Towards Defining Semantic Foundations for Purpose-Based Privacy Policies

Mohammad Jafari, Philip W.L. Fong, Reihaneh Safavi-Naini, Ken Barker
Department of Computer Science
Univ of Calgary
2500 University DR NW, Calgary, AB, Canada, T2N-1N4
{jafarm,pwlfong,rei,kbarker}@ucalgary.ca

Nicholas Paul Sheppard
Library eServices,
Queensland Univ of
Technology
GPO Box 2434, Brisbane
Australia, QLD 4001
nicholas.sheppard@ieee.org

ABSTRACT

We define a semantic model for *purpose*, based on which purpose-based privacy policies can be meaningfully expressed and enforced in a business system. The model is based on the intuition that the purpose of an action is determined by its situation among other inter-related actions. Actions and their relationships can be modeled in the form of an *action graph* which is based on the business processes in a system. Accordingly, a modal logic and the corresponding model checking algorithm are developed for formal expression of purpose-based policies and verifying whether a particular system complies with them. It is also shown through various examples, how various typical purpose-based policies as well as some new policy types can be expressed and checked using our model.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Security

Keywords

Purpose, Privacy Policy, Access Control, Modal Logic

1. INTRODUCTION

Privacy policies and enforcement technologies are crucial for mitigating risks involved in storage and processing of data in digital form and making such systems safe and reliable. *Purpose of access* is one of the core concepts in privacy which considers the data user's intent as a factor in making access control decisions. This enables differentiating between access to the same piece of data, even by the same person, when it is for a different purpose. For example, a patient may want to allow a physician to see the blood-test results for the purpose of *medical treatment*; but deny access to the same data by the same person for a purposes such as *research*.

Purpose has been considered in major privacy legislations, such as the U.S. Privacy Act (1974) and Canada's Federal Privacy Act (1983). These laws and similar ones in other countries stipulate that personal information must be used only for the purpose that was declared at collection time. More recent research considers *purpose* a decision factor in privacy-oriented access control models [5, 2, 19] and in policy languages [27, 25, 20].

One major issue is that in nearly all existing models, purposes are treated as opaque labels (i.e. a character string) with little or no semantics. This leads to ambiguity and arbitrary interpretation of purposes in privacy policies, often contrary to the interests of data owners. For example, if the privacy policy of a company states that data collected from customers may be used for the purpose of *service improvement*, this is likely to be very unclear to the customers who do not get to have a clear understanding of what exactly this entails, under what conditions it is violated, and how it is enforced.

Besides, and in our opinion as a consequence of such a lack of semantics, enforcing purpose-based policies continues to be a challenging problem. The main difficulty in purpose enforcement is how to identify the purpose of an agent when it requests to perform an action. Some common proposed mechanisms are *self-declaration* in which the agent explicitly announces the purpose of data access (e.g. [16]), and *role-based* enforcement in which the purpose is identified based on the agent's role in the system (e.g. [5]). The first method obviously cannot stop a malicious agent from claiming false purposes. The second method has been criticized to be inefficient in capturing purpose of an action since roles and purposes are not always aligned and members of the same organizational role may practice different purposes in their actions [15]. Therefore, identifying the purpose of an action, or verifying a claimed purpose remains an open question, partly because enough attention has not been paid to the link between actions and their purposes.

This paper addresses these problems by: (a) developing a formalism with which *purpose*, and its relationship to actions is clearly defined, and (b) designing a corresponding mechanism using which the purpose of an action can be identified and thereby adherence to purpose-based policies can be verified and enforced.

Section 1.1 gives an overview of our work by discussing the concept of purpose and how it relates to actions in Section 1.1.1, sketching our formal framework in Section 1.1.2, and showing how our proposed framework will be used in a practical scenario in Section 1.1.3. Section 1.2 goes over some related work and Section 2 formalizes the basics for definition of purpose. Section 3 develops a modal logic to articulate actions and their teleological relationships which is the foundation for formally defining purpose-based policies. Section 4 defines the form of a purpose-based policy in our model and illustrates how various examples of common purpose-based policies can be expressed using the developed language. This section also demonstrates how our model is capable of expressing new types of policies that are not considered in the current literature. Compliance checking for policies is discussed in Section 5 by giving a model checking algorithm that tests whether a given system adheres to some given policy. Section 6 provides a further elaboration of several key points about the framework, leaving Section 7 to make some concluding remarks.

1.1 Our Work

This section briefly describes our conceptual framework, the formal tools that we have developed on its basis, and a walk-through of how it can be used in practice.

1.1.1 Conceptual Framework

Intentional actions are often assigned a *purpose* that refers to the aim and rationale to perform them. One may ask or talk about the purpose of reading a book, increasing salaries, or collecting information. Hence, everyday usages of *purpose* presume a sense of teleology (or final aim) concerning the goal behind executing an action and its ultimate consequences. Thus, the purpose of *reading* a book may be to entertain, or by increasing salaries the ultimate intention may be to attract high-quality workforce, and information may be collected to do medical research.

Observing how *purpose* is used in the natural language reveals that purposes often refer to an action or a set of actions. A web search for *"for the purpose of"* yields top results such as: disseminating information, pricing, promoting, language verification, *etc.* all of which are names of some abstract actions. Typical purpose names mentioned in privacy standards and guidelines also refer to actions; for example, *completion and support of activity* and *website and system administration* in P3P [27], or *treatment* and *research* in Healthcare XSPA [21] and Dimitropoulos's report [8]. The correspondence of purposes and actions has also been observed by others in the literature as it will be mentioned in Section 1.2.

In the first place, purpose of an action is something that only exists in the mind of the agent performing the action. However, an agent's purpose for an action affects other actions performed, so, the set of actions that precede or follow the action are affected by the agent's purpose. Thus, related actions can be indicative of the purpose and the purpose of the action may be revealed by looking at the actions that precede or follow it. For example, when someone borrows money and later uses the money to pay a phone bill, it can be inferred that the purpose of borrowing money has been to pay the bill. Conversely, it is possible to *enforce* a purpose by restricting the surrounding actions. Thus, the purpose of *paying the bill* can be enforced by forbidding the agent from doing anything but making the payment as a consequence of borrowing the money.

Thus, purpose can be defined by the situation of an action within a larger context containing other actions and the relationships among them. This context can be defined as a network of relationships that capture the intention, or more precisely, the purpose of the action. Accordingly, we can define the purpose of an action as *its placement in a collection of other related actions*; we call such network of inter-related actions a *plan.* This is the fundamental assumption that forms the basis of our purpose model in this paper.

Intuitively, a *plan* is a collection of interrelated actions together with various relationships among them. Later in Section 2, we define *action graph* as an abstract model to capture such plans with only two types of relationships. These two types of relationships are based on the following observations of two types of purposes:

Purpose as a High-Level Action.

In some contexts, *purpose* refers to a more abstract, or semantically higher-level action in a plan. Thus, doing something for some purpose, actually means doing it as a part, or a sub-action, for that higher-level action. For example, when Alice checks some patient's blood pressure *for the purpose of surgery*, it means that checking the blood pressure is a part of a more complex and abstract action of *surgery.* Similarly, when it is said the a surgery is performed *for the purpose of treatment*, it is because the high-level action of *medical treatment* includes *surgery* as a part.

In some contexts purpose refers to a desired state of affairs, such as doing some action for the purpose of *happiness.* In such cases we assume that the purpose refers to the abstract action of *reaching* that state. Thus, doing something for the purpose of *happiness*, can be interpreted as doing something as part of the abstract action of *pursuing happiness*, which is the higher-level action.

Purpose as a Future Action.

In some contexts, purpose is used to indicate that an action is performed as a prerequisite of another action in future. For example, when Bob withdraws money from a bank account for the purpose of *paying the bills*, it means the former action is done as a prerequisite to performing the latter.

1.1.2 Formal Framework

We develop formal tools for expressing and verifying purpose-based policies. A formal model is developed for the business processes in the system. This can be used to formalize the relationships between different high-level and low-level actions in a business system in the form of a graph. Correspondingly, a formal language is also developed using which purpose-based policies can be expressed about such business processes. The model checking algorithm can be used to check whether the business processes in a particular system comply with the policies.

1.1.3 Walk-Through

We motivate our framework by explaining how it can be used in practice. Suppose there is an organizations with well-defined business processes. The aim is to check whether the business processes in this organization comply with a set of purpose-based privacy rules.

