A first-order policy language for history-based transaction monitoring

Andreas Bauer, Rajeev Goré, and Alwen Tiu

Logic and Computation Group, The Australian National University

Abstract. Online trading invariably involves dealings between strangers, so it is important for one party to be able to judge objectively the trustworthiness of the other. In such a setting, the decision to trust a user may sensibly be based on that user's past behaviour. We introduce a specification language based on linear temporal logic for expressing a *policy* for categorising the behaviour patterns of a user depending on its transaction history. We also present an algorithm for checking whether the transaction history obeys the stated policy. To be useful in a real setting, such a language should allow one to express realistic policies which may involve parameter quantification and quantitative or statistical patterns. We introduce several extensions of linear temporal logic to cater for such needs: a restricted form of universal and existential quantification; arbitrary computable functions and relations in the term language; and a "counting" quantifier for counting how many times a formula holds in the past. We then show that model checking a transaction history against a policy, which we call the history-based transaction monitoring problem, is PSPACE-complete in the size of the policy formula and the length of the history, assuming that the underlying interpreted functions and relations are polynomially computable. The problem becomes decidable in polynomial time when the policies are fixed. We also consider the problem of transaction monitoring in the case where not all the parameters of actions are observable. We formulate two such "partial observability" monitoring problems, and show their decidability under certain restrictions.

1 Introduction

Internet mediated trading is now a common way of exchanging goods and services between parties who may not have engaged in transactions with each other before. The decision of a seller/buyer to engage in a transaction is usually based on the "reputation" of the other party, which is often provided via the online trading system itself. These so-called *reputation systems* can take the form of numerical ratings, which can be computed based on feedback from users (cf. [9] for a survey of reputation systems). While many reputation systems used in practice seem to serve their purposes, they are not without problems (cf. [9]) and can be too simplistic in some cases. For example, in eBay.com, the rating of a seller/buyer consists of two components: the number of positive feedbacks she gets, and the number of negative feedbacks. A seller with, say 90 positive

feedbacks and 1 negative feedback may be considered trustworthy by some. But one may want to correlate a feedback with the monetary value of the transaction by checking if the one negative feedback was for a very expensive item, or one may want to check other more general relations between different parameters of past transactions.

Here, we consider an alternative (and complementary) method to describe the reputation of a seller/buyer, by specifying explicitly what constitutes a "good" and a "bad" seller/buyer based on the observed patterns of past transactions. More specifically, we introduce a formal language based on linear temporal logic for encoding the desired patterns of behaviours, and a mechanism for checking these patterns against a concrete history of transactions. The latter is often referred to as the *monitoring problem* since the behaviour of users is being monitored, but here, it is just a specific instance of model checking for temporal logic. The patterns of behaviours, described in the logical language, serve as a concise description of the policies for the user on whether to engage with a particular seller/buyer. The approach we follow here is essentially an instance of *history-based access control* (see e.g., [6, 8, 7, 2, 11, 3]). More precisely, our work is closely related to that of Krukow et al. [11, 12].

There are two main ideas underlying the design of our language:

Transactions vs. individual actions: Following Krukow et al., we are mainly interested in expressing properties about transactions seen as a logically connected grouping of actions, for example because they may represent a run of a protocol. A history in our setting is a list of such transactions. This is in contrast to the more traditional notion of history as a list of individual actions (i.e., a trace), e.g., as in [6, 8], which is common in monitoring program execution.

Closed world assumption: The main idea underlying the design of our quantified policies is that a policy should only express properties of objects which are observed in the history. For example, in monitoring a typical online transaction, it makes sense to talk about properties that involve "all the payments that have been made". Thus, if we consider a formalisation of events using predicates, where $pay(100)$ denotes the payment of 100 dollars (say), then we can specify a policy like the one below left which states that all payments must obey ψ:

$$\forall x.\ pay(x) \rightarrow \psi(x) \qquad\qquad \forall x.\ \neg pay(x) \rightarrow \psi(x)$$

However, it makes less sense to talk about "for all dollar amounts that a seller did not pay", like the policy above right, since this involves infinitely many possibility (e.g., the seller paid 100, but did not pay 110, did not pay 111, etc.). We therefore restrict our quantification in policies to have a "positive guard", guaranteeing that we always quantify over the finitely many values that have already been observed in the history.

An important consequence of the closed world assumption is that we can only describe relations between known individual objects. Thus we can enrich our

logical language with computable functions over these objects and computable relations between these objects without losing decidability of the model checking problem. One such useful extension is arithmetic, which allows one to describe constraints on various quantities and values of transactions.

Our base language for describing policies is the pure past fragment of linear temporal logic [14] since it has been used quite extensively by others [15, 8, 11, 3] for similar purposes. However, the following points distinguish our work from related work in the literature, within the context of history-based access control:

- We believe our work is the first to incorporate both quantified policies and computable functions/relations within the same logic. Combining unrestricted quantifiers with arbitrary computable functions easily leads to undecidability (see Section 7).
- We extend temporal logic with a "counting quantifier", which counts how many times a policy has been satisfied in the past. A similar counting mechanism was proposed in [11, 12] as a part of a meta-policy language. But in our work, it is a part of the same logic.
- We consider new monitoring problems based on a notion of *partial observability* which seem to arise quite naturally in online trading platforms where a user (or a system provider) cannot directly observe all parameters of an action. For instance, in eBay, it may not be always possible to observe whether payments have been made, or it may be possible to observe a payment but not the exact amount paid. We model unobservable parameters in an action as variables representing unknown values. Given a policy and a history containing unknown parameters, we ask whether the policy is satisfied under some substitution of the variables (the *potential satisfiability problem*), or under *all* substitutions (the *adherence problem*).

