Privacy Promises That Can Be Kept: A Policy Analysis Method with Application to the HIPAA Privacy Rule

Omar Chowdhury‡, Andreas Gampe‡, Jianwei Niu‡, Jeffery von Ronne‡, Jared Bennatt‡, Anupam Datta§, Limin Jia§

CS Department, The University of Texas at San Antonio‡ CyLab, ECE Department, Carnegie Mellon University§
{ochowdhu, agampe, niu, vonronne, jbennatt}@cs.utsa.edu ‡ {danupam, liminjia}@cmu.edu §

ABSTRACT

Organizations collect personal information from individuals to carry out their business functions. Federal privacy regulations, such as the Health Insurance Portability and Accountability Act (*HIPAA*), mandate how this collected information can be shared by the organizations. It is thus incumbent upon the organizations to have means to check compliance with the applicable regulations. Prior work by Barth *et al.* introduces two notions of *compliance*, weak compliance (WC) and strong compliance (SC). WC ensures that present requirements of the policy can be met whereas SC also ensures obligations can be met. An action is compliant with a privacy policy if it is both weakly and strongly compliant. However, their definitions of compliance are restricted to only propositional linear temporal logic ($pLTL$), which cannot feasibly specify HIPAA. To this end, we present a policy specification language based on a restricted subset of first order temporal logic ($FOTL$) which can capture the privacy requirements of HIPAA. We then formally specify WC and SC for policies of our form. We prove that checking WC is feasible whereas checking SC is undecidable. We then formally specify the property WC entails SC, denoted by Δ, which requires that each weakly compliant action is also strongly compliant. To check whether an action is compliant with such a policy, it is sufficient to only check whether the action is weakly compliant with that policy. We also prove that when a policy $\wp$ has the Δ-property, the present requirements of the policy reduce to the safety requirements imposed by $\wp$. We then develop a sound, semi-automated technique for checking whether practical policies have the Δ-property. We finally use HIPAA as a case study to demonstrate the efficacy of our policy analysis technique.

Categories and Subject Descriptors

K.4.1 [**Computers and Society**]: Public Policy Issues—*Privacy, Regulation*; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*Temporal logic*

Keywords

Privacy Policy, HIPAA, Policy Analysis, Obligations

1. INTRODUCTION

Our society is becoming increasingly dependent on computer information systems for the proper management of personal data. Medical records, financial data, and personal information collected from users are just a few examples. Organizations are required to store and share such information in a manner that conforms to specific privacy policies, which are mandated by custom, sound business practice, contract, and, often, by law. Examples of privacy policies that carry the force of law include the Health Insurance Portability and Accountability Act (HIPAA) [33], the Gramm-Leach-Bliley Act (GLBA) [3], Sarbanes-Oxley Act (SOX) [51]. Violations of these federal regulations can bring down heavy financial penalties on the organizations. For instance, Cignet Health Center was fined $1.3 million for violating §164.524 of HIPAA [10] which *obligates* the covered entity (hospital) to give patients access to their medical records when the patients request for it. It is thus important for the organizations to check compliance with applicable regulations.

Several frameworks have been proposed for specifying and analyzing privacy policies [4, 11, 13, 12, 45, 41, 21, 28, 19, 15, 22, 38]. Specifically, 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 [20]. Consider the privacy rule from §164.502(e)(1)(i) of HIPAA which states 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, 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 discussed above. Requiring the covered entity to give access to the PHI to the patient is an example of an obligatory requirement. We prove that checking WC is feasible whereas checking SC is undecidable. Current work in this area [13, 28, 41, 4, 19, 22], while checking compliance, only considers WC without taking SC into account.

In the current work, we aim to verify the Δ-property of privacy policies. If a policy $\wp$ can be shown to have the Δ-property, to check whether an action is in compliance with

$\wp$, it suffices to only check whether that action is weakly compliant with $\wp$. For this, we introduce a specification language expressive enough to encode the HIPAA Privacy Policy. We formally specify WC and SC for our language. Barth *et al.*'s definitions of compliance (WC and SC) are not sufficient as they are restricted to only propositional linear temporal logic (*pLTL*) which cannot be feasibly used for specifying privacy regulations like HIPAA. We then present a sound, semi-automated technique to verify whether a privacy policy written in our specification language satisfies the Δ-property. To show the efficacy of our policy analysis technique, we formally verify that our encoding of HIPAA satisfies the Δ-property. This implies that whether an action is in compliance with the HIPAA Privacy Policy can be checked efficiently by checking WC only. Note that although the Δ-property has been introduced before [11], we are the first to present feasible techniques to decide whether a policy has the Δ-property and apply it to a practical privacy policy like HIPAA.

We now detail our technical contributions. We present a privacy policy specification language based on a restricted subset of first order temporal logic (FOTL). We demonstrate the expressive power of our language by encoding all 84 disclosure-related clauses of HIPAA [1]. We then formally specify what it means for an action to be weakly compliant and strongly compliant with FOTL policies of our form. We also prove that WC can be checked in PSPACE in the policy size provided the policy satisfies a constraint (*mode restriction* [28, 29]) whereas checking SC is undecidable. While the complexity of checking WC is in PSPACE, research [28, 13] has shown that it can be checked efficiently in practice.

To mitigate the undecidability of SC, we formally specify the property WC entails SC (denoted by Δ) [11] of a privacy policy. Prior work presents a semantic definition and a decision procedure for checking the Δ-property restricted to only pLTL policies. A policy has the Δ-property if every weakly compliant action is also strongly compliant. The Δ-property can be checked once statically before the policy is deployed. Given a FOTL policy $\wp$, we syntactically generate a first order CTL* with linear past (denoted by FO-CTL$^*_{lp}$) [39] formula $\delta(\wp)$ from $\wp$. We prove that $\delta(\wp)$ is satisfied 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. However, model checking a FO-CTL$^*_{lp}$ formula with respect to $\mathbb{M}_\wp$ is undecidable.

