem necessary for verifying whether a policy has the Δ-property. We now show a concrete small model theorem. To this end, we first impose some restrictions which enable us to develop a concrete small model theorem of a sliced HIPAA policy.

The first restriction we impose is to disallow the $\neq$ operator or any predicate simulating it.

(**Synthetic Policy Rule**)

$$\text{inrole}(p_1, \textit{individual}) \wedge \text{inrole}(p_2, \textit{covered-entity}) \wedge \text{inrole}(q, \textit{individual}) \wedge$$
$$\text{purpose}(u, \textit{access-request}) \wedge \textit{samePerson}(p_1, q) \wedge \text{request}(p_1, p_2, q, \textit{PHI})$$

(§**164**.**524**)

$$\text{inrole}(p_1, \textit{individual}) \wedge \text{inrole}(p_2, \textit{covered-entity}) \wedge \text{inrole}(q, \textit{individual}) \wedge$$
$$\text{purpose}(u, \textit{access-request}) \wedge \textit{samePerson}(p_1, q) \wedge \text{request}(p_1, p_2, q, \textit{PHI}) \longrightarrow$$
$$\Diamond(\exists m_1 : M.(\text{send}(p_2, p_1, m_1) \wedge \text{contains}(m_1, p_1, \textit{PHI})$$
$$\wedge \text{for-purpose}(m_1, \textit{access-request})))$$

(§**164**.**502**(**a**)(**2**)(**ii**))

$$\text{inrole}(p_1, \textit{covered-entity}) \wedge \text{inrole}(p_2, \textit{individual}) \wedge$$
$$\text{inrole}(q, \textit{individual}) \wedge \text{in}(t, \textit{PHI}) \wedge \textit{samePerson}(p_2, q)$$

(§**164**.**502**(**b**))

$$\text{inrole}(p_1, \textit{covered-entity}) \wedge \text{inrole}(q, \textit{individual}) \wedge \text{in}(t, \textit{PHI}) \longrightarrow$$
$$\text{believesMinimumNecessaryForPurpose}(p_1, p_2, q, t, u) \vee$$
$$(\text{inrole}(p_1, \textit{covered-entity}) \wedge \text{inrole}(p_2, \textit{individual})$$
$$\wedge \text{inrole}(q, \textit{individual}) \wedge \text{in}(t, \textit{PHI}) \wedge \textit{samePerson}(p_2, q))$$

(§**164**.**508**(**a**)(**2**))

$$\text{inrole}(p_1, \textit{covered-entity}) \wedge \text{inrole}(q, \textit{individual}) \wedge \text{in}(t, \textit{psych-notes}) \longrightarrow$$
$$\text{obtainedAuthorization}(p_1, p_2, q, t, u)$$

Figure 5.14: Sliced HIPAA policy ($\wp_{H_{P_2}}$) norms w.r.t the obligation in §164.524 of HIPAA.

(**Synthetic Policy Rule 1**)

$$\text{inrole}(p_1, \textit{parent}) \wedge \text{inrole}(p_2, \textit{covered-entity}) \wedge \text{inrole}(q, \textit{individual}) \wedge$$
$$\text{purpose}(u, \textit{access-request}) \wedge \textit{parentOf}(p_1, q) \wedge \text{request}(p_1, p_2, q, \textit{PHI})$$

(**Synthetic Policy Rule 2**)

$$\text{inrole}(p_1, \textit{parent}) \wedge \text{inrole}(p_2, \textit{covered-entity}) \wedge \text{inrole}(q, \textit{individual}) \wedge$$
$$\text{purpose}(u, \textit{access-request}) \wedge \textit{parentOf}(p_1, q) \wedge \text{request}(p_1, p_2, q, \textit{PHI}) \rightarrow$$
$$\Diamond(\exists m_1 : M.(\text{send}(p_2, p_1, m_1) \wedge \text{contains}(m_1, q, \textit{PHI}) \wedge$$
$$\text{for-purpose}(m_1, \textit{access-request})))$$

(§**164**.**502**(**b**))

$$\text{inrole}(p_1, \textit{covered-entity}) \wedge \text{inrole}(q, \textit{individual}) \wedge \text{in}(t, \textit{PHI}) \rightarrow$$
$$(\text{believesMinimumNecessaryForPurpose}(p_1, p_2, q, t, u) \vee$$
$$\text{obtainedAuthorization}(p_1, p_2, q, t, u))$$

(§**164**.**508**(**a**)(**2**))

$$\text{inrole}(p_1, \textit{covered-entity}) \wedge \text{inrole}(q, \textit{individual}) \wedge \text{in}(t, \textit{psych-notes}) \rightarrow$$
$$\text{obtainedAuthorization}(p_1, p_2, q, t, u)$$

(§**164**.**502**(**g**)(**3**)(**ii**)(**A**))

$$\text{inrole}(p_1, \textit{covered-entity}) \wedge \text{inrole}(p_2, \textit{parent}) \wedge \text{inrole}(q, \textit{individual}) \wedge \text{in}(t, \textit{PHI})$$
$$\wedge \textit{permittedByOtherLaw}(p1, p2, q, t, u) \wedge \textit{parentOf}(p2, q)$$

(§**164**.**502**(**g**)(**3**)(**ii**)(**B**))

$$\text{inrole}(p_1, \textit{covered-entity}) \wedge \text{inrole}(p_2, \textit{parent}) \wedge \text{inrole}(q, \textit{individual}) \wedge \text{in}(t, \textit{PHI})$$
$$\wedge \textit{prohibitedByOtherLaw}(p1, p2, q, t, u) \wedge \textit{parentOf}(p2, q) \rightarrow \textit{false}$$

(§**164**.**510**(**b**)(**2**))

$$\text{inrole}(p_1, \textit{covered-entity}) \wedge \text{inrole}(p_2, \textit{parent}) \wedge \text{inrole}(q, \textit{individual}) \wedge \text{in}(t, \textit{PHI}) \wedge$$
$$\text{parentOf}(p_2, q) \wedge (\Diamond(\textit{available}(p_1, q) \wedge \textit{hasCapabilityToMakeHealthDecisions}(q))) \rightarrow ($$
$$(\exists m_1 : M.(\Diamond(\text{send}(q, p_1, m_1) \wedge \textit{isAgreement164510b2}(m_1, p_1, p_2, q, t, u)))) \vee$$
$$((\neg(\exists m_2 : M.(\text{send}(q, p_1, m_2) \wedge \textit{isObjection164510b2}(m_2, p_1, p_2, q, t, u)))) \mathcal{S}$$
$$(\exists m_3 : M.(\text{send}(p_1, q, m_3) \wedge \textit{isOpportunityToObject}(m_3, p_1, p_2, q, t, u))))$$
$$\vee (\textit{professionalJudgementIndividualDoesNotObject}(p_1, p_2, q, t, u)))$$

(§**164**.**510**(**b**)(**2**))

$$\text{inrole}(p_1, \textit{covered-entity}) \wedge \text{inrole}(p_2, \textit{parent}) \wedge \text{inrole}(q, \textit{individual}) \wedge \text{in}(t, \textit{PHI}) \wedge$$
$$\textit{parentOf}(p2, q) \wedge (\neg(\Diamond(\textit{available}(p_1, q) \wedge \textit{hasCapabilityToMakeHealthDecisions}(q)))) \rightarrow$$
$$(\textit{professionalJudgementIsInBestInterestof164510b3}(p_1, p_2, q, t, u) \wedge$$
$$\textit{relevantToInvolvement}(p_1, p_2, q, t, u))$$

Figure 5.15: Sliced HIPAA policy ($\wp_{H_{P_3}}$) norms w.r.t the synthetic obligation.

Thus, the policy cannot distinguish between two elements of the carrier. Consequently, we cannot specify in the policy that two individuals are different. The sliced HIPAA policies we consider satisfy this restriction. Moreover, we remove the message sort ($\mathcal{M}$) and also remove the predicates `contains` and `for-purpose`. We enhance the send predicate to have the signature $\mathcal{P} \times \mathcal{P} \times \mathcal{P} \times \mathcal{T} \times \mathcal{U}$. Now, the predicate send(p_1, p_2, q, t, u) holds when p_1 sends a message to p_2 about q's attribute t for purpose u. Removing the message sort prevents us from specifying that a message contains multiple attributes of multiple individuals or is sent for multiple purposes. This is not as restricting as it sounds, at least for the slices of HIPAA we consider. Assume a set of send events, each for a message with a single attribute and for a single purpose. If all events are allowed, then the send of a single message that combines all the other messages' contents is allowed by the sliced policies. Now, we provide intuitions behind developing a small model theorem for the HIPAA policy sliced with respect to the obligation in §160.310 (Figure 5.13), denoted by $\wp_{H_{P_1}}$.

The number of attributes in the $\wp_{H_{P_1}}$ is finite and they are *PHI* and *psych-notes* (in short, *PSN*). Each of *PHI* and *PSN* can be viewed as a set of attributes where $PHI, PSN \subseteq \mathcal{T}$, $PSN \subset PHI$ and $\mathcal{T}$ is the carrier of attributes. Thus, if we consider any attribute $t \in \mathcal{T}$, one of the following would hold: $t \in \mathcal{T} \setminus (PHI \cup PSN)$, $t \in (PHI \setminus PSN)$, or $t \in (PHI \cap PSN)$ (see Figure 5.16). Thus, we consider three attributes, t_1, t_2, and t_3 such that $t_1 \in (PHI \cap PSN)$, $t_2 \in (PHI \setminus PSN)$, and $t_3 \in \mathcal{T} \setminus (PHI \cup PSN)$. These three attributes can simulate all possible attributes referred to by $\wp_{H_{P_1}}$. The only purpose present in the $\wp_{H_{P_1}}$ is *compliance-investigation*. We thus consider two purposes in the system u_1 and u_2 where u_1 is the purpose *compliance-investigation* and u_2 refers to a purpose which is something other than *compliance-investigation*. These two purposes capture all the possible purposes referred by $\wp_{H_{P_1}}$. For the principal sort, we consider one principal for each role in $\wp_{H_{P_1}}$ and one additional principal not having any roles. The roles in $\wp_{H_{P_1}}$ are: covered entity, secretary, and individual. Considering one individual from each role is sound as $\wp_{H_{P_1}}$ cannot differentiate between two principals. We actually could have considered only two principals, one acting in all the roles and another not in any role. For clarity, we consider principals of different roles are different. Having multiple principals in each role does not change the result of the Δ-

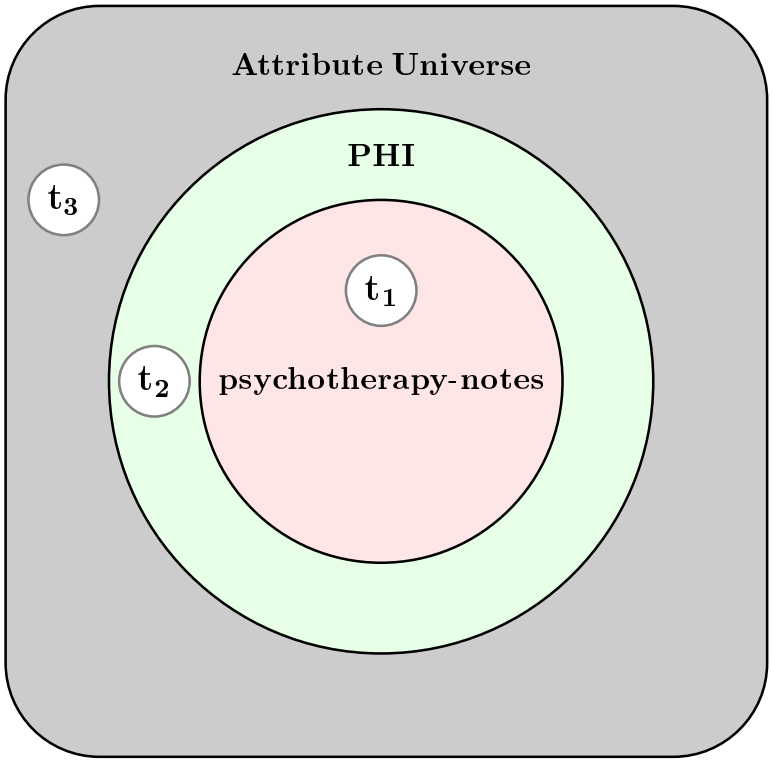

Figure 5.16: Small model property for Attribute domain

property holding, as these 4 principals can simulate all possible behaviors. We have the following small model theorem for $\wp_{HP_1}$.

Theorem 74. *The policy $\wp_{HP_1}$ has the Δ-property for any arbitrary carriers $\mathcal{P}$, $\mathcal{T}$, and $\mathcal{U}$ if and only if $\wp_{HP_1}$ has the Δ-property for finite carriers $\hat{\mathcal{P}}, \hat{\mathcal{T}}$, and $\hat{\mathcal{U}}$ in which $|\hat{\mathcal{P}}| = 4$, $|\hat{\mathcal{T}}| = 3$, and $|\hat{\mathcal{U}}| = 2$.*

Proof. The *only if* ($\rightarrow$) direction is trivial. When the $\wp_{HP_1}$ has the Δ-property for any arbitrary carriers $\mathcal{P}$, $\mathcal{T}$, and $\mathcal{U}$, then it should be case that, $\wp_{HP_1}$ has the Δ-property for specific carriers $\hat{\mathcal{P}}, \hat{\mathcal{T}}$, and $\hat{\mathcal{U}}$, where $|\hat{\mathcal{P}}| = 4$, $|\hat{\mathcal{T}}| = 3$, and $|\hat{\mathcal{U}}| = 2$.

We will now show the *if* ($\leftarrow$) direction. According to the definition of Δ-property, a policy $\wp$ has the Δ-property if and only if for all finite history σ_f and for all environment η such that

$\sigma_f, |\sigma_f|-1, \eta \models \boxminus weak(\wp)$, there exists an infinite extension σ_i such that $\sigma_f \cdot \sigma_i \models \wp$. Take any arbitrary finite trace σ_f and any arbitrary environment η with carriers $\mathcal{P}$, $\mathcal{T}$, and $\mathcal{U}$ such that $\sigma_f, |\sigma_f|-1, \eta \models \boxminus weak(\wp_{HP_1})$. We will show that we can convert (1) σ_f to another finite trace σ_f^* where $|\sigma_f| = |\sigma_f^*|$, (2) η to another environment η' such that $\sigma_f^*, |\sigma_f^*|-1, \eta' \models \boxminus weak(\wp_{HP_1})$ with carriers $\hat{\mathcal{P}}, \hat{\mathcal{T}}$, and $\hat{\mathcal{U}}$, where $|\hat{\mathcal{P}}| = 4$, $|\hat{\mathcal{T}}| = 3$, and $|\hat{\mathcal{U}}| = 2$. For this we will first show how to convert each carrier element appearing in the finite trace σ_f to carrier elements in our smaller carriers. We will show this for each carrier. Note that, we refer carrier elements of arbitrary carriers $\mathcal{P}$, $\mathcal{T}$, and $\mathcal{U}$ as concrete carrier elements whereas we refer carrier elements of the carriers $\hat{\mathcal{P}}, \hat{\mathcal{T}}$, and $\hat{\mathcal{U}}$ as abstract carrier elements.

The number of attributes explicitly mentioned in the $\wp_{HP_1}$ is finite and they are *PHI* and *psych-notes* (in short, *PSN*). From the domain knowledge of HIPAA, we know that each of *PHI* and *PSN* can be viewed as a set of attributes where $PHI, PSN \subseteq \mathcal{T}$, $PSN \subset PHI$ and $\mathcal{T}$ is the carrier of attributes. Thus, if we consider any attribute $t \in \mathcal{T}$, one of the following would hold: $t \in \mathcal{T} \setminus (PHI \cup PSN)$, $t \in (PHI \setminus PSN)$, or $t \in (PHI \cap PSN)$ (see Figure 5.16). Thus, we consider three abstract attributes, t_1, t_2, and t_3 such that $t_1 \in (PHI \cap PSN)$, $t_2 \in (PHI \setminus PSN)$, and $t_3 \in \mathcal{T} \setminus (PHI \cup PSN)$. Any concrete attribute that can be calculated from both *PHI* and *psych-notes*, can be abstractly represented as the abstract attribute t_1. Now for any concrete attribute that can only be calculated from *PHI* but not from *psych-notes* can be abstractly denoted by abstract attribute t_2. Finally for any concrete attribute which cannot be calculated from either *PHI* or *psych-notes*, can be represented as abstract attribute t_3. Thus, these three abstract attributes can represent all concrete attributes in any arbitrary carrier $\mathcal{T}$.

The only purpose present in the $\wp_{HP_1}$ is *compliance-investigation*. We thus consider two purposes in the system u_1 and u_2 where u_1 is the purpose *compliance-investigation* and u_2 refers to a purpose which is something other than *compliance-investigation*. These two purposes is sufficient to represent all the possible purposes according to any arbitrary carrier $\mathcal{U}$ with respect to $\wp_{HP_1}$. Now, we can replace any concrete purpose which is not *compliance-investigation* with u_2 and any purpose which is *compliance-investigation* can be replaced with u_1.

Recall that we make the assumption that each principal can have at most one critical role. In $\wp_{HP_1}$, the critical roles are: secretary, *covered-entity*, and individual. We now consider one principal from each role and an additional principal which does not belong to any role. Thus, these 4 principals p_c, p_s, p_i, and p_n, where p_c belongs to the role *covered-entity*, p_s belongs to the role secretary, p_i belongs to the role individual, and p_n belongs to none of the critical roles, is sufficient to represent any concrete individuals of any arbitrary carrier $\mathcal{P}$ with respect to $\wp_{HP_1}$ under the assumption that the concrete principals do not possess two conflicting roles (*e.g.*, *covered-entity*, individual). Now we can replace any principal possessing the role secretary with the abstract carrier element p_s, replace any principal possessing the role covered-entity with the abstract carrier element p_c, and replace any principal possessing the role individual with the abstract carrier element p_i.

Once we have setup the abstract individuals, we can translate σ_f and η with σ_f^* and η' such that all the concrete individuals are replaced by the corresponding abstract individuals in the carriers $\hat{\mathcal{P}}, \hat{\mathcal{T}}$, and $\hat{\mathcal{U}}$. We do this for all the predicate valuations too. Inspecting the policy we see that the any principal possessing the role of secretary can send a request to a principal with role covered entity to have access to the *PHI* of a principal with role individual for the purpose of compliance-investigation. This actions is allowed by the first positive norm (Synthetic Policy Rule) and according to the negative norm §160.310, this incurs an obligation for the principal in the covered-entity role to provide the principal with the secretary role access to the *PHI* of the principal in the role individual. The positive norm §164.502(a)(2)(ii) enables this action. Moreover, there are two negative norms in §164.502(b) and §164.508(a)(2) that restricts the fulfillment of the obligation. The negative norm §164.502(b) allows the principal in covered-entity role to fulfill the obligation provided that the predicates holds true believesMinimumNecessaryForPurpose or this action is fulfillment of the obligation. The negative norm §164.508(a)(2) requires that if the attribute intended to be sent is *psych-notes*, the predicate obtained Authorization should hold. From the inspection of the policy $\wp_{HP_1}$, we can see that $\sigma_f^*, |\sigma_f^*| - 1, \eta' \models \boxminus \mathit{weak}(\wp_{HP_1})$ with carriers $\hat{\mathcal{P}}, \hat{\mathcal{T}}$, and $\hat{\mathcal{U}}$. As we do not have the $\neq$ operator, it follows that replacing the concrete carrier elements with abstract

carrier elements will not change the result of the satisfaction of $\boxminus weak(\wp_{H_{P_1}})$.

Now from the fact that $\mathbb{M}_{\wp} \models \wp_{H_{P_1}}$ w.r.t carriers $\hat{\mathcal{P}}, \hat{\mathcal{T}}$, and $\hat{\mathcal{U}}$, gives us that there is an infinite extension of σ_f^*, σ_i^*, such that $\sigma_f^* \cdot \sigma_i^* \models \wp_{H_{P_1}}$ w.r.t to $\hat{\mathcal{P}}, \hat{\mathcal{T}}$, and $\hat{\mathcal{U}}$. From σ_i^* we will show how to construct an extension σ_i such that $\sigma_f \cdot \sigma_i \models \wp_{H_{P_1}}$ w.r.t $\mathcal{P}$, $\mathcal{T}$, and $\mathcal{U}$.

Now consider the finite trace σ_f with carriers $\mathcal{P}$, $\mathcal{T}$, and $\mathcal{U}$ where principals $s_1, s_2, s_3, \ldots$ are in role secretary, principals $c_1, c_2, c_3, \ldots$ are in role covered-entity, and finally principals $i_1, i_2, i_3, \ldots$ are in the role individuals. Let us assume in σ_f in the first state s_1 sends a request to c_1 to access the *PHI* of i_1, in the second state s_2 sends a request to c_2 to access the *PHI* of i_2, and so on. Let us also assume $|\sigma_f| = k$. We construct σ_f^* where $|\sigma_f^*| = k$ with carriers $\hat{\mathcal{P}}, \hat{\mathcal{T}}$, and $\hat{\mathcal{U}}$ in way that all s_j are replaced by p_s, all c_j replaced by p_c, and all i_j replaced by p_i. Thus, for a trace σ_f, when converted to σ_f^*, each element of σ_f^* is a request from p_s to p_c to access *PHI* of p_i. As $\mathbb{M}_{\wp} \models \wp_{H_{P_1}}$ w.r.t carriers $\hat{\mathcal{P}}, \hat{\mathcal{T}}$, and $\hat{\mathcal{U}}$, gives us that there is an infinite extension of σ_f^*, σ_i^*, such that $\sigma_f^* \cdot \sigma_i^* \models \wp_{H_{P_1}}$ w.r.t to $\hat{\mathcal{P}}, \hat{\mathcal{T}}$, and $\hat{\mathcal{U}}$. It follows that there was a state in σ_i^* in which p_c sent a message to p_s giving access to *PHI* of p_i. Note that, for all the k requests in σ_f^*, one response is sufficient according to the semantics of $\Diamond$. We will now show that we can translate σ_i^* to σ_i such that $\sigma_f \cdot \sigma_i \models \wp$ w.r.t $\mathcal{P}$, $\mathcal{T}$, and $\mathcal{U}$. We first find all the request in σ_i^*, we make them requests in σ_i by translating p_c to a specific c_j, translating p_s to a specific s_j, and translating p_j to a specific i_j. We then find states where p_c responded to p_s's request to access the *PHI* of p_i. We then check σ_f and partially constructed σ_i till the position of the response and find out all the request from s_j to c_j to access *PHI* of i_j from which we have not seen a response from c_j to s_j. We then instantiate the response from p_c to p_s for giving access to *PHI* of p_i to these open requests, for all these principals in which p_c is replaced by c_j (for all j), p_s is replaced by s_j (for all j), and p_i is replaced by i_j (for all j). We also change the valuations of the predicates accordingly. Now $\sigma_f \cdot \sigma_i \models \wp$, due to the case that after inspecting the policy $\wp_{H_{P_1}}$, we can see that as there is no $\neq$ operator, if one particular instance of a send is allowed than arbitrary instance of the send is allowed provided that all the conditions (predicates) are also instantiated for that arbitrary instance. Based on this, it follows that $\sigma_f \cdot \sigma_i \models \wp_{H_{P_1}}$, which completes our proof. □

In the same vein, we can also prove a small model theorem for the policy slice $\wp_{H_{P_2}}$ and the policy $\wp_{H_{P_3}}$. As a result, we have the following small model theorems for policy slices $\wp_{H_{P_2}}$ and $\wp_{H_{P_3}}$.

Theorem 75. *The policy $\wp_{H_{P_2}}$ has the Δ-property for any arbitrary carriers $\mathcal{P}$, $\mathcal{T}$, and $\mathcal{U}$ if and only if $\wp_{H_{P_2}}$ has the Δ-property for finite carriers $\hat{\mathcal{P}}, \hat{\mathcal{T}}$, and $\hat{\mathcal{U}}$ in which $|\hat{\mathcal{P}}| = 3$, $|\hat{\mathcal{T}}| = 3$, and $|\hat{\mathcal{U}}| = 2$.*

Theorem 76. *The policy $\wp_{H_{P_3}}$ has the Δ-property for any arbitrary carriers $\mathcal{P}$, $\mathcal{T}$, and $\mathcal{U}$ if and only if $\wp_{H_{P_3}}$ has the Δ-property for finite carriers $\hat{\mathcal{P}}, \hat{\mathcal{T}}$, and $\hat{\mathcal{U}}$ in which $|\hat{\mathcal{P}}| = 4$, $|\hat{\mathcal{T}}| = 3$, and $|\hat{\mathcal{U}}| = 2$.*

Table 5.1: Policy analysis result: HIPAA case study

Obligations	Number of norms in the slice	Number of states in tableau	Number of transitions in the tableau	Number of initial states	Number of accepting states	Number of strongly connected component	Automata generation time (s)	Graph analysis time (ms)	Analysis results
160.310	5/76	28	162	8	12	10	3	2	Passed
Synthetic-1	8/78	957	71563	126	274	131	98	16	Passed
Synthetic-2	8/78	2422	129538	204	702	516	324	31	Failed
164.524	5/76	533	12331	48	173	147	22	5	Passed

5.4.7 Policy Analysis Results.

We sliced $\wp_H$ with respect to 2 obligations in HIPAA (§160.310, §164.524) and 1 synthetic obligation (Synthetic-1). After developing the small model theorems, we converted the slices of the FOTL policy into pLTL formulas. Once we have the pLTL policies/formulas, we follow the approach proposed by Barth *et al.* [11], as discussed before. We convert each of the sliced pLTL policies to a tableau (with Büchi accepting condition) using the GOAL automata generation tool [120], then check whether all the reachable states can reach a strongly connected component with an accepting state in it. The experimental results are presented in Table 5.1. We verify that for the two obligations in HIPAA, the Δ-property holds for the sliced policies.

5.4.8 Observation.

While verifying the sliced policy with respect to the synthetic obligation, we observed something interesting. Investigating the regulations manually had led us to believe that the policy sliced with respect to the synthetic obligation (Synthetic-1) should **not** satisfy the Δ-property. However, our experimental result seemed to differ. Upon close inspection, we determined that the result is due to how HIPAA is specified. Specifically, rule §164.502(g)(3)(ii)(b) states that a covered entity cannot share an individual's *PHI* if it is forbidden by some law. In our specification of HIPAA, we keep the room that even though an action is now forbidden by some law, the law might change, and allow the forbidden action later. When we changed our specification in such a way that laws cannot change (*Synthetic-2*), then we got the desired result of the Δ-property not holding.

5.4.9 Discussion.

There are two more occurrences of obligations in HIPAA (§164.512(c)(2) and §164.502(b)) which require sending privacy notices to the patient. We assumed that privacy notices do not contain any *PHI* of the patient, and thus, we considered those obligations to be trivially allowed. Alternatively, however, if one allows privacy notices to contain PHI, then the corresponding slice will be similar to the slice of the obligation in §164.524.

5.4.10 Counter Example.

When a policy violates the Δ-property, we can traverse the tableau to find a path to the violating node. To achieve this, we can run a depth first search from all the initial states (nodes) of the tableau and try to find the states (nodes) from which it is not possible to reach a strongly connected component with at least one accepting state. This path corresponds to a finite trace showing the violation of the Δ-property. This counter example (expressed as a finite trace) can help the policy author to rewrite the policy to satisfy the Δ-property.

Chapter 6: FORMALLY ENSURING PERMISSIBILITY OF OBLIGATIONS IN SECURITY POLICIES

6.1 Introduction

Many access control and privacy policies contain some notion of actions that are *required* to be performed by a system or its users in some future time. Such required actions can be naturally modeled as obligations. Consider the following paraphrased regulation excerpt from section §164.524 of the Health Insurance Privacy and Accountability Act (HIPAA) [62]. A covered entity must respond to a request for access no later than 30 days after receipt of the request of the patient. As we can see from the regulation, the action of the covered entity is required when he receives a request from the patient. When we use obligations to capture this notion of required actions, we need a proper framework and mechanisms by which obligations can be managed efficiently.

The notion of obligations is not new. Several researchers [3, 11, 18, 31, 69, 94, 95, 97, 101, 122, 127] have proposed frameworks for modeling and managing obligations. The majority of the existing work [3, 11, 18, 31, 94, 101, 122, 127] focuses on policy specification languages for obligations rather than efficient management of obligations [11, 39, 54, 67, 86, 100, 110]. Even for works on the management of obligations, they mainly consider *system obligations*. Our goal is to address technical issues for efficient management of *user obligations*. A user (resp., system) obligation is an action that is to be carried out by a user (resp., the system) in some future time. In this current work, we only consider post-obligations. Modeling and analyzing pre-obligations and refrainments are subjects of future work. Managing user obligations is challenging as system obligations can be assumed to be always fulfilled whereas this is often not the case for user obligations. The behavior of human users unlike mechanical users cannot be enforced. More generally, we consider user obligations that can require authorization and can also alter the authorization state of the system. As a user obligation is an action, it is subjected to the authorization requirements imposed by the

The contents of this chapter is based on the joint work with Murillo Pontual, Jianwei Niu, Ting Yu, Keith Irwin, and William H. Winsborough [28].

security policy of the system. We also consider that each of the user obligations has a time interval (*e.g.*, 30 days, *etc.*) which represents the alloted time window at which the obligation should be performed. Such intervals help detect obligation violation.

When managing user obligations depend on and affect authorization, we have to consider the case in which users can incur obligations that they are not authorized to perform. Otherwise, when an obligation goes unfulfilled, it is difficult to know if it is due to insufficient authorization or lack of diligence from the user. When it is ensured that all the obligatory actions are authorized, any obligation violation will only be caused due to user negligence. Irwin *et al.* [67] introduce a property of the authorization state and the current obligation pool, *accountability*, that tries to ensure that all the obligatory actions are authorized in some part of their stipulated time interval. They consider two variations of the accountability property (*i.e.*, *strong accountability* and *weak accountability*) based on when in the time intervals the obligatory actions should be authorized. Strong accountability requires that no matter when all the previous obligations are performed in their associated time intervals, each obligation in the current obligation pool is authorized during its whole time interval. Weak accountability on the other hand requires that each obligation is at least authorized just before its deadline.

Irwin *et al.* [67] propose to maintain the accountability property as an invariant of the system. They propose to use the reference monitor of the system for maintaining accountability by denying actions that violate accountability. Extending the work of Irwin *et al.* [67], Pontual *et al.* [109] show that for an obligation system using mini-RBAC [114, 119] and mini-ARBAC [114, 119] as its authorization model, strong accountability can be decided in polynomial time whereas deciding weak accountability is co-NP complete. They also provide empirical evaluations for showing that a reference monitor can maintain the strong accountability property efficiently. They partition possible actions into two disjoint sets, *discretionary* and *obligatory* and only allow discretionary actions to incur further obligations. By doing this, they disallow *cascading obligations*.

The assumption of disallowing cascading obligations is restrictive. It significantly reduces the expressive power of the obligation model they use. For instance, consider the following scenario.