Step 1: Vocabulary.

A common terminology is necessary for referring to system's actions so that business processes and policies can use the same vocabulary. Low-level actions such as *read*, *write*, *etc.* are well-known and common across many domains with clear and standard meanings. More complex and abstract actions like *surgery*, *marketing*, *etc.* can be taken from standard vocabularies that exist in many domains such as clinical systems in healthcare (e.g. [22]). In this paper, we do not discuss how such a vocabulary is developed and assume it exists.

Step 2: Abstract Model of Actions in the System.

The next step is to make a formal model of actions in the system and their relationships. The action names in such a model are taken from the common vocabulary and their relationships can be extracted based on the business processes in the system. This model reflects how the system works and is used to evaluate whether it complies with the purpose-based policy. Definition and explanation of this model is given in Section 2.

Step 3: Policy.

Purpose-based policy is a set of rules about the purposes of actions in the system. For example, one may want to make sure some data is not used for the purpose of *marketing*, or patient files are not modified when used for the purpose of *research.* Such rules come from different origins; there are global system-wide rules that apply to an entire organization or even multiple organizations, and are usually authored by a management authority. For example, a jurisdictional policy may stipulate that employee ethnicity data must not be used for the purpose of *promotions* in any organization in the country. On the other hand, there are data-dependent policies, such as patient consents, that are specific to treatment of a particular piece of data and are effective only if that piece of data is being processed. Such policies are usually defined by the data owner.

Policy rules should be formalized using a language we develop in Section 3. The atomic propositions in the language (e.g. *marketing*, *research*, and *promotion* in the previous examples) are taken from the common vocabulary described above. The semantics of the language is defined based on the formal model of the actions in the system, and therefore, what is expressed in the language has a clear meaning about the actions in the system and their relationships.

Step 4: Model Checking and Policy Enforcement.

Our final goal is to test whether the system complies with the policies. Having a formal model of the actions in the system and after formalizing the policy rules using the developed language, a model checking algorithm is used to check whether the model satisfied the rules. The model checking algorithm can also be used to enforce the policy at run-time by blocking access if the model did not comply with it. For example, if Alice's consent requires that her blood test results cannot be *read* for the purpose of *research*, the model checking algorithm can be run when a read access is requested to her file and test whether the purpose-based rule is satisfied.

1.2 Related Work

The conceptual link between purposes and actions has been observed by a number of others in the literature and can be considered supportive to our approach of defining purpose using related actions.

van Sataden *et. al.* suggest that purpose names can be taken from the verbs in a standard dictionary [26]. Similarly, Powers *et. al.* mention that business purposes are a form of high-level action and argue that in high-level privacy policies instead of referring to low-level actions such as *read* or *write*, high-level business purposes such as *treatment* or *diagnosis* are used [23].

In the context of an object-oriented system, Yasuda *et al.* associate purpose with the caller method [28]. For example, if the *housekeeping* method of a person object calls the *withdraw* method of a bank account object, the implication is that money is withdrawn for the purpose of *housekeeping.* This is consistent with our definition of purpose in its first meaning that refers to a higher-level, more abstract action.

In their development of a formal semantics for privacy policies, Breaux and Antón propose to model purpose as an auxiliary related action [4]. For example, the policy that *data is collected for the purpose of marketing* is taken to mean the primary action of *data collection* is related to the auxiliary action of *marketing* that happens later. This is very similar to our notion of purpose as a future action.

HL7 Reference Information Model ("RIM") specifications, a data model used as the basis for designing many healthcare systems, mentions the *has reason* relation between two actions for specifying that one is the *reason* for the other [11]. In this design, *reason* is similar to *purpose* of an action, especially in its sense as as future action.

There are some proposals [9, 10, 6, 15] that suggest associating purpose with the units of work in a system. They argue that *tasks* or *workflows*, can be used to identify the purpose of an action by looking at the higher-level unit of work in which it takes place. The higher-level unit of work is basically equivalent to our definition of more abstract actions, so this approach is consistent with the first type of purpose as defined above. Our proposed model is more general and can encompass these approaches as very simple special cases. Moreover, our work extends earlier approaches by considering both meanings of purpose as future event and more abstract action. Also, it allows multiple purposes to be present for a single action which is a feature that is not considered in any other works on purpose-based models as far as we are aware (see Section 6.6).

A different line of research on *obligations* in access control systems is also concerned about actions and how they relate to future actions (e.g. [12, 13]). The part of our model that considers future actions has some similarities to this line of work, but we are also concerned about other relationships among actions which are not of interest in the study of obligations. Also, even in the study of future actions we do not follow a strict linear notion of time and our model allows multiple future actions whose order is immaterial (see Section 2).

As the formal language, we use modal logic to articulate the relationships among actions. Temporal logic, which is a special type of modal logic, has been used previously to formalize different relationships between actions; for example, control flow [17] or obligations policies [12].

We introduce action graphs as an abstract model of different actions in the system and their order. Our model of action graph is similar to *control flow graph* used to assess programs in programming languages [1] and hierarchical planning in the artificial intelligence literature [24].

2. MODELING PLANS: ACTION GRAPH

Based on the discussion in Section 1.1.1, we define a formalism for a *plan* that captures the actions and two types of relationships among them. The action graph can be thought of as an abstract model based of the business processes in a system. We define an *action graph* as a directed graph in which nodes correspond to actions (both high-level and low-level) and edges denote the relationships among them.

There are two types of edges each corresponding to one type of relationship mentioned in Section 1.1.1: *prerequisite-of* and *part-of.* The prerequisite-of relationship signifies that one action is performed as an antecedent for another action, and the part-of relationship indicates that one action is performed as part of a higher-level more abstract action. For brevity, we will refer to the prerequisite-of and part-of relationships, respectively as the F- and A-relationships, short for *future* and *abstract.*

An example of such a graph is shown in Figure 1 where various purposes can be identified based on F- and A-edges. For example, the purpose of *opening the file* (node g), is to *read* its contents (node h), which is manifested by the F-relationship between the two that says opening the file is a prerequisite for reading its content. Moreover, both *opening* and *reading* the file are for the purpose of *loading the patient's information* (node f). This means that *opening* and *reading* the file are part of the realization of the *load patient's file* action. Similarly, *loading the patient's file* is in turn for the purpose of *checking blood pressure history* (node e), and so on. Eventually, as the graph shows, all of the actions in the graph are for the purpose of *cancer treatment* (node a) as they are all part of the realization of this action.

Note that an action can be assigned numerous purposes. For example, the action of *opening the file* can at the same time be associated with the purpose of *reading the file*, *surgery preparation*, *cancer treatment*, *etc.*

2.1 Defining the Model

The action graph is a directed graph with two sets of edges, each corresponding to one type of relationship as discussed above. It is defined as $AG = (V, A, F)$ in which V is the set of vertices each of which corresponds to an action, and A and F are subsets of $V \times V$, and respectively correspond to A- and F-relationships. We will use the shorter form uu' instead of the more common (u, u') to denote pairs throughout this paper. The action graph satisfies the following conditions:

(a) $A \cap F = \emptyset$,

(b) (V, A) is a tree with its root being the only sink vertex,

(c) $uu' \in F \rightarrow \exists v.\ \{uv, u'v\} \subseteq A$, and

(d) $(V, A \cup F)$ is a directed acyclic graph.

Intuitively, an action graph is a hierarchical workflow, with nodes representing actions. Condition (a) says that the A– and F– relations cannot co-occur between the same pair of nodes. Condition (b) requires that an action can only be part of a single higher-level action, and there exists a single highest-level action. Condition (c) settles that the prerequisite of an action should be part of the same higher-level action. Finally, condition (d) forbids circularity in the graph.

Based on the intuition that F– and A– relationships cannot be circular, we assume that the action graph does not have a cycle, and hence property (d). Real workflows and business processes may however contain cycles which makes it difficult to build an action graph based on them. We leave solving this problem as a future work and assume the action graphs is acyclic for the moment.

Theorem 1 extends property (c) to the reflexive transitive closure of the F-relationship. The proof is given in Appendix A.