While checking the Δ-property for a FOTL policy 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. We prove that there are exactly two cases in which the Δ-property can be violated (Theorem 8). In the first case, taking an 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 [20]. We prove that for policies written in our specification language, the former violation case cannot happen. Thus, it is sufficient to consider the second violation case only (Corollary 11). 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 *PHI* of an individual. One can thus satisfy the HIPAA Privacy Policy by only sending messages not containing any *PHI*. Thus, from Corollary 11, it follows that $\wp_H$ 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. The size of the sliced policies in both cases is only 5% of $\wp_H$, which is a significant reduction of the policy size to be considered. We then develop a small model theorem [25] for $\wp_H$ 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 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.

Organization. Section 2 briefly overviews FOTL. We introduce our privacy policy specification language in section 3. In section 4, we formalize what it means for an action to be compliant with a privacy policy. Our main technical contribution is in section 5 where we present a sound, semi-automated technique by which we can verify the Δ-property of a privacy policy. In section 6, we demonstrate how to use our techniques for HIPAA. Related work is discussed in section 7. Section 8 discusses open problems, future work, and concludes the paper.

2. BACKGROUND ON FOTL

Linear temporal logic (LTL) [48] characterizes the behavior of reactive systems in terms of *traces* (σ), infinite sequences of states and/or events. LTL abstracts the explicit notion of time and only reasons about a relative temporal ordering of events. Our privacy policy language is a many-sorted, first-order linear temporal logic (FOTL) [24]. We briefly summarize FOTL here.

FOTL generalizes pLTL in the same way that first-order logic generalizes propositional logic. Along with boolean connectives (*e.g.*, $\wedge, \vee$, *etc.*), predicates, function symbols, and quantifiers, FOTL formulas can additionally contain unary and binary temporal operators where the operand(s) are FOTL sub-formula(s). Temporal operators can be classified as future and past. *Future Operators.* Henceforth: $\Box\phi$ says that ϕ holds in all future states. Eventually: $\Diamond\phi$ says that ϕ holds in some future state. Tomorrow: $\bigcirc\phi$ holds when the formula ϕ is true in the next step. Until: $\phi_1 \,\mathbb{U}\, \phi_2$ is true in the current state if ϕ_2 holds true in some future state (including the current one) and the formula ϕ_1 holds in all the states from the current state to the state before ϕ_2 holds. *Past Operators.* Historically: $\boxminus\phi$ says that ϕ held in all previous states. Once: $\diamondminus\phi$ says that ϕ held in some previous state. Yesterday: $\ominus\phi$ holds true when the formula ϕ held true in the 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

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

Figure 1: Forms of our privacy policy ($\wp$)

held in every state. Note that the $\mathcal{S}$ operator can be used to represent the $\boxminus$ and $\diamondminus$ operator.

A *logical environment* η maps each variable to a value in the carrier set according to the variable's sort. A formula ϕ is satisfied by a trace σ at an index i under η, denoted by $\sigma, i, \eta \models \phi$, can be defined inductively on the structure of ϕ. One says that σ satisfies ϕ, written $\sigma \models \phi$, if and only if for any η, we have $\sigma, 0, \eta \models \phi$. We use $\sigma_1 \cdot \sigma_2$ to denote the concatenation of two traces in which σ_1 is a finite trace whereas σ_2 is an infinite trace.

3. POLICY SPECIFICATION LANGUAGE

In this section, we introduce our privacy policy specification language. Our policy specification language is a restricted subset of first-order linear temporal logic (FOTL). It is inspired by the specification language, Contextual Integrity (CI), proposed by Barth *et al.* [11]. The specific differences between our language and CI are discussed in Section 7. We demonstrate the adequacy of our specification language by expressing all disclosure-related clauses of the HIPAA Privacy Rule [33] in it [1]. Note that we cannot express obligation deadlines in our language. Although enhancing our specification language to express obligation deadlines [37, 9] is plausible, we do not take obligation deadlines into account in our policy analysis. This is further discussed in section 8.

Top-level Policy. The form of our privacy policies is shown in Figure 1. We use $\wp$ to denote such policies. The sorts are 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. 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, $\text{inrole}(p, \textit{psychiatrist})$ 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 $\text{purpose}(u, \hat{u})$ 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 1 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.

Syntax of Norms. The form of the policy norms are shown in Figure 2. The formula meta-variables in the norms (*i.e.*, ψ, β, and χ) correspond to syntactic categories introduced below in Figure 3. Exception formulas $\psi_{exception}$ have the same form as ψ. In the norms (see Figure 2), 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 $\text{inrole}(p, \hat{r})$. 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 $\text{in}(t, \hat{t})$ and $\text{purpose}(u, \hat{u})$, respectively. 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 2: Norms of transmission

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

Figure 3: Meta-variables of the privacy policy.

We have already discussed some pre-defined predicates of our language (*e.g.*, inrole, *etc.*). We allow additional predicates denoted by $R(\vec{x})$ (see Figure 3) 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.

Restrictions. We now discuss the different constraints we impose in our specification language 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 section 4. More precisely, we do not allow formulas expressing *general liveness properties* ($\Box\Diamond q$). Instead we allow formulas expressing *response properties* [43]. Response properties have the general form $\square(p \rightarrow \Diamond q)$, in which p is a pure-past formula and q is a non-temporal formula. The formula $\square(p \rightarrow \Diamond q)$ 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 section 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.

Example norms from HIPAA. 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.

$$\begin{aligned}&\begin{pmatrix}\text{inrole}(p_1, \textit{covered-entity}) \wedge \text{inrole}(p_2, \textit{business-associate})\\ \wedge\text{inrole}(q, \textit{individual})\end{pmatrix} \wedge\\ &\text{in}(t, \textit{PHI}) \wedge\\ &\text{purpose}(u, \textit{creating-deidentified-info}) \wedge\\ &\text{businessAssociateOf}(p2, p1)\end{aligned}$$

A negative norm (shown below) can be found in §164.508(a)(2) of HIPAA. It specifies that a covered entity must obtain an authorization before disclosing an individual's psychotherapy notes.