When a sales assistant submits a purchase order, the clerk incurs an obligation to issue a check in the amount identified in the purchase order. As soon as the clerk issues the check, the manager incurs an obligation that requires him to check the consistency of the purchase order. If the purchase order is consistent and the manager approves it, then the accountant incurs another obligation to approve the check. Now, this situation can be easily modeled with cascading obligations, but it cannot be modeled by the obligation model of Pontual *et al.* [109]. Thus, one of the principal goals of this work is to provide a concrete model in which the policy writers can specify cascading obligations easily. Furthermore, we also present a decision procedure which can be used to decide the strong accountability property efficiently for special but practical cases of cascading obligations in the model. We only consider strong accountability in this work due to the complexity results of weak accountability.

Contributions The abstract obligation model that Irwin *et al.* [67] and Pontual *et al.* [109] use, allows specification of cascading obligations. However, their concrete model does not support the specification of cascading obligations. We adopt the concrete model of Pontual *et al.* [109] that uses mini-RBAC and mini-ARBAC as its authorization model and augment it in a way that cascading obligations can be specified. Furthermore, existing work [67, 109] does not discuss how to specify the user (obligatee) who incurs the new obligation when a user takes an action (obligatory or discretionary). We present several proposals for specifying the obligatee in a policy. The enhancement to the obligation model and proposals for obligatee selection comprise our ***first contribution***.

The specification for strong accountability presented by Pontual *et al.* [109] also takes advantage of the assumption that cascading obligations are not allowed. Our ***second contribution*** is to precisely specify the strong accountability property in presence of cascading obligations. There are two possible interpretations of strong accountability when considering cascading obligations. We define both interpretations, *existential* and *universal*, and give motivations for choosing the existential interpretation.

Our ***third contribution*** is to present a theorem which states that deciding accountability in presence of cascading obligations is in general NP-hard. We then consider several special cases which makes the problem tractable. We then provide a polynomial time algorithm (polynomial in the size of the policy, the size of the current obligation pool, and the new obligations to be considered) that can decide strong accountability for special cases of cascading obligations. This is our ***fourth contribution***.

We then present empirical evaluations of the accountability decision procedure allowing special cases of cascading obligations. Our empirical evaluations show that strong accountability can be efficiently decided for these special cases of cascading obligations. This is our ***final contribution***.

6.2 Background

In this section, we review some of the background materials necessary to understand our contributions. We then discuss the obligation model presented by Pontual *et al.* [109] that uses mini-RBAC and mini-ARBAC as its authorization model.

6.2.1 Obligation Model

We now summarize the obligation model proposed by Pontual *et al.* [109]. Note that, we augment this model for supporting cascading obligations in section 6.3. We use $U \subseteq \mathcal{U}$ to denote the finite set of users in the system at any given point of time. We use u possibly with subscripts to represents users. The finite set of objects in the system is denoted by $O \subseteq \mathcal{O}$. We use o with possibly subscripts to range over the elements of O. Note that, the universes $\mathcal{U}$ and $\mathcal{O}$ are countably infinite as we want to model systems of finite but unbounded sizes. For supporting administrative actions, we have $U \subseteq O$. The set of possible actions in the system is given by $\mathcal{A}$. The formal type of $\mathcal{A}$ is given below.

We denote a system state with $s = \langle U, O, t, \gamma, B \rangle$ where $t \in \mathcal{T}$ denotes the current system time, $\gamma \in \Gamma$ represents the mini-RBAC authorization state, and $B \subseteq \mathcal{B}$ represents the current pool of obligations. Γ here denotes the set of abstract authorization states. Obligations in the system has

the form $b = \langle u, a, \vec{o}, t_s, t_e \rangle$, the universe of which, $\mathcal{B}$ has the formal type $\mathcal{U} \times \mathcal{A} \times \mathcal{O}^* \times \mathcal{T} \times \mathcal{T}$. Note $\mathcal{O}^*$ is the Cartesian product of zero or more copies of $\mathcal{O}$. For an obligation $b = \langle u, a, \vec{o}, t_s, t_e \rangle$, $[t_s, t_e]$ denotes the interval in which the obligation should be performed. Moreover, we require that $t_s < t_e$. We use $b.u$, $b.a$, and $b.o^*$, respectively, to denote the user, the actions, and the object(s) of the obligation b.

We consider two types of actions, namely, *discretionary* and *obligatory*. The system views them uniformly as events. The universe of discretionary action $\mathcal{D}$ has the formal type $\mathcal{U} \times \mathcal{A} \times \mathcal{O}^*$. Thus, the universe of all possible events is $\mathcal{E} = \mathcal{D} \cup \mathcal{B}$. Each action $a \in \mathcal{A}$ has the formal type $(\mathcal{U} \times \mathcal{O}^*) \rightarrow (\mathcal{FP}(\mathcal{U}) \times \mathcal{FP}(\mathcal{O}) \times \Gamma) \rightarrow (\mathcal{FP}(\mathcal{U}) \times \mathcal{FP}(\mathcal{O}) \times \Gamma)$. $\mathcal{FP}(\mathcal{X}) = \{X \subset \mathcal{X} \,|X \text{ is finite}\}$ denotes the set of finite subsets of the given set $\mathcal{X}$. Actions can add or remove users and objects and can also alter the authorization state. Thus, for a given a user u and object(s) $\vec{o}$, the action $a(u, \vec{o})$ is a mapping that that maps the current set of users, objects, and the current authorization state to a new set of users, objects, and authorization state.

Each action in our system is regulated by a fixed set of positive, policy rules $\mathcal{P}$. Each policy rule $p \in \mathcal{P}$ has the form $p = a(u, \vec{o}) \leftarrow cond(u, \vec{o}, a) : F_{obl}(s, u, \vec{o})$. This represents that a user u is authorized to perform an action a that is applied to object(s) $\vec{o}$, when the predicate $cond(u, \vec{o}, a)$ is satisfied in the current authorization state γ (denoted by $\gamma \models cond(u, \vec{o}, a)$) and this in turn incurs a set of obligations (possibly empty) for u or some other users. The predicate *cond* represents the authorization requirements imposed by the policy rule. $F_{obl}(s, u, \vec{o})$ is a function that takes as input s, u, and $\vec{o}$ and returns a set of obligations (possibly empty) when the policy rule p is used to authorize the action a. For a policy rule $p \in \mathcal{P}$ of form $p = a(u, \vec{o}) \leftarrow cond(u, \vec{o}, a) : F_{obl}(s, u, \vec{o})$, when $a \in \mathcal{B}$ and the $F_{obl}(s, u, \vec{o})$ is not empty, we call the obligatory action a, a *cascading obligation*. We also require that for each action, there is at least one policy rule that governs that action.

Recall that, we consider actions that can alter the authorization state of the system. Based on whether an action alters the authorization, we classify the actions in two possible categories, namely, *administrative* and *non-administrative*. An action a is called administrative (denoted by $a \in$ administrative) when it has the form $grant(u, \vec{o})$ or $revoke(u, \vec{o})$ and called non-administrative,

otherwise.

Now, we turn our attention to how state transition occurs in the obligation system. We use $s \xrightarrow{\langle e,p\rangle} s'$ to denote the transition from state s to state s' when event e takes place and is authorized using the policy rule p. When e is of form $\langle u,a,\vec{o}\rangle$, we require that $u \in s.U$ and $\vec{o} \in s.O^*$. Furthermore, for each state s of the form $\langle U,O,t,\gamma,B\rangle$, the transition relation ensures that $\forall b \in s.B \cdot (b.u \in s.U) \wedge (b.\vec{o} \in s.O^*)$ and $s.U = s.\gamma.U$ holds. We first formalize what it means that the current authorization state γ satisfies the $cond(u,\vec{o},a)$ predicate of a policy rule p (definition 77). We then formally specify the transition relation of the system.

Definition 77. *For all $u \in \mathcal{U}$ and $\vec{o} \in O^*$, $\gamma \models cond(u,\vec{o},a)$ if and only if the following holds.*

$$
\begin{aligned}
& (\exists r).(((u,r) \in \gamma.UA) \wedge \\
(i) \quad & [a \notin \text{administrative} \rightarrow (\langle r,\langle a,\vec{o}\rangle\rangle \in \gamma.PA)] \wedge \\
(ii) \quad & (\forall u_t,r_t).[a = grant \wedge \vec{o} = \langle u_t,r_t\rangle \rightarrow \\
& \qquad (\exists c).((\langle r,c,r_t\rangle \in \Phi.CA) \wedge (u_t \models_\gamma c))] \wedge \\
(iii) \quad & (\forall u_t,r_t).[a = revoke \wedge \vec{o} = \langle u_t,r_t\rangle \rightarrow \\
& \qquad (\exists c).((\langle r,c,r_t\rangle \in \Phi.CR) \wedge (u_t \models_\gamma c))])
\end{aligned}
$$

Definition 78 (Transition relation)**.** *We use $s_{0..j}$ to denote the sequence $s_0,s_1,\ldots,s_j$ where $j \in \mathbb{N}$, and for $\ell \in \mathbb{N}$, $\ell \leq j$, $s_{0..\ell}$ denotes the prefix of $s_{0..j}$ and when $\ell < j$ the prefix is* proper*. Similarly, $\langle e,p\rangle_{0..j}$ denotes $\langle e_0,p_0\rangle,\langle e_1,p_1\rangle,\ldots,\langle e_j,p_j\rangle$. Given any sequence of event/policy-rule pairs, $\langle e,p\rangle_{0..k}$, and any sequence of system states $s_{0..k+1}$, the relation $\longrightarrow \subseteq \mathcal{S} \times (\mathcal{E} \times \mathcal{P})^+ \times \mathcal{S}$ is defined inductively on $k \in \mathbb{N}$ as follows:*

(1) $s_k \xrightarrow{\langle e,p\rangle_k} s_{k+1}$ holds if and only if, letting $p_k = a(u,\vec{o}) \leftarrow cond(u,\vec{o},a) : F_{obl}(s,u,\vec{o})$, we have $s_k.\gamma \models cond(e_k.u,\ e_k.\vec{o},\ e_k.a)$, and $s_{k+1} = \langle U'',O'',t'',\gamma'',B''\rangle$, in which $\langle U'',O'',\gamma''\rangle = a(u,\vec{o})(s_k.U,s_k.O,s_k.\gamma)$, $B'' = (s_k.B - \{e\}) \cup F_{obl}(s_k,e_k.u,e_k.\vec{o})$ when $e_k \in \mathcal{B}$, and $B'' = s_k.B \cup F_{obl}(s_k,e_k.u,e_k.\vec{o})$ otherwise. t'' denotes the system time when a is completed.

(2) $s_0 \xrightarrow{\langle e,p\rangle_{0..k}} s_{k+1}$ if and only if there exists $s_k \in \mathcal{S}$ such that $s_0 \xrightarrow{\langle e,p\rangle_{0..k-1}} s_k$ and $s_k \xrightarrow{\langle e,p\rangle_k} s_{k+1}$.

6.3 Enhancement of the Model

In this section, we extend the obligation model of Pontual *et al.* [109] to facilitate the specification and analysis of accountability in presence of cascading obligations.

6.3.1 Time Interval of the Incurred Obligation

In the previous obligation model [109], when a discretionary action a is taken at time t and it causes an obligation b to be incurred, the time interval of b depends on the time t. Thus, the time interval of b is calculated using a fixed offset from t and the interval size of b. Let us assume the fixed offset is $\delta \in \mathbb{N}$. As we have mentioned in section 6.1, in the current work we only consider post-obligations. Let us assume, if a user takes action a according to policy rule p then she incurs the obligation b. If we allow δ to be negative we can model pre-obligations where b's interval will be before the time when a is attempted. Handling pre-obligations is a subject of future work. and the interval size of b is w. So, the time interval of b will be $[t+\delta, t+\delta+w]$. Now, consider the case where an obligation b_1 with time interval $[s,e]$ incurs another obligation b_2. The time t at which b_1 can possibly be performed can be any value between s and e, inclusive. Thus, we have several possible intervals for b_2 considering each possible values of t (see figure 6.1(a)). For deciding strong accountability, we have to check whether b_2 is authorized in each of the possible time intervals. One possibility is to consider the interval $[s+\delta, e+\delta+w]$ to be the time interval of b_2, as all the possible time intervals are inside this interval. However, when b_1 is performed (we know t) we get b_2's original time interval and have to shrink the large time interval $[s+\delta, e+\delta+w]$ appropriately with respect to t. When we use this approach, it will yield runtime overhead for managing obligations and accountability will be less likely to hold due to increasingly large obligation time intervals.

To mitigate this problem, we assume that b_2's time interval will be at a fixed distance δ from the time interval of b_1 (see figure 6.1(b)). We assume δ is measured from the end time of b_1's interval. Thus, b_2's stipulated time interval in our approach will be $[e+\delta, e+\delta+w]$. This approach will ensure that the cascading obligation's time interval is fixed. For a discretionary action

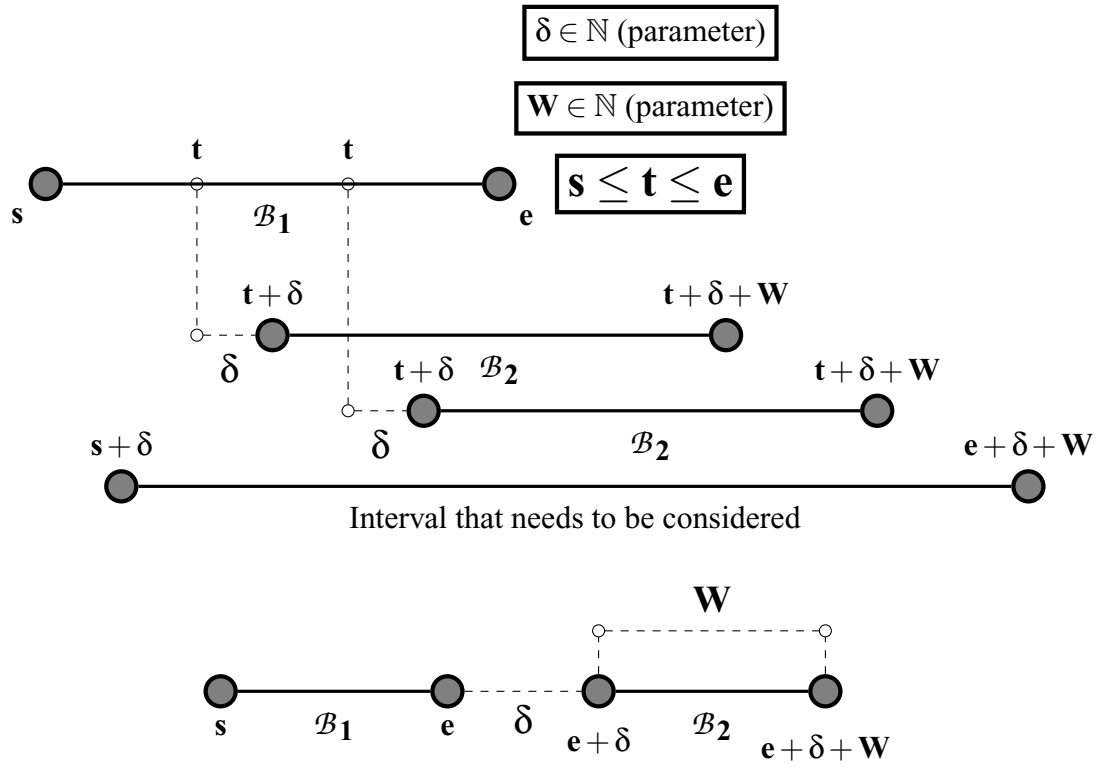

Figure 6.1: Time Interval of the Incurred Obligation. (A) Previous Approach (top). (B) Our Approach (bottom)

incurring an obligation, we replace e with t, the time at which the action is performed. Thus, in our model obligations have the form $b = \langle u, a, \vec{o}, t_s, t_e, \delta, w \rangle$. In section 6.4.4.2, we further augment our obligations to contain one additional field (repetition). We also extend the notion of the transition relation to allow cascading obligations.

6.3.2 Selection of Obligatee

We now present some strategies by which a user who incurs obligations, can be specified in the policy. When a user executes an action, this can generate other obligations to the user who initiated the action, or for other users. The user who incurs an obligation is called an *obligatee*. Existing work [67,109] does not discuss how obligatees are specified in the policy. To allow the specification of obligatees, we extend the policy rules to include an extra field called "obligatee". Thus, policies

now have the following form: $p = a(u,\vec{o}) \leftarrow cond(u,\vec{o},a) : F_{obl}(obligatee,s,u,\vec{o},\delta,w)$. Note that, F_{obl} function returns a set of obligations and is guaranteed to terminate in a constant time.

Explicit User: In this strategy, the obligatee is hard-coded in the policy rule.

Example 79 (Explicit User)**.** *Let us consider the following policy rule,* $p_0 : check(u, log) \leftarrow (u \in manager) : F_{obl}(Bob,s,u,log,\delta = 10,w = 5)$. *This rule authorizes a user in the role of manager to check the log and it will incur an obligation for Bob. The action associated with the obligation will be specified in the body of the* F_{obl} *function. For clarity, we do not show the body of the* F_{obl} *function.*

Self, Target, and Explicit User: In this strategy, the obligatee field can contain "Self", "Target", or an explicit user. When a policy rule's obligatee field contains "Self", it represents that the user who initiates the action, authorized by the current policy, will incur the associated obligations.

Example 80 (Self)**.** *Let us consider a policy rule,* $p_1 : grant(u,\langle u_t, programmer\rangle) \leftarrow ((u \in manager) \wedge (u_t \in employee)) : F_{obl}(Self,s,u,\langle u_t, programmer\rangle,\delta = 10,w = 5)$. *This rule authorizes a user u in the role of manager to grant a new role programmer to a target user* u_t *in the role of employee and this will incur an obligation for u. Let us consider that manager Bob grants the employee Alice the role programmer. This will generate a new obligation for Bob.*

On the other hand, whenever the policy rule is authorizing an administrative action and the obligatee field of that policy rule contains "Target", it signifies that the target of the original administrative action authorized by this policy would incur the obligations specified by it.

Example 81 (Target)**.** *Let us consider a policy rule,* $p_2 : grant(u,\langle u_t, programmer\rangle) \leftarrow ((u \in manager) \wedge (u_t \in employee)) : F_{obl}(Target,s,u,\langle u_t, programmer\rangle,\delta = 10,w = 5)$. *The policy rule* p_2 *is similar to* p_1 *except it incurs an obligation for the target. As in the previous example, when Bob grants the role programmer to Alice, Alice will incur an obligation as she is the target of the action.*

Role Expression: In this approach, the obligatee field can contain a boolean role expression. Each literal in the boolean expression is either a positive or a negative role assignment. The system can select a user to be the obligatee provided that the user satisfies the role expression when the original action is performed.

Example of Role Expression. Let us assume the following policy rules in figure 6.2. Policy rule p_1 allows a registered user to submit a paper and this in turn creates an obligation for a user (obligatee) in role reviewer to submit the review of the paper. Rule p_2 authorizes a user in role reviewer to submit a review of a paper and it incurs an obligation for a user in the role PC_chair that requires him to make a decision on the paper. Rule p_3 authorizes a user in role PC_chair to submit a decision for a paper and it incurs an obligation for the same user submitting the decision to notify the corresponding author. Rule p_4 authorizes a user in the role PC_chair to notify the author of a paper. Now, consider the following situation. The set of current users of the system is $\gamma.U = \{Alice, Bob, Carol\}$ and their current role assignments are $\gamma.UA = \{\langle Alice, registeredUser\rangle, \langle Bob, reviewer\rangle, \langle Carol, PC\_Chair\rangle\}$. Let us assume Alice submits a paper on 07/01/2012 and according to p_1 Bob (in role reviewer) will get the following obligation $\langle Bob, submit - Review, \langle Alice, paper\rangle, 07/03/2012, 07/10/2012\rangle$. According to p_2, this obligation in turn will incur the obligation $\langle Carol, submitDecision, \langle Alice, paper\rangle, 07/11/2012, 07/12/2012\rangle$ for Carol (in role PC_chair). According to p_3 when Carol submits the decision, she incurs the obligation $\langle Carol, notify, \langle Alice, paper\rangle, 07/13/2012, 07/14/2012\rangle$.

In the current work, we use the "Self, Target, and Explicit User" scheme to specify the obligatee. Although this approach is not the most general strategy to specify an obligatee, our accountability decision procedure requires every obligation to have an individual user statically associated with it. However, in the "Role expression" scheme multiple users can satisfy the role expression specified in the obligation policy rule. Thus, we have two possible interpretations of strong accountability. One of which says that the newly incurred obligation will maintain accountability if

$$
\begin{aligned}
&p_1 : submit(u, paper) \leftarrow (u \in registeredUser) : \\
&F_{obl}(reviewer, s, u, paper, 2\ days, 1\ week) \\
&\{ \\
&\quad (Choose\, u_1\, such\, that\, u_1 \in reviewer) \\
&\quad\quad submitReview(u_1, \langle u, paper \rangle) \\
&\} \\
&p_2 : submitReview(u, \langle author, paper \rangle) \leftarrow (u \in reviewer) : \\
&F_{obl}(PC\_Chair, s, u, \langle author, paper \rangle, 1\ day, 1\ day) \\
&\{ \\
&\quad (Choose\, u_1\, such\, that\, u_1 \in PC\_Chair)\{ \\
&\quad\quad submitDecision(u_1, \langle author, paper \rangle); \\
&\quad \} \\
&\} \\
&p_3 : submitDecision(u, \langle author, paper \rangle) \leftarrow (u \in PC\_Chair) : \\
&F_{obl}(Self, s, u, \langle author, paper \rangle, 1\ day, 1\ day) \\
&\{ \\
&\quad\quad notify(u, \langle author, paper \rangle) \\
&\} \\
&p_4 : notify(u, \langle author, paper \rangle) \leftarrow (u \in PC\_Chair) : \varnothing
\end{aligned}
$$

Figure 6.2: Example Policy Rules with Role Expressions

at least one of the users satisfying the role expression is authorized to perform the obligation during its whole time interval. The other interpretation requires that every user who satisfies the role expression must be authorized to perform the obligation during its whole time interval. Although both of the interpretations have practical utility, the choice of interpretation will influence the time complexity of the accountability decision procedure. We leave the adoption of the role expression scheme for specifying the obligatee as a future work.

6.4 Strong Accountability

When considering user obligations that depend on and affect authorization, we can have a situation where a user can incur obligations which she is not authorized to fulfill. However, without any

preemptive approach, the obligatee will realize the absence of proper authorization in the time she attempts the obligation. This can hinder the proper functioning of the system. To mitigate this, Irwin *et al.* [67] introduced a property of the authorization state and the current obligation pool, accountability, that ensures that all the obligatory actions are authorized in some part of their time interval. Based on when they are supposed to be authorized in their respective time intervals, they introduced two variations of the accountability property, weak and strong. Pontual *et al.* [109] have shown that deciding weak accountability is co-NP complete for an obligation model using mini-RBAC and mini-ARBAC whereas, deciding strong accountability is polynomial. Due to its high complexity, we do not consider weak accountability in this work. Roughly, strong accountability requires that as long as prior obligations has been performed in their stipulated time interval, each obligatory action needs to be authorized no matter what policy rules are used to authorize the other obligations and no matter when they are performed in their time interval.

In this section, we first present the definition of strong accountability presented by Pontual *et al.* [109]. As mentioned before, their definition of strong accountability does not take into account cascading obligations. We call their notion of the property *restricted strong accountability*. We then refine their notion of the property and give a recursive definition of it considering the presence of cascading obligations. We go on to show that deciding strong accountability in presence of cascading obligations in general is NP-hard. We then consider some special cases of cascading obligations and give a tractable decision procedure for deciding strong accountability in their presence.

6.4.1 Restricted Strong Accountability

Roughly stated, under the assumption that all previous obligations have been fulfilled in their time interval, strong accountability property requires that each obligation be authorized throughout its entire time interval, no matter when during that interval the other obligations are scheduled, and no matter which policy rules are used to authorize them.

Given a pool of obligations B, a *schedule* of B is a sequence $b_{0..n}$ that enumerates B, for $n =$

$|B|-1$ (including the possibility that B may be countably infinite). A schedule of B is *valid* if for all i and j, if $0 \leq i < j \leq n$, then b_i.start $\leq b_j$.end. This prevents scheduling b_i before b_j if b_j.end $< b_i$.start. Given a system state s_0, and a policy $\mathcal{P}$, a proper prefix $b_{0..j}$ of a schedule $b_{0..n}$ for B is *authorized by* policy-rule sequence $p_{0..j} \subseteq \mathcal{P}^*$ if there exists s_{j+1} such that $s_0 \xrightarrow{\langle b,p\rangle_{0..j}} s_{j+1}$.

Definition 82 (Restricted Strong accountability)**.** *Given a state $s_0 \in \mathcal{S}$ and a policy $\mathcal{P}$, we say that s_0 is* strongly accountable *(denoted by RStrongAccountable$(s_0,\mathcal{P})$) if for every valid schedule, $b_{0..n}$, every proper prefix of it, $b_{0..k}$, for every policy-rule sequence $p_{0..k} \subseteq \mathcal{P}^*$ and every state s_{k+1} such that $s_0 \xrightarrow{\langle b,p\rangle_{0..k}} s_{k+1}$, there exists a policy rule p_{k+1} and a state s_{k+2} such that $s_{k+1} \xrightarrow{\langle b,p\rangle_{k+1}} s_{k+2}$.*

6.4.2 Unrestricted Strong Accountability

In this section, we provide a formal specification of the strongly accountability property under the cascading obligation assumption. The strong accountability definition presented by Pontual *et al.* [109] disallowed cascading obligation. We refine their notion of strong accountability that takes into consideration cascading obligations. We start by defining three auxiliary functions that are going to be used in the definition of strong accountability.

Definition 83 (Ψ function)**.** Ψ *is a function that takes as input an obligation $\hat{b}$ and a fixed set of policy rules $\mathcal{P}$ and returns a set of sets of obligations $\hat{B}$ in which each element represents a set of obligations that $\hat{b}$ can incur according to the F_{obl} function of a policy rule authorizing it. The formal specification and the type of Ψ are precisely shown below.*

$$\Psi : B \times \mathcal{F}\,\mathcal{P}(\mathcal{P}) \rightarrow \mathcal{F}\,\mathcal{P}(\mathcal{F}\,\mathcal{P}(B)) \tag{6.1}$$

$$\Psi(b = (u,a,\vec{o},t_s,t_e,\delta,w),\mathcal{P}) = \tag{6.2}$$

$$\Big\{ F_{obl}(obligatee,s,u,\vec{o},\delta,w) \;\; p = (a(u,\vec{o}) \leftarrow cond(u,\vec{o},a) : \tag{6.3}$$

$$F_{obl}(obligatee,s,u,\vec{o},\delta,w)) \wedge (p \in \mathcal{P}) \Big\} \tag{6.4}$$

Definition 84 (Π function)**.** Π *is a function that takes as input a set of obligations $\bar{B}$ and a fixed set*

of policy rules $\mathcal{P}$ and returns a set of sets of obligations $\tilde{B}$ in which each element is a possible set of obligations that all the obligations of $\bar{B}$ can incur. In short, $\tilde{B}$ is the set containing all possible combination of obligations that $\bar{B}$ can incur. The formal specification of Π and its type are shown below.

$$\Pi : \mathcal{F}\,\mathcal{P}(B) \times \mathcal{F}\,\mathcal{P}(\mathcal{P}) \rightarrow \mathcal{F}\,\mathcal{P}(\mathcal{F}\,\mathcal{P}(B))$$

$$\Pi(b_{1\ldots n} = (u_{1\ldots n}, a_{1\ldots n}, \vec{o}_{1\ldots n}, t_{s_{1\ldots n}}, t_{e_{1\ldots n}}, \delta_{1\ldots n}, w_{1\ldots n}), \mathcal{P})$$
$$= \{\mathcal{B} \subseteq \mathcal{B} | \forall i \in 1\ldots n. \Psi(b_i, \mathcal{P}) \neq \emptyset \rightarrow \exists f \in \Psi(b_i, \mathcal{P}).f \subseteq BB\}$$

Definition 85 (Ξ function)**.** *Ξ is a function that takes as input a set of sets of obligations and a set of policy rules and applies Π to each of set of obligations and then combines the results. This allows us to find the set of all possible sets of obligations generated by a given set of possible obligations. For simplicity in later definitions, we also include in the output sets, the original sets which generated those obligations.*

$$\Xi : \mathcal{F}\,\mathcal{P}(\mathcal{F}\,\mathcal{P}(B)) \times \mathcal{F}\,\mathcal{P}(\mathcal{P}) \rightarrow \mathcal{F}\,\mathcal{P}(\mathcal{F}\,\mathcal{P}(B))$$

$$\Xi(\tilde{B}, \mathcal{P}) = \{\forall \bar{B} \in \tilde{B}, \bigcup_{B \in \Pi(\bar{B})} B \cup \bar{B}\}$$

Note that, each action a in our system can be authorized by multiple policy rules. Each of the policy rules authorizing a can incur different obligations. Furthermore, it can be the case that among different possible obligations incurred due to a, some of them maintain accountability and some of them do not. Provided that the policy allows infinite cascading obligations and a is authorized by multiple policy rules, each of which incurs different obligations, then all possible obligations incurred due to a can be modeled as a tree (possibly infinite). Based on this, we can have two interpretations of strong accountability, *existential* and *universal*. The existential interpretation requires that there exists a single path in the tree in which all the obligatory actions maintain accountability when added to the current pool of obligations. The universal interpretation

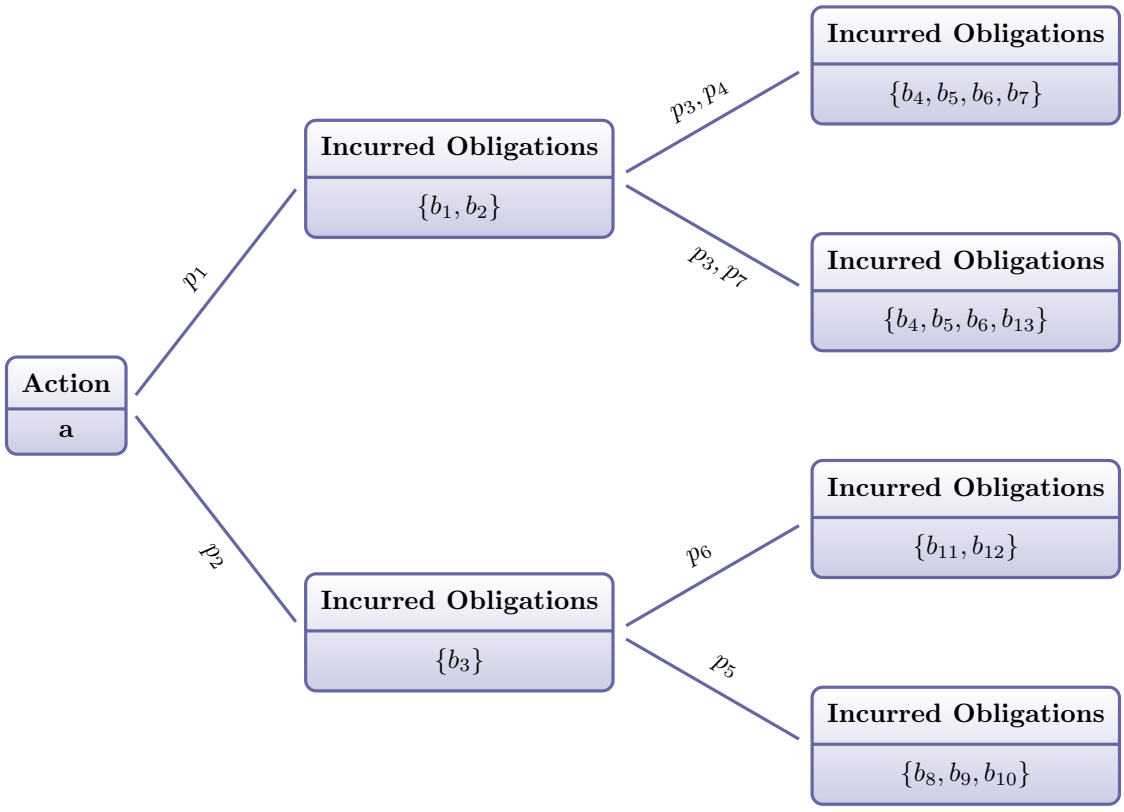

Figure 6.3: Possible obligations incurred by action **a**

is the dual and requires that all the paths in the tree maintain strong accountability. We think the universal interpretation is too strong. As a result of which, we use the existential interpretation of the strong accountability property and define it just below. However, the following definition can be extended to express the universal interpretation of strong accountability.