Theorem 1.
$uu' \in F^* \Rightarrow \exists v.\ \{uv, u'v\} \subseteq A$ where F^* is the transitive closure of F. ■

In order to accommodate action attributes into the model, we define the labeling function L. Suppose P is the set of all atomic propositions defined by the vocabulary. These propositions can refer to different facts about the actions, such as their names, locations, *etc.* The labeling function $L : V \mapsto 2^P$ maps each action to the set of all atomic propositions that hold true for that action, i.e. all of its attributes. The simplest of such attributes is the name of the action; for example the node representing the surgery action is mapped to a *surgery* proposition which belongs to the vocabulary. See Section 6.1 for a discussion of other attributes such as authorized roles, location, *etc.* and how they can be used to model more complex policies.

3. A MODAL LOGIC FOR FORMALIZING PURPOSE-BASED POLICIES

Since purpose is captured in the form of A– and F–edges in an action graph, defining purpose-based rules requires a language capable of expressing constraints on these relationships. For instance, if the purpose of *marketing* is forbidden for action a, this is interpreted in the action graph as the restriction that action a should not lead to *marketing* along F– and A–edges in the graph; in other words, it must not be a prerequisite, nor be a part of, a *marketing* action.

In this section, we define a modal logic with different modal operators useful to capture such restrictions. There are four basic modal operators and two derived forms. The

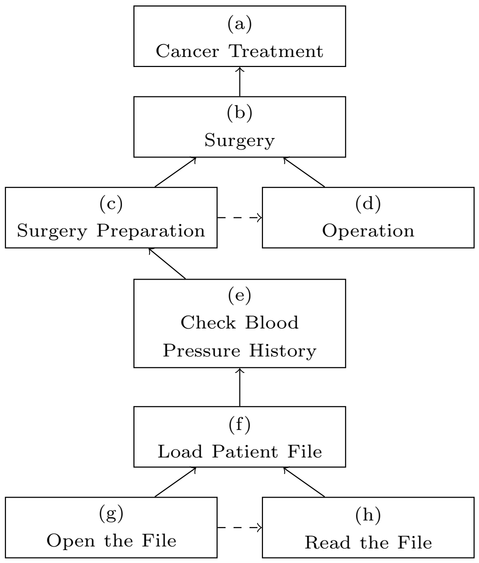


Figure 1: An example of an action graph as defined in Section 2. The F– and A–edges are shown with dashed and solid arrows respectively. Actions are labeled with letters for convenient later reference.

six modal operators are analogous to the conventional temporal logic operators, i.e. $\Diamond$, $\Box$, and $\bigcirc$, that are particularized for F– and A–relationships. We will first give the formal definition of the syntax and semantics of the language, and then, explain the meaning of the operators using some examples.

The syntax of the language is presented in Backus-Naur Form as follows:

$$\phi ::= \top \mid p \mid \neg\phi \mid \phi \wedge \phi \mid (\mathcal{A})\phi \mid (\mathcal{F})\phi \mid [\mathcal{A}]\phi \mid [\mathcal{F}]\phi \quad (1)$$

where p is any atomic proposition.

The semantics of the language is defined by a satisfaction relation ($\models$) in the context of a certain vertex v in the action graph $AG = (V, F, A)$, and a labeling function L. We write $AG, L, v \models \phi$, or more simply $v \models \phi$ where the context is clear; we also write $v \not\models \phi$ when it is not the case that $v \models \phi$. The satisfaction relation $\models$ is defined as follows:

- $v \models \top$ always holds.
- $v \models p$ holds iff $p \in L(v)$.
- $v \models \neg\phi$ holds iff $v \not\models \phi$.
- $v \models \phi_1 \wedge \phi_2$ holds iff $v \models \phi_1$ and $v \models \phi_2$.
- $v \models (\mathcal{A})\phi$ holds iff $\exists vv' \in A, v' \models \phi$.
- $v \models (\mathcal{F})\phi$ holds iff $\exists vv' \in F, v' \models \phi$.
- $v \models \langle\mathcal{A}\rangle\phi$ holds iff $\exists vv' \in A^*, v' \models \phi$.
- $v \models \langle\mathcal{F}\rangle\phi$ holds iff $\exists vv' \in F^*, v' \models \phi$.

A^* and F^* are the reflexive transitive closures of respectively A and F.

We also define the following derived forms to facilitate expressing more complex formulas:

$$\bot \overset{def}{=} \neg\top$$
$$\phi_1 \vee \phi_2 \overset{def}{=} \neg(\neg\phi_1 \wedge \neg\phi_2)$$
$$\phi_1 \rightarrow \phi_2 \overset{def}{=} \neg(\phi_1 \wedge \neg\phi_2)$$
$$[\mathcal{A}]\phi \overset{def}{=} \neg\langle\mathcal{A}\rangle(\neg\phi)$$
$$[\mathcal{F}]\phi \overset{def}{=} \neg\langle\mathcal{F}\rangle(\neg\phi)$$

Based on the action graph of Figure 1, several examples are presented to clarify the operators' meanings. To keep the examples simple, we assume that the set of atomic propositions is the same as the set of vertices and the labeling function is an identity in that each vertex maps to its name. In other words: $P = V$ and $\forall v \in V.\ L(v) = v$.

F–Next and A–Next.

$(\mathcal{F})\phi$ and $(\mathcal{A})\phi$ mean that ϕ is true at least in one of the immediately following nodes along the F– or A–edges respectively. For example, in the graph of Figure 1, we have $c \models (\mathcal{F})d$, since d is a next node of c according to the F–relation (i.e. $(c, d) \in F$), and $d \models d$. Similarly, we have $e \models (\mathcal{A})(\mathcal{F})d$, since c is a next node of e according to the A–relation (i.e. $(e, c) \in A$), and $c \models (\mathcal{F})d$ as discussed above.

F–Diamond and A–Diamond.

$\langle\mathcal{F}\rangle\phi$ and $\langle\mathcal{A}\rangle\phi$ mean that in the paths of F–, or respectively, A–edges beginning inclusively from the current node, there exists at least one node for which ϕ is true. For example, in the graph of Figure 1, we have $g \models \langle\mathcal{F}\rangle(\mathcal{A})f$ because along the path of F–edges beginning from g, there is a node, namely h, satisfying $(\mathcal{A})f$. Similarly, $e \models \langle\mathcal{A}\rangle(\mathcal{F})d$, because along the path of A-edges beginning inclusively from e, there is a node, namely c, that satisfies $(\mathcal{F})d$.

F–Box and A–Box.

$[\mathcal{F}]\phi$ and $[\mathcal{A}]\phi$ mean that along the paths of F–, or respectively, A–edges beginning inclusively from the current node, all of the nodes satisfy ϕ. For example, in the graph of Figure 1 we have $g \models [\mathcal{F}](\mathcal{A})f$, since along the only path of F–edges beginning inclusively from g, all of the nodes (i.e. g and h) satisfy $(\mathcal{A})f$. Similarly, we also have $g \models [\mathcal{A}](c \rightarrow \langle\mathcal{F}\rangle d)$, because along the path of A–edges beginning from g, the formula $c \rightarrow \langle\mathcal{F}\rangle d$ is true for all nodes. Note that since A–edges by definition form a directed tree there is only one such path.

Theorem 2 is crucial in simplifying policy formulas (see the proof in Appendix A).

Theorem 2.

Any formula of the form $\langle*\rangle...\langle*\rangle\phi$ in which $*$ can be either of $\mathcal{A}$ or $\mathcal{F}$ is equivalent to:

- $\langle\mathcal{A}\rangle\phi$ if it has the form $\langle\mathcal{A}\rangle^n\phi$,
- $\langle\mathcal{F}\rangle\phi$ if it has the form $\langle\mathcal{F}\rangle^n\phi$,
- $\langle\mathcal{F}\rangle\langle\mathcal{A}\rangle\phi$ if it has the form $\langle\mathcal{F}\rangle^n\langle\mathcal{A}\rangle^m\phi$ $(m, n \in \mathbb{N})$,
- and $\langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\phi$ otherwise.

Similarly, $[*]...[*]\phi$ in which $*$ can be either of $\mathcal{A}$ or $\mathcal{F}$, is equivalent to:

- $[\mathcal{A}]\phi$ if it has the form $[\mathcal{A}]^n\phi$,
- $[\mathcal{F}]\phi$ if it has the form $[\mathcal{F}]^n\phi$,
- $[\mathcal{F}][\mathcal{A}]\phi$ if it has the form $[\mathcal{F}]^n[\mathcal{A}]^m\phi$ $(m, n \in \mathbb{N})$,
- and $[\mathcal{A}][\mathcal{F}]\phi$ otherwise.