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

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*. Recall the clause §164.502(e)(1)(i) of HIPAA discussed in Section 1. 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 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 for compliance investigation is an obligatory requirement.

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, checking whether 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 $\textit{weak}(\wp)$.

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

DEFINITION 1 (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, we have $\sigma \cdot s, |\sigma|, \eta \models \boxminus \textit{weak}(\wp)$ where state $s \models a$.*

Although we have formally defined $\models$ only in terms of infinite traces (see Section 2), the usage of $\models$ for finite σ here is well defined because $\textit{weak}(\wp)$ is a pure-past formula: $\sigma \cdot s, |\sigma|, \eta \models \boxminus \textit{weak}(\wp)$ 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 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 $\textit{weak}(\wp)$ in every point in that trace. Note that $\textit{weak}(\wp)$ is a pure-past FOTL formula with quantifiers. Garg *et al.* [28, 29] present *mode restriction*, which ensures that quantifiers can be expressed as finite conjunctions or disjunctions. This enables an algorithm with PSPACE complexity that can check whether a finite trace satisfies a first-order logic policy. The authors [28] also show that mode restriction is still practical as the HIPAA privacy rules satisfy it. Note that their language is a proper superset of our language fragment used to express $\textit{weak}(\wp)$. We can thus translate $\textit{weak}(\wp)$ 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.

THEOREM 2. *Given a policy $\wp$, WC can be checked in PSPACE in the size of $\wp$ if $\textit{weak}(\wp)$ satisfies the mode restriction.*

For brevity, we do not present proofs for our theorems. Proofs of all theorems appear in the technical report [2].

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 present requirements of the policy [20]. Strong compliance (SC) [11] ensures that this is not the case.

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 3 (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 SC. Checking SC requires deciding whether the incurred future requirements (non-monadic FOTL formula) of an action are satisfiable. However, checking the satisfiability of non-monadic FOTL formulas is undecidable [34]. As a result, we have the following theorem.

Theorem 4. *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.*

5. PRIVACY POLICY ANALYSIS

We now formally specify the property *weak compliance entails strong compliance* (denoted by Δ) [11]. A 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 (see Appendix A) imposed by $\wp$.

For a given privacy policy $\wp$, we syntactically construct a first order CTL* with linear past (denoted by FO-CTL*$_{lp}$) [39] 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 one action from all the possible actions referred by $\wp$, is non-deterministically chosen to be performed. Considering $\mathbb{M}_\wp$ of a policy $\wp$ is reasonable because, if $\wp$ does not have the Δ-property in the $\mathbb{M}_\wp$ then $\wp$ is not well-formed. When a policy can incur obligations that cannot be met even in the most permissive model then it is unlikely that those obligations can be met in other models. Model checking a FO-CTL*$_{lp}$ specification with respect to a given model is undecidable. Thus, in section 5.2, 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 *weakly compliant finite trace* if each of the actions 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 5 (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*$_{lp}$ [39] 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. The formula $\delta(\wp)$ is defined in Figure 4. 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 WC. 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.

$$\delta(\wp) ::= \mathcal{A}\Box\left(\begin{array}{l}(\boxminus \mathit{weak}(\wp)) \longrightarrow \\ \quad E\left(\bigwedge_{\langle\lambda,\gamma\rangle\in\alpha(\wp)} \begin{array}{r}\forall p_1, p_2, q:P.\forall m:M.\forall t:T.\forall u:U. \\ \big((\neg\gamma\,\mathcal{S}\,\lambda) \rightarrow \Diamond\gamma\big) \wedge \Box \mathit{weak}(\wp)\end{array}\right)\end{array}\right)$$

Figure 4: FO-CTL*$_{lp}$ formulation of the Δ-property

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 complete definition of function α is shown in appendix C.

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

Theorem 6. *Given a policy $\wp$, $\wp$ has the Δ-property if and only if $\mathbb{M}_\wp \models \delta(\wp)$.*

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 complaint extension). We call a policy which does ***not*** allow this case, an *incrementally satisfiable policy.*

Definition 7 (Incrementally Satisfiable). *A pure past FOTL formula ϕ is* incrementally satisfiable *if 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 8 states that these two cases are necessary and sufficient.

Theorem 8. *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.*

We will now give an example for each violation case. First, consider the simple example policy in Figure 5, denoted by $\wp_1$. For brevity, we consider a pLTL policy in which A, B, and H are actions. We also assume at each step only one action can happen. An 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. The same is true for A as it satisfies the first positive norm requiring that there is a B before an A and additionally it satisfies all the negative norms. The same goes for 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 H has not happened before. The same goes for B as it violates the second negative norm. Finally, H cannot happen as an H has happened already, 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 6 (denoted by $\wp_2$) to demonstrate the second violation case. We assume that at each step only one action can happen. Let us consider the finite trace $DCBA$. Each of the actions of the trace is weakly compliant with respect to $\wp_2$. The action A, however, incurs the obligation F. Note that the last positive norm allows F under the condition that action C has not 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 $\wp_2$.

Negative Norms :
$A \rightarrow \neg(\diamondminus H)$
$B \rightarrow \neg(\diamondminus H)$
$H \rightarrow \ominus(\neg(\diamondminus H))$
Positive Norms :
$A \wedge \diamondminus B$
B
H

Figure 5: Violation (1)

Positive Norms :
$A \wedge \diamondminus B \wedge \diamondminus C$
$B \wedge \diamondminus C \wedge \diamondminus D$
$C \wedge \diamondminus D$
D
$F \wedge (\neg(\diamondminus C))$
Negative Norms :
$A \rightarrow \Diamond F$

Figure 6: Violation (2)

We now prove that the violation case 1 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.