The rest of the paper is organised as follows. Section 2 introduces our policy language $PTLTL^{FO}$, for "past time linear temporal logic with first-order (guarded) quantifiers", and defines its semantics. Section 3 presents some examples using $PTLTL^{FO}$ for specifying access control policies. Two examples are formalisations of known security policies, which are trace-based in the sense that the histories are just traces, and that go beyond the scope of online trading systems alone. The third example shows a transaction-based policy as it can be used for eBay.com type of systems. Section 4 considers the model checking problem for $PTLTL^{FO}$ which we show to be PSPACE-complete. Fixing the policies reduces the complexity to PTIME. Section 5 presents an extension of $PTLTL^{FO}$ with a counting quantifier allowing us to express that a policy depends on the number of times another policy was satisfied in the past. The model checking problem for this extension remains PSPACE-complete. In Section 6, we consider more general (undecidable) monitoring problems where not all the parameters of an action can be observed. By restricting the class of allowed functions and relations, we can obtain decidability of both the potential satisfiability and adherence problems, for example, when the term language of the logic is restricted to linear arithmetic. Section 7 discusses possible decidable extensions to the guarded quantifiers. Section 8 concludes the paper and discusses related work.

2 The policy language: definitions and notation

Since we are interested in the notion of history-based access control, our definition of history is a simplification of that of [12]. A history is organised as a list of sessions. Each session is a finite set of events, or actions. Each event is represented by a predicate. A session represents a "world" in the sense of a Kripke semantics where the underlying frame is linear and discrete.

The term structures of our policy language are made up of variables and interpreted multi-sorted function symbols. Function symbols of zero arity are called *constants*. Terms are ranged over by s, t, u. Variables of the language, denoted by x, y, z, range over certain domains, such as strings, integers, or other finite domains. We call these domains *base types* or simply *types*. We assume a distinguished type *prop* which denotes the set of propositions of the logic, and which must not be used in the types of the function symbols and variables. That is, we do not allow logical formulae to appear at the term level. Function symbols and variables are typed.

We assume an interpretation where distinct constants of the same type map to distinct elements of the type. We shall use the same symbol, say a, to refer both to an element of some type τ and the constant representing this element. Function symbols of one or more arities admit a fixed interpretation, which can be any total recursive function. We shall assume the usual function symbols for arithmetic, $+$, $-$, $\times$, etc., with the standard interpretations. The language we are about to define is open to additional interpreted function symbols, e.g., string related operations, etc. We shall use f, g, h to range over function symbols of arity one or more, and a, b, c, d to range over constants. We also assume a set of interpreted relations, in particular, those for arithmetic, e.g., $<$, $=$, $\geq$, etc. These interpreted relations are ranged over by R. All the interpreted functions and relations have first-order types, i.e., their types are of the form $\tau_1 \times \cdots \times \tau_n \rightarrow \tau$, where τ and $\tau_1, \ldots, \tau_n$ are base types. We shall restrict to computable relations R. Of course, there is also the (rigidity) assumption that the function f, constant c and relation R have the same fixed interpretation over all worlds.

Since our term language contains interpreted symbols, we assume that there is a procedure for evaluating terms into values. We also assume that each term can be evaluated to a unique value. Given a term t, we shall denote with $t\downarrow$ the unique value denoted by this term, e.g., if $t = (2{+}3)$ then $t\downarrow = 5$. Given an atomic formula $p(t_1, \ldots, t_n)$, we shall write $p(t_1, \ldots, t_n)\downarrow$ to denote $p(t_1\downarrow, \ldots, t_n\downarrow)$. The policy language is given by the following grammar:

$$\psi ::= p(t_1, \ldots, t_m) \mid R(t_1, \ldots, t_n) \mid \psi \wedge \psi \mid \neg\psi \mid \mathbf{X}^{-1}\psi \mid \psi\, \mathbf{S}\, \psi \mid \forall(x_1, \ldots, x_n) : p.\, \psi.$$

In the quantified formula $\forall(x_1, \ldots, x_n) : p.\ \psi$, where $n \geq 1$, the symbol p is an n-ary predicate of type $\tau_1 \times \cdots \times \tau_n \rightarrow \mathit{prop}$, and each x_i is of type τ_i. The intended interpretation of this quantification is that the predicate p defines a subtype of $\tau_1 \times \cdots \times \tau_n$, which is determined by the occurrence of p in the world (session) in which the formula resides. For example, in a world consisting of $\{p(1,1), p(1,2), p(1,3), q(4)\}$ the predicate p represents the set $\{(1,1), (1,2), (1,3)\}$,

$(h,i) \models p(t_1,\ldots,t_n)$ iff $p(t_1\downarrow,\ldots,t_n\downarrow) \in h_i$
$(h,i) \models R(t_1,\ldots,t_n)$ iff $R(t_1\downarrow,\ldots,t_n\downarrow)$ is true
$(h,i) \models \psi_1 \wedge \psi_2$ iff $(h,i) \models \psi_1$ and $(h,i) \models \psi_2$
$(h,i) \models \neg\psi$ iff $(h,i) \not\models \psi$
$(h,i) \models \mathbf{X}^{-1}\psi$ iff $i > 1$ and $(h,i-1) \models \psi$
$(h,i) \models \psi_1 \,\mathbf{S}\, \psi_2$ iff there exists $j \leq i$ such that $(h,j) \models \psi_2$ and
for all k, if $j < k \leq i$ then $(h,k) \models \psi_1$
$(h,i) \models \forall(x_1,\ldots,x_n) : p.\ \psi$ iff for all $c_1,\ldots,c_n$, if $p(c_1,\ldots,c_n) \in h_i$
then $(h,i) \models \psi[x_1 := c_1,\ldots,x_n := c_n]$.

Fig. 1. Semantics of $PTLTL^{FO}$

i.e., a subset of $N \times N$. We shall often abbreviate $\forall(x_1,\ldots,x_n) : p.\ \psi$ as simply $\forall\vec{x} : p.\ \psi$ when the exact arity and the information about each x_i is not important or can be inferred from context. The notions of free and bound variables are defined as usual. A formula is *closed* if it has no occurrences of free variables.