■

4. PURPOSE-BASED POLICY

A purpose-based policy is a set of rules based on the purposes of actions. A simple example of such rules is to forbid reading a some piece of information for the purpose of *marketing*. In this paper, we only consider purpose-based authorization policies, i.e. those policies that allow or forbid some actions in the system based on their purposes. More complex types of policies that go beyond a yes-or-no decision about an action are left as future work but briefly glanced at in Section 6.3.

Based on our conceptual framework (Section 1.1.1), we assume that a purpose-based policy can be expressed in the form of a set of restrictions on an action graph. For instance, the simple policy that a file should not be read for the purpose of *marketing*, can be interpreted as a restriction that the action of *reading the file* should neither be part of *marketing* action, nor be a prerequisite for a future *marketing* action. The modal logic defined in Section 3 provides a language to express such restrictions, so, we can define the purpose-based policy by assigning formulas of that language to actions. A purpose-based policy will be a set of such rules.

On this basis, we define the purpose-based policy $POLICY$ as a set of formulas of the form $a_i \rightarrow \phi_i$ in which a_i's are action names (i.e. atomic propositions belonging to the vocabulary), and ϕ_i's are formulas belonging to the modal logic language defined in Section 3. Each such rule states that the purpose-based formula ϕ_i should hold when action a_i is to be performed. If this is not the case, the action graph in question (and hence the corresponding business process) is deemed as non-compliant, or a reference monitor blocks the action.

An action graph $AG = (V, A, F)$ satisfies a policy if all the rules of the policy hold in all of its nodes:

$$\forall v \in V. \forall r \in POLICY. v \models r$$

4.1 Types of Policy Rules

This section describes several types of rules in purpose-based policies that can be expressed using the developed language. The current purpose-based policies found in the literature fall into the first three types discussed here. The rest are new types that are a contribution of this paper.

4.1.1 Required Purposes

One type of rule is to require that some action be for some particular purpose; for example, "action a must be for the purpose of *treatment*". To formulate this, one should essentially say that the *treatment* purpose should be visited at some point by traversing along F– or A–edges. According to Theorem 2 any formula of such a form can be reduced to either $\langle\mathcal{A}\rangle\langle\mathcal{F}\rangle$treatment, or one of: $\langle\mathcal{A}\rangle$treatment, $\langle\mathcal{F}\rangle$treatment, or $\langle\mathcal{F}\rangle\langle\mathcal{A}\rangle$treatment. Since the latter three cases all imply the former, the disjunction of the four is equivalent to the former, and hence, the rule can be written in the following form:

$$POLICY \ni (a \rightarrow \langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\text{treatment})$$

Since in our model, it is possible that an action be associated multiple purposes (see Section 6.6), a rule can require more than one purpose. For example, an action may be required to be both for the purpose of *order-processing* and *delivery*. Using the logical conjunction, this can be formulated as:

$$(\langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\text{order-processing}) \wedge (\langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\text{delivery})$$

4.1.2 Forbidden Purposes

There are cases where a purpose must be forbidden for an action; for example, "action a should not be for the purpose of *marketing*". This is actually the negation of the type discussed in Section 4.1.1 and can be formulated as:

$$POLICY \ni a \rightarrow (\neg\langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\text{marketing})$$

or equivalently, as:

$$POLICY \ni a \rightarrow ([\mathcal{A}][\mathcal{F}]\neg\text{marketing})$$

4.1.3 Compound Forbidden and Required Purposes

A rule may arbitrarily forbid or allow purposes. For example, a rule may forbid performing action a for the purpose of *marketing* unless the *treatment* purpose is also involved. In other words, it should either be that the *marketing* purpose is not involved, or both *marketing* and *treatment* are present. Using the types of rules already discussed in Sections 4.1.1 and 4.1.2, we can write this as:

$$POLICY \ni a \rightarrow (\langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\text{marketing} \rightarrow \langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\text{treatment})$$

4.1.4 Order-Based Rules

There are cases where the order of purposes matters; for example, suppose that an insurance company covers the costs of the activities performed for the purpose of *surgery*, but it also wants to make sure that the *surgery* is in turn for the purpose of *treatment*, and not for other purposes such as *cosmetics*, or *birth control*. In such cases, not only the presence of certain purposes, but also their order is important. The above rule, for instance, requires that the *surgery* purpose appear and also be in sequence with the *treatment* purpose. The first part (existence of the *surgery* purpose) is a simple rule of the type discussed in Section 4.1.1. The latter part dealing with the order is a new type with which we are concerned here.

As shown in Section 4.1.1, the requirement that the *treatment* purpose should appear is formulated as $\langle\mathcal{A}\rangle\langle\mathcal{F}\rangle$treatment and this should hold for the *surgery* node, hence:

$$\phi = (\text{surgery} \rightarrow \langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\text{treatment})$$

Now, we want to say that ϕ should hold at all of the nodes accessible from the current node, that is the right-hand side of the implication must hold for any *surgery* node visitable by traversing along the A– and F–edges. This can be stated using a formula of the form discussed in Theorem 2, which is eventually reduced to the following simple form according to that theorem:

$$POLICY \ni a \rightarrow ([\mathcal{A}][\mathcal{F}]\,(\text{surgery} \rightarrow \langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\text{treatment}))$$

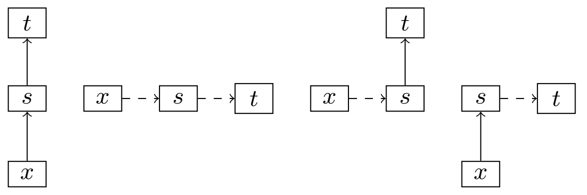


Figure 2: Four cases for a *treatment* (t) purpose at the distance of two.

Note that a simpler rule that requires both *surgery* and *treatment* purposes does not necessarily describe the same requirement. Imagine a scenario where someone wants to perform a cosmetic surgery but has a blood pressure problem that makes such a surgery dangerous and therefore needs to go through a treatment process to cure that problem first. In such a case, both *treatment* and *surgery* purposes are involved, but the order is contrary to what is desired, that is, the *treatment* is for the purpose of *surgery*.

It is possible to simplify order-based rules by assigning them to a different node in the action graph. For instance, the above rule can be simplified by finding all the *surgery* nodes (accessible from a) and requiring them to be for the purpose of *treatment* by a simpler rule of the type discussed in Section 4.1.1:

$$POLICY \ni \text{surgery} \rightarrow (\langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\text{treatment})$$

This can help an organization to simplify such rules by converting them into a simpler form and assigning them to different nodes of the action graph.

4.1.5 Distance-Based Rules

Another possible type of rules limits the distance between the purpose and the action in order to forbid access even for valid purposes, when they are very indirect and rendered irrelevant due to the degree of indirectness. For example, a rule may allow access for *treatment* purpose only within a distance of 3, or in other words, when the *treatment* purpose is involved but there are at most two other intervening purposes.

Distance-based policies are a bit more complex to formulate. First, we consider exact (rather than maximum) distances. For example, a distance of 2 for *treatment* can be formulated with the following which captures different cases of a distance of 2. These cases are shown in Figure 2.

$$(\mathcal{A})^2\text{treatment} \vee (\mathcal{F})^2\text{treatment}$$
$$\vee(\mathcal{F})(\mathcal{A})\text{treatment} \vee (\mathcal{A})(\mathcal{F})\text{treatment}$$

Higher distances can be formulated similarly, with longer formulas. A maximum distance rule then, can be formulated by the disjunction of all distances lower than the maximum; for example, 3, or 2, or 1, for the maximum distance of 3. The following notation [3] can facilitate formalizing distance-based rules. Note that * is used to represent either of $\mathcal{F}$ or $\mathcal{A}$:

$$\langle * \rangle^{\leq d}\phi \overset{def}{=} \bigvee_{0\leq i\leq d} (*)^i\phi$$

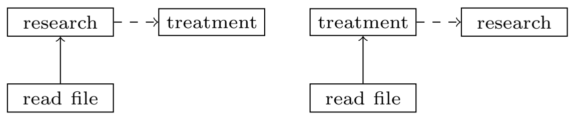


Figure 3: An example showing how the difference between *purpose as prerequisite-of* and *purpose as part-of* can affect the policy.