THEOREM 9. *A closed, pure-past, and satisfiable policy $\wp$ without the $\ominus$ temporal operator, is incrementally satisfiable.*

Privacy policies of our form ($\wp$) do not allow the $\ominus$ temporal operator, which yields the following two corollaries.

COROLLARY 10. *For a given privacy policy $\wp$, weak($\wp$) is incrementally satisfiable if weak($\wp$) is satisfiable.*

This follows by Theorem 9, as *weak*($\wp$) is closed, has no future operators, and does not allow the $\ominus$ temporal operator.

COROLLARY 11. *A satisfiable privacy policy $\wp$ satisfies Δ-property if and only if no weakly compliant action of $\wp$ incurs any unsatisfiable future obligations.*

Corollary 11 follows from Theorem 8 and Corollary 10.

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 (see Appendix A) imposed by $\wp$.

THEOREM 12. *For a given privacy policy $\wp$ with the Δ-property, weak($\wp$) expresses the strongest safety property that contains the property expressed by $\wp$.*

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 a specification written in FO-CTL*$_{lp}$ with respect to a model is undecidable. The complexity of model checking a propositional CTL*$_{lp}$ formula with respect to a model is in EXPSPACE [39] 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 policy is in FOTL, we cannot directly use this technique to check the Δ-property.

5.2 Analysis Technique for Checking the Δ-property

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.

1. Privacy policy slicing.
2. Developing a small model theorem [25].
3. pLTL policy analysis.

By Corollary 11, 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 it has the Δ-property is still undecidable.

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 [25]. 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 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.

Finally, we analyze the pLTL policy $\wp$ (step 3). We obtain a CTL*$_{lp}$ formula from $\wp$ (see Figure 4). 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.* [39], 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* [44] (with Büchi accepting condition [16]) 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.

5.2.1 *Privacy Policy Slicing*

For our policy analysis, the privacy policy size is a bottleneck. As it turns out, our policy specification language allows us to use a divide-and-conquer approach for verification. Without loss of generality, we assume that one send event occurs in a single state. 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, we introduce *privacy policy slicing* analogous to *program slicing* [54]. Slicing decomposes the privacy policy with respect to an obligation. The requirements of privacy policy slicing, which make it interesting for our analysis, are the following: (A) The slicing preserves the Δ-property of the original policy with respect to the slicing criterion. (B) 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: $\text{send}(p_1, p_2, m) \wedge \text{contains}(m, q, t) \wedge \text{for-purpose}(m, u) \wedge \mathbb{C}_{\text{sender}} \wedge \mathbb{C}_{\text{receiver}} \wedge \mathbb{C}_{\text{subject}} \wedge \mathbb{C}_{\text{attribute}} \wedge \mathbb{C}_{\text{purpose}}$. 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.

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.

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. 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. A send event is called *regulatory* 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 send event is given in appendix B.

Consistency. We say a send event Q_1 is *consistent* with another send event 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 send events 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 then 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, $\text{inrole}(p, \text{doctor}) \leftrightsquigarrow \text{inrole}(p, \text{psychiatrist})$.
- $\text{in}(t, \hat{t}) \leftrightsquigarrow \text{in}(t, \bar{t})$ 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, $\text{in}(t, \mathit{PHI}) \leftrightsquigarrow \text{in}(t, \mathit{psych\text{-}notes})$ as the attribute "*diagnosis*" can be calculated from both *PHI* (protected health information) and *psych-notes*.
- $\text{purpose}(u, \hat{u}) \leftrightsquigarrow \text{purpose}(u, \bar{u})$ when $\hat{u} = \bar{u}$.
- $\mathbb{C}_i^x \leftrightsquigarrow \mathbb{C}_i^y$ if and only if there exists an atomic formula a_x of $\mathbb{C}_i^x$ that is consistent with an 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 $\text{inrole}(p_1, \text{doctor}) \leftrightsquigarrow \text{inrole}(p_1, \text{psychiatrist})$.

We assume that an empty constraint is consistent with any constraint. We can now inductively check whether two send events are consistent.

Dependency. The next notion we need for defining privacy policy slicing is called the norm *dependencies*.

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 ϕ_2 can influence one of the conditional sends of ϕ_1. 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 ϕ_2 can influence one of the obligatory sends of ϕ_1. A norm ϕ_1 has an *anti-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 event based on possibly different conditions.

We say norm ϕ_1 *depends* on norm ϕ_2 if ϕ_1 has either a positive-, negative-, or anti-dependency on ϕ_2.

We can now precisely define privacy policy slicing.

Definition 13 (Slice of a Privacy Policy). *Given a privacy policy $\wp$ with the norm set ϕ, a slicing criterion P, $\wp_{s(P)}$ with the norm set $\phi_{s(P)}$ is a slice of $\wp$ with respect to P if it satisfies the following.*

1. $\phi_{s(P)} \subseteq \phi$.
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 (A) and (B) mentioned above. To show that the slicing procedure satisfies both the requirements (A) and (B), we have the following theorems. The first (Theorem 14) formalizes the requirement that the slicing procedure preserves the Δ-property of the original policy with respect to the slicing criterion.

Theorem 14. *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, and environment η where $\sigma_f, |\sigma_f|-1, \eta \models \boxminus weak(\wp)$, $\sigma_f, |\sigma_f|-1, \eta \models \boxminus weak(\wp_s)$, and $s, \eta \models P$, 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 $\sigma_f \cdot s \cdot \sigma_j \models \wp$.*

The next theorem (Theorem 15) precisely formalizes the requirement that the results of the decomposed policies can be composed together to get the result of the original policy. Recall that we use obligations as our slicing criterion.

Theorem 15. *For a policy $\wp$, if for all obligations P in $\wp$, $\mathbb{M}_{\wp_{s(P)}} \models \delta(\wp_{s(P)})$ where $\wp_{s(P)}$ is the slice of $\wp$ with respect to P, 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 in appendix D.