Definition 1. *An* event *(or an* action*) is a predicate $p(c_1,\ldots,c_n)$ where each c_i is a constant and p is an uninterpreted predicate symbol. A* session *is a finite set of events. A* history *is a finite list of sessions.*

A standard definition for the semantics of first-order logic uses a mapping of free variables in a formula to elements of the types of the variables. To simplify the semantics, we shall consider only closed formulae. The semantics for quantified statements is then defined by closing these statements under variable mappings. We use the notation σ and θ to range over partial maps from variables to elements of types. We usually enumerate them as, e.g., $[x_1 := a_1,\ldots,x_n := a_n]$. Since we identify a constant with the element represented by that constant, a variable mapping is both a semantic and a syntactic concept. The latter means that we can view a variable mapping as a substitution. Given a formula ψ and variable mapping σ, we write $\psi\sigma$ to denote a formula resulting from replacing each free variable x in ψ with the constant $\sigma(x)$. From now on, we shall use the term variable mapping and substitution interchangeably.

The semantic judgement that we are interested in is of the form $(h,i) \models \psi$, where h is a history, i is an index referring to the i-th session in h, and ψ is a closed formula. The judgement reads "ψ is true at the i-th world in the history h". We denote with $|h|$ the length of h, and with h_i the i-th element of h when $i \leq |h|$.

Definition 2. *The* forcing relation *$(h,i) \models \psi$, where h is a history, i an integer, and ψ a formula, is defined inductively as shown in Figure 1 where $1 \leq i \leq |h|$. We denote with $h \models \psi$ the relation $(h,|h|) \models \psi$. The boolean connectives $\vee$ (disjunction) and $\rightarrow$ (implication) are defined in the standard way using negation and conjunction. We derive the operators $\mathbf{F}^{-1}\varphi \equiv \top\,\mathbf{S}\,\varphi$ ("sometime in the past"), and $\mathbf{G}^{-1}\varphi \equiv \neg\mathbf{F}^{-1}(\neg\varphi)$ ("always in the past"), where $\top$ ("true") is short for $p \vee \neg p$.*

Note that allowing unrestricted quantifiers can cause model checking to become undecidable, depending on the interpreted functions and relations. For example, if we allow arbitrary arithmetic expressions in the term language, then we can express solvability of Diophantine equations, which is undecidable [13, Chapter 5].

3 Some example policies

Let us now examine some example policies known from the literature, and our means of expressing them concisely and accurately. We also examine some policies from applications other than monitoring users in online trading systems to demonstrate that our language can model the requirements of other related domains as well if they can be expressed as trace-based properties.

One-out-of-k policy. The *one-out-of-k policy* as described in [6] concerns the monitoring of web-based applications. More specifically, it concerns monitoring three specific situations: connection to a remote site, opening local files, and creating subprocesses. We model this as follows, with the set of events being

$open(file, mode)$: request to open the file $file$ in mode, $mode$, where $file$ is a string containing the absolute path, and $mode$ can be either ro (for read-only) or rw (for read-write). There can be other modes but for simplicity we assume just these two;
$read/write/create(file)$: request to read/write/create a file;
$connect$: request to open a socket (to a site which is irrelevant for now);
$subproc$: request to create a subprocess.

We assume some operators for string manipulation: the function $path(file)$ which returns the absolute path to the directory in which the file resides, and the equality predicate $=$ on strings. The history in this setting is restricted to one in which every session is a singleton set. We now show how to encode one of the policies as described in [6]: allow a program to open local files in user-specified directories for modifications if and only if it has created them, and it has neither tried to connect to a remote site nor tried to create a sub-process. Suppose that we allow only one user-specified directory called "Document". Then this policy can be expressed as:

$$\forall(x, m) : open.m = \text{rw} \rightarrow [\, path(x) = \text{“Document”} \wedge\ \mathbf{F}^{-1}\, create(x) \wedge \neg\mathbf{F}^{-1}\, connect\ \wedge\ \neg\mathbf{F}^{-1}\, subproc].$$

Chinese wall policy. The chinese wall policy [5] is a common access control policy used in financial markets for managing conflicts of interests. In this setting, each object for which access is requested, is classified as belonging to a *company dataset*, which in turn belongs to a *conflict of interest class*. The idea is that a user (or subject) that accessed an object that belonged to a company A in the past will not be allowed to access another object that belongs to a company B which is in the same conflict of interest class as A.

To model this policy, we assume the following finite sets: U for users, O for objects, D for company datasets, and C for the names of the conflict of interest class. The event we shall be concerned with is access of an object o by a user u. We shall assume that this event carries information about the company dataset to which the object belongs, and the name of the conflict of interest class to which the company dataset belongs. That is, *access* is of type $U \times O \times D \times C \to prop$. A history in this case is a sequence of singletons containing the *access* event. The policy, as given in [5], specifies among others that

> "access is only granted if the object requested: 1.) is in the same company dataset as an object already accessed by that subject, or 2.) belongs to an entirely different conflict of interest class."

Implicit in this description is that first access (i.e., no prior history) is always allowed. We can model the case where no prior history exists simply using the formula $\neg \mathbf{X}^{-1} \top$. This policy can be expressed in our language as follows:

$$\begin{aligned} &\forall (u, o, d, c) : access.\ \neg \mathbf{X}^{-1} \top \vee \\ &\quad (\mathbf{X}^{-1} \mathbf{F}^{-1}\, \exists (u', o', d', c') : access.\ u = u' \wedge d = d') \vee \\ &\quad (\mathbf{X}^{-1} \mathbf{G}^{-1}\, \forall (u', o', d', c') : access.\ u = u' \to \neg (c = c')). \end{aligned}$$

eBay.com. In this example, we consider a scenario where a potential buyer wants to engage in a bidding process on an online trading system like eBay.com, but the buyer wants to impose some criteria on what kind of sellers she trusts. A simple policy would be something like "only deal with a seller who was never late in delivery of items". In this model, a session in a history represents a complete exchange between buyer and seller, e.g., the bidding process, winning the bid, payment, confirmation of payment, delivery of items, confirmation of delivery, and the feedbacks. We consider the following events (we are considering the history of a seller):

$win(X, V)$: the bidder won the bid for item X for value V.
$pay(T, X, V)$: payment of item X at date T of the sum V (numerical value of dollars).
$post(X, T)$: the item X is delivered within T days. [1]
negative, *neutral*, *positive*: represents negative, neutral and positive feedbacks.