In the example (in figure 6.3), let us consider the current accountable pool of obligations is B. We want to know whether performing a would maintain accountability. Let us consider that a can be authorized by policy rule p_1 or p_2. When a is authorized using p_1, it incurs obligations b_1 and b_2. However, when p_2 is used to authorize a, it incurs obligation b_3. Then, b_1 can be authorized by p_3 and b_2 can be authorized by either p_4 or p_7 and so on. In the existential interpretation, if one of the following sets is accountable then adding a would maintain accountability: $B \cup \{b_1, b_2, b_4, b_5, b_6, b_7\}$, $B \cup \{b_1, b_2, b_4, b_5, b_6, b_{13}\}$, $B \cup \{b_3, b_8, b_9, b_{10}\}$, and $B \cup \{b_3, b_{11}, b_{12}\}$. The universal interpretation requires all the above sets to be accountable.

In order to formalize this, we need to first define the set of possible future sets of obligations using the Ξ function. In particular, we wish to define a series of sets of sets of obligations. We will define $\Xi^0(\tilde{B}) = \tilde{B}$ and for all integers $i > 0$, we define $\Xi^i(\tilde{B}) = \Xi(\Xi^{i-1}(\tilde{B}))$. Further, we define $\Xi^\infty = \lim_{i\to\infty} \Xi^i$. Thus, given a starting set of obligations B, we can define the set of all sets of possible obligations which can arise from B as $\Xi^\infty(\{B\})$. Note that this is a countable, but possibly infinite, set of countable, but possibly infinite, sets of obligations. Because we are allowing for potentially infinitely cascading obligations, this is necessary.

Definition 86 (Strong accountability)**.** *Given a state $s_1 \in \mathcal{S}$, in which $s_1.B$ is a strongly accountable pool of obligations, a policy $\mathcal{P}$, a set of new obligations B_c that can generate cascading obligations, we say that the state s (where $s.B := s_1.B \cup B_c$) is existentially strongly accountable (denoted by StrongAccountable$(s, \mathcal{P})$) if and only if*

$$\exists B'_c \in \Xi^\infty(s.B, \mathcal{P}).RStrongAccountable(s[B := B'_c \cup s.B], \mathcal{P})$$

6.4.3 Computational Complexity

This section discusses the computational complexity of deciding strong accountability in presence of cascading obligations. Prior work [67, 109] disallowed cascading obligations while deciding strong accountability. They give intuitive discussions about why deciding strong accountability in presence of cascading obligation is difficult. However, they did not present any theoretical results regarding this.

We now discuss the computational complexity of deciding strong accountability in presence of cascading obligations. We have the following theorem which states that the strong accountability decision problem is NP-hard. We reduced the Hamiltonian path problem for graphs to the decision of unrestricted strong accountability property.

Theorem 87. *Given a strongly accountable pool of obligations B, a new obligation b, an initial authorization state γ, and a mini-ARBAC policy Φ that allows cascading obligations and also*

allows each action to be authorized by multiple policy rules, deciding whether $B \cup B_c \cup \{b\}$ is strongly accountable (either in existential or universal interpretation) is NP-hard in the size of B, γ, and Φ, where B_c is the set of cascading obligations incurred by b.

Proof. To show that deciding accountability is NP-hard when the input mini-ARBAC policy allows cascading of obligations and also allows each action to be authorized by multiple policy rules, we reduce the *Hamiltonian path problem* for directed graphs to it. Given a directed graph $G(V,E)$ where V is the set of vertices and E is the set of edges, the Hamiltonian path problem asks whether there is a simple path in the graph $G(V,E)$, such that it contains all the vertices and each of the vertices in V are visited only once in that path. To this end, we present a polynomial time algorithm which reduces a Hamiltonian path problem instance to the problem instance of deciding accountability in presence of cascading obligations. Thus, the graph will have a Hamiltonian path when the reduced problem instance of deciding accountability (when we say deciding accountability, we actually mean deciding the stronger version of the accountability) yields true.

Let us consider a Hamiltonian path problem instance where the directed graph $G(V,E)$ is given. Let us also assume that the total number of vertices in the graph is n (*i.e.*, $|V| = n$). We also consider the vertices in V are uniquely labeled using a number from $1...n$. Each edge $e \in E$ has the form (v_i, v_j) where $1 \leq v_i, v_j \leq n$.

Now, we will try to construct a problem instance of deciding accountability that corresponds to the Hamiltonian path problem instance. In the accountability decision problem instance, consider that we have only two users, namely u_0 and u_1. Furthermore, let us also consider we have the following roles, $ar_1, r_0, \hat{r}, r_{1...n}, v_{1...n}$. Each of the role $v_{1...n}$ corresponds to the vertex of the graph. At each point of time, the roles (in $v_{1...n}$) that the user u_1 currently possesses, denote the vertices that we have already visited in the current path in the graph. Additionally, each of the role $r_{1...n}$ also corresponds to the vertices of the graph. This is used to determine the current vertex being inspected and will be explained later.

The current role assignment of users in the system is as following.

$$user\_assignment\{\langle u_0, \{ar_1\}\rangle, \langle u_1, \{\emptyset\}\rangle\}$$

Now, we turn our attention to the input mini-ARBAC policy of the problem instance. The policy for the corresponding problem instance would be like following:

For each vertex i in the graph $G(V,E)$, we have a *can_assign* rule, each of which has the following form where $1 \leq i \leq n$.

$$\begin{aligned} &can\_assign(ar_1, true, r_0): \\ &\qquad \{ \\ &\qquad \langle u_0, Grant, u_1, r_i, 1, 1\rangle, \\ &\qquad \langle u_0, Grant, u_1, \hat{r}, 5*n, 10\rangle \\ &\qquad \} \end{aligned} \tag{6.5}$$

Furthermore, for each vertex i in the graph $G(V,E)$, we have another *can_assign* rule, each of which has the following form where $1 \leq i \leq n$.

$$can\_assign(ar_1, true, v_i): \{\emptyset\}, 1 \leq i \leq n \tag{6.6}$$

For each edge $e = (x,y) \in E$, we have one policy rule like the following,

$$\begin{aligned} &can\_assign(ar_1, \neg v_x \wedge \neg v_y \wedge \neg r_y, r_x) : \\ &\qquad \{ \\ &\qquad \langle u_0, Grant, u_1, v_x, 1, 1\rangle, \\ &\qquad \langle u_0, Grant, u_1, r_y, 1, 1\rangle \\ &\qquad \} \end{aligned} \tag{6.7}$$

We also have the following policy rules for each of the vertices. The policy specifies that if r_i is the last role among the roles $r_{1\ldots n}$ that is being granted to u_1 then there is no need to incur any further obligations and it could be the end of the simple path where each vertex is representing granting of a role to u_1.

$$can\_assign(ar_1, \bigwedge_{(1\leq j\leq n)\wedge(i\neq j)} r_j, r_i) : \{\emptyset\}, 1 \leq i \leq n \tag{6.8}$$

The following policy rule allows a user in role ar_1 to grant the user who has all the roles in $r_{1\ldots n}$ to be assigned the role $\hat{r}$.

$$can\_assign(ar_1, \bigwedge_{1\leq i\leq n} r_i, \hat{r}) : \{\emptyset\} \tag{6.9}$$

In the problem instance, consider $B = \emptyset$ and the obligation we want to add is $b = \langle u_0, Grant, u_1, r_0, [1,2]\rangle$. Furthermore, we consider that when an action is authorized by multiple policy rules, then we select the one to use non-deterministically. We also consider that adding an obligation b in the accountable pool of obligations B will not violate the accountability property as long as there is a cascading pool of obligations B_c incurred by b, in which all the obligations in $B \cup B_c$ are authorized. However, in our instance, as B is empty, we just need to check whether there exists a B_c in which all the obligations are authorized.

Now when we consider the new obligation $b = \langle u_0, Grant, u_1, r_0, [1,2] \rangle$, no matter what policy rules of the form (1) we use, it will incur an obligation $\hat{b} = \langle u_0, Grant, u_1, \hat{r}, [5*n+2, 5*n+12] \rangle$. Thus, one of the pre-condition of b not violating accountability is that the obligation $\hat{b}$ be authorized. We can see from the policy rule (5) that it is the only policy rule that can possibly authorize it. It however requires that the user u_1 possesses all the roles in $r_{1...n}$. Now u_1 to get all the roles the policy rules that can be used are either of form (1), (3), or (4). These policy rules make sure that the only way a user can get all the roles in $r_{1...n}$ if there is a simple path in the corresponding graph containing all the vertices only once. The policy rules of form (1) ensure that one can start looking for such a path in any of the vertices of the graph. The policy rules of form (3) on the other hand encodes the edge relationship and also impose a constraint that the only way to get a role r_i through an obligation incurred due to granting a predecessor role (predecessor vertex) and if she did not have the role before (not visiting a vertex twice). The policy rules of form (4) precisely specifies that whenever u_1 possesses the last role of the roles in $r_{1...n}$, there is no need to search anymore as we have already found a simple path containing all the roles. Thus, $G(V,E)$ will have a Hamiltonian Path if the accountability decision procedure yields true.

6.4.4 Special Cases of Cascading Obligation

As in section 6.4.3, deciding accountability in presence of cascading obligations is NP-hard. Our goal is to find certain special cases of cascading obligations for which the accountability decision is tractable. This section introduces two such special cases.

6.4.4.1 Finite Cascading Obligation

In this special case of cascading obligation, we consider that the policy is written in a way that the maximum number of new obligations incurred by a single obligation is bounded by a constant. Furthermore, we also consider that each action, object pair is authorized by only one policy rule. We also assume that the policy rules are free of cycles prohibiting infinite cascading. This can be achieved by the following static policy analysis.

Static Policy Analysis to Check for Cycles. We now present an algorithm that can statically verify the absence of cycles among policy rules. An absence of any cycles in the policy guarantees that there is no provisions for discretionary actions to incur infinite cascading obligations. The following algorithm (cf. algorithm 6.1) proceeds in a depth first search manner inspecting all possible action, object pair. Considering they are authorized with respect to a specific policy rule, it then gathers the obligations incurred due to that action, object pair according to the policy rule. It then calls the procedure *findCycles* (cf. algorithm 6.2) to check whether we can reach an action, object pair we have already seen before. If this is the case then it terminates and reports that there is a cycle in the policy.

Algorithm 6.1 *InfiniteCascading*($\langle \gamma, \psi \rangle$)

Input: A policy $\langle \gamma, \psi \rangle$
Output: Returns **true** if $\langle \gamma, \psi \rangle$ allows infinite cascading obligations

```
1: map⟨pair⟨action, object⟩, boolean⟩ obligationSeen
2: obligationSeen.clear()
3: for each possible action, object pair ⟨a,o⟩ in the system do
4:    obligationSeen.insert(⟨⟨a,o⟩,true⟩)
5:    if findCycles(a,o,obligationSeen) == true then
6:       return true
7:    obligationSeen.delete(⟨⟨a,o⟩,true⟩)
8: return false
```

Algorithm 6.2 *findCycles*(*action a*, *object o*, *map obligationSeen*)

Input: An action a, object o, and a map data structure which represents the obligations/action, object pairs we have already seen.
Output: Returns **true** if $\langle a, o \rangle$ can generate an infinite cascading of obligations

```
1: for each possible policy rule p ∈ P do
2:    if p.a = a ∧ p.o⃗[0] = o then
3:       obligations B = p.Fobl()
4:       for each obligation b ∈ B do
5:          if obligationSeen.find(⟨b.a, b.o⃗[0]⟩) = true then
6:             return true/* cycle found */
7:          else
8:             obligationSeen.insert(⟨⟨b.a, b.o⃗[0]⟩, true⟩)
9:             if findCycles(b.a, b.o⃗[0], obligationSeen) = true then
10:               return true/* cycle found */
11:            obligationSeen.delete(⟨⟨b.a, b.o⃗[0]⟩, true⟩)
12: return false
```

6.4.4.2 Repetitive Obligation

Repetitive obligations occur recurrently after a fixed amount of time. A real life example of repetitive obligation can be found in the chapter 6803(a) of Gramm-Leach-Bliley Act (GLBA) [2]. According to the regulation, a financial institution must send a customer an annual privacy notice as long as the individual is a customer. Note that, we cannot specify repetitive obligations in our model directly. For this, we follow Ni *et al.* [100] to augment our obligations with an extra field that specifies the number of repetition (denoted by ρ). We allow both finite and infinite repetitive obligation. Now, let us consider an obligation $b = \{u, a, \vec{o}, t_s, t_e, \delta, \rho, w\rangle\}$. This obligation is considered to be infinite repetitive when $\rho = I$ or finite repetitive when $\rho \in \mathbb{N}$ and $\rho > 1$. For finite and infinite repetition of the obligation the possible time intervals of the recurring obligation are the following.

- **Finite Repetitive:** $[t_s, t_e], [t_e + \delta, t_e + \delta + w], \cdots [t_s + (\rho - 1)(w + \delta), t_e + (\rho - 1)(w + \delta)]$.
- **Infinite Repetitive:** $[t_s, t_e], [t_e + \delta, t_e + \delta + w], \cdots [t_s + (n - 1)(w + \delta), t_e + (n - 1)(w + \delta)], \cdots$ where $n \in \mathbb{N}$.

Finite Repetitive Obligations. This kind of obligation recurs finitely after a fixed amount of time. For instance, $b = \{Bob, check, log, t_s = 5, t_e = 8, \delta = 2, \rho = 3, w = 3\}$ will generate 3 obligations $\{Bob, check, log, 5, 8\}$, $\{Bob, check, log, 10, 13\}$, and $\{Bob, check, log, 15, 18\}$.

Infinite Repetitive Obligations. This kind of obligations on the other hand recurs indefinitely. For example, $b = \{Bob, check, log, t_s = 5, t_e = 8, \delta = 2, \rho = I, w = 3\}$ will generate the following infinite number of obligations: $\{Bob, check, log, 5, 8\}$, $\{Bob, check, log, 10, 13\}$, $\cdots$.

6.4.5 Algorithm

As deciding accountability in presence of cascading obligations is NP-hard, we simplify our accountability decision problem by imposing several restrictions on the problem. The restrictions

are: (1) We consider each action, object pair is authorized by one policy rule, prohibiting disjunctive choices. (2) We require that the policy is free of cycles which prohibits obligations to incur infinite number of new obligations. (3) We disallow role expressions to specify the obligatee of the new obligation. (4) We also disallow finite cascading obligations to incur repetitive obligations. (5) We also disallow repetitive obligations to incur non-repetitive cascading obligations.

Considering these restrictions, strong accountability can be decided in polynomial time of the policy size, number of obligations, and the number of new obligations that needs to be considered. The algorithm (algorithm 6.3) decides whether adding an obligation to an accountable pool of obligations, maintains accountability. The algorithm takes as input the accountable pool of obligations B (containing the finite cascading, finite repetitive, and infinite repetitive obligations), the current authorization state γ of the system, a mini-ARBAC policy Φ, and the new obligation b. It returns true when adding $B \cup \{b\} \cup B_c$ is strongly accountable where B_c is the new set of obligations incurred by b. Note that, the time complexity of the algorithm additionally depends on the type of the new obligation to be added and also the number of infinite repetitive obligations that needs to be unrolled. The complexity of the algorithm is precisely described later.

In the algorithm 6.3, the new obligation b can either incur no new obligations, finite cascading obligations, finite repetitive obligations, or infinite repetitive obligations. Based on what kind of new obligation(s) b incurs, we have to take different course of actions. The main idea behind the algorithm is to unroll a finite amount of new obligations and use the non-incremental algorithm presented by Pontual *et al.* [109] to decide whether the original pool of obligation in addition with the new obligation and finitely unrolled obligation is strongly accountable. The way in which each type of obligation is unrolled is presented in the following discussion.

Unrolling Finite Cascading Obligations To unroll the chain of cascading obligations incurred by b, Algorithm 6.3 uses procedure *UnrollCascading* described in Algorithm 6.4. This procedure is an adaptation of the breadth-first search algorithm. Recall that, we disallow infinite cascading obligations which guarantees that the procedure *UnrollCascading* will terminate. Furthermore, we

Algorithm 6.3 *StrongAccountableCascading* (γ, Φ, B, b)

Input: A policy $\langle \gamma, \Phi \rangle$, a strongly accountable obligation set B, and a new obligation b that generates cascading obligations.

Output: returns **true** if addition of b to the system preserves strong accountability.

```
1:  if b.ρ = 1 then
2:      B_final := B ∪ UnrollCascading(γ, Φ, b);
3:  else if b.ρ = I then
4:      B_final := B ∪ {b};
5:  else
6:      B_final := B ∪ UnrollFiniteRepetitive(γ, Φ, b);
7:  m := MaxEndTime(B_final);
8:  B_final := B_final ∪ UnrollInfiniteRepetitive(γ, Φ, B_final, m);
9:  for each obligation b* ∈ B_final do
10:     if b*.a = grant or revoke then
11:         InsertIntoDataStructure(b*);
12: for each obligation b* ∈ B_final do
13:     if Authorized (γ, Φ, B_final, b*)= false then
14:         return false
15: return true
```

also impose the restriction that each action, object pair can be authorized by only one policy rule. As a result of which, the new obligations incurred by a fixed obligation will be finite and fixed. For this, we use the function Ψ (discussed in section 6.4.2) that takes an obligation b and set of policy rules and returns a set of set of obligations which can be possibly incurred by b. Due to the restriction above, the result of Ψ will be a single set of obligations B_f that can be incurred by b. The different fields of each obligation $\hat{b} \in B_f$ will depend of the fields of b and the policy rule that authorizes b.

Algorithm 6.4 *UnrollCascading* (γ, Φ, b)

Input: A policy $\langle \gamma, \Phi \rangle$ and a new obligation b.

Output: returns a set of cascading obligations B that is generated by b.

```
1:  B = ∅;
2:  queue < obligation > q;
3:  q.push(b);
4:  while !q.empty() do
5:      b = q.front(); B := B ∪ {b};
6:      q.pop(); B' := Ψ(b, Φ);
7:      for each obligation b* ∈ B' do
8:          q.push(b*);
9:  return B
```

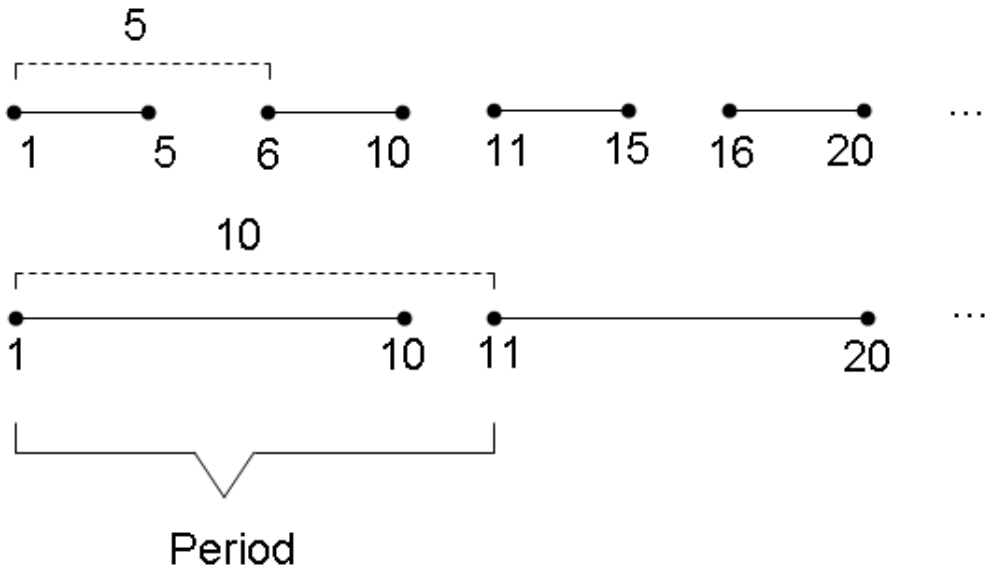

Figure 6.4: Computing Period of Infinite Repetitive

Unrolling Finite Repetitive Obligations When the new obligation we want to add (b) is a finite repetitive obligation ($b.\rho \in \mathbb{N}$ and $b.\rho > 1$), we use the procedure *UnrollFiniteRepetitive* described in Algorithm 6.5 to unroll it appropriately. We follow the procedure presented in section 6.4.4 to unroll finite repetitive obligations. Thus, for the obligation b, the procedure *UnrollFiniteRepetitive* clones b, varying only the time intervals of the new obligations based on $b.\delta$. The exact number of copies of b that are unrolled will depend on $b.\rho$.

Algorithm 6.5 *UnrollFiniteRepetitive* (γ, Φ, b)

Input: A policy $\langle \gamma, \Phi \rangle$ and a finite repetitive obligation b.
Output: returns a set of unrolled obligations B that is generated by b.

1: $B = \emptyset$; $i := 1$;
2: **while** $i \leq b.\rho$ **do**
3: $\quad b_i := b$; $b_i.t_e := (b.w - b.\delta) \times i + b.t_s - b.\delta$;
4: $\quad b_i.t_s := b_i.t_e - w$; $B := B \cup \{b_i\}$; $i := i + 1$;
5: **return** B

Unrolling Infinite Repetitive Obligations When the obligation we want to add (b) is an infinite repetitive obligation, Algorithm 6.3 uses procedure *UnrollInfiniteRepetitive*, described in algorithm 6.6, to unroll a finite amount of it. Let us consider $B_i \subseteq B$ is the set of infinite repetitive obligations. Note that, $b \in B_i$. First, we find the *overall period* of all the obligations in B_i at which

the infinite repetitive obligations repeat themselves. In figure 6.4 we have two infinite repetitive obligations, $b_1 = \{u_1, a, o, t_s = 1, t_e = 5, \delta = 1, \rho = I, w = 4\}$ and $b_2 = \{u_1, a, o, t_s = 1, t_e = 10, \delta = 1, \rho = I, w = 9\}$. It is clear that after time 11, we see a pattern formed by the obligations, this is the overall period. The overall period is the least common multiple (LCM) of the periods of each $b_i \in B_i$. For each infinite repetitive obligation b_i, the period of b_i is given by $b_i.\delta + b_i.w$. Once the

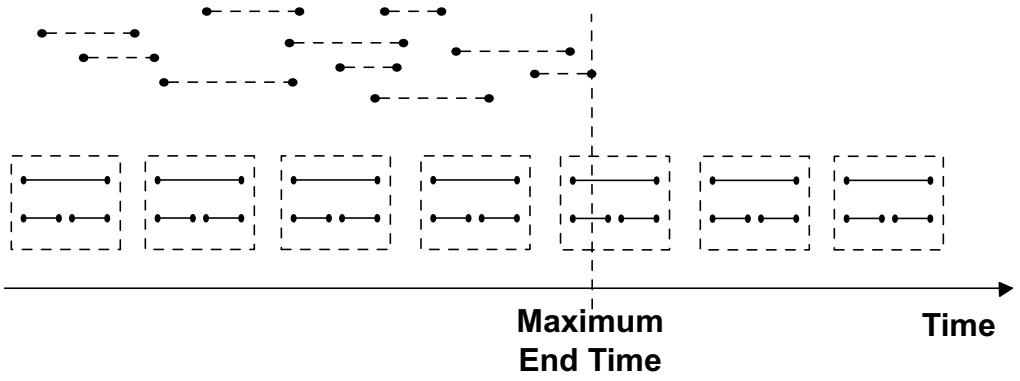

Figure 6.5: Unrolling Infinite Repetitive Obligations

period is computed, we check to see whether the overall period is greater than the maximum end time of the finite obligations (repetitive or non-repetitive). If this is the case, we just need to unroll the infinite repetitive obligations three periods (to be safe). Otherwise, we unroll the infinite obligations until the maximum time, and then we unroll two additional periods (figure 6.5). In the current pool of obligations let us assume that the only type of obligations present are the infinite repetitive obligations. When we have calculated the overall period of these infinite repetitive obligations, part of the authorization state influencing the permissibility of the infinite repetitive obligations, after each of these period should be equivalent to the authorization state before, if the system is accountable. If the authorization state is not necessarily equivalent, this will be revealed when the second repetition is analyzed. Thus, we do not need to analyze the infinite repetitive obligations beyond two repetitions. Similarly, when we have other types of obligations residing in the current pool of obligations, we can safely unroll the infinite repetitive obligations for two additional period after the maximum end time of the finite obligations and soundly decide accountability.

We now briefly summarize the non-incremental algorithm for deciding strong accountability

Algorithm 6.6 *UnrollInfiniteRepetitive* (γ, Φ, B, m)

Input: A policy $\langle \gamma, \Phi \rangle$, a set of obligations, where $B_i \subseteq B$ is a set of infinite repetitive obligations, and m representing the last time point where a non-infinite obligation happens.

Output: returns a set of unrolled obligations B' that is generated by B.

1: $B' = \emptyset$; $period = LCM(B)$
2: **if** $period > m$ **then**
3: $\quad finalTime := period * 3$;
4: **else**
5: $\quad finalTime := (\lceil (m/period) \rceil + 2) \times period$;
6: **for** each obligation $b' \in B_i$ **do**
7: $\quad end := b'.t_e$;
8: $\quad$ **while** $end <= finalTime$ **do**
9: $\quad\quad b_i := b'$; $b_i.t_e := (b'.w - b'.\delta) \times i + b'.t_s - b'.\delta$;
10: $\quad\quad b_i.t_s := b_i.t_e - w$; $B' := B' \cup \{b_i\}$; $end := b_i.t_e$;
11: **return** B'

due to Pontual *et al.* [109] which we use as a procedure for deciding accountability in presence of special cases of cascading obligations. We refer readers to Pontual *et al.* [109] for a detailed presentation. A pseudo-code of the algorithm is presented just below.

Algorithm 6.7 *Authorized* (γ, Φ, B, b)

Input: A mini-ARBAC policy Φ, an authorization state γ, an obligation set B, and an obligation b.

Output: returns **true** if b is authorized with respect to $\gamma.UA$ and B

1: **if** $b = \langle u, grant, \langle r_t, u_t \rangle, [start, end] \rangle$ **then**
2: $\quad$ **return** $(\forall [s,e] \in subint(B \cup \{t_s, t_e\}))$.
$\quad\quad (overlaps([s,e],[t_s,t_e]) \rightarrow$
$\quad\quad\quad (\exists \langle r_a, c, r_t \rangle \in \Phi.CA).(hasRole(\gamma, \Phi, B, u \models r_a, [s,e])$
$\quad\quad\quad \wedge (\forall l \in c).(hasRole(\gamma, \Phi, B, u_t \models l, [s,e]))))$
3: **else if** $b = \langle u, revoke, \langle r_t, u_t \rangle, [start, end] \rangle$ **then**
4: $\quad$ **return** $(\forall [s,e] \in subint(B \cup \{t_s, t_e\}))$.
$\quad\quad (overlaps([s,e],[t_s,t_e]) \rightarrow$
$\quad\quad\quad (\exists \langle r_a, r_t \rangle \in \Phi.CR).(hasRole(\gamma, \Phi, B, u \models r_a, [s,e]))$
5: **else** /* $b = \langle u, a, \langle \vec{o} \rangle, [start, end] \rangle$ */
6: $\quad$ **return** $(\forall [s,e] \in subint(B \cup \{t_s, t_e\}))$.
$\quad\quad (overlaps([s,e],[t_s,t_e]) \rightarrow$
$\quad\quad\quad (\exists \langle r_a, \langle a, \vec{o} \rangle \rangle \in \gamma.PA)$.
$\quad\quad\quad (hasRole(\gamma, \Phi, B, u \models r_a, [s,e]))$

The non-incremental algorithm takes as input a set of obligations, an authorization state, and a mini-ARBAC policy and returns true when the set of obligations is strong accountable. For this, the algorithm inserts all the administrative obligations in the set to a modified interval search tree. Then it checks whether each of the obligation is authorized in its whole time interval. For this, the

Algorithm 6.8 *hasRole* $(\gamma, \Phi, B, u \models_\gamma l, [s,e])$

Input: A mini-ARBAC policy Φ, an authorization state γ, an obligation set B, a query $u \models_\gamma l$ in which l has either the form r or $\neg r$, and a time interval $[s,e]$.

Output: Returns **true** if $u \models_\gamma l$ is guaranteed to hold throughout the interval $[s,e]$.

```
1:  if l = r then /* positive role constraint */
2:      if (∃⟨u′, revoke, ⟨r,u⟩, [start, end]⟩ ∈ B).(
        overlap([s,e], [start, end])) then
3:          return false
4:      if ⟨u,r⟩ ∈ γ.UA then
5:          if (∃⟨u′, revoke, ⟨r,u⟩, [start, end]⟩ ∈ B).(end < s) then
6:              Select such a tuple so that end is maximized
7:              if (∃⟨u″, grant, ⟨r,u⟩, [start′, end′]⟩ ∈ B).
                (start′ > end ∧ end′ < s) then
8:                  return true
9:              else
10:                 return false
11:         else
12:             return true
13:     else /* ⟨u,r⟩ ∉ γ.UA */
14:         if (∃⟨u′, grant, ⟨r,u⟩, [start, end]⟩ ∈ B).
            (end < s) then
15:             Select such a tuple so that start is maximized
16:             if (∃⟨u″, revoke, ⟨r,u⟩, [start′, end′]⟩ ∈ B).
                (overlap([start, e], [start′, end′])) then
17:                 return false
18:             else
19:                 return true
20:         else
21:             return false
22: else /* l = ¬r negative role constraint */
23:     In case of negative role checking (u ⊨_γ l where l = ¬r), the algorithm follows similar steps, reversing
        the roles of “GRANT" and “REVOKE" and reversing the negative and positive role tests.
```

algorithm inspects whether the user performing the obligation has the necessary roles in the whole time interval. For simplicity, let us consider the user u needs the role r to perform the obligation. Then, the algorithm checks whether u has role r in the current authorization state. If yes, then it checks whether there is an obligation overlapping with the current obligation that revokes r (Queries like this can be done efficiently ($O(\log n)$) using the modified interval search tree.). If no, then u is guaranteed to have role r in the whole time interval. In case, u currently does not have role r, then the algorithm checks whether there is a grant of the role r to u and no one is revoking it. If that is the case, then u is guaranteed to have role r in the whole time interval.