5.2.2 *Small Model Theorem (SMT)*

Our policies are specified in a restricted subset of FOTL which is not a decidable fragment of full FOTL [34]. The fragment we consider can have more than one free variable for subformulas of form $\psi_1 \mathcal{S} \psi_2$ (non-*monadic*) [34]. On the contrary, pLTL is decidable. A small model theorem (or, a finite model theorem) [25] will establish that any *behavior of interest* of a policy in our specification language 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, the full infinite carrier $\mathcal{P}$ of principals might be able to 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. By the theorem, the resulting propositional policy will capture all the behaviors, necessary to prove the property we are interested in, as the original FOTL policy. That means checking the pLTL policy will suffice.

It is not clear whether there exists a small model theorem for all privacy policies specified in our language. This is what results in the incompleteness in our technique. This means that small model theorems must be derived for specific policy instances and specific properties of it (*e.g.*, consistency, Δ-property, *etc.*). 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 6, 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 our specification language will have the following general form. The Theorem 16 in section 6 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.

6. HIPAA: A CASE STUDY

In this section, we demonstrate the adequacy of our policy analysis techniques by using HIPAA as a case study.

Specification of HIPAA. We have specified all 84 disclosure related clauses in our language [1]. We considered the HIPAA privacy rule in Subpart E of CFR §164. We have 66 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.

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 $\wp_H$.

Incremental satisfiability of HIPAA. In our policy language, we do not allow the $\ominus$ operator. Moreover, $\wp_H$ is trivially satisfiable as discussed just above. By Corollary 10, it follows that *weak*($\wp_H$) is incrementally satisfiable.

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 74 norms. The algorithm runs in 480 milliseconds in the worst case on an Intel Core i7 1.73 GHz machine with 4GB of RAM running Ubuntu 12.04.

Making the slicing procedure more precise. To preserve soundness, the dependence relations we define over-approximate the most-precise dependence relations. This is due to the fact that our dependence relation does not take into account the condition of the norms (when the condition is not a send event). However, it is not apparent how to incorporate conditions 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. 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 policy in both cases has 4 norms (1 positive and 3 negative norm) out of total 74 norms (see Figure 7). This is a significant reduction in the size of the norms. We also sliced $\wp_H$ 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. The sliced policy has 1 positive norm and 6 negative norms out of total 75 norms.

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. 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, m, 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 con-

$$(\S\mathbf{160.310}): \text{inrole}(p_1, \textit{secretary}) \wedge \text{inrole}(p_2, \textit{covered-entity}) \wedge \text{inrole}(q, \textit{individual}) \wedge \text{purpose}(u, \textit{compliance-investigation}) \wedge$$
$$\text{request}(p_1, p_2, q, \textit{PHI}) \longrightarrow \Diamond(\exists m_1 : M.(\text{send}(p_2, p_1, m_1) \wedge \text{contains}(m_1, q, \textit{PHI}) \wedge \text{for-purpose}(m_1, \textit{compliance-investigation})))$$
$$(\S\mathbf{164.502(a)(2)(ii)}): \text{inrole}(p_1, \textit{covered-entity}) \wedge \text{inrole}(p_2, \textit{secretary}) \wedge$$
$$\text{inrole}(q, \textit{individual}) \wedge \text{in}(t, \textit{PHI}) \wedge \text{purpose}(u, \textit{compliance-investigation})$$
$$(\S\mathbf{164.502(b)}): \text{inrole}(p_1, \textit{covered-entity}) \wedge \text{inrole}(q, \textit{individual}) \wedge \text{in}(t, \textit{PHI}) \longrightarrow \text{believesMinimumNecessaryForPurpose}(p_1, p_2, q, t, u)$$
$$\vee (\text{inrole}(p_1, \textit{covered-entity}) \wedge \text{inrole}(p_2, \textit{secretary}) \wedge \text{inrole}(q, \textit{individual}) \wedge \text{in}(t, \textit{PHI}) \wedge \text{purpose}(u, \textit{compliance-investigation}))$$
$$(\S\mathbf{164.508(a)(2)}): \text{inrole}(p_1, \textit{covered-entity}) \wedge \text{inrole}(q, \textit{individual}) \wedge \text{in}(t, \textit{psych-notes}) \rightarrow \text{obtainedAuthorization}(p_1, p_2, q, t, u)$$

Figure 7: Sliced HIPAA policy ($\wp_{H_P}$) norms with respect to the obligation in §160.310 of HIPAA.

tains 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 7), denoted by $\wp_{H_P}$.

The number of attributes in the $\wp_{H_P}$ 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 $t \in (PHI \cap PSN)$. 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 $\wp_{H_P}$. The only purpose present in the $\wp_{H_P}$ 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 $\wp_{H_P}$. For the principal sort, we consider one principal for each role in $\wp_{H_P}$ and one additional principal not having any roles. The roles in $\wp_{H_P}$ are: covered entity, secretary, and individual. Considering one individual from each role is sound as $\wp_{H_P}$ 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-property holding, as these 4 principals can simulate all possible behaviors. We have the following small model theorem for $\wp_{H_P}$.

Theorem 16. *The policy $\wp_{H_P}$ has the Δ-property for infinite carriers of sorts $\mathcal{P}$, $\mathcal{T}$, and $\mathcal{U}$ if and only if $\wp_{H_P}$ 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 $|\hat{\mathcal{U}}| = 2$.*

Obligation	Norms in the slice	Automata generation time (s)	Graph analysis time (ms)	Analysis results
§160.310	4	3	2	Passed
Synthetic-1	6	98	16	Passed
Synthetic-2	6	324	31	Failed
§164.524	4	22	5	Passed

Table 1: Policy analysis result (HIPAA)

Policy Analysis Results. We slice $\wp_H$ with respect to 2 obligations in HIPAA (§160.310, §164.524) and 1 synthetic obligation (Synthetic-1). Once we have the small model theorem, we convert the FOTL sliced policy to a pLTL policy. Once we have the pLTL policy, 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 [53], 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 1. We verify that for the two obligations in HIPAA, the Δ-property holds for the sliced policies.