There are of course other actions and parameters that we can formalise, but these are sufficient for an illustration. Now, suppose the buyer sets a criterion such that a posting delay greater than 10 days after payment is unacceptable. This can be expressed as:

$$\mathbf{G}^{-1}\, [\forall (t, x, v) : pay.\ \exists (y, t') : post.\ x = y \wedge t' \leq 10]. \tag{1}$$

Of course, for such a simple purpose, one can rely on eBay's rating system, which basically computes the number of feedbacks in each category (positive,

[1] Note that in the actual eBay system, no concrete number of days is given, but instead buyers can rate the time for posting and handling in the feedback forums in a range of 1 to 5.

$$(id)\ \frac{\text{if } p(\vec{t})\downarrow \in h_i \text{ then } v := \mathbf{t} \text{ else } v := \mathbf{f}}{\langle h, i, p(\vec{t})\rangle \Downarrow v} \qquad (R)\ \frac{\text{if } R(\vec{t})\downarrow \text{ is true then } v := \mathbf{t} \text{ else } v := \mathbf{f}}{\langle h, i, R(\vec{t})\rangle \Downarrow v}$$

$$(\neg)\ \frac{\langle h, i, \psi\rangle \Downarrow v}{\langle h, i, \neg\psi\rangle \Downarrow \neg v} \qquad (\wedge)\ \frac{\langle h, i, \psi_1\rangle \Downarrow v_1 \quad \langle h, i, \psi_2\rangle \Downarrow v_2}{\langle h, i, \psi_1 \wedge \psi_2\rangle \Downarrow v_1 \wedge v_2}$$

$$(\forall)\ \frac{\langle h, i, \varphi(\vec{t}_1)\rangle \Downarrow v_1 \quad \cdots \quad \langle h, i, \varphi(\vec{t}_n)\rangle \Downarrow v_n}{\langle h, i, \forall \vec{x} : p.\varphi(\vec{x})\rangle \Downarrow \bigwedge_{i=1}^{n} v_i} \quad \text{where } \{\varphi(\vec{t}_1), \cdots, \varphi(\vec{t}_n)\} = \{\varphi(\vec{x}) \mid p(\vec{x}) \in h_i\}$$

$$(\mathbf{S})\ \frac{\langle h, i, \psi_1\rangle \Downarrow v_1 \quad \langle h, i, \psi_2\rangle \Downarrow v_2 \quad \langle h, i-1, \psi_1 \mathbf{S} \psi_2\rangle \Downarrow v_3}{\langle h, i, \psi_1 \mathbf{S} \psi_2\rangle \Downarrow v_2 \vee (v_1 \wedge v_3)}\ i > 1$$

$$(\mathbf{S}_1)\ \frac{\langle h, 1, \psi_2\rangle \Downarrow v}{\langle h, 1, \psi_1 \mathbf{S} \psi_2\rangle \Downarrow v} \qquad (\mathbf{X}^{-1})\ \frac{\langle h, i-1, \varphi\rangle \Downarrow v}{\langle h, i, \mathbf{X}^{-1}\varphi\rangle \Downarrow v}\ i > 1 \qquad (\mathbf{X}_1^{-1})\ \frac{v := \mathbf{f}}{\langle h, 1, \mathbf{X}^{-1}\varphi\rangle \Downarrow v}$$

Fig. 2. Evaluation rules for deciding whether $(h, i) \models \varphi$.

neutral and negative). However, the seller's rating may sometimes be too coarse a description of a seller's reputation. For instance, one is probably willing to trust a seller with some negative feedbacks, as long as those feedbacks refer to transactions involving only small values. A buyer can specify that she would trust a seller who never received negative feedbacks for transactions above a certain value, say, 200 dollars. This can be specified as follows: $\mathbf{G}^{-1}\,[\forall(t, x, v) : \mathit{pay}.\ v \geq 200 \rightarrow \neg \mathit{negative}]$.

4 Model checking $PTLTL^{FO}$

We now consider the model checking problem for $PTLTL^{FO}$, i.e., deciding whether $h \models \varphi$ holds. We show that the model checking problem is PSPACE-complete, even in the case where no interpreted functions or relations occur in the formula.

We prove the complexity of our model checking problem via a terminating recursive algorithm. The algorithm is presented abstractly via a set of rules which successively transform a triple $\langle h, i, \varphi\rangle$ of a history, an index and a formula, and return a truth value of either $\mathbf{t}$ or $\mathbf{f}$ to indicate that $(h, i) \models \varphi$ (resp. $(h, i) \not\models \varphi$). We write $\langle h, i, \varphi\rangle \Downarrow v$ to denote this relation and overload the logical connectives $\wedge$, $\vee$ and $\neg$ to denote operations on boolean values, e.g., $\mathbf{t} \wedge \mathbf{t} = \mathbf{t}$, etc. Since $\psi_1 \mathbf{S} \psi_2 \equiv \psi_2 \vee (\psi_1 \wedge \mathbf{X}^{-1}(\psi_1 \mathbf{S} \psi_2))$, we shall use the following semantic clause for $\psi_1 \mathbf{S} \psi_2$ which is equivalent:

$(h, i) \models \psi_1 \mathbf{S} \psi_2$ iff $(h, i) \models \psi_2$ or $[(h, i) \models \psi_1$ and $i > 1$ and $(h, i-1) \models \psi_1 \mathbf{S} \psi_2]$.