Complexity Analysis of the Algorithm Let us consider the current pending pool of obligations is B where $|B| = n$. Moreover, let us consider $B_i \subseteq B$ denotes the set of infinite repetitive obligations in the current pending pool of obligations where $|B_i| = d$. Let us consider the number of policy rules Φ is k. (1) When the new obligation b we want to add incurs a finite number of cascading obligations, the number of finite cascading obligations due to b can be approximated by k. This is due to our restriction that our policies are free of cycles. Furthermore, let us consider that the number of times the infinite repetitive obligations are unrolled is α. Thus, the total number of obligations for which we need to check accountability in this case is $\eta_c = \alpha \times d + k + (n-d)$. Then, we check each of the η obligations are all authorized, which can be done using the non-incremental algorithm presented by Pontual *et al.* [109] in $O(k\eta_c^2 \times \log(\eta_c))$. (2) In the case b being a finite repetitive obligation, the number of times b needs to be unrolled is $b.\rho$. Let us denote it by m. Thus, the total number of obligations for which we need to check accountability in this case is $\eta_r = \alpha \times d + m + (n-d)$. The resulting complexity of the algorithm in this case will be $O(k\eta_r^2 \times \log(\eta_r))$ (3) For the case, b is an infinite repetitive obligation, we have to compute the overall period of the obligations in B_i and b. Let, β denote the number of times the obligations in B_i and b needs to be unrolled. Thus, the total number of obligations for which we need to check accountability in this case is $\eta_i = \beta \times (d+1) + (n-d)$. This results in a time complexity of $O(k\eta_i^2 \times \log(\eta_i))$.

6.5 Empirical Evaluation

The goal of the empirical evaluations is to determine whether strong accountability can be decided efficiently for some special cases of cascading obligations. For those cases, our empirical evaluations illustrate that it is actually feasible to decide the strong accountability property.

6.5.1 Experimental Environment

The algorithm for deciding strong accountability for special cases of cascading obligations is implemented using C++ and compiled with g++ version 4.4.3. All experiments are performed using

an Intel i7 2.0GHz computer with 6GB of memory running Ubuntu 11.10.

6.5.2 Input Instance Generation

As in the case for many security researcher, we do not have access to real life access control policies that contain obligations. Thus, we synthetically generate problem instances for our empirical evaluations. We belief the values of the different parameters we assume are appropriate for a medium sized organization.

In our experiments, we consider 1007 users, 1051 objects, and 551 roles. We also consider 53 types of actions, 2 of which are administrative in nature (grant and revoke). We handcrafted a mini-RBAC and mini-ARBAC policy with 1251 permission assignment rules, 560 role assignment rules (maximum 5 pre-conditions in each), and 560 role revocation rules. Among the policy rules, 100 of them can incur new obligations. Each of which can incur a maximum of 10 new obligations totaling 1000 new cascading obligations.

To generate the obligations, we handcrafted 6 strongly accountable sets of obligations in which each set has 50 obligations. Each set has a different ratio of administrative to non-administrative obligations (*rat*). We then replicated each set of obligations for different users to obtain the desired number of obligations. Similarly, we generate the infinite and finite repetitive obligations, we use 6 sets of repetitive obligations that are strongly accountable. The execution times shown are the average of 100 runs of each experiment.

6.5.3 Empirical Results

We now discuss our empirical results. Recall that, our accountability decision procedure takes as input an accountable pool of obligations B, the current authorization state γ, a mini-ARBAC policy Φ, and a new obligation b. It returns true when adding b and its associated new obligations maintain accountability. In these empirical evaluations, we consider cases where b can incur a finite amount of new obligations and can be finitely (resp., infinitely) repetitive.

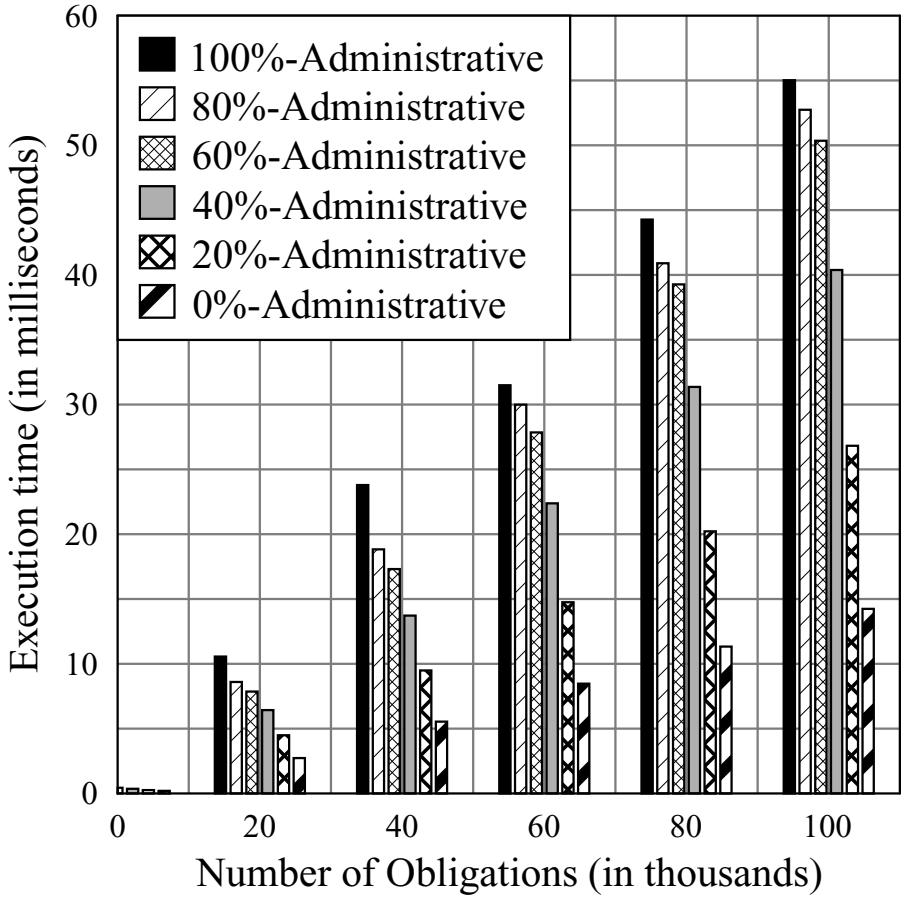

Figure 6.6: Base Line (No Cascading Obligations).

Finite Cascading Obligations In this case study, we add an obligation to a strongly accountable set of obligations. This obligation in turn incurs 1000 new obligations. Then, the algorithm needs to decide whether these 1000 obligation along with the original strongly accountable obligation set is still strongly accountable. Figure 6.7 presents the results for the strong accountability algorithm for this case. Although the number of cascading obligations is fixed (1000) throughout this experiment, we vary the number of obligations by changing the number of pending obligations in the pool from 0 to 99000. We follow the same strategy for all the other case studies.

The time required by the strong accountability algorithm grows roughly linearly in the number of obligations. In the worst case (99,000 administrative obligations plus 1000 finite cascading obligations), the algorithm runs in 103 milliseconds to determine that the set is strongly accountable. This is roughly two times slower than the non-incremental strong accountability algorithm

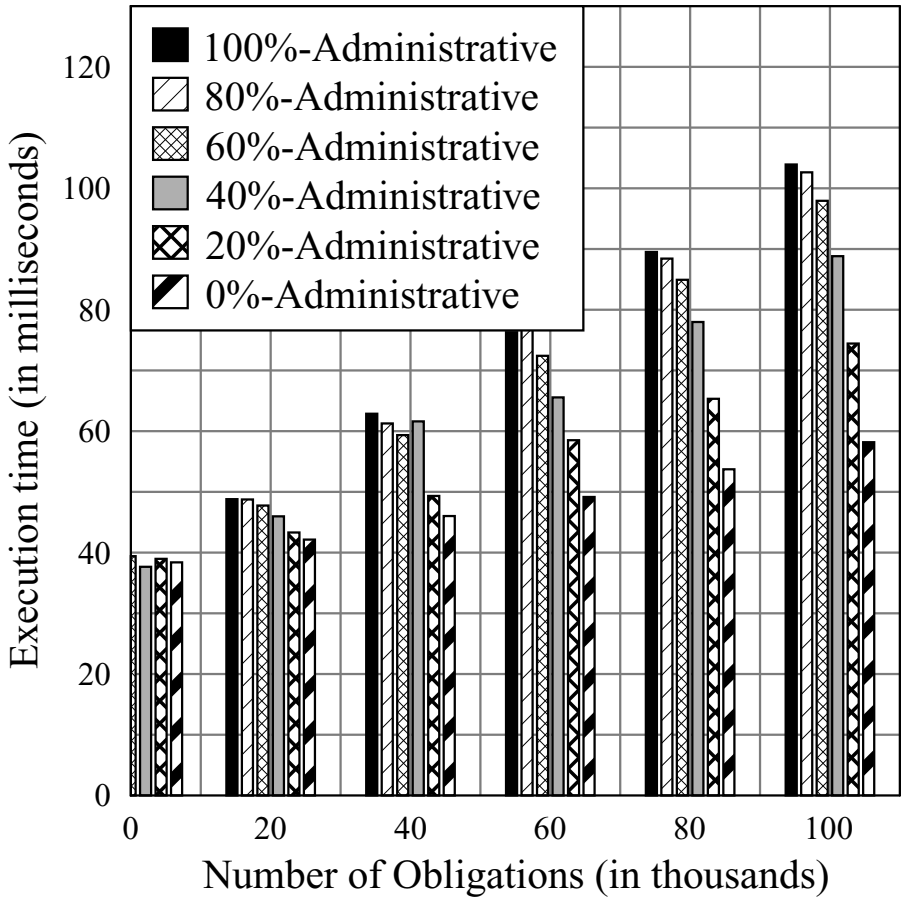

Figure 6.7: Finite Cascading Obligations.

presented by Pontual *et al.* [109] without cascading of obligations. This is due to the overhead of unfolding the cascading obligations (algorithm 6.5). As the algorithm must inspect every obligation following each administrative obligation, *rat* influences the execution time of the algorithm. In addition, we have also simulated (cf. figure 6.8) the same case when the original set of obligations have infinite repetitive obligations, in this case the worst execution time is still 103 milliseconds. This is due to the fact that the time of procedure *UnrollCascading* dominates the time of procedure *UnrollInfiniteRepetitive*.

Finite Repetitive Obligations In this experiment, we add a finite repetitive obligation to a strongly accountable obligation set. This new obligation repeats 1000 times ($\rho = 1000$). The algorithm decides whether the old set of obligations plus the 1000 copies of the repetitive obligation

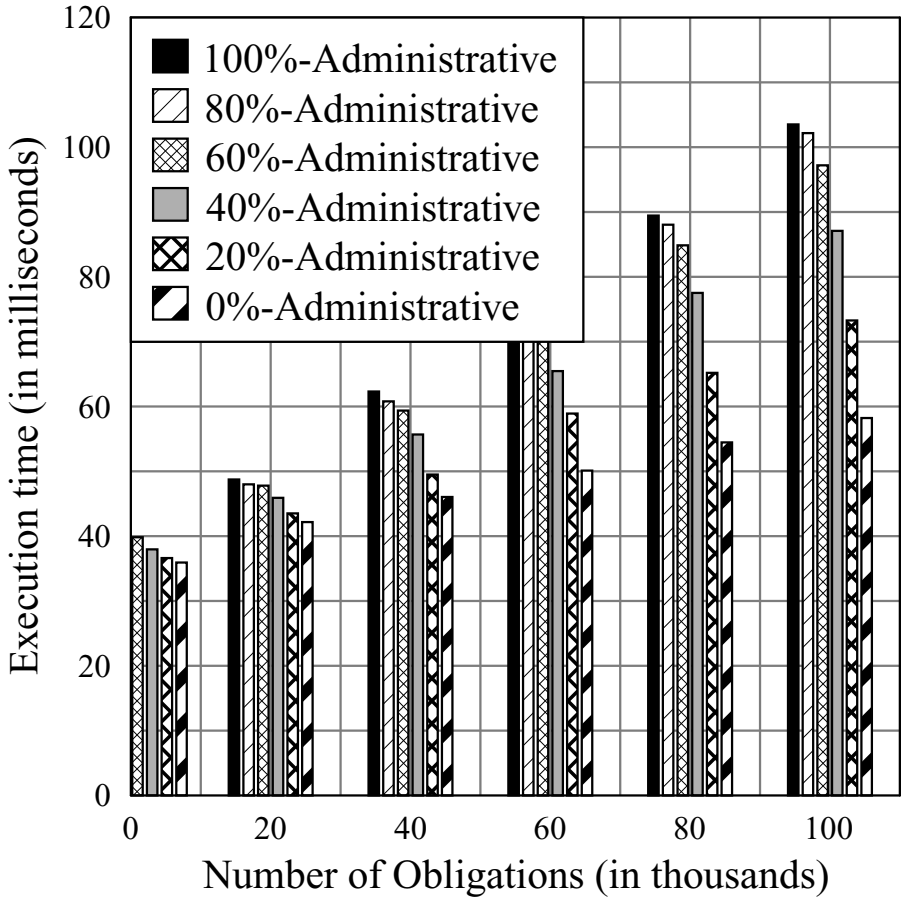

Figure 6.8: Finite Cascading Obligations (with Infinite Repetitive Obligations).

is strongly accountable. Figure 6.9 shows the results for the strong accountability algorithm for this case. The execution time of the strong accountability algorithm grows roughly linearly in the number of obligations. In the worst case, the algorithm runs in 66 milliseconds to decide whether the set is strongly accountable. In general, if the number of obligations generated by the finite repetitive obligations is not too large (when compared with the original set), the time necessary to decide accountability is not affected by the addition of finite repetitive obligations. As algorithm 6.5 can unroll the repetitive obligations in a trivial way, the overhead of this procedure will be small provided that the number of repetition is small. In addition, we have also simulated (cf. figure 6.10) the same case when the original set of obligations have infinite repetitive obligations. The worst case execution time, for this case, is 66 milliseconds.

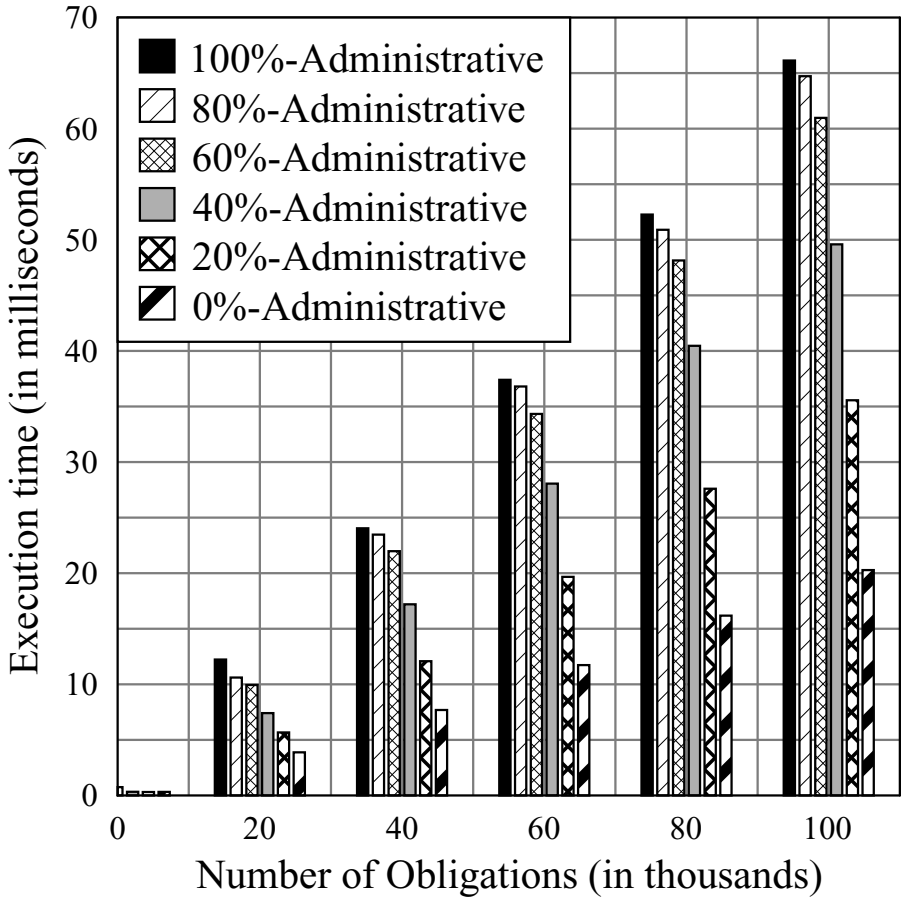

Figure 6.9: Finite Repetitive Obligations.

Infinite Repetitive Obligations In these experiments, we add an infinite repetitive obligation to a strongly accountable obligation set that already contains some infinite repetitive obligations. These infinite repetitive obligations together with *b* is cloned for a total of 519 times. Figure 6.11 shows the results for the strong accountability algorithm for this case. The execution time of the strong accountability algorithm grows roughly linearly in the number of obligations. In the worst case, the algorithm runs in 66 milliseconds.

Remarks One of the problems that algorithm 6.3 can suffer from is that the number of finitely unrolled obligations can be large. This is possible in three different cases. First, an obligation added to the system can incur a large number of obligations (finite cascading obligations). Although this is possible, we do not consider this as we prohibit existence of cycles in the policy rules.

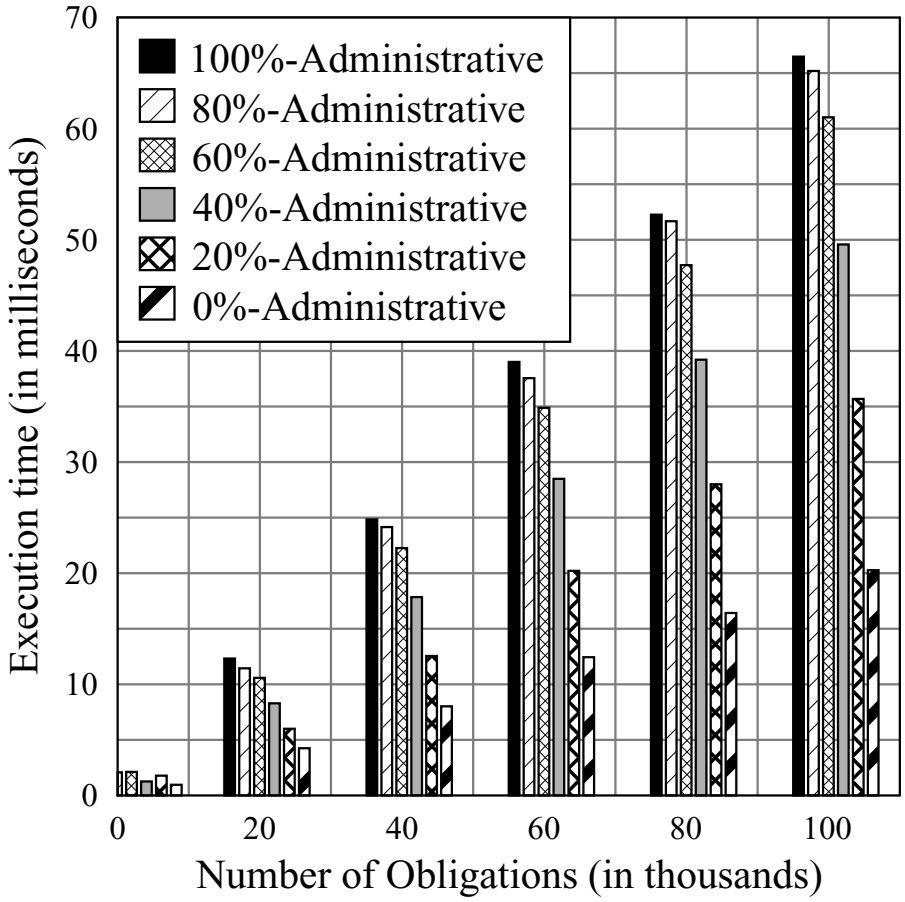

Figure 6.10: Finite Repetitive Obligations (with Infinite Repetitive Obligations).

Thus, the number of new obligations that can be incurred by an obligation is bounded by the size of the policy. Secondly, one can add a finite repetitive obligation that repeats a large number of times. In this case, we can try to bound the number of times a finite repetitive obligation can recur. Finally, when someone adds an infinite repetitive obligation, the overall period of all the infinite repetitive obligations can be very large. In this case, we might end up unrolling the infinite repetitive obligations many times. A possible solution to this problem is to impose a restriction that finite and infinite repetitive obligations will only be non-administrative. As non-administrative obligations do not alter the authorization state, we only need to unroll the infinite repetitive obligations just beyond the maximum end time of non-repetitive obligations.

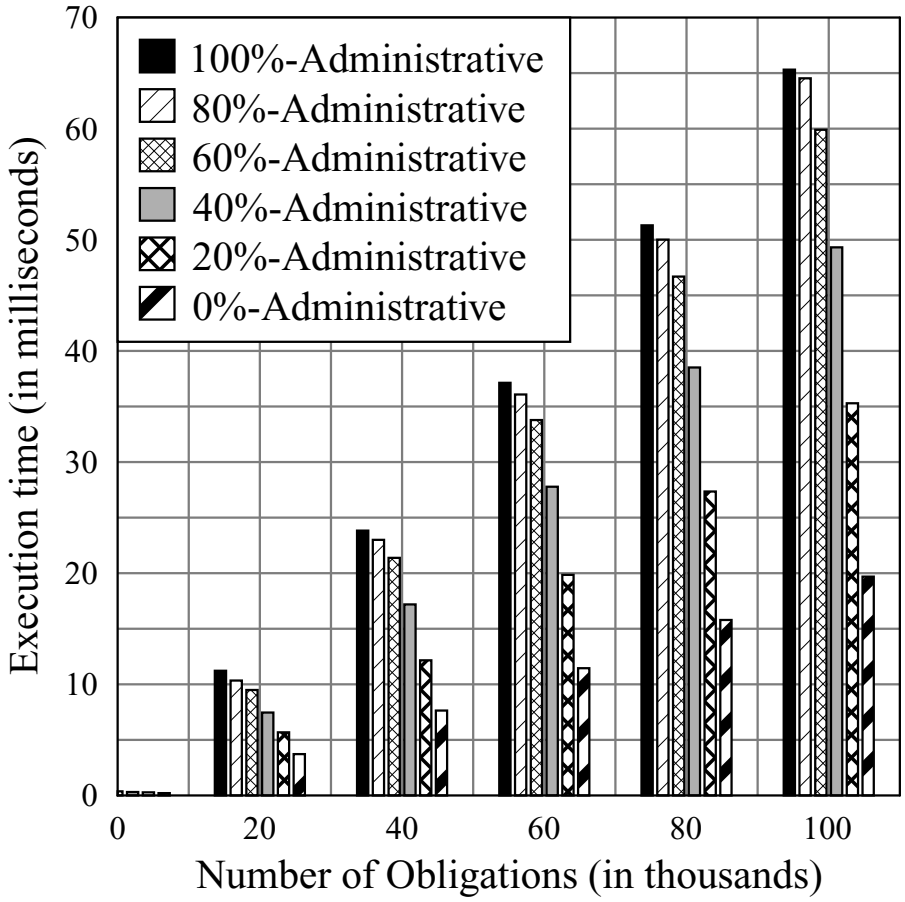

Figure 6.11: Infinite Repetitive Obligations.

6.6 Summary

In current work, we have refined the notion of strong accountability due to Irwin *et al.* [67] to allow cascading obligations. We also enhance the obligation model used by Pontual *et al.* [109] to support the specification of cascading obligations. We present several proposals to specify the obligatee in the policy. We then show that deciding accountability in general is NP-hard. Thus, we consider several simplifications for which the strong accountability decision becomes tractable. We provide an algorithm, its complexity, and also present empirical evaluations of the algorithm. Our experiments show that accountability can be efficiently decided for special cases of cascading obligations.

Chapter 7: RELATED WORK

7.1 Access Control and Obligations

Obligations have received attention from numerous researchers [3, 11, 18, 31, 39, 54, 67, 69, 86, 94, 95, 97, 100, 101, 110, 122, 127]. Some of them are interested in efficiently specifying obligatory requirements [3, 11, 18, 31, 94, 101, 122, 127] and others are interested in the management of obligations [11, 39, 54, 67, 86, 100, 110].

Katt *et al.* [70] augmented the UCON model [105] to support post-obligations. Their system considers two types of obligatory actions , *non-trusted obligations* and *trusted obligations*. Trusted obligations are performed by the system, so they consider that they are never violated. Non-trusted obligations; however, are user obligations and can be violated. They proposed a mechanism that makes decisions based on the status of fulfillment of the non-trusted obligations. However, they did not consider interaction of authorization systems and obligations.

Dougherty *et al.* [39] introduced an obligation model that supports several types of obligations, negative, pre, and post-obligations. Moreover, their model also support some cases of cascading obligations (finite repetitive obligations). For deciding whether an obligation is fulfilled they use static analysis. In contrast, we manage special cases of cascading user obligations where they only consider system obligations.

Ni *et al.* [100] presented a user obligation model based on an extended role based access control for privacy preserving data mining (PRBAC) [101]. Their model supports repeated obligations, cascading obligations, pre and post-obligations and also conditional obligations. In addition, they also present how to detect infinite obligation cascading in a policy. Their work is complimentary to ours, since we study the impact of different types of cascading obligations when deciding accountability.

Ali *et al.* [4] presented an enforcement mechanism for obligations in service oriented architectures. Their model supports many different aspects of real obligation systems (*e.g.*, repeated

obligations, conditional and pre-obligations), but do not support cascading obligations. Although their model is more expressive than ours, they assume that obligations have all the necessary permissions.

Elrakaiby *et al.* [41] borrow the concepts of Event Condition Action from the area of database to present an obligation model. It supports pre and post-obligations, on-going obligations. Obligations can have relative or absolute deadlines. They also introduce the concept of continuous obligations, which is a variation of conditional obligation. To cope with violations, conflicts, and lack of permissions in obligations, they adopt a set of strategies such as sanctions for users that violate obligations, cancellation of obligations, delay of obligations, and re-compensation for users that fulfill their obligations. In contrast, although our obligation model is less expressive, we use accountability to detect violations before they occur.

Li *et al.* [86] extended XACML [127] to support a richer notion of obligations. They view obligations as state machines and can express pre-obligations, post-obligations, stateful-obligations, *etc.* However, they do not consider obligations requiring authorizations and in turn do not concentrate on deciding accountability. Although, the model we extend [109] is not as expressive as theirs, we concentrate on those obligations that require authorizations and can alter the authorization state of the system. Particularly, we are interested in expressing some special cases of cascading obligations and deciding accountability in presence of it. In this sense, our view of managing obligations is different than theirs.

7.2 Privacy Policy Specification Language

The privacy policy specification language we use (section 3) is derived the privacy policy specification framework Contextual Integrity (CI) presented by Barth *et al.* [11]. The goal of this thesis and that of CI are fundamentally different. The work by Barth *et al.*, are geared towards developing a framework for specification of practical privacy policies whereas our goal additionally is to perform static policy analysis and to develop efficient compliance checking algorithm for practical privacy policies like HIPAA. We propose several modifications of their specification language.

Some of them are restrictions and some of them are enhancements of their specification language.

Notice in particular the limited way in which future temporal operators are used (figure 3.2). Aside from the $\Box$ at the outer-most level, the only future subformulas are of the form given by β and $\Diamond$ can be applied only to non-temporal formulas. This restriction allows us to syntactically extract the present conditions and obligatory requirements imposed by the privacy policy and thus gracefully define what it means for a transmission action to be weakly compliant with a privacy policy of our form. Although, Gabbay [52] provides a syntactic way of separating the present conditions and obligations from pLTL policy, it is not trivial to extend his approach for FOTL due to predicates sharing variables among each other.

Furthermore following the example of DeYoung *et al.* [35], we extend CI [11] to contain two extra pre-defined predicates for-purpose(m, pr) and purpose$(pr, \hat{pr})$ to express the purpose of the disclosure. This is inherent to HIPAA as it sometimes decides whether to allow (resp., deny) a disclosure based on the intended purpose of that disclosure (*e.g.*, treatment, payment). Additionally, we support arbitrary regulation specific predicates which are not supported by CI. Thus, due to absence of these regulation specific predicates, their specification language cannot be used to capture fine-grained privacy requirements imposed by practical privacy regulations like HIPAA [62]. Moreover, CI supports having multiple context which provides a mechanism for combining different privacy policies applicable to different contexts. In this thesis, we concentrate on the HIPAA privacy rules, thus we have one context only, and consequently we remove the sort referring to the context in question.

The FOTL formulas that we allow in the positive norms are of form $\psi \wedge \beta$ whereas Barth *et al.* allow arbitrary FOTL formulas (past and future). In case of negative norms, they do not allow any temporal operators in the antecedent of the norms whereas, we allow pure past temporal formulas of form ψ in the antecedent. However, they allow arbitrary FOTL formulas in the consequent of the negative norms. On the other hand, the FOTL formulas that we allow in the consequent of the negative norms are of form χ.

Lam *et al.* [79] propose a privacy policy specification language called pLogic based on stratified

datalog [113]. To demonstrate the adequacy of their specification language they specify §164.506 of HIPAA in pLogic. Due to the limitations of datalog, pLogic cannot specify temporal conditions similar to ours. Furthermore, they cannot explicitly express obligations. We can actually specify all 84 disclosure related clauses of HIPAA in our specification language. In that sense, our specification language is richer than pLogic.

In light of CI [11, 12], DeYoung *et al.* [35] propose an expressive FOTL based policy specification language *PrivacyLFP*. Their specification language is strictly more expressive than our specification language. They have shown the adequacy of their specification language by capturing all fine-grained privacy requirements imposed by all 84 disclosure related clauses of HIPAA. Moreover, they specify the GLBA [2] privacy regulation in their PrivacyLFP. We adopted some of their improvements over CI (*e.g.*, purpose of a disclosure, arbitrary regulation specific predicates) but left out others that were not relevant for HIPAA (fixed point operators). Note that, the goal of their work is to precisely capture the privacy requirements imposed by practical privacy policies like HIPAA, GLBA. Our goal additionally is to perform static policy analysis and developing efficient compliance checking algorithm for these regulations. To this end, we have make some more restricting assumptions about the form of the policies.

Garg *et al.* [55] propose an expressive, first-order-logic based privacy policy specification language. They have also specified all 84 disclosure related clauses of the HIPAA privacy rule in their specification language. Note that, their specification language is also strictly more expressive than our specification language. However, we share some of the same goals, our goals differ in some places. Specifically, one of our goal is to perform static policy analysis which requires us to impose some syntactic restrictions on our policy specification language.

Basin *et al.* [13] propose an expressive, metric first order temporal logic (MFOTL) based privacy policy specification language. Their specification language is also more expressive than our specification language. The principal goal of their work is to check whether whether an action is compliant with a privacy policy specified in MFOTL. Note that, the goal of this thesis is to additionally check in static time whether a policy has a desirable Δ-property. Also note that as we

have mentioned before provided that obligations do not interact with each other and the only place deadlines are placed is on the obligations, then deadlines do not have any impact on our analysis. Moreover, they do not have an encoding of HIPAA privacy rule in their specification language.