4.1.6 Other Miscellaneous Rules

In all rule types discussed so far, we have treated A– and F–relationships similarly and did not distinguish between purposes resulting from these two relationships. However, there are cases where the rule is based on such a distinction. As an example, consider the following two scenarios:

- A patient's data is used in some research project, and the results of the research will eventually be used to improve the treatment of some disease.
- The patient's data is used in the course of treatment, but the results of the treatment will later be used for research purposes.

In both cases the data is used for the purpose of *research* and *treatment*, but the difference between *purpose as part-of* and *purpose as prerequisite-of* distinguishes them. Figure 3 shows these two scenarios. Thus, the patients can be more precise in their consent to allow the data be used only as part of a treatment process and not as part of some research that will subsequently be used in treatment. Such a rule can be formulated as:

$$\langle\mathcal{A}\rangle\text{treatment}$$

Figure 3 illustrates how this rule allows the first case, but denies the latter. Note that a simpler rule of the type discussed in Section 4.1.1 that require *treatment* purpose, would be satisfied by both cases, contrary to what is desired.

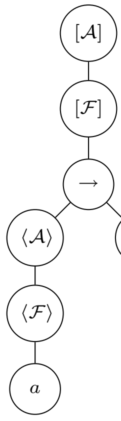
As another example, consider a case where a customer is giving contact information to a company and does not like this information to be used for *marketing* purposes with the exception of receiving free promotion product samples. Thus, the rule is that access is not allowed for *marketing* unless there is also a *delivery* purpose involved as *part of* the marketing. Note that a delivery that *leads to* marketing, that is, done as a prerequisite for a marketing action in future, is not allowed because it may, for example, correspond to the case where an item is delivered and the address is kept for later use when sending marketing mails, which is unacceptable to the customer. Such a case can be formulated as:

$$\langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\text{marketing} \rightarrow$$
$$(\langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\text{delivery} \wedge [\mathcal{A}][\mathcal{F}]\,(\text{delivery} \rightarrow \langle\mathcal{A}\rangle\text{marketing}))$$

This rule says that if the *marketing* purpose is present, the *delivery* purpose must also be present and it must be *part of* a *marketing* purpose.

5. MODEL CHECKING

Model checking is the problem of deciding whether or not a node in a given action graph satisfies a formula. The input to the algorithm is the formula ϕ, an action graph AG,



❶	a
❷	$\langle\mathcal{F}\rangle a$
❸	$\langle\mathcal{A}\rangle\langle\mathcal{F}\rangle a$
❹	b
❹	$b \rightarrow \langle\mathcal{A}\rangle\langle\mathcal{F}\rangle a$
❻	$[\mathcal{F}](b \rightarrow \langle\mathcal{A}\rangle\langle\mathcal{F}\rangle a)$
❼	$[\mathcal{A}][\mathcal{F}](b \rightarrow \langle\mathcal{A}\rangle\langle\mathcal{F}\rangle a)$

Figure 4: The abstract syntax tree of the formula $[\mathcal{A}][\mathcal{F}](b \rightarrow \langle\mathcal{A}\rangle\langle\mathcal{F}\rangle a)$ and the ordered set of its subformulas based on its post-order traversal.

a certain node v, and the labeling function L; the output is a yes-or-no answer that indicates whether $AG, L, v \models \phi$ holds. In other words, model checking is the process of testing whether a plan adheres to a policy rule.

Figure 5 outlines a model checking algorithm. This algorithm is based on the standard model checking algorithm for CTL [7] and the model checking algorithm for history-based access control policies [18], so we only give a brief discussion here.

The main idea of the algorithm is that evaluating a formula at a certain node can be performed recursively by (a) evaluating its subformulas on that node, and (b) evaluating it on the current and immediately proceeding nodes along the F– and A–edges in the action graph. For example, checking $\langle\mathcal{A}\rangle\phi$ can be performed recursively by checking whether ϕ holds in the current node, and $\langle\mathcal{A}\rangle\phi$ holds in any of the immediately next nodes along A-edges.

Using dynamic programming, the result of this recursive algorithm can be built bottom-up. For this purpose, a mechanism is needed for ordering both subformulas and the nodes of the action graph so that the two-dimensional array of the dynamic programming can be formed. For ordering the subformulas, we use the post-order traversal of the abstract syntax tree of the formula. Figure 4 shows an example of such a tree. This guarantees that the subformulas of a complex formula are evaluated first and allows evaluating a formula in a bottom-up manner, beginning from the atomic propositions. The algorithm for post-order traversal is straightforward. For ordering the nodes of the action graph, we rely on a topological sort based on A– and then F–edges, beginning from the sink. As an example, the order of nodes for the action graph of Figure 1 is: a, b, d, c, e, f, h, and g.

The algorithm fills a two-dimensional array, like the one of Figure 6, in which each cell indicates whether or not the corresponding sub-formula holds true at the corresponding node. It begins by evaluating the first sub-formula (an atomic proposition) on the first node of the action graph (the sink node) and continues to fill the array using the following rules:

Input: $AG = (V, A, F)$, L, ϕ
Output: Res: evaluation of ϕ on every node in AG.

```
Subformulas= post-order traversal of $\phi$'s abstract syntax tree
Nodes = Topological sort of AG.V
Res //stores the results
foreach (n in Nodes)
  foreach (f in Subformulas)
    if (f.isAtomic())
      Res[f,n]=(f $\in$ L(n))
    else if (f.operator== $\neg$)
      Res[f,n]=$\neg$(Res[f.operand,n])
    else if (f.operator== $\wedge$)
      Res[f,n]=Res[f.operand1,n]$\wedge$ Res[f.operand2,n]
    else if (f.operator== $(\mathcal{A})$)
      Res[f,n]=false
      foreach (a so that (n,a) $\in$ AG.A)
        if (Res[f.operand,a]==true)
          Res[f,n]=true
    else if (f.operator== $(\mathcal{F})$)
      Res[f,n]=false
      foreach (a so that (n,a) $\in$ AG.F)
        if (Res[f.operand,a]==true)
          Res[f,n]=true
    else if (f.operator== $\langle\mathcal{A}\rangle$)
      Res[f,n]=false
      if (Res[f.operand,n]==true)
        Res[f,n]=true
      else
        foreach (a so that (n,a) $\in$ AG.A)
          if (Res[f,a]==true)
            Res[f,n]=true
    else if (f.operator== $\langle\mathcal{F}\rangle$)
      Res[f,n]=false
      if (Res[f.operand,n]==true)
        Res[f,n]=true
      else
        foreach (a so that (n,a) $\in$ AG.F)
          if (Res[f,a]==true)
            Res[f,n]=true
return Res
```

Figure 5: The pseudo-code of the model checking algorithm.

- Evaluation of an atomic proposition on a node n is simply checking whether it is a member of $L(n)$.
- Evaluation of logical operators are done based to the rules of logic. Note that because of the post-order traversal, the operand subformulas are already evaluated.
- Formulas of the form $(\mathcal{A})\phi$ and $(\mathcal{F})\phi$ are evaluated by checking whether ϕ is evaluated as true at least in one of the nodes immediately following the current node along the A– and F–edges respectively. Note that because of the topological sort, proceeding nodes are already evaluated and the results already exist in the table.
- Formulas of the form $\langle\mathcal{A}\rangle\phi$ and $\langle\mathcal{F}\rangle\phi$ are evaluated to true if they are true for one of the immediately proceeding nodes along the A– and F–edges respectively, or if ϕ has been evaluated to true for the current node.
- Formulas of the form $[\mathcal{A}]\phi$ and $[\mathcal{F}]\phi$ are evaluated to true if they are true in all immediately proceeding nodes, respectively along the A– and F–edges, and also ϕ has been evaluated to true at the current node.

Figure 6 shows the dynamic programming array evaluated for the formula of Figure 4 on the action graph of Figure 1. We have assumed a simple labeling function that maps nodes to their names as a proposition. Note that this algorithm

		a	b	d	c	e	f	h	g
❶	a	1	0	0	0	0	0	0	0
❷	$\langle\mathcal{F}\rangle$❶	1	0	0	0	0	0	0	0
❸	$\langle\mathcal{A}\rangle$❷	1	1	1	1	1	1	1	1
❹	b	0	1	0	0	0	0	0	0
❺	❹$\rightarrow$❸	1	1	1	1	1	1	1	1
❻	$[\mathcal{F}]$❺	1	1	1	1	1	1	1	1
❼	$[\mathcal{A}]$❻	1	1	1	1	1	1	1	1

Figure 6: The dynamic-programming array for evaluating the formula of Figure 4 on the action graph of Figure 1.

will end up evaluating the formula for all nodes in the action graph; it can be changed to halt as soon as it evaluates the formula on the particular node of interest. It is also noteworthy that for evaluating multiple formulas it will be more efficient to extract all the subformulas and then evaluate all formulas in a single pass.