The rules for the evaluation judgement are given in Figure 2. To evaluate the truth value of $\langle h, i, \varphi\rangle$, we start with the judgement $\langle h, i, \varphi\rangle \Downarrow v$ where v is still unknown. We then successively apply the transformation rules bottom up,

according to the main connective of φ and the index i. Each transformation step will create n-child nodes with n unknown values. Only at the base case (i.e., *id*, R, or $\mathbf{X}_1^{-1}$) the value of v is explicitly computed and passed back to the parent nodes. A run of this algorithm can be presented as a tree whose nodes are the evaluation judgements which are related by the transformation rules. A straightforward simultaneous induction on the derivation tree of the evaluation judgements yields:

Lemma 1. *The judgement $\langle h, i, \varphi\rangle \Downarrow \boldsymbol{t}$ is derivable if and only if $(h, i) \models \varphi$ and the judgement $\langle h, i, \varphi\rangle \Downarrow \boldsymbol{f}$ is derivable if and only if $(h, i) \not\models \varphi$.*

Theorem 1. *Let φ be a $PTLTL^{FO}$ formula and h a history. If the interpreted functions and relations in φ are in* PSPACE*, then deciding whether $h \models \varphi$ holds is* PSPACE*-complete.*

Although the model checking problem is PSPACE-complete, in practice, one often has a fixed policy formula which is evaluated against different histories. Then, it makes sense to ask about the complexity of the model checking problem with respect to the size of histories only (while restricting ourselves to interpreted functions and relations computable in polynomial time).

Theorem 2. *The decision problem for $h \models \varphi$, where φ is fixed, is solvable in polynomial time.*

An easy explanation for the above hardness result is via a polynomial time encoding of the PSPACE-complete QBF-problem (cf. [16] and Appendix). Given a boolean expression like $E(x_1, x_2, x_3) \equiv (x_1 \vee \neg x_2) \wedge (\neg x_2 \vee x_3)$ and the QBF-formula $F \equiv \forall x_1.\ \exists x_2.\ \forall x_3.\ E(x_1, x_2, x_3)$, we can construct a corresponding $PTLTL^{FO}$-formula, $\varphi \equiv \forall x_1 : p_1.\ \exists x_2 : p_2.\ \forall x_3 : p_3.\ E'(x_1, x_2, x_3)$ where $E'(x_1, x_2, x_3) \equiv (true(x_1) \vee \neg true(x_2)) \wedge (\neg true(x_2) \vee true(x_3))$, and a history, h below, representing all possible interpretations of F's variables in a single session:

$$h = \{p_1(0), p_1(1), p_2(0), p_2(1), p_3(0), p_3(1), true(1)\}.$$

It is then easy to see that F evaluates to $\top$ if and only if $h \models \varphi$ holds.

On the surface it seems that this "blow up" is caused by the multiple occurrences of the same predicate symbol in a single session. It is therefore natural to ask whether the complexity of the problem can be reduced if we consider histories where every predicate symbol can occur at most once in every session. Surprisingly, however, even with this restriction, model checking remains PSPACE-complete. Consider, for example, the following polynomial encoding of the above QBF-instance, using this restriction:

$$\{p_3(0), true(1)\}; \{p_3(1), true(1)\}; \ldots; \{p_1(0), true(1)\}; \{p_1(1), true(1)\} \models \\ \mathbf{G}^{-1}\, \forall x_1 : p_1.\ \mathbf{F}^{-1} \exists x_2 : p_2.\ \mathbf{G}^{-1}\, \forall x_3 : p_3.\ E'(x_1, x_2, x_3)).$$

Definition 3. *A history h is said to be* trace-like *if for all i such that $1 \leq i \leq |h|$, for all p, $\vec{t}$ and $\vec{s}$, if $p(\vec{t}) \in h_i$ and $p(\vec{s}) \in h_i$, then $\vec{t} = \vec{s}$.*

Theorem 3. *Let φ be a $PTLTL^{FO}$ formula and h a trace-like history. If the interpreted functions and relations in φ are in* PSPACE*, then deciding whether $h \models \varphi$ holds is* PSPACE*-complete.*

Note that we have implemented the above in terms of a prototypic model checker for $PTLTL^{FO}$, which can be freely downloaded and evaluated.[2] The model checker primarily accepts two user inputs: a $PTLTL^{FO}$ policy and a history which is then checked against the policy. We use FOL-RuleML [4] as the input format for the policy since it is due for standardisation as the W3C's first-order logic extension to RuleML [1]. Thus users can even specify policies using graphical XML-editors with a FOL-RuleML DTD extended by our temporal operators.

The model checker is currently not optimised for high performance, but demonstrates the feasibility and practicality of our approach to tackling these problems, as its main algorithm is based directly on the rules in Figure 2. The above web site contains Ocaml source code (as well as a statically linked binary for Linux) and some example policies from Section 3 stored in XML-format.

5 Extending $PTLTL^{FO}$ with a counting quantifier

We now consider an extension of our policy language with a counting quantifier. The idea is that we want to count how many times a policy was satisfied in the past, and use this number to write another policy. The language of formulae is extended with the construct $\mathbf{N}x : \psi.\ \phi(x)$ where x binds over the formula $\phi(x)$ and is not free in ψ. The semantics is as follows:

$$(h,i) \models \mathbf{N}x : \psi.\ \phi(x) \text{ iff } (h,i) \models \phi(n), \text{ where } n = |\{j \mid 1 \leq j \leq i \text{ and } (h,j) \models \psi\}|.$$

Krukow et al. also consider a counting operator, #, which applies to a formula. Intuitively, $\#\psi$ counts the number of sessions in which ψ is true, and can be used inside other arithmetic expressions like $\#\psi \leq 5$. The advantage of our approach is that we can still maintain a total separation of these arithmetic expressions and other underlying computable functions from the logic, thus allowing us to modularly extend these functions. Another notable difference is that our extension resides in the logic itself, instead of a separate "meta" policy language like theirs.