May *et al.* [94] present a formalism, based on HRU access control matrix model [59], called Privacy APIs to encode practical privacy policies like HIPAA [62]. They also provide techniques to convert privacy regulations to their formal language. They extended the HRU access control matrix model to achieve the expressive power to specify useful practical privacy policies. Their extensions include support for logging and notification, extending the syntax and semantics of HRU conditional commands, specification of conditions, purposes, and obligations. They only translate §164.506 of the 2000 and 2003 version of HIPAA in their formalism. It is also not clear whether their specification language is expressive enough to specify the whole HIPAA privacy rule. Moreover, ***FOPSL*** can capture fined grained privacy requirements (*e.g.*, temporal conditions) which their language cannot capture.

Ni *et al.* [101] present a family of model named P-RBAC (Privacy-aware Role Based Access Control) that extends the traditional RBAC [8, 50] to support specification of practical but complex privacy policies. Their model can specify conditions, purposes, obligations, *etc.*, which are necessary for privacy policy specification. However, it is not clear how these complex (temporal) conditions are managed in their system. Moreover, they do not provide any explicit management of obligations incurred due to an action. It is also not apparent how to encode some HIPAA rules into their specification language due to absence of examples of such encodings.

The Enterprise Privacy Authorization Language (EPAL) [3] is a specification language used to express organizational privacy policies. EPAL supports purpose of a disclosure or usage and also allows obligations to be incurred when an action is either allowed or denied. However, in EPAL's policy rules there is no support for restricting the attributes of the sender of a information and also the subject of the information. They can just impose restrictions on the attributes of the receiver of a disclosure. Furthermore, they do not support past temporal conditions to be added to the condition of a rule.

Wu *et al.* [126] use a two step approach to specify HIPAA privacy regulation into a declarative style programming approach called Answer Set Programming (ASP) [21]. In the first step, the natural language policy is converted to a abstract representation of the policy (generic pattern based policy), which they introduce for interoperability of policies and ease of use. In the second step, they take the generic pattern based policy and translate it to a logic based representation on ASP. In their case study, they only use clause §164.506 of HIPAA. It is not clear whether all the fine-grained privacy requirements imposed by HIPAA can be captured in their framework.

Breaux and Antón [20] propose a systematic methodology of extracting access rights and obligations from the regulation text. They consider both HIPAA security and privacy rule. Note that in this thesis we consider only the HIPAA privacy rule. Although, the access rights and obligations are extracted they do not perform any kind of analysis to give static guarantees about whether incurred privacy promises or obligations can be fulfilled. Moreover, they are more interested in capturing the privacy regulations accurately whereas our goal is to additional check consistency of the regulation and also developing efficient compliance checking algorithms. In that sense, their work is complementary to our work.

In the same vein as Breaux and Antón, Maxwell and Antón [93] propose a framework for capturing a set of software requirements for checking compliance with the regulations like HIPAA. They propose a production rule framework which can be used by software engineers to specify compliance requirements of their developed software. Then they go on to check whether iTrust which is an open source electronic medical records system, is compliant with the HIPAA security rules. The goal of this thesis is however checking consistency of HIPAA privacy rules and developing efficient compliance checking algorithm for it.

7.3 Privacy Policy Analysis and Compliance Checking

Garg *et al.*, [55] propose an expressive, first-order-logic based privacy policy specification language. They present a semi-automated auditing algorithm that incrementally inspects the log of a system against a policy and detects violations. Their specification language is expressive enough

to capture the requirements of all 84 disclosure related clauses of HIPAA. Their algorithm takes a privacy policy specified in their language and an incomplete log of the system and tries to evaluate as much as the log as possible. They assume that the interpretation of some of the predicates cannot be determined automatically or needs manual inspection from a human. In such a case, they return a residual policy which can be further evaluated once part of the unavailable logs become available. In their setting, they consider two kinds of incompleteness: (i) Past incompleteness: this incompleteness is due to unavailable interpretation of some predicates, (ii) Future incompleteness: this incompleteness is due to not enough time being passed. Incompleteness case (i) happens when the policy has predicates about subjected belief which has to be manually inspected by a human or the log did not capture all the necessary information required to reason about whether an action is allowed by the policy. Incompleteness case (ii) happens when according to the policy a principal incurs an obligation but the deadline of the obligation has not passed yet. In this case, it is not reasonable to come to any conclusion about whether the obligation will be fulfilled or not.

Moreover, they allow quantification over infinite domains. To achieve finiteness, they have developed a static linear time policy checking algorithm which checks whether the input-output modes of the different predicates of the policy are well-moded. The well-modedness of the policy is actually borrowed from the well-modedness of logic programs. The well-modedness of the policy guarantees that while evaluating the policy with respect to a (possibly, incomplete) log, the number of individuals that needs to be guaranteed is finite. Note that, they do not make the assumption that the number of valuations which make a predicate true might not be finite. However, their algorithm is not efficient due to the fact that while evaluating the policy, they have to go back and forth in the log to check for satisfiable valuations of certain sub-formula of the policy. As we have shown in this thesis, it is possible to keep a summary structure of the log which ensure that we do not have to go back to the log which is a disk intensive operation according to our assumption. Provided that a summary structure is kept in the memory, only the summary structure needs to accessed to find satisfiable valuations of sub-formulas of the policy. However, note that, the policies for which they can check compliance of is more expressive than the policies we consider. Moreover, in our

algorithm, we do not consider incompleteness of the finite execution history (incompleteness case 1). Also note that, considering that the incompleteness case (i) cannot happen, their algorithm can be used as a runtime monitor to detect any violation of the policy in preemptive manner.

Basin *et al.* [13] propose a specification language based on metric first order temporal logic (MFOTL). MFOTL differs with traditional FOTL in regards to the constraints imposed to the temporal logic operators (*e.g.*, $\mathcal{S}$, $\mathbb{U}$). In MFOTL, there a time restriction on each temporal logic formula as interval of form $[c,d]$ where $c,d \in \mathbb{N}$. The restriction $[c,d]$ specifies when a formula ϕ is being evaluated in the time t then the formula ϕ must hold true with the interval $[t-c,t-d]$ for past time formulas (resp., $[t+c,t+d]$ for future time formulas). Note that, in their language they also allow future time temporal operators. The time interval associated with the future temporal operators enable them to wait only a finite amount of time (time dependent on the interval) before deciding whether certain future formula has held. Due to the time interval restrictions, the properties that can be specified in their language are safety properties. In our language, we assume past temporal operators do not have any time restrictions whereas we do not have any future temporal operators in the formula $weak(\wp)$ which we need to check for compliance. They provide a PSPACE-complete algorithm for incrementally checking compliance of a policy specified in their language with respect to a history. Similar to mode checking, they come up with a static time checking algorithm called the "safe-range checking". The safe-range checking algorithm basically verify whether all the free variables of a formula is in the output mode of the formulas. This actually boils down to requiring that every predicate a finite number of valuations for which the formula holds true. Our temporal mode checking is more expressive than their safe-range check and allows more expressive policies to be checked for compliance. They also assume strict temporal operators whereas we consider non-strict version of the temporal operators. Note that, unlike the work of Garg *et al.* [55], they assume that logs are complete and they view their algorithm for checking compliance to be used online as a reference monitor deciding whether to allow or disallow certain actions.

Basin *et al.* [15] extended their previous work on compliance checking to handle disagreeing

and incomplete logs. To this end, they propose a three valued logic where along with the traditional `t` (true) and `f` (false), they have an extra value $\perp$ which denotes "don't know". They provide the interpretation of different operators based on the three-valued operands. Moreover, they assume the number of valuations that makes a formula true can be one of the following: FIN, CO-FIN, and NONE. FIN represents that the number of valuations which make a certain formula true is finite. CO-FIN represents that the number of valuations which make a certain formula true is co-finite, meaning that the complement of that set is finite. NONE represents that the number of valuations which make the formula true is infinite. They then extend their previous compliance checking algorithm to handle these sets and define different operations on the sets as relational algebra operators like projection, join, *etc.* Note that, our compliance checking algorithm cannot handle incompleteness of the logs or disagreement of the logs.

Dinesh *et al.* [37, 38] translate regulations in statements in predicate linear temporal logic (predLTL) where the constraints imposed by the regulations are either obligatory or permitted. They use traces to model the operations of the organization. Then they check whether these traces satisfy the restrictions imposed by the formalized regulations in the formal logic. They specifically consider the Food and Drug Administration's Code of Federal Regulations [102–104] (FDA CFR). Then they introduce another logic (RefL) that can be used to analyze and reason about references to other regulations and their implications. In contrast, our work focuses on HIPAA privacy rules and developing techniques to check whether a privacy policy has a desired Δ-property and use it to develop efficient compliance checking algorithm. In that sense, their work is complementary to our work.

Krukow *et al.* [73] develop a specification language which can specify history-based requirements for transaction monitoring. Their language is past-only fragment of the first order temporal logic (FOTL). They also assume that the only predicates that are allowed are unary predicates (predicates with one argument). They also assume a special counting quantifier which enables the policy to refer to the number of times a formula has held in the past. They then come up with a compliance checking algorithm that incrementally checks whether a finite trace is compliant with

the history-based requirements specified in their language. Moreover, in their language, they only assume *action predicates*, which has the implication that the number of individuals that can satisfy action predicate at one point of time is finite. Although, the domain they support for the different sorts can be infinite, this assumption ensures that at each point of time, the number of individuals that has to be considered are the ones that appear in the history which in turn is finite. First, while specifying HIPAA, it is required to have predicates which are not unary. Secondly, according to the specification of HIPAA, sometimes it is required to have non-action predicates to specify subjected belief, which violates their requirements to have only action predicates. Thus, their approach of checking weak compliance is too restrictive and is not applicable for privacy regulations like HIPAA.

Bauer *et al.* [16] also propose a specification language which can specify history-based requirements for transaction monitoring. Similar to the work of Krukow *et al.* [73], they consider past only fragment of first order temporal logic. However, the language proposed by Bauer *et al.*, is strictly more expressive than the language proposed by Krukow *et al.* In their language, they consider arbitrary predicates instead of just unary predicates assumed by Krukow *et al.* They also assume interpreted function symbols which are in PSPACE. Then they come up with a compliance checking algorithm that incrementally checks whether a history satisfies the policy specified in their language. Note that, in their specification language they also consider action/event predicates and the implication of that is that the number of valuations that makes the predicate hold in a state is finite sized. They also prove that the complexity of their compliance checking algorithm is PSPACE-complete provided that interpreted functions and relations specified in the policy is in PSPACE. Due to the fact that they consider each predicate has finite number of satisfiable valuations in each state, prohibits to specify HIPAA in their specification language as HIPAA privacy rules might refer to predicates which do not satisfy this constraint. In that sense, our specification language is strictly more expressive than their language as we do not consider any such restriction about all the predicates should have finite satisfiable valuations.

Lam *et al.* [79] proposed a stratified datalog based policy specification language which they

used to encode the section §164.506 of HIPAA. Recall that, their specification language is not expressive enough to capture obligations imposed by HIPAA. One can then use a datalog engine to check whether an action is compliant with the privacy policy which is expressed as a datalog program. In a follow up work, Lam *et al.* [78] developed a generalized way of coming up with a finite domain (small model) for the policy which is sufficient to check any properties of the policy. In our policy analysis technique, we also come up with small model theorems for slices of HIPAA privacy rules. However, their approach is more generalized than ours due to the restricted syntax of their specification language. Then they develop techniques which enable them to automatically generate a form of access control policy used in Attribute-based Encryption (ABE). They use this approach to transmit sensitive health information over untrusted servers and clouds. In contrast, our goal is to develop formal verification techniques to check compliance and consistency of the privacy regulations.

May *et al.* [94] present a formalism, based on HRU access control matrix model [59], called Privacy APIs to encode practical privacy policies like HIPAA [62]. As discussed before, they only specify clause §164.506 of HIPAA, translate it to the SPIN model checker's modeling language PROMELA, and then use the SPIN model checker [65] to check different desired invariants of the specification. They consider conditions and fulfillment of obligations to be represented as environmental flags. However, they do not explain how these flags are set or how to explicitly manage obligations. HIPAA contains complex temporal conditions and it is not very clear how these conditions can scalably be managed in their formalism. Additionally, they only translate §164.506 of HIPAA in their formalism, it is not clear whether their specification language is expressive enough to translate the whole HIPAA. Roughly, their work concentrates on specifying privacy policy and analyze whether some simple invariants hold. Their analysis do not provide any kind of assurance about whether obligations or privacy promises can be fulfilled. The goal of this thesis is to first statically check whether a policy has a desirable Δ-property (all incurred obligations can be discharged) and then develop efficient compliance checking algorithm for policies that satisfy this property.

Safety properties can be enforced by *security automata* [14,48,66,92,115], an approach that has received a lot of attention. The actions of a system can be mediated by a wrapper that maintains the automaton's state and terminates the system if it attempts to perform an action for which no legal transition exists. Determination of legality is made one action at a time. Walker [124] introduced programming language techniques for enforcing a policy expressed as a security automaton. Ligatti *et al.* [88] extended security automata by using edit automata to postpone realizing actions from when the system being monitored attempts them until the system being monitored has attempted additional properties that will restore the renewal property . This is clearly unworkable as a way of enforcing HIPAA, because a doctor cannot decide to "not realize" a patient's request for their medical records until the doctor has attempts to supply those records. Another issue with using edit automata to enforce HIPAA privacy rule is that obligations are not supported, making security automata inappropriate for our purposes. Moreover, the literature on security automata has only consider propositional temporal logic properties, which is inadequate for our purposes.

Chapter 8: CONCLUSION

8.1 Summary

Organizations collect personal information from customers to provide them with services. They use computer information systems to store, manage, and disclose the collected information. Federal privacy regulations like HIPAA, GLBA, and SOX, mandate how the collected information can be used or disclosed by the organizations. These federal regulations carry the force of law and violation of these federal regulations can bring down heavy financial penalties on the organization and the accountable person. Thus, it is of paramount importance for the organizations to have means to check compliance of their information system with applicable privacy regulations.

Privacy regulations like HIPAA impose two kinds restrictions when allowing certain disclosure operation, present requirements and obligatory requirements. Present requirements imposed by the privacy policy restricts disclosure operation based on the system's finite execution history. An obligatory requirement (or, simply obligation) is a future requirement and it requires the obligatee (a particular individual or the system) to perform an action in some future time interval. Prior work by Garg *et al.* [55] and Basin *et al.* [13], has shown that to check whether an action is consistent with the present requirements can be decided automatically and efficiently. Obligations, particularly user obligations, cannot be enforced as it is not feasible to force an individual to take an action. However, it is possible to monitor the fulfillment of an obligation. Precisely, it is feasible to check whether certain pending obligation has been fulfilled, as most of the time a deadline is associated with an obligation.

When an obligation is violated, an enforcement mechanism that checks to see whether certain action satisfies the present requirements or certain obligation is violated, cannot differentiate whether the obligation was not permitted by the policy or the user was not diligent enough to carry out the obligation. At a first approximation, this distinction might not seem interesting or important, but we argue that this information is important. When an individual working for a hospital

violates federal privacy regulations like HIPAA, the hospital is steeply punished for violating the regulation. Specifically consider the case of Cignet Health Center which was fined a staggering $1.3 million for violating the obligation associated with §164.524 [112]. Now hypothetically consider the situation, where, under the correct interpretation, the HIPAA policy did not allow covered entities to fulfill the obligation in §164.524, in that case, if Cignet Health Center was fined, it would have been unreasonable. Thus, before punishing someone for violating the law, we should have formal assurance that the law is well-formed. The notion well-formedness in our case, is allowing incurred obligation to be performed in a policy conforming way.

To provide such formal assurance about policy well-formedness, in this thesis we first converted the natural language specification of regulation into a formal policy specification language. We considered two candidate formal policy specification languages, namely, XACML [26,127] and ***FOPSL*** [27]. We first evaluated XACML [127] as a candidate specification language for HIPAA. We showed that XACML has some important features but lacks some other features to specify HIPAA. We then proposed extensions to XACML which will enable it to specify HIPAA. However, XACML's inadequacy to specify HIPAA in a flexible way and to capture the fine grained requirements of HIPAA, we considered another policy specification language, ***FOPSL***. ***FOPSL*** is inspired by the specification languages presented by Barth *et al.* [11] and DeYoung *et al.* [35]. ***FOPSL*** has well-defined semantics and can capture the fine grained requirements of HIPAA. ***FOPSL*** is a restricted fragment of many sorted first order temporal logic (FOTL). To demonstrate the expressive power of ***FOPSL***, we have encoded all 84 disclosure related clauses of HIPAA in ***FOPSL*** (see Appendix A).

We then formally specified what it means for an action to be compliant with privacy policies written in ***FOPSL***. We borrowed the notion of compliance from the work of Barth *et al.* [11]. Although, their specification language is a restricted fragment of FOTL, their definition of policy compliance is only applicable to policies specified in propositional linear temporal logic (pLTL). There is a general consensus among the researchers in the academia that practical privacy policies like HIPAA cannot be concisely represented in pLTL [13, 35, 55]. We formally specified two

notions of compliance, weak compliance and strong compliance. An action is weakly compliant with a policy if the action satisfies all the present conditions imposed by the policy. On the contrary, an action is strongly compliant if it does not incur an unsatisfiable obligation or it does not prohibit a pending obligation to be carried out. We provided an algorithm for checking weak compliance, that takes as input a finite history, a privacy policy, and checks to see whether there is any violation of the policy in the trace. Our algorithm has a runtime complexity of $O(|\mathcal{L}|^{|\varphi|})$ where $|\mathcal{L}|$ represents the history length and $|\varphi|$ represents the policy length. Our algorithm has a space complexity which is linear to the policy length. We then showed that for policies written in ***FOPSL***, to check whether an action is strongly compliant is undecidable by reducing the Turing machine halting problem [47] to checking strong compliance for policies written in ***FOPSL***.

To mitigate the undecidability result of strong compliance, we formally specified the Δ-property (weak compliance entails strong compliance) borrowed from Barth *et al.* [11]. Although, they were the first to specify the property, their semantic definition of Δ-property and its associated decision procedure were only applicable to policies written in pLTL. Informally, a policy has the Δ-property if all the weakly compliant action of the policy is also strongly compliant. To check compliance of a policy which has the Δ-property, it is sufficient just to check the weak compliance. Moreover, when a policy has the Δ-property, it guarantees that all the incurred obligations can be met. It is sufficient to check the Δ-property once for each policy and before it is deployed for enforcement.

We are the first to formally specify what it means for a policy to have the Δ-property. Given a policy $\wp$ written in ***FOPSL***, we syntactically extracted a formula $\delta(\wp)$ in first order CTL* with linear past logic (FO-CTL*$_{lp}$). We showed that $\wp$ has the Δ-property, if the most permissive model of $\wp$, $\mathbb{M}_{\wp}$, satisfies the formula $\delta(\wp)$ ($\mathbb{M}_{\wp} \models^{?} \delta(\wp)$). The most permissive model of a policy $\wp$, denoted by $\mathbb{M}_{\wp}$, is the model in which at each step one of the all possible actions referred to by the $\wp$ is chosen to be performed. Considering $\mathbb{M}_{\wp}$ of a policy $\wp$ is reasonable because when $\wp$ can incur obligations that cannot be met even in the most permissive model then it is not possible that those obligations can be met in other more restricted models. Also, more restrictive models can not reach any policy "states" that are not reachable with the most permissible model, so other

models would not be able to incur any obligations that would not be able to be incurred by the most permissive model. Other models exhibit a subset of behaviors which are of the interest to the analysis of the Δ-property. However, model checking of FO-CTL$^*_{lp}$ is undecidable in general and thus checking whether a policy has the Δ-property is in general undecidable.

We then made some assumptions, which we showed that practical privacy policies like HIPAA satisfies, and based on them we came up with sound, semi-automated technique to check whether a policy has the Δ-property. We first showed that there are two cases in which a policy might fail to possess the Δ-property. We showed that due to a syntactic restriction of ***FOPSL***, only one violation case can actually occur for policies of our form. Thus, for policies specified in ***FOPSL***, the only way the Δ-property can be violated is when the policy allows a weakly compliant action to incur an unsatisfiable obligation. We then made the assumption that obligations do not interact with each other. Based on this assumption, we came up with a privacy policy slicing algorithm which takes as input a privacy policy and an obligation, and returns a sub-policy of the input policy which is sufficient for analyzing whether the input obligation is satisfiable. After that it is required to come up with a small model theorem which converts an analytical problem of unbounded carriers to an analytic problem of small, finite sized carrier. Once we have developed a small model theorem, we can rewrite the universal and existential quantifiers as finite conjunctions and disjunctions and replace the relations with propositions. The resulting policy is in pLTL and to check whether the resulting pLTL policy has the Δ-property we can apply two possible techniques. One possible technique is tableau-based and is proposed by Barth *et al.* The other technique is proposed by us and requires model checking a CTL$^*_{lp}$ specification. The complexity of both of the approaches are same and they are both EXPSPACE-complete. We currently do not have access to a CTL$^*_{lp}$ model checker, so we used the approach of Barth *et al.* to show the efficacy of our policy technique.

To demonstrate the applicability and efficacy of our policy analysis technique, we ran our policy analysis technique on our interpretation of HIPAA. According to two obligations in HIPAA (§160.310 and §164.524), we first slice the HIPAA policy. The algorithm runs linear to the policy size and with an optimization (human support) yields a sub-policy of size 5 norms which is 6.5% of

the original policy size. We then showed how to develop a small model theorem for the two policy slices. There are two additional obligations in HIPAA which require sending privacy notices. In this thesis, we assumed privacy notices do not contain any individually identifiable information and thus is trivially allowed by the HIPAA privacy rule. Once we had the small model theorem, we converted the sliced policy to a pLTL policy. We then used the GOAL [120] automata generation tool to generate a tableau automata with Büchi accepting condition. We then checked whether all the reachable states from the initial state, can reach a strongly connected component with at least one accepting state. Our analysis ran less than 10 minutes and verified that our interpretation of HIPAA has the Δ-property.

Note that, organizations that are required to check compliance of their computer information system with applicable privacy regulations will require access control policies to safe guard their resources from unauthorized access. For instance, consider the HIPAA privacy policy rule in §164.508 which requires that a covered entity can disclose a patient's psychotherapy note provided that it has already received a valid authorization that allows the covered entity's action. Now the covered entity must ensure that one of the authorized employee can get access to it rather than any arbitrary employee of the covered entity. To ensure this, the covered entity should put forward an access control policy that only allows the authorized employee to access the patient's psychotherapy notes. Access control policies can have some notion of obligations. Research has shown different applications of obligations in access control policies: policy management [17, 18], risk management and tackling insider threat [10, 25], managing pervasive systems [117], usage control [105], data protection [63], *etc.* There are generally two kinds of obligations, user obligations and system obligations. A user (*resp*., system) obligation is an action that a user (*resp*., system) must fulfill in some future time interval. Although, system obligations can be assumed to be fulfilled, user obligations cannot be assumed to be fulfilled due to the fact that it is not feasible to force a user to take an action. Moreover, user obligations like any other actions require proper authorization to be fulfilled. Without formally ensuring a user will have appropriate authorizations to fulfill her obligations, it is not possible to distinguish whether the user was not diligent

or the user did not have proper authorization to fulfill the obligation. If the company were to take into account a user's obligation fulfillment rate to evaluate its performance, there should be assurance that incurred the user has the necessary authorization privileges to fulfill all his obligations. To this end, prior work [67, 109, 110] introduces a property of the authorization state and pending obligations, called "*accountability*". The accountability property roughly requires that all the pending obligatory actions are authorized, the obligatee has the necessary authorizations to fulfill them. They propose to maintain accountability as an invariant and denying any action that disturbs the accountability. The accountability property guarantees that any obligation violation is due to the lack of diligence from the user not due to lack of appropriate privileges. However, prior work assumes that obligatory actions cannot further incur additional obligations (no *cascading obligations*). They achieve this by separating the set of possible actions into two disjoint sets, discretionary actions and obligatory actions and making sure that obligatory actions cannot further incur obligations. This is a very restricted constraint and due to it their obligation model is incapable to encode a lot of real life scenarios. In this thesis, we first defined what it means by a state is accountable in presence of cascading user obligations. We showed that there are two different interpretations of accountability, existential and universal. We then gave justifications of choosing the existential interpretation of accountability. We then proved that checking both interpretations of accountability is in general NP-hard in presence of cascading obligations. We then proposed some special yet practical cases of cascading obligations and for which we provided a tractable decision procedure. Finally, we demonstrated through empirical evaluations that maintaining accountability in presence of these special yet practical cases of cascading obligations is feasible in practice. In the worst case, it takes 103 milliseconds, on a reasonable desktop machine, to decide whether accountability holds for a state with 10^5 pending obligations.

8.2 Open Problems

The notion of strong compliance due to Barth *et al.* [11] is appropriate for closed systems where it is reasonable to assume that all the agents in the system will cooperate to achieve the policy

satisfaction. However, this is not a valid assumption for an open system. This is due to the fact that one cannot safely assume that the environment will cooperate for policy satisfaction instead a safe assumption would be to consider environment as an adversary. Based on this, one can have a stronger notion of strong compliance, namely, *adversarial strong compliance*.

One way to precisely define the adversarial strong compliance by explicitly distinguishing between cooperating agents and potentially non-cooperating agents (environment) of the system. Our formulation requires the cooperating agents to have a strategy that enables them to satisfy the policy, no matter what actions are initiated by the non-cooperating agents (or, by the environment).

Definition 88 (Adversarial Strong Compliance)**.** *For a finite history σ'_f and a contemplated action a where $\sigma_f = \sigma'_f \cdot a$, a is* adversarially strongly compliant *(denoted by ASC) with respect to a privacy policy $\wp$ if there exists a strategy for the agents (both software and human) of the system (denoted by A) such that no matter what actions the environment takes, the agents in the system following the strategy would be able to force the system to take an infinite trace (extension) σ_j such that $\sigma_f \cdot \sigma_j \models \wp$. Here, σ'_f can be viewed as the current system history whereas a can be viewed as the contemplated action. More formally using* $\text{mp}^{-}\text{ATL}^{*}$ *we can define adversarial strong compliance in the following way. Given a CGS $\mathcal{G}$, an environment η, a finite history σ', and a contemplated action a where $\sigma = \sigma' \cdot a$, we say that action a is* adversarially strongly compliant *with respect to a privacy policy $\wp$ if $\mathcal{G}, \sigma, \eta \models \langle\langle A \rangle\rangle \wp$ holds.*

Adversarial strong compliance is stronger than the notion of strong compliance. As a result, when an action is adversarially strongly compliant it is also strongly compliant. However, it can be the case that an action is strongly compliant but not adversarially strongly compliant. One possible example where an action is strongly compliant but not adversarially strongly compliant can be as follows.

Example 89 (Not Adversarially Strongly Compliant)**.** *Let us consider there is a prison facility and it has a clinic to treat the inmates. Let us also assume they have the following simple but unrealistic privacy requirements.*

(A) When one doctor (d_1) requests for a patient's (inmate) medical records for treatment to another doctor (d_2), then it is doctor d_2's responsibility (obligation) to provide doctor d_1 with the necessary medical records.

(B) A doctor can disclose a patient's, who is also an inmate, medical records provided that he received a court order that allows him to do so.

Now consider that there are two doctors, John and Kathy, in the prison clinic. John treated patient (inmate) Rob and has access to his medical records. Now, Kathy is treating Rob and wants his medical records. So, Kathy sends a request to John for the medical records. As a result of the request, according to privacy requirement (A), John incurs an obligation to provide the medical records of Rob. However, according to privacy requirement (B), John can only disclose Rob's medical records if he has received a court order that approves him. Recall that, strong compliance assumes that all the agents of the system (inside or outside) would cooperate to achieve the policy satisfaction. Thus, the action of Kathy is strongly compliant as the court can cooperate and send an order to John to release Rob's medical records enabling John to fulfill his obligation. However, the court is an outside (environmental) entity. Adversarial strong compliance does not assume that the court would cooperate and send an order to John to release Rob's information. As a result, John would not be able to fulfill his obligation, making the action of Kathy not adversarially strongly compliant.

Computational Complexity of Adversarial Strong Compliance. For a given finite history σ', and a contemplated action a, to check whether that action a is *adversarially strongly compliant* with respect to a privacy policy $\wp$, can be decided in 2EXPTIME-complete of the policy size, provided that the $\wp$ is specified in propositional LTL. As we have already seen in the formal definition of adversarial strong compliance, to check whether an action is adversarially strongly compliant with respect to a policy $\wp$ one has to check whether the mp^{-}ATL* formula $\langle\langle A\rangle\rangle\wp$[1] is satisfiable with respect to the most permissive CGS. Model checking a mp^{-}ATL* formula is 2EXPTIME-complete

[1] A denotes the set of agents (human and machine) in the system.

with respect to the formula size [98]. In our case, our policy $\wp$ is an FOTL formula. As a result, the complexity of deciding whether a certain action is adversarially strongly compliant (model checking a FOmp^{-}ATL* formula $\langle\langle A\rangle\rangle\wp$) is undecidable.

Theorem 90 (Complexity of Adversarial Strong Compliance)**.** *For a given finite history σ', and a contemplated action a, to check whether that action a is* adversarially strongly compliant *with respect to an FOTL privacy policy $\wp$ is undecidable.*

As deciding whether an action is adversarially strongly compliant with a policy $\wp$ is undecidable in general. In the same vein as Δ-property, one can possibly come up with a similar property which we decided to call the $\bigstar$-property. When a privacy policy $\wp$ of an open system has the $\bigstar$-property it signifies that any action that is weakly compliant is also adversarially strongly compliant. More precisely, any weakly compliant action that incurs an obligation, there is a strategy for the agents in the system such that the obligation can be discharged. In that case, it suffices just to enforce the weak compliance. Note that, the $\bigstar$-property gives a more stronger assurance that what Δ-property gives, as it mentions that for fulfilling the obligation no collaboration with the environment is necessary as the environment might act against the system. We now specify what it means for a privacy policy $\wp$ to have the $\bigstar$-property.

Definition 91 ($\bigstar$-property)**.** *The $\bigstar$-property holds for a given privacy policy $\wp$, if for any possible environment η and for any possible finite trace σ_f that satisfies $\sigma_f, |\sigma_f|-1, \eta \models \boxminus weak(\wp)$, there exists a strategy for the agents*[2] *of the system (denoted by A) such that no matter what actions the environment takes, the agents in the system following the strategy would be able to force the system to take an infinite trace (extension) σ_i such that $\sigma_f \cdot \sigma_i \models \wp$. More formally, for a given privacy policy $\wp$, the $\bigstar$-property holds if for all CGS $\mathcal{G}$, for all environments η, for all finite history σ_f such that $\mathcal{G}, \sigma_f, \eta \models \boxminus weak(\wp)$ then $\mathcal{G}, \sigma_f, \eta \models \langle\langle A\rangle\rangle\wp$ holds.*

The $\bigstar$-property for a privacy policy can be precisely specified using the FOmp^{-}ATL*. Note that, if for a policy the following FOmp^{-}ATL* is valid then it guarantees that as long as each action

[2] software and human

$$\langle\langle\varnothing\rangle\rangle\Box[\boxminus weak(\wp) \longrightarrow \langle\langle A\rangle\rangle\wp]$$

Figure 8.1: FOmp^{-}ATL* formulation of the ★-property for $\wp$. We denote this formula by $\partial(\wp)$.

is weakly compliant, it is also adversarially strongly compliant.