The correctness of the algorithm can be checked by noticing that the recursive part of the dynamic programming calculation matches the definition of the operators. The complexity of the algorithm is $O(|\phi|(e+n))$ in which $|\phi|$ is the size of the formula (i.e. the total number of atomic propositions and operators), e is the number of edges in the action graph, and n is the number of its nodes. Note that based on the loops of line 4 and 5 of Figure 5, all subformulas (with a total number of $|\phi|$) are evaluated at each node. The evaluation in the worst case needs examining that node and all of its outgoing edges. If d_i shows the out-degree of the ith node, the complexity is $O(|\phi|((1+d_1)+...+(1+d_n)))$ which in turn yields $O(|\phi|(e+n))$.

6. DISCUSSION

Several additional aspects of this work should be considered:

6.1 Modeling More Complex Policies

The labeling function L maps actions to the set of propositions that hold true for that node (see Section 3). So far, we have only used very simple labeling functions that map each node to its name as a proposition. However, the labeling function can model other attributes of actions, such as the authorized roles, input data types, time and location constraints, *etc.* This enables defining more complex policies, particularly those that combine purpose constraints with other types of constraints, e.g. role-based, or time- and location-based. For example, a policy that settles *any action for the purpose of research must be limited to business hours* can be formulated as the following, assuming that *biz-hours* is a proposition that holds true for actions that are restricted to business hours:

$$[\mathcal{A}][\mathcal{F}](\langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\text{treatment} \rightarrow \text{biz-hours})$$

Another interesting example is the case of an action's location which can be important in privacy policies. Suppose that *third-party* is a proposition that is true for actions performed by collaborator organization, and false for actions that are performed within the organization. Using such a scheme, an inter-organizational workflow can be modeled as a single unified action graph in which the location of actions is captured using this *third-party* attribute. A policy rule such as "reading for the purpose of research is not allowed by third parties" can then be modeled as:

$$[\mathcal{A}][\mathcal{F}](\text{reading} \wedge \langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\text{research} \rightarrow \neg\ \text{third-party})$$

Note that these formulas do not consider the semantics of the *biz-hours* or *third-party* propositions; but, given that the evaluation of this proposition is otherwise taken care of, they regulate the relationships between purpose-based rules and other types of constraints in a system. We leave further exploration of this extension as a future work.

6.2 Setting Distance-Based Policies

Since the granularity of an action graph depends on how the system is designed, distance-based policies (recall Section 4.1.5) are very system-dependent. For example, a maximum distance of three may be very far in one system and very close in another, based on the granularity of the action graph. Therefore, the policy maker should be aware of the internal structure of the system to build meaningful distance-based policies. This can be true in case of organizational policies where the policy maker is aware of the details of the organization and the systems in use, but in cases such as the ethics consent in healthcare, the policy maker is an outsider to the organization and such policies cannot be meaningfully settled.

Distance can also be defined between two purposes and make the ground for a new type of policy. For example, one scenario may require that the *surgery* purpose be within 3 steps of the *treatment* purpose. We did not discuss such policies here, as we could not imagine a pragmatic case for them.

6.3 Purpose-Based Obligations

Obligations are commitments that should be incurred after an agent performs an action [12]. Currently our model only considers allowing or blocking an action based on the purpose-based policy. A further step would be to enable assigning obligations to agents as a result of performing an action. For example, the billing system in a hospital may charge a patient's or insurer's account with different amounts, depending whether the *surgery* is for the purpose of *treatment* or *cosmetics.* Or, different fees may be charged when a movie is played for the purpose of *entertainment* or *education.* We did not discuss purpose-based obligations in this paper but believe it is an interesting topic of future work.

6.4 Obligations and the F-Relationship

Suppose action a is related to a_F by an F- relationship. At run-time, if the reference monitor allows an agent to perform a based on a purpose-based policy that relies on the performance of a_F, it can be said that it is under the condition that the action a_F is performed in future. In fact, if a_F is not performed the policy would be violated. This matches the classic notion of obligation and the reference monitor can issue an obligation in such cases.

This observation points out the relationship between obligation policies and purpose-based policies which is another interesting topic for future work.

6.5 Use of First-Order Predicate Logic

The logic we have built for formalizing purposes is a modal logic over a propositional logic. However, using propositional logic has a scalability problem. For example, if there are one million health records in a system, atomic propositions should be used to express the policy regarding each of them which will make the system very complex. In other words, lack of quantifiers and predicates may lead to scalability issues in formalizing policies. On the other hand, first-order predicate will make the verification of policies undecidable in the general case. One direction of future work could be to extend the logic to first-order logic and then study the conditions under which an efficient algorithm for policy evaluation can be developed.

6.6 Multiple and Nested Purposes

The current literature generally assumes that a single purpose is associated with an action. In practice, however, a single action may have multiple purposes that are independent or tangentially related. For example, the purpose of reading a book may be both to enjoy the literature and to prepare for an exam. In turn, the purpose of preparing for the exam may be to pass the course and subsequently to get a degree. One can observe that a single action may be associated with a chain of multiple nested purposes and/or multiple independent purposes. Capturing this broader aspect of purposes is one contribution of our model.

6.7 Simplified Version of the Logic

If we simplify the logic to only contain the A–relationships, the action graph will be reduced to a directed tree. In such a tree, the purposes of an action can be captured as a *chain* of *nested purposes* and a simplified version of our logic, resembling a simple temporal logic, would be adequate to express the restrictions on such a chain. This simpler case of the model allows the chain of nested purposes to be associated with the stack context in a programming language, or a workflow as it shows the hierarchy of nested actions each of which is being performed as part of the other. Therefore, a reference monitor can be implemented very straightforwardly by watching the stack. This result has not appeared in the literature yet but is available through our tech report series [14].

6.8 Other Types of Relationships

In our current model of action graph, we only consider two types of relationships between actions. These two have obviously an implication of purpose as we discussed in Section 1.1.1. Other types of relationships such as temporal precedence or causality exist between actions and they may also be captured in an extended model of the action graph. Deciding whether or not to include other types of relationships in the action graph and how they may or may not imply a purpose is a topic of future work.

7. CONCLUSION

We developed a framework for expressing and enforcing purpose-based privacy policies. We defined the semantics of an action's purpose in terms of its situation among other related actions. A modal logic was developed to enable expressing restrictions about the relationships among actions; thereby expressing purpose-based policies. Finally, a model checking algorithm was described to check the compliance of a system with such policies. We also showed how our framework is capable of formalizing common purpose-based policy rules and also introduced some new types of such rules. The main future work is to evaluate the model by studying a practical case of formalizing and enforcing purpose-based policies in a business system.

8. REFERENCES

[1] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools, 2nd Edition*. Addison-Wesley, 2006.

[2] C. A. Ardagna, S. De Capitani di Vimercati, and P. Samarati. Enhancing user privacy through data handling policies. In *Data and Applications Security*, pages 224–236, Sophia Antipolis, France, 2006.

[3] C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.

[4] T. D. Breaux and A. I. Antón. Deriving semantic models from privacy policies. In *IEEE POLICY'05*, pages 67–76, Stockholm, Sweden.

[5] J.-W. Byun, E. Bertino, and N. Li. Purpose based access control of complex data for privacy protection. In *SACMAT '05: Proceedings of the tenth ACM symposium on Access control models and technologies*, pages 102–110, New York, NY, USA, 2005. ACM.

[6] W. Cheung and Y. Gil. Towards privacy aware data analysis workflows for e-science. In *Proceedings of the 2007 Workshop on Semantic e-Science (SeS2007), Vancouver, Canada*, pages 17–25, July 2007.

[7] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs, Workshop*, pages 52–71, London, UK, 1982. Springer-Verlag.