Examples: For example, we show how to state a "meta" policy such as: "engage only with a seller whose past transactions with negative feedbacks constitute at most a quarter of the total transactions". This can be expressed succinctly by the following formula since $\mathbf{N}y : \top$ instantiates y to be the length of the transaction history to date:

$$\mathbf{N}x : \mathit{negative}.\ \mathbf{N}y : \top.\ \frac{x}{y} \leq \frac{1}{4}.$$

[2] see http://code.google.com/p/ptltl-mc/

A more elaborate example is the formula in Equation 1 without the $\mathbf{G}^{-1}$-operator:

$$\psi \equiv \forall(t, x, v) : pay.\ \exists(y, t') : post.\ x = y \wedge t' \leq 10.$$

Then one can specify a policy that demands that "the seller's delivery is *mostly* on-time", where *mostly* can be given as a percentage, such as 90%, via:

$$\mathbf{N}x : \psi.\ \mathbf{N}y : \top.\ \frac{x}{y} \leq 0.9.$$

The proof of the next theorem is a straightforward extension of the proof of Theorem 1.

Theorem 4. *Assuming that the interpreted functions and relations are in* PSPACE*, the model checking problem for $PTLTL^{FO}$ with the counting quantifier is* PSPACE*-complete.*

6 Partial observability

In some online transaction systems, like eBay, certain events may not be wholly observable all the time, even to the system providers, e.g., payments made through a third-party outside the control of the provider. [3] We consider scenarios where some information is missing from the history of a client (buyer or seller) and the problem of enforcing security policies in this setting.

Examples: Consider the policy $\psi \equiv \mathbf{G}^{-1}\,[\forall(x, v) : win.\exists(t, y, u) : pay.x = y \wedge v = u]$ which states that every winning bid must be paid with the agreed dollar amount. The history below, where X represents an unknown amount, can *potentially satisfy* ψ when $X = 100$ (say):

$$\begin{aligned} h = &\{win(a, 100), pay(1, a, 100), post(a, 5)\}; \\ &\{win(a, 100), pay(2, a, X), post(a, 4), positive\} \end{aligned}$$

Of course it is also possible that the actual amount paid is less than 100, in which case the policy is not satisfied. There are also cases in which the values of the unknowns do not matter. For instance, a system provider may not be able to verify payments, but it may deduce that if a buyer leaves a positive remark, that payment has been made. That is, a policy like the following:

$$\varphi' \equiv \mathbf{G}^{-1}\,[\forall(x, v) : win.\exists(t, y, u) : pay.x = y \wedge (u = v \vee positive)]$$

which checks that a payment was made and it was made for exactly the same amount as the winning bid, or the transaction is concluded with a positive feedback (which presumably means everything is fine). In this case, we see that h still satisfies φ' under all substitutions for X.

We consider two problems arising from partial observability. For this, we extend slightly the notion of history and sessions.

[3] eBay asks the user for confirmation of payment, but does not check whether the payment goes through. This is modelled here by an unknown amount in the payment parameters.

Definition 4. *A* partially observable session*, or* po-session *for short, is a finite set of predicates of the form $p(u_1, \ldots, u_n)$, where p is an uninterpreted predicate symbol and each u_i is either a constant or a variable. A* partially observable history *(*po-history*) is a finite list of po-sessions.*

Given a po-history h, we denote with $V(h)$ the set of variables occurring in h. In the following, we shall consider formulae which may have occurrences of free variables. We use the notation $V(\psi)$ to denote the set of free variables in the formula ψ.

Definition 5. *Given a po-history h, a natural number i, and a formula ψ such that $V(\psi) \subseteq V(h)$, we say that h* potentially satisfies *ψ at i, written $(h, i) \vdash \psi$, if there exists a substitution σ such that $dom(\sigma) = V(h)$ and $(h\sigma, i) \models \psi\sigma$. We say that h* adheres to *ψ at i, written $(h, i) \Vdash \psi$, if $(h\sigma, i) \models \psi\sigma$ for all σ such that $dom(\sigma) = V(h)$.*

Notice that the adherence problem is just the dual of the potential satisfiability problem. That is, $(h, i) \Vdash \psi$ if and only if $(h, i) \nvdash \neg\psi$. In general the potential satisfiability problem is undecidable, since one can easily encode solvability of general Diophantine equations, which is known to be undecidable. To see this, let us suppose that the term language of the logic includes standard arithmetic operators (including exponentiation). Then we can express directly any Diophantine equations within our term language. Let us denote with $D(x_1, \ldots, x_n)$ a set of Diophantine equations whose variables are among $x_1, \ldots, x_n$. Assume that we have n uninterpreted unary predicate symbols $p_1, \cdots, p_n$ which take an integer argument. Then solvability of $D(x_1, \ldots, x_n)$ is reducible to the satisfiability problem

$$\{p_1(x_1), \ldots, p_n(x_n)\} \vdash \exists x_1 : p_1. \cdots \exists x_n : p_n. \psi(x_1, \ldots, x_n)$$

where $\psi(x_1, \ldots, x_n)$ is the conjunction of all the equations in $D(x_1, \ldots, x_n)$.

However, we can obtain decidability results if we restrict the term language. We consider here such a restriction where the term language is the language of linear arithmetic over integers, i.e., terms of the form (modulo associativity and commutativity of $+$): $k_1x_1 + \cdots + k_nx_n + c$, where c and each k_i are integers. We also assume the standard relations on integers $=$, $\geq$ and $\leq$. It is useful to introduce a class of *constraint formulae* generated from the following grammar:

$$C ::= \top \mid \bot \mid t_1 = t_2 \mid t_1 \leq t_2 \mid t_1 \geq t_2 \mid C_1 \wedge C_2 \mid C_1 \vee C_2 \mid \neg C.$$

We say that a constraint C is *satisfiable* if there exists a substitution σ such that $C\sigma$ is true. Satisfiability of constraint formulae is decidable (see [10] for a list of algorithms). The decidability proof of the potential satisfiability problem involves a transformation of the judgement $(h, i) \vdash \psi$ into an equivalent constraint formula.

In the following, given a finite set of formula S, we shall write $\bigvee S$ to denote the disjunction of all the formulae in S. If S is the empty set, $\bigvee S$ denotes $\bot$. Likewise, $\bigwedge S$ denotes the conjunction of the formulae in S and when S is empty, it denotes $\top$.