A FOmp^{-}ATL* specification for ★-property We use $\partial(\wp)$ to denote the formula in figure 8.1. In essence, the formula says the following for every path starting at the initial state and at each step along that path. If WC has held at every step up to the current one, then no matter what decisions the other agents make thereafter, the agents in A (all the agents of the system excluding the environment) can force an infinite sequence of global decisions that yields a trace satisfying the privacy policy.

Based on the semantics of FOmp^{-}ATL*, we can see that the formula in figure 8.1 precisely captures the notion of ★-property. When the formula $\partial(\wp)$ is satisfied in the most permissive computational game structure $\mathcal{G}_{\wp}$, we can say that $\wp$ has the desired property of ★-property. The following theorem states that our specification correctly captures the ★-property.

Theorem 92. *Given a privacy policy $\wp$, $\wp$ has the ★-property if and only if there exists an environment η such that $\mathcal{G}_{\wp}, \eta \models \partial(\wp)$ holds where $\mathcal{G}_{\wp}$ is the most permissive computational game structure of $\wp$.*

It however remains open whether our technique for checking Δ-property is applicable to checking the ★-property of a policy written in ***FOPSL***. Moreover, it also remains open to come up with techniques to check whether a policy specified in pLTL has ★-property. To this end, it is necessary to develop a mp^{-}ATL* model checker based on the algorithm described in [98]. In this work, we check the Δ-property of propositional slices of HIPAA using the technique proposed by Barth *et al.* [11]. It would be desirable to develop a CTL$^{*}{}_{lp}$ model checker to demonstrate that the CTL$^{*}{}_{lp}$ formula of the Δ-property is usable.

Recall that, we do not have any general small model theorem results for the Δ-property and

the policies written in ***FOPSL***. This step in the policy analysis is what causes our technique to be semi-automated and incomplete. To show whether all policies written in ***FOPSL*** have the small model theorem with respect to Δ-property remains an open problem. If one can have a positive general small model theorem result (every policy specified in ***FOPSL*** has a general small model theorem) then it might be interesting to explore whether it is possible to fully automate the policy analysis technique we proposed.

The procedure for checking the Δ-property, described in this thesis, checks to see whether the model general model of the policy $\wp$, $\mathbb{M}_{\wp}$ satisfies the $\delta(\wp)$ formula. Organizations that deal with storing, managing, and disclosing collected personal information, has specific workflow that sometime prescribes what it possible and what is not permitted. The most permissive model of a policy $\wp$, $\mathbb{M}_{\wp}$ has all possible executions that a system can have which tries to check compliance with $\wp$. However, the model of a real organization can be more restrictive than $\mathbb{M}_{\wp}$ with respect to $\wp$. In that case, one first have to verify that for a given policy $\wp$, the Δ-property holds. Once this is verified it provides the assurance that the policy is well-formed. Now one can have a situation where the policy $\wp$ has the Δ-property, but the specific model of the organization does not have prescribed flows that allow principals to discharge their obligations. This is due to the fact that one can incur an obligation according to the $\wp$ but the model of the organization does not have any prescribed execution where the obligation is attempted to be fulfilled. In that case, one has to verify that the specific model of the organization $\mathbb{M}_o$ satisfies the $\delta(\wp)$ formula ($\mathbb{M}_o \models \delta(\wp)$). This will ensure that the organization's model allows any incurred obligations to be successfully discharged. Considering these real life models (workflows) from hospitals and checking whether these models allows all incurred obligations to be successfully discharged is open.

When checking weak compliance, our weak compliance checking algorithm reports whether there is a policy violation. However, it cannot report any useful diagnostic information which can help the user to understand the exact cause of violation for future references. Enhancing the algorithm to output proper diagnostic information is also open. Moreover, our current weak compliance checking algorithm assumes the log or the history to be past-complete. It signifies

that everything that is true or false is completely captured in the log or the history. However, in a real life scenario, this might not be a valid assumption. This is due to the fact that the logs can be distributed and also that the logger might not be correct, and it might cause the information to be not logged properly. In such case, one can extend our compliance checking algorithm to handle incompleteness. Moreover, our compliance checking algorithm cannot handle bounded liveness properties (or, obligations). More precisely, it cannot check whether an obligation is violated or incurred. Extending the language and the algorithm to support monitoring of obligation fulfillment is open. This dissertation does not contain any experimental results for the scalability of the compliance checking algorithm. Evaluating the algorithm with real life log or execution traces is also open.

Currently, we only consider HIPAA in our policy analysis case study. Although, HIPAA is one of the largest and most complicated privacy policies, one might also consider other practical privacy policies like Google's [57] privacy policy, Facebook's [49] privacy policy, SOX [116], and GLBA [2], to understand the efficacy of our policy analysis technique.

In our work on providing assurance that all the incurred obligations will have proper authorization according to some access control policy in presence of obligations, we do not consider how to recover from accountability violation. Prior work [110] provide techniques to recover from accountability violation but makes the simplifying assumption that cascading obligations cannot happen. One might explore whether it is possible to relax this constraint. Moreover, we [28, 67, 109, 110] propose to maintain accountability as an invariant by denying actions that violate accountability. However, when one wants to provide diagnostic information why certain actions were denied one might end up revealing the security policy to the user whose action was denied. This is undesirable in most cases. One might take a different route where one provides a user with an alternative plan of actions which might enable the user to perform its desired action. Prior work [111] has provided a solution based on partial order planners [19,71,99,106]. However, they make the simplifying assumption that no cascading obligations can occur. One can explore whether it is possible to relax this assumption and come up with a reasonably feasible solution to

this problem. Moreover, one might explore whether it is possible to assign blames to user [68] when an obligation goes unfulfilled and there can be cascading obligations in the system.

Appendix A: HIPAA PRIVACY RULE SPECIFICATION IN *FOPSL*

To have a better understanding of how the norms are derived from the regulation text, please consult [35] Note that, the policies are abstracted in some places. The abstractions do not influence the policy analysis results. To get a finer-grained HIPAA policy, see the above pdf document. Moreover, the current encoding has a simplified version of 164.524.

Positive Norms

```
% <164/502/a/1/i>
inrole(p1, covered-entity) ∧ inrole(p2, individual) ∧ inrole(q, individual) ∧ samePerson(p2, q) ∧
    in(t, PHI)

% <164/502/a/1/iii>
inrole(p1, covered-entity) ∧ inrole(q, individual) ∧ in(t, PHI) ∧ incidentToUseDisclosure(p1, p2, q,
    t, u)

% <164/502/a/1/iv> % Abstracted obtained authorization
inrole(p1, covered-entity) ∧ inrole(q, individual) ∧ in(t, PHI) ∧
( ∃ m27 : M. ( (⟐(send(q, p1, m27))) ∧ satisfiesAllValidAuthorizationRequirement(m27, p1, p2, q,
    t, u)
∧ (¬(violatesValidAuthorizationRequirement(m27, p1, p2, q, t, u)))
) )

% <164/502/a/2/ii>
inrole(p1, covered-entity) ∧ inrole(p2, secretary) ∧ inrole(q, individual) ∧ in(t, PHI) ∧ purpose(u,
    compliance-investigation)

% <164/502/d/1>
inrole(p1, covered-entity) ∧ inrole(p2, business-associate) ∧ inrole(q, individual) ∧ in(t, PHI) ∧
    purpose(u, creating-deidentified-info) ∧ businessAssociateOf(p2, p1)

% <164/502/e> % FINISHED
inrole(p1, covered-entity) ∧ inrole(p2, business-associate) ∧ inrole(q, individual) ∧ in(t, PHI) ∧
    satisfactoryAssuranceWillSafeGuardInfo(p1, p2, q, t, u) ∧ businessAssociateOf(p2, p1)
∧ ( ¬ ( inrole(p1, covered-entity) ∧ inrole(p2, provider) ∧ providerOf(p2, q) ∧ purpose(u, treatment)
    ) )
```

∧ (¬ (((**inrole**(p1, group-health-plan) ∨ **inrole**(p1,health-insurance-issuer) ∨ **inrole**(p1, HMO)) ∧
inrole(p2, plan-sponsor))
∧
(
restrictUsageAndDisclosure(p1, p2, q, t, u)
)
))
∧ (¬ (**inrole**(p1, agency) ∧ **inrole**(p2, goverment-health-plan) ∧ **purpose**(u,
determine-eligibility-enrollment) ∧ (
((¬(samePerson(p1, p2))) ∧ determineEligibilityEnrollment(p1, p2)) ∨
(¬(eligibilityEnrollmentInfoCollectedBy(p2)))
)))

% <164/502/g/3/ii/A>
inrole(p1, covered-entity) ∧ (**inrole**(p2, parent) ∨ **inrole**(p2, guardian) ∨ **inrole**(p2, loco-parentis)) ∧
inrole(q, individual) ∧ in(t, PHI)
∧ permittedByOtherLaw(p1, p2, q, t, u) ∧ (parentOf(p2, q) ∨ guardianOf(p2, q) ∨ locoParentis(p2,
q))

% <164/502/j/1> % Divide into 3 norms based on receipient role

% PART (1)
(**inrole**(p1, work-force-member) ∨ **inrole**(p1, business-associate))
∧ (**inrole**(p2, health-oversight-agency) ∨ **inrole**(p2, public-health-authority))
∧ in(t, PHI) ∧ (**purpose**(u, investigate-allegation) ∨ **purpose**(u, oversee-conduct-condition))∧
authorizedByLawForPurpose(p2, u) ∧
believesCoveredEntityInvolvedInUnethicalOrDangerous(p1) ∧ coveredEntityWorkForceMember(p1) ∧
coveredEntityBusinessAssociate(p1)

% PART (2)
(**inrole**(p1, work-force-member) ∨ **inrole**(p1, business-associate)) ∧
inrole(p2, healthcare-accreditation-organization) ∧
in(t, PHI) ∧ **purpose**(u, report-unethical-conduct-allegation) ∧
believesCoveredEntityInvolvedInUnethicalOrDangerous(p1) ∧ coveredEntityWorkForceMember(p1) ∧
coveredEntityBusinessAssociate(p1)

% PART (3)
(**inrole**(p1, work-force-member) ∨ **inrole**(p1, business-associate))
inrole(p2, attorney) ∧
in(t, PHI) ∧ **purpose**(u, determine-legal-options) ∧

```
believesCoveredEntityInvolvedInUnethicalOrDangerous(p1) ∧ coveredEntityWorkForceMember(p1) ∧
    coveredEntityBusinessAssociate(p1)
∧ attorneyOf(p2, p1)
```

% <164/502/j/2>

(**inrole**(p1, work-force-member) ∧ **inrole**(p1, victim-of-crime)) ∧ **inrole**(p2, law-enforcement-official) ∧ **inrole**(q, suspected-perpetrator) ∧ in(t, info-164-512-f2i) ∧ coveredEntityWorkForceMember(p1) ∧ victimeOfCrime(p1, q)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% END OF 502 %%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% BEGIN OF 506 %%%

% <164/506/b/1>

inrole(p1, covered-entity) ∧ **inrole**(q, individual) ∧ in(t, PHI) ∧ (**purpose**(u, treatment) ∨ **purpose**(u, payment) ∨ **purpose**(u, health-care-operations)) ∧ obtainedConsent(p1, p2, q, t, u)

% <164/506/c/1>

inrole(p1, covered-entity) ∧ **inrole**(p2, work-force-member) ∧ **inrole**(q, individual) ∧ in(t, PHI) ∧ (**purpose**(u, treatment) ∨ **purpose**(u, payment) ∨ **purpose**(u, health-care-operations)) ∧ workForceMemberOf(p2, p1)

% <164/506/c/2>

inrole(p1, covered-entity) ∧ **inrole**(p2, provider) ∧ **inrole**(q, individual) ∧ in(t, PHI) ∧ **purpose**(u, treatment) ∧
providerOf(p2, q)

% <164/506/c/3> % p2's payment

inrole(p1, covered-entity) ∧ (**inrole**(p2, covered-entity) ∨ **inrole**(p2, provider)) ∧ **inrole**(q, individual) ∧ in(t, PHI) ∧ **purpose**(u, payment) ∧
(**inrole**(p2, provider) ⟶ providerOf(p2, q))

% <164/506/c/4>

inrole(p1, covered-entity) ∧ **inrole**(p2, covered-entity) ∧ **inrole**(q, individual) ∧ in(t, PHI) ∧ bothHasRelationShip(p1, p2, q) ∧
(**purpose**(u, healthcare-op-paras-1-2) ∨ **purpose**(u, healthcare-fraud-abuse-detection) ∨ **purpose**(u, healthcare-fraud-abuse-compliance))

```
% <164/506/c/5>
inrole(p1, covered-entity)  ∧ inrole(p2, covered-entity) ∧ inrole(q, individual) ∧ in(t, PHI) ∧ purpose
     (u, OHC-health-care-operations) ∧
participantOfSameOrganizedHealthcareArrangement(p1, p2)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% END OF 506%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%% BEGIN OF 508 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% <164/508> has no positive norms .....
% It has only two negative norms .....

%%%%%%%%%%%%%%%%%%% END OF 508 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% BEGIN OF 510
     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% <164/510/a>
%%%%%% Separate into two norms
%%%%%%%% Missing exception 164.510(a)(3)(ii), cannot express weak until

%%%%%%%%%%%% PART(1)
inrole(p1, covered-entity) ∧ inrole(p2, clergy) ∧ inrole(q, individual) ∧ in(t, directory-information)
     ∧ purpose(u, directory)
∧ (
 (  (¬(∃ m11 : M. (send(q, p1, m11) ∧ directoryObjection510a(m11, p1, p2, q, t, u)))) 𝒮 ( ∃ m12 :
     M. (send(p1, q, m12) ∧
 opportunityToObject(m12, p1, p2, q, t, u))) ) ∨
 ( (¬( practicalToProvideOpportunityToObject(p1, p2, q, t, u)))
 ∧ consistentWithPriorPreference(p1, p2, q, t, u) ∧ believesInBestInterest(p1, p2, q, t, u))
)

%%%%%%%%%%%%%%%%% PART (2)
inrole(p1, covered-entity) ∧ inrole(q, individual) ∧ (in(t, directory-information) ∧ (¬ (in(t,
     religious-affiliation))))∧ purpose(u, directory)
∧ (
```

```
 (  (¬(∃ m13 : M. (send(q, p1, m13) ∧ directoryObjection510a(m13, p1, p2, q, t, u)))) S ( ∃ m14 :
    M. (send(p1, q, m14) ∧
 opportunityToObject(m14, p1, p2, q, t, u))) ) ∨
 ( (¬( practicalToProvideOpportunityToObject(p1, p2, q, t, u)))
 ∧ consistentWithPriorPreference(p1, p2, q, t, u) ∧ believesInBestInterest(p1, p2, q, t, u))
)

% <164/510/b/1/i>
inrole(p1, covered-entity) ∧ (inrole(p2, family-member) ∨ inrole(p2, relative) ∨ inrole(p2,
    close-friend) ∨ inrole(p2, identified-164510b))
∧ inrole(q, individual) ∧ in(t, PHI) ∧ relevant-to-involvement(t, p2, q) ∧
( familyMemberOf(p2, q) ∨ relativeOf(p2, q) ∨ closeFriendOf(p2, q) ∨ identifiedPerson(p2, q))

% <164/510/b/1/ii> %Separate into two positive norms for keeping the syntax
% PART (1)
inrole(p1, covered-entity) ∧ (inrole(p2, family-member) ∨ inrole(p2, personal-rep) ∨ inrole(p2,
    responsible-for-care))
∧ in(t, location-condition-death) ∧ purpose(u, notification-164510b) ∧
(personalRepOf(p2, q) ∨ familyMemberOf(p2, q) ∨ responsibleForCareOf(p2, q))

%PART (2)
inrole(p1, covered-entity) ∧ (inrole(p2, family-member) ∨ inrole(p2, personal-rep) ∨ inrole(p2,
    responsible-for-care))
∧ in(t, PHI) ∧ purpose(u, assist-notification-164510b) ∧
(personalRepOf(p2, q) ∨ familyMemberOf(p2, q) ∨ responsibleForCareOf(p2, q))

% <164/510/b/4>

inrole(p1, covered-entity) ∧ inrole(p2, authorizedByLaw-CharterToAssistInDisasterRelief) ∧
in(t, PHI) ∧ purpose(u, coordinate-disclosure-under-164510b1ii) ∧
(
(¬ (profJudgementReqDoNotInterfereWithEmergencyResponse(p1, p2, q, t, u)) )
∨
(
(
inrole(p1, covered-entity) ∧
(inrole(p2, family-member) ∨ inrole(p2, relative) ∨ inrole(p2, close-friend) ∨ inrole(p2,
    identified-164510b)
∨ inrole(p2, personal-rep) ∨ inrole(p2, responsible-for-care)) ∧
inrole(q, individual) ∧ in(t, PHI) ∧
( familyMemberOf(p2, q) ∨ relativeOf(p2, q) ∨ closeFriendOf(p2, q) ∨
```

```
identifiedPerson(p2, q) ∨ personalRepOf(p2, q) ∨ familyMemberOf(p2, q) ∨
    responsibleForCareOf(p2, q))
∧ (⟐(available(p1, q) ∧ hasCapabilityToMakeHealthCareDecisions(q)))
⟶
(
(
∃ m15 : M. (⟐(send(q ,p1, m15) ∧ isAgreement164510b2(m15, p1, p2, q, t, u)))
)
∨
(
(¬(∃ m16 : M. (send(q, p1, m16) ∧ isObjection164510b2(m16, p1, p2, q, t, u))))
S
(∃ m17 : M. (send(p1, q, m17) ∧ isOpportunityToObject(m17, p1, p2, q, t, u)))
)
 ∨
 (professionalJudgementIndividualDoesNotObject(p1, p2, q, t, u))
)
)
∧
(
inrole(p1, covered-entity) ∧
(inrole(p2, family-member) ∨ inrole(p2, relative) ∨ inrole(p2, close-friend) ∨ inrole(p2,
    identified-164510b)
∨ inrole(p2, personal-rep) ∨ inrole(p2, responsible-for-care)) ∧
inrole(q, individual) ∧ in(t, PHI) ∧
( familyMemberOf(p2, q) ∨ relativeOf(p2, q) ∨ closeFriendOf(p2, q) ∨
identifiedPerson(p2, q) ∨ personalRepOf(p2, q) ∨ familyMemberOf(p2, q) ∨
    responsibleForCareOf(p2, q))
∧ (¬(⟐(available(p1, q) ∧ hasCapabilityToMakeHealthCareDecisions(q))) )
⟶
(
professionalJudgementIsInBestInterestof164510b3(p1, p2, q, t, u)
∧
relevantToInvolvement(p1, p2, q, t, u)
)
)
)
)

%%%%%%%%%%%%%%%%%%%%%%%%% END OF 510 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

%%%%%%%%%%%%%%%%%%%%%%%%% *BEGIN OF 512* %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% <164/512/a/1>

%%%%%%%%%%%% THIS MIGHT NEED TO BE REWRITTEN TO BE COMPREHENSIBLE

%%%%%%%%%%%% CURRENTLY THIS NORM IS INCOMPREHENSIBLE

%%%%%%%%%%%% A LITTLE TRICK LIKE BOOLEAN SIMPLIFICATION MIGHT HELP TO

%%%%%%%%%%%% TO SEPARATE THIS INTO MULTIPLE POSITIVE NORMS

% <164/512/c> %%%%%% Separate into two norms

%%%%%%%%%%%% For this norm only the definition needs to be fixed for

%%%%%%%%%% the function which returns all pair of obligations

%%%%%%%%% PART (1)

inrole(p1, covered-entity) ∧ **inrole**(p2, government-authority) ∧
inrole(q, individual) ∧ in(t, PHI) ∧ **purpose**(u, reports-of-abuse) ∧
isRequiredByLaw(p1, p2, q, t, u) ∧
authorizedByLawForPurpose(p2, u) ∧ believesVictimOfAbuse(p1, q) ∧
(isRequiredByLaw(p1, p2, q, t, u) ∨ individualHasAgreed(p1, p2, q, t, u)
∨ (authorizedByStatueRegulation(p1, p2, q, t, u) ∧ (
    believesDisclosureNecessaryToPreventHarm(p1, p2, q, t, u) ∨
(incapacitated(q) ∧ assuranceDisclosureNotUsedAgainstIndividual(p1, p2, q, t, u) ∧
believesWaitingForAgreementWouldHinderEnforcement(p1, p2, q, t, u)))))
∧ ((
∃ m25 : M. ((◈(**send**(p1, q, m25))) ∧ isNoticeOfReport(m25, p1, p2, q, t, u))
) ∨ believesNoticeWouldRiskIndividual(p1, p2, q, t, u))

%%%%%%%%%%%%%%%%%% PART (2)

inrole(p1, covered-entity) ∧ **inrole**(p2, government-authority) ∧
inrole(q, individual) ∧ in(t, PHI) ∧ **purpose**(u, reports-of-abuse) ∧
isRequiredByLaw(p1, p2, q, t, u) ∧
authorizedByLawForPurpose(p2, u) ∧ believesVictimOfAbuse(p1, q) ∧
(isRequiredByLaw(p1, p2, q, t, u) ∨ individualHasAgreed(p1, p2, q, t, u)
∨ (authorizedByStatueRegulation(p1, p2, q, t, u) ∧ (
    believesDisclosureNecessaryToPreventHarm(p1, p2, q, t, u) ∨
(incapacitated(q) ∧ assuranceDisclosureNotUsedAgainstIndividual(p1, p2, q, t, u) ∧
believesWaitingForAgreementWouldHinderEnforcement(p1, p2, q, t, u)))))
∧ (∃ m26 : M. (◇(**send**(p1, q, m26) ∧ isNoticeOfReport(m26, p1, p2, q, t, u))))

% <164/512/e/1/i> % can be abstracted % a little different from Henry's

inrole(p1, covered-entity) ∧ (**inrole**(p2, court) ∨ **inrole**(p2, judicial-administrative-tribunal)) ∧
in(t, PHI) ∧ **purpose**(u, judicial-administrative-proceeding) ∧
isRequiredByLaw(p1, p2, q, t, u) ∧
(∃ m4 : M. (⟐(**send**(p2, p1, m4)) ∧ isOrder(m4, p1, p2, q, t)))

% <164/512/e/1/ii>
inrole(p1, covered-entity) ∧ in(t, PHI) ∧ **purpose**(u, judicial-administrative-proceeding)
∧ isRequiredByLaw(p1, p2, q, t, u)
∧ (∃ m5 : M. (⟐(**send**(p2, p1, m5)) ∧ isLawfulProcess(m5, p1, p2, q, t, u)))
∧
(
(
∃ m18 : M. (⟐(**send**(p2, p1, m18) ∧ satisfiesConditionOf512e1iii(m18, p1, p2, q, t, u)))
)
∨
(
∃ m19 : M. (⟐(**send**(p2, p1, m19) ∧ satisfiesConditionOf512e1iv(m19, p1, p2, q, t, u)))
)
)

% <164/512/e/1/vi>
inrole(p1, covered-entity) ∧ in(t, PHI) ∧ **purpose**(u, judicial-administrative-proceeding)
∧ isRequiredByLaw(p1, p2, q, t, u) ∧
(∃ m20 : M. (⟐(**send**(p2, p1, m20) ∧ isLawfulProcess(m20, p1, p2, q, t, u) ∧
madeReasonableEffortToNotify(p1, p2, q, t, u))))

% <164/512/f/1/i> %%%%%%%%%%%% This is different from Henry's
inrole(p1, covered-entity) ∧ **inrole**(p2, law-enforcement-official) ∧ in(t, PHI) ∧
(**purpose**(u, law-enforcement) ∧ (¬(**purpose**(u, reports-of-child-abuse))))
∧ isRequiredByLaw(p1, p2, q, t, u) ∧ (¬(isRequiredByLaw512c1i(p1, p2, q, t, u)))

% <164/512/f/1/ii>
inrole(p1, covered-entity) ∧ **inrole**(p2, law-enforcement-official) ∧ in(t, PHI) ∧ **purpose**(u,
law-enforcement)
∧ isRequiredByLaw(p1, p2, q, t, u) ∧
((inComplianceWithCourtOrder(p1, p2, q, t, u)) ∨ (inComplianceWithGrandJurySubpoena(p1, p2, q,
t, u))

∨ ((inComplianceWithAdministrativeRequest(p1, p2, q, t, u)) ∧ (minimumNecessary(p1, p2, m, u)) ∧
(deIdentifiedInformationNotSufficient(u))))

% *<164/512/f/2>*

inrole(p1, covered-entity) ∧ **inrole**(p2, law-enforcement-official) ∧ (in(t, info-512-f-2) ∧ in(t, PHI))
∧ **purpose**(u, law-enforcement-relevant-identification-location)
∧ isRequiredByLaw(p1, p2, q, t, u) ∧
(∃ m23: M. ((⟐(**send**(p2, p1, m23))) ∧ isRequestFor(m23, p1, p2, q, t, u)))

% *<164/512/f/3/i>*

inrole(p1, covered-entity) ∧ **inrole**(p2, law-enforcement-official) ∧ (**inrole**(q, victim-of-crime) ∨ **inrole**(q, suspected-victim-of-crime))
∧ in(t, PHI) ∧ **purpose**(u, law-enforcement) ∧ isRequiredByLaw(p1, p2, q, t, u) ∧
(∃ m22 : M. ((⟐(**send**(p2, p1, m22))) ∧ isRequestFor(m22, p1, p2, q, t, u)))
∧
(∃ m21 : M. ((⟐(**send**(q, p1, m21))) ∧ isAgreementTo(m21, p1, p2, q, t, u)))

% *<164/512/f/3/ii>*

inrole(p1, covered-entity) ∧ **inrole**(p2, law-enforcement-official) ∧
((**inrole**(q, victim-of-crime) ∨ **inrole**(q, suspected-victim-of-crime)) ∧ **inrole**(q, emergency-circumstance))
∧ in(t, PHI) ∧ **purpose**(u, law-enforcement) ∧
isRequiredByLaw(p1, p2, q, t, u) ∧
(∃ m24 : M. ((⟐(**send**(p2, p1, m24))) ∧ (isRequestFor(m24, p1, p2, q, t, u)))) ∧
isNeededToDetermineCrime(p1, p2, q, t, u) ∧ notUsedAgainstVictim(p1, p2, q, t, u) ∧
isActivityAdverselyAffectedByWait(p1, p2, q, t, u) ∧ believesInBestInterest(p1, p2, q, t, u)

% *<164/512/f/4>*

inrole(p1, covered-entity) ∧ **inrole**(p2, law-enforcement-official) ∧ **inrole**(q, deceased)
∧ in(t, PHI) ∧ **purpose**(u, suspicious-death-notification) ∧ isRequiredByLaw(p1, p2, q, t, u) ∧
deathMayBeResultOfACrime(p1, q)

% *<164/512/f/5>*

inrole(p1, covered-entity) ∧ **inrole**(p2, law-enforcement-official) ∧ in(t, PHI) ∧ **purpose**(u, report-possible-crime-on-premise)
∧ isRequiredByLaw(p1, p2, q, t, u) ∧ believesEvidenceOfCrimeOnPremises(p1, p2, q, t, u)

% *<164/512/f/6>*

(**inrole**(p1, provider) ∧ **inrole**(p1, covered-entity))∧

inrole(p2, law-enforcement-official) ∧ in(t, PHI) ∧ **purpose**(u,
    alert-of-crime-commission-location-victims-perpetrator)

∧ isRequiredByLaw(p1, p2, q, t, u) ∧

providingEmergencyHealthCare(p1, q)

∧ appearsNecessaryToAlertOfCrimeCommissionLocationOfVictimsPerpetrator(p1, p2, q, t, u)

∧ (¬(believesEmergencyResultOfAbuseNeglectOrDomesticViolence(p1, q)))

% <164/512/b/1/i> % separate into two norms

%PART (1)

inrole(p1, covered-entity) ∧ **inrole**(p2, public-health-authority) ∧ in(t, PHI) ∧
    authorizedByLawForPurpose(p2, u)

∧ (**purpose**(u, disease-prevention-control) ∨ **purpose**(u, public-health-surveillance) ∨ **purpose**(u,
    public-health-investigation)

∨ **purpose**(u, public-health-intervention))

%PART (2) about foreign-government-agency

inrole(p1, covered-entity) ∧ **inrole**(p2, foreign-government-agency) ∧ in(t, PHI) ∧
    authorizedByLawForPurpose(p2, u)

∧ (**purpose**(u, disease-prevention-control) ∨ **purpose**(u, public-health-surveillance) ∨ **purpose**(u,
    public-health-investigation)

∨ **purpose**(u, public-health-intervention))

% <164/512/b/1/ii>

inrole(p1, covered-entity) ∧ (**inrole**(p2, public-health-authority) ∨ **inrole**(p2, government-authority)) ∧

authorizedByLawForPurpose(p2, u) ∧ in(t, PHI) ∧ **purpose**(u, reports-of-child-abuse)

% <164/512/b/1/iii>

inrole(p1, covered-entity) ∧ **inrole**(p2, responsible-for-FDA-regulated-product) ∧ in(t, PHI) ∧ **purpose**
    (u, quality-safety-effectiveness-activities)

% <164/512/b/1/iv>

inrole(p1, covered-entity) ∧ **inrole**(p2, risk-of-contracting-or-spreading-disease) ∧ in(t, PHI) ∧
    purpose(u, notify-for-public-health-intervention)

% <164/512/b/1/v> % Separate into two norms

%%%%%%%% PART (1)

((**inrole**(p1, covered-entity) ∧ **inrole**(p1, covered-health-care-provider)) ∧ (**inrole**(p1,
work-force-member) ∨ **inrole**(p1, provides-medical-surveillance)
∨ **inrole**(p1, provides-injury-evaluation))) ∧
inrole(p2, employer) ∧ (**inrole**(q, individual) ∧ **inrole**(q, work-force-member))
∧ in(t, workplace-injury-findings) ∧ **purpose**(u, obligation-to-record-workplace-injury)
∧ workForceMemberOf(q, p2) ∧ (workForceMemberOf(p1, p2) ∨ providerOfMedicalSurveillance(p1, p2) ∨
providerOfInjuryEvaluation(p1, p2))
∧ (∃ m10 : M. (⟐(**send**(p1, q, m10)) ∧ isNoticeOfWorkplaceDisclosure(m10)))

%%%%%%% PART (2)
((**inrole**(p1, covered-entity) ∧ **inrole**(p1, covered-health-care-provider)) ∧ (**inrole**(p1,
work-force-member) ∨ **inrole**(p1, provides-medical-surveillance)
∨ **inrole**(p1, provides-injury-evaluation))) ∧
inrole(p2, employer) ∧ (**inrole**(q, individual) ∧ **inrole**(q, work-force-member))
∧ in(t, medical-surveillance-findings) ∧ **purpose**(u, obligation-to-perform-medical-surveillance)
∧ workForceMemberOf(q, p2) ∧ (workForceMemberOf(p1, p2) ∨ providerOfMedicalSurveillance(p1, p2) ∨
providerOfInjuryEvaluation(p1, p2))
∧ (∃ m910 : M. (⟐(**send**(p1, q, m910)) ∧ isNoticeOfWorkplaceDisclosure(m910)))

% <164/512/d/3> %%%%% Different from Henry's, Limin's looks more reasonable
inrole(p1, covered-entity) ∧ **inrole**(p2, health-oversight-agency) ∧ in(t, PHI)
authorizedByLawForPurpose(p2, u) ∧
(**purpose**(u, over-sight-government-benefit-programs-health-eligibility) ∧
purpose(u, over-sight-government-regulated-entity-health-compliance)∧
purpose(u, over-sight-subject-to-civil-rights-health-compliance)
) ∧ ((¬(isSubjectOfInvestigation(q))) ∧ ((relatedToReceiptHealthCare(u)) ∨
(relatedToPublicHealthBenefits(u))
∨ (relatedToPublicBenefitsQualificationDependsOnHealth(u))))

% <164/512/g/1> % different from Henry's
inrole(p1, covered-entity) ∧ (**inrole**(p2, coroner) ∨ **inrole**(p2, medical-examiner)) ∧ **inrole**(q, deceased)
∧ in(t, PHI)
∧ (**purpose**(u, identification) ∨ **purpose**(u, determining-cause-of-death) ∨ **purpose**(u,
other-duties-authorized-by-law))

% <164/512/g/2> % separate into two norms

%PART (1)

inrole(p1, covered-entity) ∧ **inrole**(p2, funeral-director) ∧ **inrole**(q, deceased) ∧ in(t, PHI) ∧ **purpose**(u, funeral-director-duties)
∧ necessaryForDuties(p1, p2, q, t, u)

%PART (2)

inrole(p1, covered-entity) ∧ **inrole**(p2, funeral-director) ∧ **inrole**(q, almost-deceased) ∧ in(t, PHI) ∧ **purpose**(u, funeral-director-duties)
∧ necessaryForDuties(p1, p2, q, t, u) ∧ earlyDisclosureNecessary(p1, p2, q, t, u)

% <164/512/h>

inrole(p1, covered-entity) ∧ (**inrole**(p2, organ-procurement-org) ∨ **inrole**(p2, engaged-in-procurement-banking-transplantation-of-organs-eyes-tissue)) ∧
in(t, PHI) ∧ **purpose**(u, facilitate-organ-eye-tissue-donation-transplantation)

% <164/512/i/1>

inrole(p1, covered-entity) ∧ **inrole**(p2, researcher) ∧ **inrole**(q, deceased) ∧ in(t, PHI) ∧ **purpose**(u, research)
∧ (∃ p4 : P. (∃ m6 : M. ((**inrole**(p4, IRB) ∨ **inrole**(p4, privacy-board)) ∧ (⟐(**send**(p4, p1, m6))) ∧
isApprovalAuthorizationWaiver(m6, p4, p1, p2, q, t) ∧ otherNecessaryRequirement164512i1(p4, p1, p2, q, t, u)
)))

% <164/512/j/1/i>

inrole(p1, covered-entity) ∧ in(t, PHI) ∧ **purpose**(u, lessen-health-threat) ∧ consistentWithApplicableLaw(p1, p2, q, t, u)
∧ believesNecessaryToLessenHealthThreat(p1, p2, q, t, u) ∧ believesCanLessenThreat(p1, p2, q, t, u)

% <164/512/j/1/ii/A>

inrole(p1, covered-entity) ∧ **inrole**(p2, law-enforcement-official) ∧ **inrole**(q, individual) ∧ in(t, PHI) ∧ **purpose**(u, identify-apprehend)
∧ consistentWithApplicableLaw(p1, p2, q, t, u) ∧ (∃ m7 : M. (⟐(**send**(q, p1, m7)) ∧ isAdmissionOfCrime(m7, q) ∧ believesCrimeCausedSeriousHarm(p1, m7)))
∧
(¬(learnedWhileTreatingPropensityForCrime(p1, q)))
∧
(¬(learnedThroughRequestForTreatmentOfPropensityForCrime(p1, q, t)))
∧
(

(∃ m8 : M.((⟐(**send**(q, p1, m8)) ∧ isAdmissionOfCrime(m8, q) ∧ containsMessage(m8, m) ∧
contains(m8, q, t))))
∧ in(t, some-PHI-512j3)
)

% *<164/512/j/1/ii/B>*
inrole(p1, covered-entity) ∧ **inrole**(p2, law-enforcement-official) ∧ **inrole**(q, individual) ∧
in(t, PHI) ∧ **purpose**(u, identify-apprehend) ∧ consistentWithApplicableLaw(p1, p2, q, t, u) ∧
beleivesEscapedLawfulCustody(p1, q)

% *<164/512/k/1/i>*
inrole(p1, covered-entity) ∧ **inrole**(q, armed-forces-personnel) ∧ in(t, PHI)
∧ (∃ p5: P. (deemedNecessaryForMission(p5, p1, p2, q, t, u) ∧
publishedInFrCommandAuthorityForDisclosure(p5, p1, p2, q, t, u)
∧ publishedInFrPurposeForDisclosure(p5, p1, p2, q, t, u)))

% *<164/512/k/1/ii>*
(**inrole**(p1, covered-entity) ∧ **inrole**(p1, component-of-DOD-or-DOT)) ∧ **inrole**(p2, DVA) ∧ **inrole**(q,
previous-armed-force-personnel)
∧ in(t, PHI) ∧ **purpose**(u, eligibilityDeterminationForVeteransBenefit)

% *<164/512/k/1/iii>*
(**inrole**(p1, covered-entity) ∧ **inrole**(p1, component-of-DVA)) ∧ **inrole**(p2, component-of-DVA) ∧ **inrole**(q,
previous-armed-force-personnel)
∧ in(t, PHI) ∧ (**purpose**(u, eligibilityDeterminationForVeteransBenefit) ∧ **purpose**(u,
provisionOfVeteransBenefits))

% *<164/512/k/1/iv>*
inrole(p1, covered-entity) ∧ **inrole**(q, foreign-armed-forces-personnel) ∧ in(t, PHI)
∧ (∃ p6: P. (deemedNecessaryForMission(p6, p1, p2, q, t, u) ∧
publishedInFrCommandAuthorityForDisclosure(p6, p1, p2, q, t, u)
∧ publishedInFrPurposeForDisclosure(p6, p1, p2, q, t, u)))

% *<164/512/k/2>*
inrole(p1, covered-entity) ∧ **inrole**(p2, authorized-federal-official) ∧ in(t, PHI) ∧ **purpose**(u,
nationalSecurityActivities)

% *<164/512/k/3>*
inrole(p1, covered-entity) ∧ **inrole**(p2, authorized-federal-official) ∧ in(t, PHI) ∧ **purpose**(u,
nationalSecurityActivities)