[8] L. L. Dimitropoulos. Privacy and Security Solutions for Interoperable Health Information Exchange. `http://healthit.ahrq.gov/portal/server.pt/gateway/PTARGS_0_241358_0_0_18/IAVR_ExecSumm.pdf`, 2006.

[9] S. Fischer-Hübner. *IT-Security and Privacy: Design and Use of Privacy-Enhancing Security Mechanisms*. Springer, Berlin, Germany, 2001.

[10] Q. He. Privacy enforcement with an extended role-based access control model. Technical Report TR-2003-09, North Carolina State University, 2003.

[11] Health Level Seven Inc. HL7 Reference Information Model, ANSI/HL7 V3 RIM, R1-2003, 2003.

[12] M. Hilty, D. Basin, and A. Pretschner. On obligations. In *ESORICS 2005: Proceedings of the 10th European Symposium On Research in Computer Security.*

[13] K. Irwin, T. Yu, and W. H. Winsborough. On the modeling and analysis of obligations. In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, pages 134–143, Alexandria, Virginia, USA, 2006.

[14] M. Jafari. Nested purposes. Technical report, (unpublished), December 2009.

[15] M. Jafari, R. Safavi-Naini, and N. P. Sheppard. Enforcing purpose of use via workflows. In *WPES '09: Proceedings of the 8th ACM workshop on Privacy in the electronic society*, pages 113–116, 2009.

[16] M. Jawad, P. S. Alvaredo, and P. Valduriez. Design of PriServ, a privacy service for DHTs. In *International Workshop on Privacy and Anonymity in the Information Society*, pages 21–26, Nantes, France, 2008.

[17] T. Jensen, D. Le Metayer, and T. Thorn. Verification of control flow based security properties. pages 89 –103, Oakland, CA, USA, May 1999.

[18] 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.

[19] Q. Ni, E. Bertino, J. Lobo, and S. B. Calo. Privacy-aware role-based access control. *IEEE Security and Privacy*, 7(4):35–43, 2009.

[20] Organisation for the Advancement of Structured Information Standards. Privacy policy profile of XACML v2.0. `http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-privacy_profile-spec-os.pdf`, 2005.

[21] Organisation for the Advancement of Structured Information Standards. Cross-Enterprise Security and Privacy Authorization (XSPA) Profile of Security Assertion Markup Language (SAML) for Healthcare Version 1.0, 2009.

[22] I. H. T. S. D. Organization. SNOMED CT, Systematized Nomenclature of Medicine-Clinical Terms. `http://www.ihtsdo.org/snomed-ct/`.

[23] C. S. Powers, P. Ashley, and M. Schunter. Privacy promises, access control, and privacy management. In *ISEC '02: Proceedings of the Third International Symposium on Electronic Commerce*, pages 13–21, Research Triangle Park, North Carolina, US, 2002. IEEE Computer Society.

[24] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach (3rd Edition)*. Prentice Hall, 2009.

[25] M. Schunter and C. Powers. The Enterprise Privacy Authorization Language (EPAL 1.1). `http://www.zurich.ibm.com/security/enterprise-privacy/epal`, 2003.

[26] W. van Staden and M. S. Olivier. Purpose organisation. In *ISSA2005: Proceedings of the Fifth Annual Information Security South Africa Conference*, Sandton, South Africa, 2005.

[27] World-Wide Web Consortium. The Platform for Privacy Preferences 1.1 (P3P1.1) Specification, 2006.

[28] M. Yasuda, T. Tachikawa, and M. Takizawa. A purpose-oriented access control model. In *Proceedings of Twelfth International Conference on Information Networking*, pages 168–173, Jan. 1998.

APPENDIX

A. PROOFS

THEOREM 1. *$uu' \in F^* \Rightarrow \exists v\ \{uv, u'v\} \subseteq A$ where F^* is the transitive closure of F.*

PROOF. $uu' \in F^*$ implies either $uu' \in F$, in which case $\exists v\ \{uv, u'v\} \subseteq A$ holds based on the property (c), or there exist $u_1, ..., u_n$, so that:

$$uu_1 \in F \land u_1u_2 \in F \land \ldots \land u_nu' \in F$$

Applying property (c) for each of the terms yields
$\exists v_1\ \{uv_1, u_1v_1\} \subseteq A$, $\exists v_2\ \{u_1v_2, u_2v_2\} \subseteq A$, and so on.

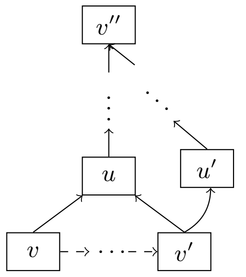


Figure 7: Proof of Lemma 1.

And since according to the property (b) (V, A) is a tree, $u_1v_1 \in A \land u_1v_2$ implies $v_1 = v_2$. Similarly it follows that all v_i's are the same and hence Q.E.D. □

LEMMA 1. *The following is a valid formula:*

$$\langle\mathcal{F}\rangle\langle\mathcal{A}\rangle\phi \leftrightarrow \langle\mathcal{F}\rangle\phi \lor \langle\mathcal{A}\rangle\phi$$

PROOF. Assuming that v satisfies the left-hand side, and based on the definition of $\langle\mathcal{F}\rangle$, it follows that $\exists v' \in V.vv' \in F^* \land v' \models \langle\mathcal{A}\rangle\phi$. Extending the definition of $\langle\mathcal{A}\rangle$, it follows that:
$\exists v' \in V.\exists v'' \in V.vv' \in F^* \land v'v'' \in A^* \land v'' \models \phi$
If $v' = v''$ it follows that $\exists v' \in V.vv' \in F^* \land v' \models \phi$ which in turn leads to: $v \models \langle\mathcal{F}\rangle\phi$.

Now suppose that $v' \neq v''$. The fact that $vv' \in F^*$ implies that $\exists u.\{vu, v'u\} \subseteq A$ according the Theorem 1. On the other hand, $v'v'' \in A^*$ implies one of the following cases when $v' \neq v''$: (A.1) $v'v'' \in A$
(A.2) $\exists v', v'u' \in A \land u'v'' \in A^*$.

In case (A.1), $v'v'' \in A$, and since we already had $v'u \in A$ it follows that $u = v''$ as (V, A) is a tree. Then, since $v'' \models \phi$, it follows that $u \models \phi$, and as $vu \in A$, it follows that $v \models \langle\mathcal{A}\rangle\phi$ which implies the right-hand side.

Now consider Figure 7 for case (A.2). since we have already proved that $v'u \in A$, the fact that $v'u' \in A$ implies $u = u'$ as (V, A) is a tree, which leads to the conclusion that $vv'' \in A^*$, since we already proved that $vu \in A$ above, and have $u'v'' \in A^*$ from the assumption of case (A.2). And as $v'' \models \phi$, we have $v \models \langle\mathcal{A}\rangle\phi$ which implies the right-hand side.

Proving the converse is easy since it straightforwardly follows from the semantic definitions that each of the terms in the right-hand side imply the left-hand side. □

LEMMA 2. *The box and diamond operators are idempotent, i.e. the following are valid formulas for all $n \in \mathbb{N}$:*

$$\langle\mathcal{F}\rangle^n\phi \leftrightarrow \langle\mathcal{F}\rangle\phi$$
$$\langle\mathcal{A}\rangle^n\phi \leftrightarrow \langle\mathcal{A}\rangle\phi$$
$$[\mathcal{F}]^n\phi \leftrightarrow [\mathcal{F}]\phi$$
$$[\mathcal{A}]^n\phi \leftrightarrow [\mathcal{A}]\phi$$

PROOF. For the diamond operators, we only prove the case for $\langle\mathcal{A}\rangle$ as the proof for $\langle\mathcal{F}\rangle$ is straightforwardly similar. Also, we only need to prove the case $\langle\mathcal{A}\rangle\langle\mathcal{A}\rangle\phi \leftrightarrow \langle\mathcal{A}\rangle\phi$ and the general case for n follows inductively.

According to the definition of $\langle\mathcal{A}\rangle$, $v \models \langle\mathcal{A}\rangle\langle\mathcal{A}\rangle\phi$ implies $\exists v' \in V.vv' \in A^* \land \exists v'' \in v.v'v'' \in A^* \land v'' \models \phi$. But then, it follows by transitivity that $vv'' \in A^*$. Therefore, $v \models \langle\mathcal{A}\rangle\phi$.