Lemma 2. *For every h, i, and ψ, there exists a constraint formula C such that $(h,i) \vdash \psi$ if and only if C is satisfiable.*

Proof. We construct C by induction on ψ and i. If $i < 1$ or $i > |h|$ then $C = \bot$. Obviously, C is satisfiable iff $(h,i) \vdash \psi$. We show some of the remaining cases here (the other cases are straightforward):

1. If ψ is either $t_1 = t_2$, $t_1 \leq t_2$ or $t_1 \geq t_2$ then $C = \psi$.
2. Suppose $\psi = p(t_1, \ldots, t_n)$. Then
$$C = \bigvee \{u_1 = t_1 \wedge \cdots \wedge u_n = t_n \mid p(u_1, \ldots, u_n) \in h_i\}.$$
3. Suppose $\psi = \psi_1 \,\mathbf{S}\, \psi_2$. By induction hypothesis (on the size of ψ) we have
   (i) C_1 such that $(h,i) \vdash \psi_1$ iff C_1 is satisfiable, and
   (ii) C_2 such that $(h,i) \vdash \psi_2$ iff C_2 is satisfiable.
   If $i = 1$ then let $C = C_2$. Otherwise, $i > 1$ and by induction hypothesis, we have C_3 such that $(h, i-1) \vdash \psi_1 \,\mathbf{S}\, \psi_2$ iff C_3 is satisfiable. In this case, let $C = C_2 \vee (C_1 \wedge C_3)$.
4. Suppose $\psi = \forall(x_1, \ldots, x_n) : p.\psi'(x_1, \ldots, x_n)$. By induction hypothesis, for each tuple $\vec{u} = (u_1, \ldots, u_n)$, we have a $C_{\vec{u}}$ such that $(h,i) \vdash \psi'(u_1, \ldots, u_n)$ iff $C_{\vec{u}}$ is satisfiable. Then $C = \bigwedge \{C_{\vec{u}} \mid p(u_1, \ldots, u_n) \in h_i\}$.

By inspecting the clauses of the forcing relation and the definition of $(h,i) \vdash \psi$, it is straightforward to check that in each case above C is satisfiable if and only if $(h,i) \vdash \psi$. □

Theorem 5. *The potential satisfiability problem and the adherence problem for $PTLTL^{FO}$ with linear arithmetic are decidable.*

We note that the transformation of $(h,i) \vdash \psi$ to C above may result in an exponential blow-up. But if we fix the formula, we may be able to obtain a polynomial translation. Details are left for future work.

7 Extended guarded quantifiers

As we have mentioned earlier, an underlying design principle for our quantified policies is the closed-world assumption (CWA). The guarded quantifier in $PTLTL^{FO}$ is the most basic quantifier, and by no means the only one that enforces this CWA principle. It is a natural theoretical question to ask what other possible extensions achieve the same effect, although we have not so far seen the need for them in practice.

We have mentioned earlier that introducing negation in the guard easily leads to undecidability. Surprisingly, simple extensions with unrestricted disjunction or the $\mathbf{S}$-operator also lead to undecidability, as we shall see shortly. Let us first fix the language with extended guarded quantifiers. The syntax of quantified formulae is as follows:

$$\forall \vec{x} : \psi(\vec{x}).\ \varphi(\vec{x}) \qquad \exists \vec{x} : \psi(\vec{x}).\ \varphi(\vec{x}).$$

Here the formula $\psi(\vec{x})$ is a guard, and $\vec{x}$ are its only free variables. The semantics of the quantifiers are a straightforward extension of that of $PTLTL^{FO}$, i.e.,

$(h,i) \models \forall(x_1,\ldots,x_n) : \psi(x_1,\ldots,x_n).\ \varphi$ iff
for all $c_1,\ldots,c_n,$ if $(h,i) \models \psi(c_1,\ldots,c_n)$ then $(h,i) \models \varphi[x_1 := c_1,\ldots,x_n := c_n]$.

Now consider a guarded quantifier that allows unrestricted uses of disjunction. Suppose $\varphi(\vec{x})$, where $\vec{x}$ range over integers, is a formula encoding some general Diophantine equation. Let $\psi(\vec{x},y)$ be a guard formula $p(\vec{x}) \vee q(y)$, for some predicate p and q of appropriate types. Then satisfiability of the entailment $\{q(0)\} \models \exists(\vec{x},y) : \psi(\vec{x},y).\ \varphi(\vec{x})$ is equivalent to the validity of the first-order formula $\exists\vec{x}.\ \varphi(\vec{x})$, which states the solvability of the Diophantine equations in $\varphi(\vec{x})$. This means that the model checking problem for $PTLTL^{FO}$ with unrestricted disjunctive guards is undecidable. The cause of this undecidability is that satisfiability of the guard, relative to the history, is independent of the variables $\vec{x}$. Similar observations can be made regarding the unrestricted uses of the "since" operator, e.g., if we replace the guard $\psi(\vec{x},y)$ with $p(\vec{x})\,\mathbf{S}\,q(y)$, we get the same undecidability result.

Another restriction that needs to be imposed on guarded quantifiers concerns the use of function symbols: their uses easily lead to a violation of CWA, and again, undecidability of model checking. For instance, in checking

$$\{p(0)\} \models \forall(x,y) : p(x+y).\ \varphi(x,y)$$

we have to consider infinitely many combinations of x and y such that $x+y=0$.