```
∧ (purpose(u, provisionToProtectiveServices) ∧ purpose(u, conductInvestigations18USC871-9))

% <164/512/k/4>
(inrole(p1, covered-entity) ∧ inrole(p1, component-of-DOS)) ∧ inrole(p2, DOS-official) ∧ inrole(q,
    individual) ∧ in(t, medical-suitability)
∧ (purpose(u, securityClearance-EO-10450-12698) ∨ purpose(u,
    determineAvailabilityForForeignService) ∨ purpose(u, determineFamilyCompany))

% <164/512/k/5/i>
inrole(p1, covered-entity) ∧ (inrole(p2, correctional-institution) ∨ inrole(p2,
    law-enforcement-official)) ∧ inrole(q, inmate)
∧ in(t, PHI) ∧ lawfulCustodyOf(q, p2) ∧ receipientBelievesDisclosureNecessary(p2, p1, p2, q, t,
    u)

% <164/512/k/6/i>
(inrole(p1, health-plan) ∧ inrole(p1, government-public-benefits-program)) ∧ inrole(p2,
    government-public-benefits-program)
∧ in(t, PHI) ∧ relatesToEligibilityEnrollmentInHealthPlan(q, t, p1) ∧
    disclosureRequiredOrAuthorizedByStatuteOrRegulation(p1, p2, q, t, u)

% <164/512/k/6/ii>
(inrole(p1, covered-entity) ∧ inrole(p1,
    government-agency-administering-public-health-benefits-program))
∧ (inrole(p2, covered-entity) ∧ inrole(p2,
    government-agency-administering-public-health-benefits-program))
∧ in(t, PHI) ∧ servesSimilarPopulation(p1, p2) ∧ relatesToTheProgram(q, t, p1) ∧
( necessaryForCoordination(p1, p2, q, t, u) ∨ necessaryForImprovingManagement(p1, p2, q, t, u))


% <164/512/l>
inrole(p1, covered-entity) ∧ inrole(q, individual) ∧ in(t, PHI) ∧ purpose(u,
    complying-with-laws-worker-compensation-or-other-programs) ∧
    authorizedAndNecessaryForWorkersCompensationLaws(p1, p2, q, t, u)


%%%%%%%%%%% END OF 512 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% p7, m9 %%%%%%%%%%

%%%%%%%%%%% BEGIN OF 514 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

% *<164/514/e/1>*

inrole(p1, covered-entity) ∧ in(t, PHI) ∧ (**purpose**(u, research) ∨ **purpose**(u, public-health) ∨ **purpose**(u, health-care-operations))

∧ isLimitedData(t) ∧ ⟐(∃ m9 : M. (**send**(p2, p1, m9) ∧ isLimitedDataUseAgreement(m9) ∧ satisfiesOtherLimitedDataUseAgreementRequirement(p1, p2, m9))

)

% *<164/514/e/3/ii>* %% ¬ *IN CMU DOCUMENT, NEEDS TO BE ENCODED ANYWAYS*

inrole(p1, covered-entity) ∧ **inrole**(p2, business-associate) ∧ **inrole**(q, individual) ∧ in(t, PHI) ∧ **purpose**(u, creating-limited-data-set)

∧ businessAssociateOf(p2, p1)

% *<164/514/f/1>*

inrole(p1, covered-entity) ∧ (**inrole**(p2, business-associate) ∨ **inrole**(p2, related-foundation))

∧ (in(t, demographic-info) ∨ in(t, healthcare-dates)) ∧ **purpose**(u, fundraising)

∧ (businessAssociateOf(p2, p1) ∨ relatedFoundationOf(p2, p1))

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% *END OF 514* %%

%%%%%%%% *p7, m10* %%%%%%

% % % % % % % % *Synthetic Policy Rule 1* %%%%%%%

inrole(p1, secretary)∧ **inrole**(p2, covered-entity)∧ **inrole**(q, individual)∧

purpose(u, compliance-investigation)∧ request(p1, p2, q, PHI)

% % % % % % % % % % % % % % *Synthetic Policy Rule 2* %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

inrole(p1, individual)∧ **inrole**(p2, covered-entity)∧ **inrole**(q, individual)∧

purpose(u, access-request) ∧ samePerson(p1, q) ∧ request(p1, p2, q, PHI)

% % % % % % % % % % *Synthetic Policy Rule 3* %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

inrole(p1, parent)∧ **inrole**(p2, covered-entity) ∧ **inrole**(q, individual) ∧

purpose(u, accessrequest) ∧ parentOf(p1, q) ∧ request(p_1, p_2, q, PHI)

% % % % % % % % % % % % % % % % % % % *END OF POSITIVE NORMS* %%%%%%%%%%%%%%%%%%%%

Negative Norms

%%%%%%%%%%%%%%%%%%%%%%%% *START OF NEGATIVE NORM* %%%%%%%%%%%%%%%%%%%%%%%%%%%%

% *<164/502/b>*

inrole(p1, covered-entity) ∧ **inrole**(q, individual) ∧ in(t, PHI)

$\longrightarrow$

(

(believesMinimumNecessaryForPurpose(p1, p2, q, t, u))

$\vee$

(**inrole**(p2, provider) $\wedge$ **purpose**(u, treatment))

$\vee$

(**inrole**(p1, covered-entity) $\wedge$ **inrole**(p2, individual) $\wedge$ **inrole**(q, individual) $\wedge$ samePerson(p2, q) $\wedge$
in(t, PHI))

$\vee$

(

inrole(p1, covered-entity) $\wedge$ **inrole**(q, individual) $\wedge$ in(t, PHI) $\wedge$

($\exists$ m127 : M. (($\diamondminus$(**send**(q, p1, m127))) $\wedge$ satisfiesAllValidAuthorizationRequirement(m127, p1, p2,
q, t, u)

$\wedge$ ($\neg$(violatesValidAuthorizationRequirement(m127, p1, p2, q, t, u)))

))

)

$\vee$

(**inrole**(p1, covered-entity) $\wedge$ **inrole**(p2, secretary) $\wedge$ **inrole**(q, individual) $\wedge$ in(t, PHI) $\wedge$ **purpose**(u,
compliance-investigation))

$\vee$

(

% <164/512/c> %%%%%% Separate into two norms

%%%%%%%%%%% For this norm only the definition needs to be fixed for

%%%%%%%%%% the function which returns all pair of obligations

%%%%%%%%% PART (1)

inrole(p1, covered-entity) $\wedge$ **inrole**(p2, government-authority) $\wedge$

inrole(q, individual) $\wedge$ in(t, PHI) $\wedge$ **purpose**(u, reports-of-abuse) $\wedge$

isRequiredByLaw(p1, p2, q, t, u) $\wedge$

authorizedByLawForPurpose(p2, u) $\wedge$ believesVictimOfAbuse(p1, q) $\wedge$

(isRequiredByLaw(p1, p2, q, t, u) $\vee$ individualHasAgreed(p1, p2, q, t, u)

$\vee$ (authorizedByStatueRegulation(p1, p2, q, t, u) $\wedge$ (

believesDisclosureNecessaryToPreventHarm(p1, p2, q, t, u) $\vee$

(incapacitated(q) $\wedge$ assuranceDisclosureNotUsedAgainstIndividual(p1, p2, q, t, u) $\wedge$

believesWaitingForAgreementWouldHinderEnforcement(p1, p2, q, t, u)))))

$\wedge$ ((

$\exists$ m125 : M. (($\diamondminus$(**send**(p1, q, m125))) $\wedge$ isNoticeOfReport(m125, p1, p2, q, t, u))

) $\vee$ believesNoticeWouldRiskIndividual(p1, p2, q, t, u))

)

$\vee$

(

inrole(p1, covered-entity) $\wedge$ **inrole**(p2, government-authority) $\wedge$

inrole(q, individual) ∧ in(t, PHI) ∧ **purpose**(u, reports-of-abuse) ∧
isRequiredByLaw(p1, p2, q, t, u) ∧
authorizedByLawForPurpose(p2, u) ∧ believesVictimOfAbuse(p1, q) ∧
(isRequiredByLaw(p1, p2, q, t, u) ∨ individualHasAgreed(p1, p2, q, t, u)
∨ (authorizedByStatueRegulation(p1, p2, q, t, u) ∧ (
    believesDisclosureNecessaryToPreventHarm(p1, p2, q, t, u) ∨
(incapacitated(q) ∧ assuranceDisclosureNotUsedAgainstIndividual(p1, p2, q, t, u) ∧
believesWaitingForAgreementWouldHinderEnforcement(p1, p2, q, t, u)))))
∧ (∃ m226 : M. (◇(**send**(p1, q, m226) ∧ isNoticeOfReport(m226, p1, p2, q, t, u))))
)
∨
(
% <164/512/e/1/i> % can be abstracted % a little different from Henry's
inrole(p1, covered-entity) ∧ (**inrole**(p2, court) ∨ **inrole**(p2, judicial-administrative-tribunal)) ∧
    in(t, PHI) ∧ **purpose**(u, judicial-administrative-proceeding) ∧
isRequiredByLaw(p1, p2, q, t, u) ∧
(∃ m554 : M. (◈(**send**(p2, p1, m554)) ∧ isOrder(m554, p1, p2, q, t))
))
∨(
% <164/512/e/1/ii>
inrole(p1, covered-entity) ∧ in(t, PHI) ∧ **purpose**(u, judicial-administrative-proceeding)
∧ isRequiredByLaw(p1, p2, q, t, u)
∧ (∃ m665 : M. (◈(**send**(p2, p1, m665)) ∧ isLawfulProcess(m665, p1, p2, q, t, u)))
∧
(
(
 ∃ m118 : M. (◈(**send**(p2, p1, m118) ∧ satisfiesConditionOf512e1iii(m118, p1, p2, q, t, u)))
)
∨
(
∃ m119 : M. (◈(**send**(p2, p1, m119) ∧ satisfiesConditionOf512e1iv(m119, p1, p2, q, t, u)))
)
)

∨(
% <164/512/e/1/vi>
inrole(p1, covered-entity) ∧ in(t, PHI) ∧ **purpose**(u, judicial-administrative-proceeding)
∧ isRequiredByLaw(p1, p2, q, t, u) ∧
(∃ m120 : M. (◈(**send**(p2, p1, m120) ∧ isLawfulProcess(m120, p1, p2, q, t, u) ∧
    madeReasonableEffortToNotify(p1, p2, q, t, u))))

```
)
∨(

% <164/512/f/1/i> %%%%%%%%%%%% This is different from Henry's
inrole(p1, covered-entity) ∧ inrole(p2, law-enforcement-official) ∧ in(t, PHI) ∧
( purpose(u, law-enforcement) ∧ (¬(purpose(u, reports-of-child-abuse))))
∧ isRequiredByLaw(p1, p2, q, t, u)  ∧ (¬(isRequiredByLaw512c1i(p1, p2, q, t, u)))
)
∨(

% <164/512/f/1/ii>
inrole(p1, covered-entity) ∧ inrole(p2, law-enforcement-official) ∧ in(t, PHI) ∧ purpose(u,
    law-enforcement)
∧ isRequiredByLaw(p1, p2, q, t, u) ∧
( (inComplianceWithCourtOrder(p1, p2, q, t, u)) ∨ (inComplianceWithGrandJurySubpoena(p1, p2, q,
    t, u))
∨ ((inComplianceWithAdministrativeRequest(p1, p2, q, t, u)) ∧ (minimumNecessary(p1, p2, m, u)) ∧
    (deIdentifiedInformationNotSufficient(u))))
)
∨(
% <164/512/f/2>
inrole(p1, covered-entity) ∧ inrole(p2, law-enforcement-official) ∧ ( in(t, info-512-f-2) ∧ in(t,
    PHI))
∧ purpose(u, law-enforcement-relevant-identification-location)
∧ isRequiredByLaw(p1, p2, q, t, u) ∧
(∃ m223: M. ( (⋄(send(p2, p1, m223))) ∧ isRequestFor(m223, p1, p2, q, t, u) ))
)
∨(
% <164/512/f/3/i>
inrole(p1, covered-entity) ∧ inrole(p2, law-enforcement-official) ∧ (inrole(q, victim-of-crime) ∨ inrole
    (q, suspected-victim-of-crime))
∧ in(t, PHI) ∧ purpose(u, law-enforcement) ∧ isRequiredByLaw(p1, p2, q, t, u) ∧
(∃ m122 : M. ( (⋄(send(p2, p1, m122))) ∧ isRequestFor(m122, p1, p2, q, t, u)))
∧
(∃ m121 : M. ( ( ⋄(send(q, p1, m121))) ∧ isAgreementTo(m121, p1, p2, q, t, u)))
)
∨(

% <164/512/f/3/ii>
inrole(p1, covered-entity) ∧ inrole(p2, law-enforcement-official) ∧
```

((**inrole**(q, victim-of-crime) ∨ **inrole**(q, suspected-victim-of-crime)) ∧ **inrole**(q,
emergency-circumstance))
∧ in(t, PHI) ∧ **purpose**(u, law-enforcement) ∧
isRequiredByLaw(p1, p2, q, t, u) ∧
(∃ m124 : M. ((⟐(**send**(p2, p1, m124))) ∧ (isRequestFor(m124, p1, p2, q, t, u)))) ∧
isNeededToDetermineCrime(p1, p2, q, t, u) ∧ notUsedAgainstVictim(p1, p2, q, t, u) ∧
isActivityAdverselyAffectedByWait(p1, p2, q, t, u) ∧ believesInBestInterest(p1, p2, q, t, u)
)
∨(
% *<164/512/f/4>*
inrole(p1, covered-entity) ∧ **inrole**(p2, law-enforcement-official) ∧ **inrole**(q, deceased)
∧ in(t, PHI) ∧ **purpose**(u, suspicious-death-notification) ∧ isRequiredByLaw(p1, p2, q, t, u) ∧
deathMayBeResultOfACrime(p1, q)
)
∨(

% *<164/512/f/5>*
inrole(p1, covered-entity) ∧ **inrole**(p2, law-enforcement-official) ∧ in(t, PHI) ∧ **purpose**(u,
report-possible-crime-on-premise)
∧ isRequiredByLaw(p1, p2, q, t, u) ∧ believesEvidenceOfCrimeOnPremises(p1, p2, q, t, u)
)
∨(
% *<164/512/f/6>*
(**inrole**(p1, provider) ∧ **inrole**(p1, covered-entity))∧
inrole(p2, law-enforcement-official) ∧ in(t, PHI) ∧ **purpose**(u,
alert-of-crime-commission-location-victims-perpetrator)
∧ isRequiredByLaw(p1, p2, q, t, u) ∧
providingEmergencyHealthCare(p1, q) ∧
appearsNecessaryToAlertOfCrimeCommissionLocationOfVictimsPerpetrator(p1, p2, q, t, u)
∧ (¬(believesEmergencyResultOfAbuseNeglectOrDomesticViolence(p1, q)))
)
)
%%%
%%%%%%%%%%%%%%%%%%%%%%%% *END OF NEGATIVE NORM* %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%% *START OF NEGATIVE NORM* %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% *<164/502/g/3/ii/B>*
inrole(p1, covered-entity) ∧ (**inrole**(p2, parent) ∨ **inrole**(p2, guardian) ∨ **inrole**(p2, loco-parentis)) ∧
inrole(q, individual) ∧ in(t, PHI)
∧ prohibitedByOtherLaw(p1, p2, q, t, u) ∧ (parentOf(p2, q) ∨ guardianOf(p2, q) ∨
locoParentis(p2, q))

$\longrightarrow$ (FALSE)

%%%%%%%%%%%%%%%%%%%%%%%% *END OF NEGATIVE NORM* %%%%%%%%%%%%%%%%%%%%%%%%%%

%%%

%%%%%%%%%%%%%%%%%%%%%%%% *START OF NEGATIVE NORM* %%%%%%%%%%%%%%%%%%%%%%%%%%

% <164/508/a/2> %%% [DONE]

inrole(p1, covered-entity) $\wedge$ in(t, psychotherapy-notes) $\longrightarrow$

(

($\exists$ m129 : M. (($\diamondsuit$(**send**(q, p1, m129))) $\wedge$ satisfiesAllValidAuthorizationRequirement(m129, p1, p2, q, t, u)

$\wedge$ ($\neg$(violatesValidAuthorizationRequirement(m129, p1, p2, q, t, u)))

)

)

$\vee$

(

inrole(p1, covered-entity) $\wedge$ **purpose**(u, counseling-training-programs) $\wedge$ counselingTrainingProgramsOf(p1)

)

$\vee$

(

inrole(p1, covered-entity) $\wedge$ **purpose**(u, defense-in-legal-proceeding) $\wedge$ defenseInLegalProceeding(p1, q)

)

$\vee$

(

inrole(p1, covered-entity) $\wedge$ (**inrole**(p2, coroner) $\vee$ **inrole**(p2, medical-examiner)) $\wedge$ **inrole**(q, deceased) $\wedge$ in(t, PHI)

$\wedge$ (**purpose**(u, identification) $\vee$ **purpose**(u, determining-cause-of-death) $\vee$ **purpose**(u, other-duties-authorized-by-law))

)

$\vee$

(

inrole(p1, covered-entity) $\wedge$ in(t, PHI) $\wedge$ **purpose**(u, lessen-health-threat) $\wedge$ consistentWithApplicableLaw(p1, p2, q, t, u)

$\wedge$ believesNecessaryToLessenHealthThreat(p1, p2, q, t, u) $\wedge$ believesCanLessenThreat(p1, p2, q, t, u)

)

$\vee$

```
(
% <164/512/c> %%%%%% Separate into two norms
%%%%%%%%%%% For this norm only  the definition needs to be fixed for
%%%%%%%%%% the function which returns all pair of obligations
%%%%%%%%% PART (1)
inrole(p1, covered-entity) ∧ inrole(p2, government-authority) ∧
inrole(q, individual) ∧ in(t, PHI) ∧ purpose(u, reports-of-abuse) ∧
isRequiredByLaw(p1, p2, q, t, u) ∧
authorizedByLawForPurpose(p2, u) ∧ believesVictimOfAbuse(p1, q) ∧
(isRequiredByLaw(p1, p2, q, t, u) ∨ individualHasAgreed(p1, p2, q, t, u)
∨ (authorizedByStatueRegulation(p1, p2, q, t, u) ∧ (
    believesDisclosureNecessaryToPreventHarm(p1, p2, q, t, u) ∨
(incapacitated(q) ∧ assuranceDisclosureNotUsedAgainstIndividual(p1, p2, q, t, u) ∧
believesWaitingForAgreementWouldHinderEnforcement(p1, p2, q, t, u)))))
∧ ((
∃ m225 : M. ( (⟐(send(p1, q, m225))) ∧ isNoticeOfReport(m225, p1, p2, q, t, u))
) ∨ believesNoticeWouldRiskIndividual(p1, p2, q, t, u))
)
∨
(
inrole(p1, covered-entity) ∧ inrole(p2, government-authority) ∧
inrole(q, individual) ∧ in(t, PHI) ∧ purpose(u, reports-of-abuse) ∧
isRequiredByLaw(p1, p2, q, t, u) ∧
authorizedByLawForPurpose(p2, u) ∧ believesVictimOfAbuse(p1, q) ∧
(isRequiredByLaw(p1, p2, q, t, u) ∨ individualHasAgreed(p1, p2, q, t, u)
∨ (authorizedByStatueRegulation(p1, p2, q, t, u) ∧ (
    believesDisclosureNecessaryToPreventHarm(p1, p2, q, t, u) ∨
(incapacitated(q) ∧ assuranceDisclosureNotUsedAgainstIndividual(p1, p2, q, t, u) ∧
believesWaitingForAgreementWouldHinderEnforcement(p1, p2, q, t, u)))))
∧ (∃ m126 : M. ( ◇(send(p1, q, m126) ∧ isNoticeOfReport(m126, p1, p2, q, t, u))))
)
∨
(
% <164/512/e/1/i> % can be abstracted % a little different from Henry's
inrole(p1, covered-entity) ∧ (inrole(p2, court) ∨ inrole(p2, judicial-administrative-tribunal)) ∧
    in(t, PHI) ∧ purpose(u, judicial-administrative-proceeding) ∧
isRequiredByLaw(p1, p2, q, t, u) ∧
(∃ m664 : M. ( ⟐(send(p2, p1, m664)) ∧ isOrder(m664, p1, p2, q, t))
))
∨(
% <164/512/e/1/ii>
```

inrole(p1, covered-entity) ∧ in(t, PHI) ∧ **purpose**(u, judicial-administrative-proceeding)
∧ isRequiredByLaw(p1, p2, q, t, u)
∧ (∃ m555 : M. (⟐(**send**(p2, p1, m555)) ∧ isLawfulProcess(m555, p1, p2, q, t, u)))
∧
(
(
∃ m218 : M. (⟐(**send**(p2, p1, m218) ∧ satisfiesConditionOf512e1iii(m218, p1, p2, q, t, u)))
)
∨
(
∃ m219 : M. (⟐(**send**(p2, p1, m219) ∧ satisfiesConditionOf512e1iv(m219, p1, p2, q, t, u)))
)
)
)

∨(
% *<164/512/e/1/vi>*
inrole(p1, covered-entity) ∧ in(t, PHI) ∧ **purpose**(u, judicial-administrative-proceeding)
∧ isRequiredByLaw(p1, p2, q, t, u) ∧
(∃ m220 : M. (⟐(**send**(p2, p1, m220) ∧ isLawfulProcess(m220, p1, p2, q, t, u) ∧
madeReasonableEffortToNotify(p1, p2, q, t, u))))
)
∨(

% *<164/512/f/1/i>* %%%%%%%%%%%% *This is different from Henry's*
inrole(p1, covered-entity) ∧ **inrole**(p2, law-enforcement-official) ∧ in(t, PHI) ∧
(**purpose**(u, law-enforcement) ∧ (¬(**purpose**(u, reports-of-child-abuse))))
∧ isRequiredByLaw(p1, p2, q, t, u) ∧ (¬(isRequiredByLaw512c1i(p1, p2, q, t, u)))
)
∨(

% *<164/512/f/1/ii>*
inrole(p1, covered-entity) ∧ **inrole**(p2, law-enforcement-official) ∧ in(t, PHI) ∧ **purpose**(u,
law-enforcement)
∧ isRequiredByLaw(p1, p2, q, t, u) ∧
((inComplianceWithCourtOrder(p1, p2, q, t, u)) ∨ (inComplianceWithGrandJurySubpoena(p1, p2, q,
t, u))
∨ ((inComplianceWithAdministrativeRequest(p1, p2, q, t, u)) ∧ (minimumNecessary(p1, p2, m, u)) ∧
(deIdentifiedInformationNotSufficient(u))))
)
∨(

% *<164/512/f/2>*

inrole(p1, covered-entity) ∧ **inrole**(p2, law-enforcement-official) ∧ (in(t, info-512-f-2) ∧ in(t, PHI))

∧ **purpose**(u, law-enforcement-relevant-identification-location)

∧ isRequiredByLaw(p1, p2, q, t, u) ∧

(∃ m123: M. ((⟐(**send**(p2, p1, m123))) ∧ isRequestFor(m123, p1, p2, q, t, u)))

)

∨(

% *<164/512/f/3/i>*

inrole(p1, covered-entity) ∧ **inrole**(p2, law-enforcement-official) ∧ (**inrole**(q, victim-of-crime) ∨ **inrole**(q, suspected-victim-of-crime))

∧ in(t, PHI) ∧ **purpose**(u, law-enforcement) ∧ isRequiredByLaw(p1, p2, q, t, u) ∧

(∃ m222 : M. ((⟐(**send**(p2, p1, m222))) ∧ isRequestFor(m222, p1, p2, q, t, u)))

∧

(∃ m221 : M. ((⟐(**send**(q, p1, m221))) ∧ isAgreementTo(m221, p1, p2, q, t, u)))

)

∨(

% *<164/512/f/3/ii>*

inrole(p1, covered-entity) ∧ **inrole**(p2, law-enforcement-official) ∧

((**inrole**(q, victim-of-crime) ∨ **inrole**(q, suspected-victim-of-crime)) ∧ **inrole**(q, emergency-circumstance))

∧ in(t, PHI) ∧ **purpose**(u, law-enforcement) ∧

isRequiredByLaw(p1, p2, q, t, u) ∧

(∃ m224 : M. ((⟐(**send**(p2, p1, m224))) ∧ (isRequestFor(m224, p1, p2, q, t, u)))) ∧

isNeededToDetermineCrime(p1, p2, q, t, u) ∧ notUsedAgainstVictim(p1, p2, q, t, u) ∧

isActivityAdverselyAffectedByWait(p1, p2, q, t, u) ∧ believesInBestInterest(p1, p2, q, t, u)

)

∨(

% *<164/512/f/4>*

inrole(p1, covered-entity) ∧ **inrole**(p2, law-enforcement-official) ∧ **inrole**(q, deceased)