Observation. While verifying the sliced policy with respect to the synthetic obligation, we observed something interesting. Investigating the regulations manually 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 seems to differ. Upon close inspection, we figured out 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.

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 consider that privacy notices do not contain any *PHI* of the patient. Thus, those obligations are trivially allowed. However, in the case that notices contain *PHI* of the patient, then the corresponding slice of the policy is similar to the slice of §164.524.

Counter Example. When a policy violates the Δ-property, we can traverse the tableau to find a path to the violating node. 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.

7. RELATED WORK

May *et al.* [45], based on the extension of the HRU model [30], present a formalism, called Privacy APIs, to encode HIPAA. They only consider §164.506 of the 2000 and 2003 version of HIPAA. They convert their formalism into the specification language of the SPIN model checker [35] and check whether it satisfies some desired invariants.

Ni *et al.* [46] present a family of models named P-RBAC (Privacy-aware Role Based Access Control) that extends the traditional RBAC [26] 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 apparent whether HIPAA can be encoded in their language due to absence of examples of such encodings.

Lam *et al.* [41] propose a privacy policy specification language called pLogic based on Datalog. They can only encode §164.502, §164.506, and §164.508 of HIPAA in pLogic. pLogic cannot specify any kinds of temporal conditions. Our specification language is richer than pLogic. Lam *et al.* [40] later developed a small model theorem for pLogic.

Recall that our specification language is inspired by CI [11]. We now discuss the distinctions between our language and CI. 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 [27]. 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. In light of CI [11, 12], DeYoung *et al.* [21] propose an expressive policy specification language, PrivacyLFP. We adopted some of their improvements over CI (*e.g.*, purpose, subjected belief, *etc.*) but left out others that were not relevant for HIPAA (fixed point operators). The goal of their work is developing a specification language of HIPAA whereas our work focuses on static policy analysis of privacy policies like HIPAA. Garg *et al.* [28] propose an expressive, first-order logic-based privacy policy specification language. HIPAA can be completely encoded in their specification language. They present an auditing algorithm that incrementally inspects the system log against a policy and detects violations. Their approach only considers WC. Our work on the contrary presents a sound, semi-automated technique for verifying the Δ-property of a policy. Once verified, one can use their auditing algorithm to check WC. In the same vein, Basin *et al.* [13, 14] present a monitoring algorithm for policies written in metric first-order temporal logic. Their monitoring algorithm can be also used to check WC for a policy. Chowdhury *et al.* [17] propose extensions of XACML [55] for specifying HIPAA. In their work, they assume all the obligatory actions are authorized. Moreover, their approach can only check WC. Havelund and Roşu [31, 32, 50] propose a dynamic programming approach for monitoring pLTL formula which can be used for checking WC of pLTL policies. Krukow *et al.* [38] provide an algorithm which can be used to check WC of very restrictive FOTL policies.

Several work[36, 49, 18] have addressed permissibility of user obligations in context of access control policies. They introduce a property "*accountability*" of the access control policy and authorization state which ensures that all the incurred obligations be authorized. The accountability property has to be maintained as an invariant of the system whereas the Δ-property is a statically checkable property of the privacy policy which one needs to check once for a policy. Dougherty *et al.* [23] present an expressive obligation model and present analysis technique to approximate obligations for monitoring. The goal of our work is to provide static guarantee of a privacy policy's well-formedness and is thus complementary to [23].

8. CONCLUSION

In this paper, we propose a privacy policy specification language which is expressive enough to capture HIPAA. We then formally specify two different notions of privacy policy compliance (WC and SC). We show that although deciding WC is feasible, checking SC is undecidable. We then formally specify the Δ-property. To check compliance of an action with a privacy policy satisfying the Δ-property, it suffices to only check whether the action is weakly compliant with the policy. The Δ-property can be checked statically before the policy is deployed. While checking the Δ-property for a policy is undecidable, we have developed a sound, semi-automated technique, which we show is adequate for checking the Δ-property for our formalization of the HIPAA Privacy Policy.

Open Problems & Future Work. Our final policy analysis step requires model checking the propositional CTL$^*_{lp}$ formula with respect to $\mathbb{M}_{\wp}$. There are currently no such tools which can model check a propositional CTL$^*_{lp}$ specification against a given model. In an on-going work, we are developing a model checker following [39]. For preliminary results and to show feasibility we currently rely on the approach of Barth *et al.* [11].

LTL abstracts away the explicit notion of time and only reasons about relative orderings of different events. Although the obligations in HIPAA have explicit deadlines, our specification language cannot express explicit deadlines. An extension covering deadlines is plausible (see [13]). However, extending the policy analysis is not trivial, due to the fact that propositional branching time logic with explicit time is undecidable [8]. In HIPAA, deadlines appear only in the context of obligations. Our analysis of a policy without obligation deadlines is extendable to policies, which only have deadlines on 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.

Currently we verify the Δ-property for a simplified version of the obligation in §164.524 of HIPAA. We plan to verify it for the general obligation in §164.524. We also want to explore techniques to check whether specific models of the system satisfy the Δ-property. We also would like to apply our policy analysis techniques to other regulations like GLBA [3], SOX [51], *etc.*

Additional Authors

William H. Winsborough contributed in this research until he passed away in August, 2011.

Acknowledgement

This material is based upon work supported by the National Science Foundation under Grant No. CNS-0964710 and by the Department of Health & Human Services under Grant No. HHS 90TR0003/01. The authors thank Ninghui Li, Keith Irwin, Murillo Pontual, Deepak Garg, Sruthi Bandhakavi, and the anonymous reviewers for their helpful suggestions. The authors also thank the team members of the GOAL tool [53].

9. REFERENCES

[1] HIPAA ENCODING. http://galadriel.cs.utsa.edu/∼ochowdhu/HIPAA-POLICY/.

[2] Privacy Promises That Can Be Kept: A Static Policy Analysis Method with Application to the HIPAA Privacy Rules. Technical Report CS-TR-2013-004. UT San Antonio.