Now, suppose $v \models \langle\mathcal{A}\rangle\phi$. Then, $\exists v' \in A. vv' \in A^* \wedge v' \in \phi$. Note that $vv \in A^*$. Thus, $v \models \langle\mathcal{A}\rangle\langle\mathcal{A}\rangle\phi$.

For the box operators, we can prove that $[\mathcal{F}]^2\phi \equiv [\mathcal{F}]\phi$ as follows, using the first part of the theorem:

$$\begin{aligned}[\mathcal{F}][\mathcal{F}]\phi &\leftrightarrow \neg\neg[\mathcal{F}][\mathcal{F}]\phi \leftrightarrow \neg\langle\mathcal{F}\rangle\neg[\mathcal{F}]\phi \leftrightarrow \neg\langle\mathcal{F}\rangle\langle\mathcal{F}\rangle\neg\phi \\ &\leftrightarrow \neg\langle\mathcal{F}\rangle\neg\phi \leftrightarrow [\mathcal{F}]\phi.\end{aligned}$$

□

Theorem 2. *Any formula of the form $\langle *\rangle ... \langle *\rangle\phi$ in which $*$ can be either of $\mathcal{A}$ or $\mathcal{F}$ is equivalent to:*

- $\langle\mathcal{A}\rangle\phi$ *if it has the form* $\langle\mathcal{A}\rangle^n\phi$,
- $\langle\mathcal{F}\rangle\phi$ *if it has the form* $\langle\mathcal{F}\rangle^n\phi$,
- $\langle\mathcal{F}\rangle\langle\mathcal{A}\rangle\phi$ *if it has the form* $\langle\mathcal{F}\rangle^n\langle\mathcal{A}\rangle^m\phi$ $(m, n \in \mathbb{N})$,
- *and* $\langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\phi$ *otherwise.*

Similarly, $[*]...[*]\phi$ *in which* $*$ *can be either of* $\mathcal{A}$ *or* $\mathcal{F}$, *is equivalent to:*

- $[\mathcal{A}]\phi$ *if it has the form* $[\mathcal{A}]^n\phi$,
- $[\mathcal{F}]\phi$ *if it has the form* $[\mathcal{F}]^n\phi$,
- $[\mathcal{F}][\mathcal{A}]\phi$ *if it has the form* $[\mathcal{F}]^n[\mathcal{A}]^m\phi$ $(m, n \in \mathbb{N})$,
- *and* $[\mathcal{A}][\mathcal{F}]\phi$ *otherwise.*

Proof. We begin by the first part about diamond operators. Applying Lemma 2 to remove all consecutive repetitions of $\langle\mathcal{A}\rangle$ and $\langle\mathcal{F}\rangle$ will reduce the formula either to one of the trivial cases, or to one of the following forms ($n \geq 2$):
(A.1)$(\langle\mathcal{A}\rangle\langle\mathcal{F}\rangle)^n\phi$
(A.2)$(\langle\mathcal{A}\rangle\langle\mathcal{F}\rangle)^n\langle\mathcal{A}\rangle\phi$
(A.3)$(\langle\mathcal{F}\rangle\langle\mathcal{A}\rangle)^n\phi$
(A.4)$(\langle\mathcal{F}\rangle\langle\mathcal{A}\rangle)^n\langle\mathcal{F}\rangle\phi$

Case (A.1):

$$\begin{aligned}(\langle\mathcal{A}\rangle\langle\mathcal{F}\rangle)^n\phi &\leftrightarrow \langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\langle\mathcal{A}\rangle\langle\mathcal{F}\rangle(\langle\mathcal{A}\rangle\langle\mathcal{F}\rangle)^{n-2}\phi \\ &\leftrightarrow \langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\langle\mathcal{A}\rangle\phi'\end{aligned}$$

where $\phi' = \langle\mathcal{F}\rangle(\langle\mathcal{A}\rangle\langle\mathcal{F}\rangle)^{n-2}\phi$. Applying the Lemma 1 we will have $\langle\mathcal{A}\rangle(\langle\mathcal{F}\rangle\phi' \vee \langle\mathcal{A}\rangle\phi')$ and since diamond operators can be distributed over disjunctions, it follows that $\langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\phi' \vee \langle\mathcal{A}\rangle\langle\mathcal{A}\rangle\phi'$ and then $\langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\phi' \vee \langle\mathcal{A}\rangle\phi'$. But this yields to $\langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\phi'$ since $\langle\mathcal{A}\rangle\phi' \rightarrow \langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\phi'$. Replacing ϕ' and applying Lemma 2 we will have:

$$\begin{aligned}\langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\langle\mathcal{F}\rangle(\langle\mathcal{A}\rangle\langle\mathcal{F}\rangle)^{n-2}\phi &\leftrightarrow \langle\mathcal{A}\rangle\langle\mathcal{F}\rangle(\langle\mathcal{A}\rangle\langle\mathcal{F}\rangle)^{n-2}\phi \\ &\leftrightarrow (\langle\mathcal{A}\rangle\langle\mathcal{F}\rangle)^{n-1}\phi\end{aligned}$$

This will recursively lead to $\langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\phi$.

Case (A.2): Applying case (A.1) we will get $\langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\langle\mathcal{A}\rangle\phi$ and then by applying Lemma 1:

$$\begin{aligned}\langle\mathcal{A}\rangle(\langle\mathcal{F}\rangle\phi \vee \langle\mathcal{A}\rangle\phi) &\leftrightarrow \langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\phi \vee \langle\mathcal{A}\rangle\langle\mathcal{A}\rangle\phi \\ &\leftrightarrow \langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\phi \vee \langle\mathcal{A}\rangle\phi\end{aligned}$$

And since $\langle\mathcal{A}\rangle\phi \rightarrow \langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\phi$, if will follow that $\langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\phi$.

Case (A.3): Rewriting the formula as $\langle\mathcal{F}\rangle(\langle\mathcal{A}\rangle\langle\mathcal{F}\rangle)^{n-1}\langle\mathcal{A}\rangle\phi$ and applying the result of case (A.1) we will get to $\langle\mathcal{F}\rangle\langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\langle\mathcal{A}\rangle\phi$. And then using Lemma 1:

$$\begin{aligned}\langle\mathcal{F}\rangle\langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\langle\mathcal{A}\rangle\phi &\leftrightarrow \langle\mathcal{F}\rangle\langle\mathcal{F}\rangle\langle\mathcal{A}\rangle\phi \vee \langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\langle\mathcal{A}\rangle\phi \\ &\leftrightarrow \langle\mathcal{F}\rangle\langle\mathcal{A}\rangle\phi \vee \langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\langle\mathcal{A}\rangle\phi \\ &\leftrightarrow \langle\mathcal{F}\rangle\phi \vee \langle\mathcal{A}\rangle\phi \vee \langle\mathcal{A}\rangle(\langle\mathcal{F}\rangle\phi \vee \langle\mathcal{A}\rangle\phi) \\ &\leftrightarrow \langle\mathcal{F}\rangle\phi \vee \langle\mathcal{A}\rangle\phi \vee \langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\phi \vee \langle\mathcal{A}\rangle\langle\mathcal{A}\rangle\phi \\ &\leftrightarrow \langle\mathcal{F}\rangle\phi \vee \langle\mathcal{A}\rangle\phi \vee \langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\phi\end{aligned}$$

And since $\langle\mathcal{F}\rangle\phi \rightarrow \langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\phi$ and $\langle\mathcal{A}\rangle\phi \rightarrow \langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\phi$ the formula can be simplified to $\langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\phi$.

Case (A.4): Using the result of case (A.3), it follows that $\langle\mathcal{F}\rangle\langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\phi$ and then $\langle\mathcal{F}\rangle\langle\mathcal{F}\rangle\phi \vee \langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\phi$, and again, since $\langle\mathcal{F}\rangle\phi \rightarrow \langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\phi$ this is equivalent to $\langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\phi$.

The second part, for box operators can be proved straightforwardly by applying the result of the first part:

$$\begin{aligned}[*]...[*]\phi &\leftrightarrow \neg\neg[*]...[*]\phi \leftrightarrow \neg\langle *\rangle...\langle *\rangle\neg\phi \\ &\leftrightarrow \neg\langle\mathcal{A}\rangle\langle\mathcal{F}\rangle\neg\phi \leftrightarrow [\mathcal{A}][\mathcal{F}]\phi\end{aligned}$$

□