∧ in(t, PHI) ∧ **purpose**(u, suspicious-death-notification) ∧ isRequiredByLaw(p1, p2, q, t, u) ∧ deathMayBeResultOfACrime(p1, q)

)

∨(

% *<164/512/f/5>*

inrole(p1, covered-entity) ∧ **inrole**(p2, law-enforcement-official) ∧ in(t, PHI) ∧ **purpose**(u, report-possible-crime-on-premise)

∧ isRequiredByLaw(p1, p2, q, t, u) ∧ believesEvidenceOfCrimeOnPremises(p1, p2, q, t, u)

)

```
∨(
% <164/512/f/6>
(inrole(p1, provider) ∧ inrole(p1, covered-entity))∧
inrole(p2, law-enforcement-official) ∧ in(t, PHI) ∧ purpose(u,
    alert-of-crime-commission-location-victims-perpetrator)
∧ isRequiredByLaw(p1, p2, q, t, u) ∧
providingEmergencyHealthCare(p1, q) ∧
appearsNecessaryToAlertOfCrimeCommissionLocationOfVictimsPerpetrator(p1, p2, q, t, u)
∧ (¬( believesEmergencyResultOfAbuseNeglectOrDomesticViolence(p1, q)))
)
∨
(
% <164/512/d/3> %%%%% Different from Henry's, Limin's looks more reasonable
inrole(p1, covered-entity) ∧ inrole(p2, health-oversight-agency) ∧ in(t, PHI)
∧
authorizedByLawForPurpose(p2, u) ∧
(purpose(u, over-sight-government-benefit-programs-health-eligibility) ∧
purpose(u, over-sight-government-regulated-entity-health-compliance)∧
purpose(u, over-sight-subject-to-civil-rights-health-compliance)
) ∧ ((¬(isSubjectOfInvestigation(q))) ∧ ( (relatedToReceiptHealthCare(u)) ∨
    (relatedToPublicHealthBenefits(u))
∨ (relatedToPublicBenefitsQualificationDependsOnHealth(u)) ) )
)
%_____________________
)

%%%%%%%%%%%%%%%%%%%%%%% END OF NEGATIVE NORM %%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%% START OF NEGATIVE NORM %%%%%%%%%%%%%%%%%%%%%%%%%
% <164/508/a/3/i>
inrole(p1, covered-entity) ∧ inrole(q, individual) ∧ in(t, PHI) ∧ purpose(u, marketing)
⟶
(
( ∃ m128 : M. ( (◇(send(q, p1, m128))) ∧ satisfiesAllValidAuthorizationRequirement(m128, p1, p2,
    q, t, u)
∧ (¬(violatesValidAuthorizationRequirement(m128, p1, p2, q, t, u)))
))
∨
(inrole(p1, covered-entity) ∧ samePerson(p2, q) ∧ faceToFace(p1, p2, q, t, u))
∨
```

```
(inrole(p1, covered-entity) ∧ promotionalGiftOfNominalValue(p1, p2, q, t, u)
)
)

%%%%%%%%%%%%%%%%%%%%%%% END OF NEGATIVE NORM %%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%% START OF NEGATIVE NORM %%%%%%%%%%%%%%%%%%%%%%%%%
% <164/510/b/2> %%%%%%%%%%% Negative Norms
inrole(p1, covered-entity) ∧
(inrole(p2, family-member) ∨ inrole(p2, relative) ∨ inrole(p2, close-friend) ∨ inrole(p2,
    identified-164510b)
∨ inrole(p2, personal-rep) ∨ inrole(p2, responsible-for-care)) ∧
inrole(q, individual) ∧ in(t, PHI) ∧
( familyMemberOf(p2, q) ∨ relativeOf(p2, q) ∨ closeFriendOf(p2, q) ∨
identifiedPerson(p2, q) ∨ personalRepOf(p2, q) ∨ familyMemberOf(p2, q) ∨
    responsibleForCareOf(p2, q))
∧ (⟐(available(p1, q) ∧ hasCapabilityToMakeHealthCareDecisions(q)))
⟶
(
(
∃ m115 : M. (⟐(send(q ,p1, m115) ∧ isAgreement164510b2(m115, p1, p2, q, t, u)))
)
∨
(
(¬(∃ m116 : M. (send(q, p1, m116) ∧ isObjection164510b2(m116, p1, p2, q, t, u))))
S
(∃ m117 : M. (send(p1, q, m117) ∧ isOpportunityToObject(m117, p1, p2, q, t, u)))
)
 ∨
 (professionalJudgementIndividualDoesNotObject(p1, p2, q, t, u))
)
%%%%%%%%%%%%%%%%%%%%%%% END OF NEGATIVE NORM %%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%% START OF NEGATIVE NORM %%%%%%%%%%%%%%%%%%%%%%%%%
% <164/510/b/3> %%%%%%%%%%% NEgative Norms
inrole(p1, covered-entity) ∧
(inrole(p2, family-member) ∨ inrole(p2, relative) ∨ inrole(p2, close-friend) ∨ inrole(p2,
    identified-164510b)
```

∨ **inrole**(p2, personal-rep) ∨ **inrole**(p2, responsible-for-care)) ∧
inrole(q, individual) ∧ in(t, PHI) ∧
(familyMemberOf(p2, q) ∨ relativeOf(p2, q) ∨ closeFriendOf(p2, q) ∨
identifiedPerson(p2, q) ∨ personalRepOf(p2, q) ∨ familyMemberOf(p2, q) ∨
responsibleForCareOf(p2, q))
∧ (¬(⟐(available(p1, q) ∧ hasCapabilityToMakeHealthCareDecisions(q))))
⟶
(
professionalJudgementIsInBestInterestof164510b3(p1, p2, q, t, u)
∧
relevantToInvolvement(p1, p2, q, t, u)
)

%%%%%%%%%%%%%%%%%%%%%%%% *END OF NEGATIVE NORM* %%%%%%%%%%%%%%%%%%%%%%%%%%

%%

%%%%%%%%%%%%% *EXTRA THREE NORMS* %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%% *START OF NEGATIVE NORM* %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%% *<160/310>*
inrole(p1, secretary) ∧ **inrole**(p2, covered-entity) ∧ **inrole**(q, individual) ∧ in(t, dii) ∧ **purpose**(u,
investigation)
⟶
◇
(
(
∃ t1 : T. (∃ m999 : M. (**send**(p2, p1, m999) ∧ **inrole**(p2, covered-entity) ∧ **inrole**(p1, secretary) ∧
inrole(q, individual) ∧ contains(m999, q, t1) ∧ in(t1, PHI)))
)
)

%%%%%%%%%%%%%%%%%%%%%%%% *END OF NEGATIVE NORM* %%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%% *START OF NEGATIVE NORM* %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%% *<164.524> -------> VERY SIMPLIFIED* %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

inrole(p1, individual) ∧ **inrole**(p2, covered-entity) ∧ **inrole**(q, individual) ∧ in(t, dii) ∧ **purpose**(u,
access) ∧ samePerson(p1, q)
⟶
(

◇

(

∃ t2 : T. (∃ m1999 : M. (**send**(p2, p1, m999) ∧ **inrole**(p2, covered-entity) ∧ **inrole**(p1, individual) ∧ **inrole**(q, individual) ∧ contains(m1999, q, t2) ∧ in(t2, PHI) ∧ samePerson(p1, q)))

)

)

%%%%%%%%%%%%%%%%%%%%%%%% *END OF NEGATIVE NORM* %%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%% *<MY OWN NORM FOR TEST>* %%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%% *START OF NEGATIVE NORM* %%%%%%%%%%%%%%%%%%%%%%%%%%

(**inrole**(p1, parent) ∨ **inrole**(p1, guardian) ∨ **inrole**(p1,loco-parentis))

∧

inrole(p2, covered-entity) ∧ **inrole**(q, individual) ∧ in(t, dii) ∧ **purpose**(u, access) ∧

(parentOf(p1, q) ∨ guardianOf(p1, q) ∨ locoParentis(p1, q))

⟶

(

◇

(

∃ t3 : T. (∃ m2999 : M. (**send**(p2, p1, m999) ∧ **inrole**(p2, covered-entity) ∧ (**inrole**(p1, parent) ∨ **inrole**(p1, guardian) ∨ **inrole**(p1,loco-parentis)) ∧ **inrole**(q, individual) ∧ contains(m2999, q, t3) ∧ in(t3, PHI) ∧ (parentOf(p1, q) ∨ guardianOf(p1, q) ∨ locoParentis(p1, q))))

)

)

%%%%%%%%%%%%%%%%%%%%%%%% *END OF NEGATIVE NORM* %%%%%%%%%%%%%%%%%%%%%%%%%%

BIBLIOGRAPHY

[1] The Medicare and Medicaid EHR Incentive Program. http://www.cms.gov/Regulations-and-Guidance/Legislation/EHRIncentivePrograms/index.html?redirect=/ehrincentiveprograms/.

[2] Senate banking committee, Gramm-Leach-Bliley Act, 1999. Public Law 106-102.

[3] Enterprise privacy authorization language (EPAL) version 1.2, November 2003. http://www.zurich.ibm.com/pri/projects/epal.html.

[4] Muhammad Ali, Laurent Bussard, and Ulrich Pinsdorf. Obligation Language and Framework to Enable Privacy-Aware SOA. In *Data Privacy Management and Autonomous Spontaneous Security*, volume 5939 of *Lecture Notes in Computer Science*, pages 18–32. Springer Berlin, Heidelberg, 2010.

[5] Bowen Alpern and Fred B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, October 1985.

[6] Bowen Alpern and Fred B. Schneider. Recognizing safety and liveness. *Distributed Computing*, 2:117–126, 1987.

[7] Rajeev Alur and ThomasA. Henzinger. Logics and models of real time: A survey. In *Real-Time: Theory in Practice*, Lecture Notes in Computer Science. 1992.

[8] ANSI. American national standard for information technology – role based access control. ANSI INCITS 359-2004, February 2004.

[9] KrzysztofR. Apt and Elena Marchiori. Reasoning about prolog programs: From modes through types to assertions. *Formal Aspects of Computing*, 6(1):743–765, 1994.

[10] Nathalie Baracaldo and James Joshi. Beyond accountability: using obligations to reduce risk exposure and deter insider attacks. In *Proceedings of the 18th ACM symposium on*

Access control models and technologies, SACMAT '13, pages 213–224, New York, NY, USA, 2013. ACM.

[11] Adam Barth, Anupam Datta, John C. Mitchell, and Helen Nissenbaum. Privacy and contextual integrity: Framework and applications. *IEEE Symposium on Security and Privacy*.

[12] Adam Barth, John Mitchell, Anupam Datta, and Sharada Sundaram. Privacy and utility in business processes. In *IEEE CSF '07*.

[13] David Basin, Felix Klaedtke, and Samuel Müller. Monitoring security policies with metric first-order temporal logic. In *ACM SACMAT '10*.

[14] David Basin, Ernst-Ruediger Olderog, and Paul E. Sevinc. Specifying and analyzing security automata using csp-oz. In *ASIACCS*, pages 70–81, 2007.

[15] David A. Basin, Felix Klaedtke, Srdjan Marinovic, and Eugen Zalinescu. Monitoring compliance policies over incomplete and disagreeing logs. In *RV*, 2012.

[16] Andreas Bauer, Rajeev Gore, and Alwen Tiu. A first-order policy language for history-based transaction monitoring. In M. Leucker and C. Morgan, editors, *Proceedings of the 6th International Colloquium on Theoretical Aspects of Computing (ICTAC)*, volume 5684 of *Lecture Notes in Computer Science*, pages 96–111, Berlin, Heidelberg, August 2009. Springer-Verlag.

[17] Claudio Bettini, Sushil Jajodia, X. Sean Wang, and Duminda Wijesekera. Provisions and obligations in policy management and security applications. In *Proceedings of the 28th international conference on Very Large Data Bases*, VLDB '02, pages 502–513. VLDB Endowment, 2002.

[18] Claudio Bettini, Sushil Jajodia, X. Sean Wang, and Duminda Wijesekera. Provisions and obligations in policy rule management. *J. Netw. Syst. Manage.*, 11(3):351–372, 2003.

[19] Avrim Blum and Merrick Furst. Fast planning through planning graph analysis. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 95)*, pages 1636–1642, 1995.

[20] Travis Breaux and Annie Antón. Analyzing regulatory rules for privacy and security requirements. *IEEE TSE '08*.

[21] Gerhard Brewka, Thomas Eiter, and Mirosław Truszczyński. Answer set programming at a glance. *Commun. ACM*, 54(12):92–103, December 2011.

[22] Julius R. Büchi. On a decision method in restricted second order arithmetic. In *Proceedings of LMPS'60*, 1962.

[23] M. Casassa and F. Beato. On Parametric Obligation Policies: Enabling Privacy-Aware Information Lifecycle Management in Enterprises. In *Policies for Distributed Systems and Networks.*, pages 51 –55, jun. 2007.

[24] Edward Y. Chang, Zohar Manna, and Amir Pnueli. Characterization of temporal property classes. In *ICALP*, pages 474–486, 1992.

[25] Liang Chen, Jason Crampton, Martin J. Kollingbaum, and Timothy J. Norman. Obligations in risk-aware access control. In *Proceedings of the 2012 Tenth Annual International Conference on Privacy, Security and Trust (PST)*, PST '12, pages 145–152, Washington, DC, USA, 2012. IEEE Computer Society.

[26] Omar Chowdhury, Haining Chen, Jianwei Niu, Ninghui Li, and Elisa Bertino. On xacml's adequacy to specify and to enforce hipaa. In *HealthSec'12*.

[27] Omar Chowdhury, Andreas Gampe, Jianwei Niu, Jeffery von Ronne, Jared Bennatt, Anupam Datta, Limin Jia, and William H. Winsborough. Privacy promises that can be kept: a policy analysis method with application to the hipaa privacy rule. In *Proceedings of the*

18th ACM symposium on Access control models and technologies, SACMAT '13, pages 3–14, New York, NY, USA, 2013. ACM.

[28] Omar Chowdhury, Murillo Pontual, William H. Winsborough, Ting Yu, Keith Irwin, and Jianwei Niu. Ensuring Authorization Privileges for Cascading User Obligations. In *ACM SACMAT '12*.

[29] Edmund M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Language and Systems (TOPLAS)*, 8(2):244–263, 1986.

[30] Edmund M. Clarke and E. 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.

[31] D. Damianou, N. Dulay, E. Lupu, and M. Sloman. The Ponder Policy Specification Language. In *IEEE POLICY '01*.

[32] Rocco De Nicola and Frits Vaandrager. Action versus state based logics for transition systems. In *Proceedings of the LITP spring school on theoretical computer science on Semantics of systems of concurrent processes*, pages 407–419, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

[33] Frank Dederichs and Rainer Weber. Safety and liveness from a methodological point of view. *Information Processing Letters*, 1990.

[34] Piotr Dembinski and Jan Maluszynski. And-parallelism with intelligent backtracking for annotated logic programs. In *SLP*, pages 29–38, 1985.

[35] Henry DeYoung, Deepak Garg, Limin Jia, Dilsun Kaynar, and Anupam Datta. Experiences in the logical specification of the hipaa and glba privacy laws. In *ACM WPES '10*.

[36] Henry DeYoung, Deepak Garg, Limin Jia, Dilsun Kirli Kaynar, and Anupam Datta. Technical report CMU-CyLab-10-007: Experiences in the logical specification of the HIPAA and GLBA privacy laws, 2010.

[37] Nikhil Dinesh, Aravind Joshi, Insup Lee, and Oleg Sokolsky. Checking traces for regulatory conformance. In *RV '08*.

[38] Nikhil Dinesh, Aravind Joshi, Insup Lee, and Oleg Sokolsky. Reasoning about conditions and exceptions to laws in regulatory conformance checking. In *Proceedings of the 9th international conference on Deontic Logic in Computer Science*, DEON '08, pages 110–124, Berlin, Heidelberg, 2008. Springer-Verlag.

[39] Daniel J. Dougherty, Kathi Fisler, and Shriram Krishnamurthi. Obligations and their interaction with programs. In *Proceedings of ESORICS 2007*, pages 375–389.

[40] Daniel J. Dougherty, Kathi Fisler, and Shriram Krishnamurthi. Obligations and their interaction with programs. In *12th European Symposium On Research In Computer Security, Dresden, Germany, September 24-26, Proceedings*, pages 375–389, 2007.

[41] Yehia Elrakaiby, Frédéric Cuppens, and Nora Cuppens-Boulahia. Formal enforcement and management of obligation policies. *Data Knowl. Eng.*, 71:127–147, January 2012.

[42] E. Allen Emerson. Temporal and modal logic. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 16, pages 995–1072. Elsevier Science, 1990.

[43] E. Allen Emerson. Automated temporal reasoning about reactive systems. In *Proceedings of the VIII Banff Higher order workshop conference on Logics for concurrency : structure versus automata: structure versus automata*, pages 41–101, Secaucus, NJ, USA, 1996. Springer-Verlag New York, Inc.

[44] E. Allen Emerson and Joseph Y. Halpern. Sometimes and not never revisited: on branching versus linear time temporal logic. *J. ACM*, 33(1):151–178, January 1986.

[45] E. Allen Emerson and Joseph Y. Halpern. "Sometimes" and "not never" revisited: on branching versus linear time temporal logic. *J. ACM*, 1986.

[46] E. Allen Emerson and Kedar S. Namjoshi. Reasoning about rings. In *POPL '95*, 1995.

[47] E. Engeler. *Introduction to the theory of computation*. Computer science and applied mathematics. Academic Press, 1973.

[48] Úlfar Erlingsson and Fred B. Schneider. SASI enforcement of security policies: a retrospective. In *Proceedings of the workshop on New security paradigms*, pages 87–95, 2000.

[49] Inc. Facebook. Facebook's Privacy Policy. Available at https://www.facebook.com/about/privacy.

[50] David F. Ferraiolo, Ravi S. Sandhu, Serban Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. Proposed NIST standard for role-based access control. *ACM TISSEC '01*.

[51] Richard E. Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3 - 4):189 – 208, 1971.

[52] Dov Gabbay. The imperative future. chapter : The declarative past and imperative future, pages 35–67. John Wiley & Sons, Inc., New York, NY, USA, 1996.

[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] Pedro Gama and Paulo Ferreira. Obligation policies: An enforcement platform. In *6th IEEE International Workshop on Policies for Distributed Systems and Networks*, Stockholm, Sweden, June 2005. IEEE Computer Society.

[55] Deepak Garg, Limin Jia, and Anupam Datta. Policy auditing over incomplete logs: theory, implementation and applications. In *ACM CCS '11*.

[56] Deepak Garg, Limin Jia, and Anupam Datta. A logical method for policy enforcement over evolving audit logs. *CoRR*, abs/1102.2521, 2011.

[57] Inc. Google. Google's Privacy Policy. Available at http://www.google.com/policies/privacy/.

[58] D. Harel and A. Pnueli. On the development of reactive systems. In *Logics and Models of Concurrent Systems*, pages 477–498, 1985.

[59] Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. Protection in operating systems. *CACM '76*.

[60] Klaus Havelund and Grigore Roşu. Efficient monitoring of safety properties. *Int. J. Softw. Tools Technol. Transf., '04*.

[61] Klaus Havelund and Grigore Rosu. Synthesizing monitors for safety properties. In *TACAS' 02*.

[62] Health Resources and Services Administration. Health insurance portability and accountability act, 1996. Public Law 104-191.

[63] Manuel Hilty, David A. Basin, and Alexander Pretschner. On obligations. In *ESORICS*, pages 98–117, 2005.

[64] Ian M. Hodkinson, Frank Wolter, and Michael Zakharyaschev. Decidable fragment of first-order temporal logics. *Ann. Pure Appl. Logic '00*.

[65] Gerard J. Holzmann. The model checker SPIN. *TSE '97*.

[66] Marieke Huisman and Alejandro Tamalet. A formal connection between security automata and jml annotations. In *FASE*, pages 340–354, 2009.

[67] Keith Irwin, Ting Yu, and William H. Winsborough. On the modeling and analysis of obligations. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 134–143, New York, NY, USA, 2006. ACM.

[68] Keith Irwin, Ting Yu, and William H. Winsborough. Assigning responsibilities for failed obligations. In *IFIPTM Joined iTrust and PST Conference on Privacy, Trust Management and Security*, pages 327–342. Springer Boston, 2008.

[69] A. J. I. Jones. On the relationship between permission and obligation. In *ICAIL '87*, New York, NY, USA. ACM.

[70] Basel Katt, Xinwen Zhang, Ruth Breu, Michael Hafner, and Jean-Pierre Seifert. A general obligation model and continuity: enhanced policy enforcement engine for usage control. In *Proceedings of the 13th ACM symposium on Access control models and technologies*, pages 123–132, New York, NY, USA, 2008. ACM.

[71] Henry A. Kautz and Bart Selman. Planning as satisfiability. In *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI'92)*, pages 359–363, 1992.

[72] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Syst.*, 1990.

[73] Karl Krukow, Mogens Nielsen, and Vladimiro Sassone. A logical framework for history-based access control and reputation systems. *J. Comput. Secur.*, 16(1):63–101, 2008.

[74] O. Kupferman and M.Y. Vardi. Vacuity detection in temporal model checking. In *10th Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, volume 1703 of *Lecture Notes in Computer Science*, pages 82–96. Springer-Verlag, 1999.

[75] O. Kupferman and M.Y. Vardi. Memoryful branching-time logics. In *Proc. 21st IEEE Symp. on Logic in Computer Science*, 2006.

[76] Orna Kupferman and Amir Pnueli. Once and for all. In *Proceedings of the 10th Annual IEEE Symposium on Logic in Computer Science*, pages 25–, Washington, DC, USA, 1995. IEEE Computer Society.

[77] Orna Kupferman, Amir Pnueli, and Moshe Y. Vardi. Once and for all. *J. Comput. Syst. Sci. '12*.

[78] Peifung E. Lam, John C. Mitchell, Andre Scedrov, Sharada Sundaram, and Frank Wang. Declarative privacy policy: finite models and attribute-based encryption. In *ACM IHI '12*.

[79] Peifung E. Lam, John C. Mitchell, and Sharada Sundaram. A Formalization of HIPAA for a Medical Messaging System. In *TrustBus '09*.

[80] L. Lamport. Proving the correctness of multiprocess programs. *IEEE TSE'77*.

[81] Leslie Lamport. "sometime" is sometimes "not never": on the temporal logic of programs. In *Proceedings of the 7th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '80, pages 174–185, New York, NY, USA, 1980. ACM.

[82] Leslie Lamport. Proving possibility properties. *Theoretical Computer Science*, 206(1-2):341 – 352, 1998.

[83] François Laroussinie, Nicolas Markey, and Ph. Schnoebelen. Temporal logic with forgettable past. In *IEEE LICS '02*.

[84] François Laroussinie, Nicolas Markey, and Ph. Schnoebelen. Temporal logic with forgettable past. In *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science*, LICS '02, pages 383–392, Washington, DC, USA, 2002. IEEE Computer Society.

[85] Martin Leucker and Christian Schallhart. A brief account of runtime verification. *Journal of Logic and Algebraic Programming*, 78(5):293–303, 2009.

[86] Ninghui Li, Haining Chen, and Elisa Bertino. On practical specification and enforcement of obligations. In *Proceedings of the second ACM conference on Data and application security and privacy*, 2012.

[87] Ninghui Li, John C. Mitchell, and William H. Winsborough. Design of a role-based trust-management framework. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2002.

[88] Jay Ligatti, Lujo Bauer, and David Walker. Run-time enforcement of nonsafety policies. *ACM Trans. Inf. Syst. Secur.*, 12:19:1–19:41, January 2009.

[89] Zohar Manna and Amir Pnueli. The anchored version of the temporal framework. In *REX Workshop '88*.

[90] Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer, 1992.

[91] Zohar Manna and Amir Pnueli. *Temporal verification of reactive systems: safety*. Springer-Verlag New York, Inc., New York, NY, USA, 1995.

[92] Fabio Martinell and Ilaria Matteucci. Through modeling to synthesis of security automata. *Electron. Notes Theor. Comput. Sci.*, 179:31–46, 2007.

[93] Jeremy C. Maxwell and Annie I. Antón. The production rule framework: developing a canonical set of software requirements for compliance with law. In *Proceedings of the 1st ACM International Health Informatics Symposium*, IHI '10, pages 629–636, New York, NY, USA, 2010. ACM.

[94] Michael J. May, Carl A. Gunter, and Insup Lee. Privacy APIs: Access control techniques to analyze and verify legal privacy policies. In *IEEE CSFW '06*.

[95] L.T. McCarty. Pemissions and obligations. In *Proceedings IJCAI-83*, 1983.

[96] C.S. Mellish and University of Edinburgh. Dept. of Artificial Intelligence. *The Automatic Generation of Mode Declarations for Prolog Programs*. Research paper / Department of Artifical Intelligence, University of Edinburgh. Department of Artificial Intelligence, University of Edinburgh, 1981.

[97] Naftaly H. Minsky and Abe D. Lockman. Ensuring integrity by adding obligations to privileges. In *Proceedings of the 8th international conference on Software engineering*, pages 92–102, Los Alamitos, CA, USA, 1985. IEEE Computer Society Press.

[98] Fabio Mogavero, Aniello Murano, and Moshe Y. Vardi. Relentful strategic reasoning in alternating-time temporal logic. In *international conference on Logic for programming, artificial intelligence, and reasoning*, pages 371–386, 2010.

[99] XuanLong Nguyen and Subbarao Kambhampati. Reviving partial order planning. In *IJCAI*, pages 459–466, 2001.

[100] Qun Ni, Elisa Bertino, and Jorge Lobo. An obligation model bridging access control policies and privacy policies. In *SACMAT' 08*, New York, NY, USA. ACM.

[101] Qun Ni, Alberto Trombetta, Elisa Bertino, and Jorge Lobo. Privacy -aware role based access control. In *ACM SACMAT '07*.

[102] Food and Drug Administration - Department of Health and Human Services. FDA Regulations Relating to Good Clinical Practice and Clinical Trials. http://www.fda.gov/ScienceResearch/SpecialTopics/RunningClinicalTrials/ucm155713.htm.

[103] Food and Drug Administration - Department of Health and Human Services. Privacy Protection - 21 CFR part 21.71 for disclosure of records in privacy act record systems; accounting required. http://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfCFR/CFRSearch.cfm?fr=21.71.

[104] Food and Drug Administration - Department of Health and Human Services. Public Information - 21 CFR part 20.63 for personnel, medical, and similar files, disclosure of which constitutes a clearly unwarranted invasion of personal privacy. http://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfcfr/CFRSearch.cfm?fr=20.63.

[105] Jaehong Park and Ravi Sandhu. The uconabc usage control model. *ACM Trans. Inf. Syst. Secur.*, 7(1):128–174, 2004.

[106] J. Scott Penberthy. Ucpop: A sound, complete, partial order planner for adl. pages 103–114. Morgan Kaufmann, 1992.

[107] Guillaume Piolle and Yves Demazeau. Obligations with deadlines and maintained interdictions in privacy regulation frameworks. In *IAT '08: Proceedings of the 2008 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 162–168, Sidney, Australia, December 2008. IEEE Computer Society.

[108] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science*, volume 526, pages 46–67, 1977.

[109] Murillo Pontual, Omar Chowdhury, William Winsborough, Ting Yu, and Keith Irwin. Toward Practical Authorization Dependent User Obligation Systems. In *ASIACCS' 10*, pages 180–191. ACM Press, 2010.

[110] Murillo Pontual, Omar Chowdhury, William H. Winsborough, Ting Yu, and Keith Irwin. On the management of user obligations. SACMAT '11, New York, NY, USA. ACM.

[111] Murillo Pontual, Keith Irwin, Omar Chowdhury, William H. Winsborough, and Ting Yu. Failure feedback for user obligation systems. In *The Second IEEE International Conference on Information Privacy, Security, Risk and Trust*, 2010.

[112] Paul Roberts. HIPAA Bares Its Teeth: $4.3m Fine For Privacy Violation. Available at https://threatpost.com/en_us/blogs/hipaa-bares-its-teeth-43m-fine-privacy-violation-022311.

[113] Kenneth A. Ross. Modular stratification and magic sets for datalog programs with negation. *J. ACM*, 41(6):1216–1266, November 1994.

[114] A. Sasturkar, Ping Yang, S.D. Stoller, and C.R. Ramakrishnan. Policy analysis for administrative role based access control. In *Computer Security Foundations Workshop, 2006. 19th IEEE*, 2006.

[115] Fred B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security*, 3:2000, 2000.

[116] Securities and Exchange Commission. Sarbanes-oxley act, 2002. Public Law 107-204.

[117] Chetan Shankar and Roy Campbell. Managing pervasive systems using role-based obligation policies. In *Proceedings of the 4th annual IEEE international conference on Pervasive Computing and Communications Workshops*, PERCOMW '06, pages 373–, Washington, DC, USA, 2006. IEEE Computer Society.

[118] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *J. ACM '85*.

[119] Scott D. Stoller, Ping Yang, C R. Ramakrishnan, and Mikhail I. Gofman. Efficient policy analysis for administrative role based access control. In *CCS '07*, New York, NY, USA, 2007. ACM.

[120] Yih-Kuen Tsay, Ming-Hsien Tsai, Jinn-Shu Chang, and Yi-Wen Chang. Büchi store : An open repository of büchi automata. In *TACAS '11*.

[121] Michael Tschantz, Anupam Datta, and Jeanette Wing. Formalizing and enforcing purpose restrictions in privacy policies. In *Proceedings of 33rd IEEE Symposium on Security and Privacy*. IEEE, 2012.

[122] A. Uszok, J. Bradshaw, R. Jeffers, N. Suri, P. Hayes, M. Breedy, L. Bunch, M. Johnson, S. Kulkarni, and J. Lott. Kaos policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement. In *POLICY '03*, Washington, DC, USA, 2003. IEEE Computer Society.

[123] Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the 1st Annual Symposium on Logic in Computer Science (LICS'86)*, pages 332–344. IEEE Comp. Soc. Press, June 1986.

[124] David Walker. A type system for expressive security policies. In *POPL*, pages 254–267, 2000.

[125] Mark Weiser. Program slicing. In *ICSE '81*.

[126] Ruoyu Wu, Gail-Joon Ahn, and Hongxin Hu. Towards hipaa-compliant healthcare systems. In *Proceedings of the 2nd ACM SIGHIT International Health Informatics Symposium*, IHI '12, pages 593–602, New York, NY, USA, 2012. ACM.

[127] XACML TC. Oasis extensible access control markup language (xacml).

VITA

Omar Haider Chowdhury received his B.Sc. in Computer Science and Engineering at the Bangladesh University of Engineering and Technology in Dhaka, Bangladesh in 2007. He started pursuing his Ph.D. in the department of Computer Science at the University of Texas at San Antonio from Fall 2007. He worked under the supervision of Dr. William H. Winsborough from Fall 2008 until Dr. Winsborough passed away in Fall 2011. After that, he has been advised by Dr. Jianwei Niu. His research interest broadly lies in the field of computer security, formal verification techniques, and computer privacy. He is currently working on developing rigorous formal verification techniques to check whether organizations are compliant with applicable privacy regulations. He has worked as a senior software developer in Structured Data Systems Ltd., in Dhaka, Bangladesh. He has also worked in Cylab, Carnegie Mellon University as a visiting researcher. His professional objective is to get a faculty position in the field of computer security and privacy.