[3] Senate banking committee, Gramm-Leach-Bliley Act, 1999. Public Law 106-102.

[4] Enterprise privacy authorization language (EPAL) version 1.2, Nov. 2003. http://www.zurich.ibm.com/pri/projects/epal.html.

[5] M. W. Alford and *et al.*, editors. *Distributed Systems: Methods and Tools for Specification '85*, LNCS.

[6] B. Alpern and F. B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, Oct. 1985.

[7] B. Alpern and F. B. Schneider. Recognizing safety and liveness. *Distributed Computing*, 2:117–126, 1987.

[8] R. Alur. *Techniques for automatic verification of real-time systems*. PhD thesis, Stanford, CA, 1992.

[9] R. Alur and T. Henzinger. Logics and models of real time: A survey. In *Real-Time: Theory in Practice*, Lecture Notes in Computer Science. 1992.

[10] amednews.com. Clinics fined $4.3 million for hipaa violations, 2011. Available at *http://www.ama-assn.org/amednews/2011/03/07/bisb0307.htm*.

[11] A. Barth, A. Datta, J. C. Mitchell, and H. Nissenbaum. Privacy and contextual integrity: Framework and applications. *IEEE Symposium on Security and Privacy.*

[12] A. Barth, J. Mitchell, A. Datta, and S. Sundaram. Privacy and utility in business processes. In *IEEE CSF '07.*

[13] D. Basin, F. Klaedtke, and S. Müller. Monitoring security policies with metric first-order temporal logic. In *ACM SACMAT '10.*

[14] D. A. Basin, F. Klaedtke, S. Marinovic, and E. Zalinescu. Monitoring compliance policies over incomplete and disagreeing logs. In *RV*, 2012.

[15] T. Breaux and A. Antón. Analyzing regulatory rules for privacy and security requirements. *IEEE TSE '08.*

[16] J. R. Büchi. On a decision method in restricted second order arithmetic. In *Proceedings of LMPS'60*, 1962.

[17] O. Chowdhury, H. Chen, J. Niu, N. Li, and E. Bertino. On xacml's adequacy to specify and to enforce hipaa. In *HealthSec'12.*

[18] O. Chowdhury, M. Pontual, W. H. Winsborough, T. Yu, K. Irwin, and J. Niu. Ensuring Authorization Privileges for Cascading User Obligations. In *ACM SACMAT '12.*

[19] D. Damianou, N. Dulay, E. Lupu, and M. Sloman. The Ponder Policy Specification Language. In *IEEE POLICY '01.*

[20] F. Dederichs and R. Weber. Safety and liveness from a methodological point of view. *Information Processing Letters*, 1990.

[21] H. DeYoung, D. Garg, L. Jia, D. Kaynar, and A. Datta. Experiences in the logical specification of the hipaa and glba privacy laws. In *ACM WPES '10.*

[22] N. Dinesh, A. Joshi, I. Lee, and O. Sokolsky. Checking traces for regulatory conformance. In *RV '08.*

[23] D. J. Dougherty, K. Fisler, and S. Krishnamurthi. Obligations and their interaction with programs. In *Proceedings of ESORICS 2007*, pages 375–389.

[24] E. A. Emerson. Handbook of theoretical computer science. chapter Temporal and modal logic, pages 995–1072. 1990.

[25] E. A. Emerson and K. S. Namjoshi. Reasoning about rings. In *POPL '95*, 1995.

[26] D. F. Ferraiolo, R. S. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *ACM TISSEC '01.*

[27] D. Gabbay. The imperative future. chapter : The declarative past and imperative future, pages 35–67. John Wiley & Sons, Inc., New York, NY, USA, 1996.

[28] D. Garg, L. Jia, and A. Datta. Policy auditing over incomplete logs: theory, implementation and applications. In *ACM CCS '11.*

[29] D. Garg, L. Jia, and A. Datta. A logical method for policy enforcement over evolving audit logs. *CoRR*, abs/1102.2521, 2011.

[30] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman. Protection in operating systems. *CACM '76.*

[31] K. Havelund and G. Roşu. Efficient monitoring of safety properties. *Int. J. Softw. Tools Technol. Transf., '04.*

[32] K. Havelund and G. Rosu. Synthesizing monitors for safety properties. In *TACAS' 02.*

[33] Health Resources and Services Administration. Health insurance portability and accountability act, 1996. Public Law 104-191.

[34] I. M. Hodkinson, F. Wolter, and M. Zakharyaschev. Decidable fragment of first-order temporal logics. *Ann. Pure Appl. Logic '00.*

[35] G. J. Holzmann. The model checker SPIN. *TSE '97.*

[36] K. Irwin, T. Yu, and W. 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.

[37] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Syst.*, 1990.

[38] K. Krukow, M. Nielsen, and V. Sassone. A logical framework for history-based access control and reputation systems. *J. Comput. Secur.*, 16(1):63–101, 2008.

[39] O. Kupferman, A. Pnueli, and M. Y. Vardi. Once and for all. *J. Comput. Syst. Sci. '12.*

[40] P. E. Lam, J. C. Mitchell, A. Scedrov, S. Sundaram, and F. Wang. Declarative privacy policy: finite models and attribute-based encryption. In *ACM IHI '12.*

[41] P. E. Lam, J. C. Mitchell, and S. Sundaram. A Formalization of HIPAA for a Medical Messaging System. In *TrustBus '09.*

[42] L. Lamport. Proving the correctness of multiprocess programs. *IEEE TSE'77.*

[43] Z. Manna and A. Pnueli. The anchored version of the temporal framework. In *REX Workshop '88.*

[44] Z. Manna and A. Pnueli. *Temporal verification of reactive systems: safety*. Springer-Verlag New York, Inc., New York, NY, USA, 1995.

[45] M. J. May, C. A. Gunter, and I. Lee. Privacy APIs: Access control techniques to analyze and verify legal privacy policies. In *IEEE CSFW '06.*

[46] Q. Ni, A. Trombetta, E. Bertino, and J. Lobo. Privacy -aware role based access control. In *ACM SACMAT '07.*

[47] S. Owicki and L. Lamport. Proving liveness properties of concurrent programs. *ACM TOPLAS '82.*

[48] 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.

[49] M. Pontual, O. Chowdhury, W. H. Winsborough, T. Yu, and K. Irwin. On the management of user obligations. SACMAT '11, New York, NY, USA. ACM.

[50] G. Roşu and K. Havelund. Synthesizing dynamic programming algorithms from linear temporal logic formulae. Technical report, 2001.

[51] Securities and Exchange Commission. Sarbanes-oxley act, 2002. Public Law 107-204.

[52] A. P. Sistla. Safety, liveness and fairness in temporal logic. *Formal Aspects of Computing*, 6:495–511, 1994.

[53] Y.-K. Tsay, M.-H. Tsai, J.-S. Chang, and Y.-W. Chang. Büchi store : An open repository of büchi automata. In *TACAS '11.*

[54] M. Weiser. Program slicing. In *ICSE '81.*

[55] XACML TC. Oasis extensible access control markup language (xacml).

APPENDIX

A. SAFETY AND LIVENESS PROPERTIES.

A temporal property is a set of traces. We can divide temporal properties into mainly two categories, safety [42, 5] and liveness [6]. Such a classification of properties can enable choosing the right proof methodology for proving correctness. For instance, methods based on global invariants can be used to prove safety properties whereas methods based on proof lattices or well-founded induction [47] can be used to prove liveness properties.

A safety property asserts that something bad never happens. For instance, a doctor cannot share a patient's information without receiving an authorization from the patient. Liveness properties on the contrary asserts that something good eventually happens. For instance, if a patient requests for accessing her own information, the doctor should provide her access eventually. LTL formulas can express both safety and liveness properties. Furthermore, Alpern and Schneider [7] have shown that any LTL formula can be written as conjunctions of safety and liveness properties. A past only LTL formula can express only safety properties whereas future only LTL formula can express both safety and liveness properties [52]. We now formally define what it means for a property to be safety property or a liveness property.

Definition 17 (Safety Property). *A property P is a safety property if the following holds: $\forall \sigma : \sigma \in S^{\omega} : (\sigma \models P \iff (\forall i : i \geq 0 : (\exists \beta : \beta \in S^{\omega} : \sigma[\ldots i] \cdot \beta \models P)))$. 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.*

Definition 18 (Liveness Property [7]). *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.*

B. EXAMPLE OF REGULATORY SEND

Consider the following negative norm:

inrole(p_1, *covered-entity*)$\wedge$inrole(q, *individual*)$\wedge$in(t, *PHI*) $\rightarrow$ believesMinimumNecessaryForPurpose(p_1, p_2, q, t, u)

The regulated send of the above negative norm is:

send(p_1, p_2, m) $\wedge$ contains(m, q, t) $\wedge$ for-purpose(m, u) $\wedge$ inrole(p_1, *covered-entity*) $\wedge$ inrole(q, *individual*) $\wedge$ in(t, *PHI*)

C. DEFINITION OF THE α FUNCTION

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, α function would return the following $\langle \lambda, \gamma \rangle$ 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 same as the original negative norms except that they do not have any future temporal operators in them (modified ones).

The form of our negative norms are 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 norms. For rest of them, let us consider β is not trivially true. For a negative norm ϕ_j^- whose χ is of form case (1), α function would return the following $\langle \lambda, \gamma \rangle$ 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, represents modified negative and positive norms. When the χ has the form (3) or (4), the function α would return the following $\langle \lambda, \gamma \rangle$ pair: $\langle \mathbb{C} \wedge \psi \wedge \neg(\psi_{exception}) \wedge \psi_1 \wedge \bigwedge_{k \neq j} \hat{\phi}_k^- \wedge \bigvee_i \hat{\phi}_i^+, \beta \rangle$.

Algorithm 1 Slice(Φ, P)

```
Input: A privacy policy represented as a set of norms $\Phi$ and a
    slicing criterion $P$.
Output: returns a sub-policy of the input policy represented as
    a set of norms $\Phi_r$.
 1: /* We assume the following variables to be global */
 2: $\Phi_r$ = 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 $\phi \in \Phi$ do
 9:     if $\phi \notin$ProcessedNorms then
10:       if $\phi$ is of form $(R \wedge \psi \wedge \beta) \vee \psi_{exception}$ and $R \rightsquigarrow Q$
          then
11:         $\Phi_r = \Phi_r \cup \phi$ /* add $\phi$ to the result */
12:         Insert $\phi$ to the map processedNorms
13:         for all Send $x \in \psi \vee x \in \psi_{exception} \vee x \in \beta$ do
14:           AddSend($x$)
15:       if $\phi$ is of form $R \wedge \psi \rightarrow \chi \vee \psi_{exception}$ and $R \rightsquigarrow Q$
          then
16:         $\Phi_r = \Phi_r \cup \phi$ /* add $\phi$ to the result */
17:         Insert $\phi$ to the map processedNorms
18:         for all Send $x \in \psi \vee x \in \psi_{exception} \vee x \in \chi$ do
19:           AddSend($x$)
```

D. SLICING ALGORITHM

In this section, we 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 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 2 is a utility procedure used by algorithm 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 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 $\phi$ is of form $(R \wedge \psi \wedge \beta) \vee \psi_{exception}$ and $Q$ appears as
        an obligatory send in $\beta$ then
 4:       $\Phi_r = \Phi_r \cup \phi$, Insert $\phi$ 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 $\phi$ is of form $R \wedge \psi \rightarrow \chi \vee \psi_{exception}$ and $Q$ appears as
        an obligatory send in $\chi$ then
 9:       $\Phi_r = \Phi_r \cup \phi$, Insert $\phi$ 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($x$)
```