Based on the above considerations, we design the following guarded extensions to the quantifiers of $PTLTL^{FO}$. The language of guards are defined as follows. *Simple guards* are formulae generated by the following grammar:

$$\gamma ::= p(\vec{u}) \mid \gamma \wedge \gamma \mid \mathbf{G}^{-1}\,\gamma \mid \mathbf{F}^{-1}\,\gamma$$

Here the list $\vec{u}$ is a list of variables and constants. We write $\gamma(\vec{x})$ to denote a simple guard whose only free variables are $\vec{x}$. *Positive guards $G(\vec{x})$ over variables $\vec{x}$* are formulae whose only variables are $\vec{x}$, as generated by the following grammar:

$$G(\vec{x}) ::= \gamma(\vec{x}) \mid G(\vec{x}) \wedge G(\vec{x}) \mid G(\vec{x}) \vee G(\vec{x}) \mid \mathbf{G}^{-1}\,G(\vec{x}) \mid \mathbf{F}^{-1}\,G(\vec{x}) \mid G(\vec{x})\,\mathbf{S}\,G(\vec{x}).$$

We denote with $PTLTL^{FO+}$ the language obtained by extending $PTLTL^{FO}$ with positive guards. We show that the model checking problem for $PTLTL^{FO+}$ is decidable. The key to this is the finiteness of the set of "solutions" for a guard formula.

Definition 6. *Let $G(\vec{x})$ be a positive guard and let h be a history. The* guard instantiation problem*, written $(h, G(\vec{x}))$, is the problem of finding a list $\vec{u}$ of constants such that $h \models G(\vec{u})$ holds. Such a list is called a* solution *of the guard instantiation problem.*

Lemma 3. *Let $G(\vec{x})$ be a positive guard over variables $\vec{x}$ and let h be a history. Then the set of solutions for the problem $(h, G(\vec{x}))$ is finite. Moreover, every solution uses only constants that appear in h.*

Proof. By induction on the size of $G(\vec{x})$ and by definition of the forcing relation. □

Theorem 6. *Let φ be a $PTLTL^{FO+}$ formula and h a history. The model checking problem $h \models \varphi$ is decidable.*

Proof. The proof follows the same structure as the decidability proof for $PTLTL^{FO}$, using Lemma 3 for the quantifier cases. □

8 Conclusions and related work

We have presented a formal language for expressing history-based access control policies based on the pure past fragment of linear temporal logic, extended to allow certain guarded quantifiers and arbitrary computable functions and relations. As our examples show, these extensions allow us to write complex policies concisely, while retaining decidability of model checking. Adding a counting quantifier allows us to express some statistical "meta" properties in policies. We also consider the monitoring problem in the presence of unobservable or unknown action parameters. We believe this is the first formulation of the problem in the context of monitoring.

There is much previous work in the related area of history-based access control [6, 8, 7, 2, 11, 3]. As mentioned in the introduction, our transaction-based approach to defining policies separates us from the more traditional trace-based approaches in program execution monitoring. Our work is closely related to Krukow, et al. [11, 12], but there are a few important differences. Their definition of sessions allows events to be partially ordered using *event structures* [17] whereas our notion of a session as a set with no structure is simpler. For the application domains we are interested in, we see no need for sessions to have extra structure built into their semantics since such relations between events can be explicitly encoded in our set up using first-order quantifiers and a rich term language allowing extra parameters, interpreted functions, timestamps and arithmetic. In the first-order case, they forbid multiple occurrences of the same event in a session; roughly, their histories in this case correspond to our trace-like histories (see Section 4). Their language does not allow arbitrary computable functions and relations, since as we have seen, allowing these features in the presence of quantifiers can easily lead to undecidability of model checking. Our policy language is thus more expressive than theirs in describing quantitative properties of histories.

Although we have presented an instance of history-based transaction monitoring in the sense of [11] or [6], the complexity of our decision procedure depends on the length of a history, which is in contrast to more efficient means of monitoring as presented for propositonal LTL, e.g., in [8, 3]. There a so-called *monitor*

device is generated for a policy which reads a history as it unfolds and, therefore, does not need to re-apply a costly model checking procedure when new sessions are added. Instead in [8], only the truth values of certain subformulae of the policy are kept with respect to the previous session, in order to compute the truth value of the subformulae with respect to the new session; that is, the complexity of the monitor does not depend on the length of a history.

Not all policies in $PTLTL^{FO}$ can be monitored independently of the history. For example, in a policy such as $\forall x : p.\ \mathbf{G}^{-1}\, \exists y : q.\ y \leq x$, we must, for each new $x : p$, check all the previous sessions whether or not there exists a $y : q$, such that $y \leq x$ holds, and this cannot be done without storing the entire history. A policy such as the one given in Section 3 involving a fictional eBay.com rule, however, can be monitored efficiently as it does not involve the same nesting of temporal modalities and quantifiers:

$$\varphi \equiv \mathbf{G}^{-1}\, \varphi_1 \text{ where } \varphi_1 \equiv \forall (t, x, v) : pay.\ \exists (y, t') : post.\ x = y \wedge t' \leq 10.$$

We can evaluate it with respect to the current session only, and keep track of the results from previous evaluations using two arrays of truth values, *pre* and *now* like in Havelund and Rosu's work. The idea of these arrays is to store the truth values of subformulae with respect to the current session (*now*) and the previously seen session (*pre*). In this example, it is sufficient that *pre* and *now* each have two entries; the first corresponds to the truth value of the subformula φ_1 and the second to φ. The values of *now* are updated for each new session, and subsequently copied to *pre*. The condition induced by the $\mathbf{G}^{-1}$-operator is that φ_1 has to be true now, and previously, for all sessions, i.e., $now[2] \leftarrow now[1] \wedge pre[2]$.

An obvious class of monitorable policy is one obtained by susbtituting propositional variables in a propositional LTL formula with closed first-order formulae (without temporal operators). For this class of formulae, with straightforward modifications, Havelund and Rosu's construction can be applied to construct efficient monitors. Details of this and other possible classes of monitorable policy are left for future work.

References

1. The RuleML Initiative. Document located on-line at http://www.ruleml.org/.
2. M. Bartoletti, P. Degano, and G. L. Ferrari. History-based access control with local policies. In *FoSSaCS*, volume 3441 of *LNCS*. Springer, 2005.
3. A. Bauer, M. Leucker, and C. Schallhart. Monitoring of real-time properties. In *FSTTCS*, volume 4337 of *LNCS*. Springer, 2006.
4. H. Boley, M. Dean, B. Grosof, M. Sintek, B. Spencer, S. Tabet, and G. Wagner. FOL RuleML: The First-Order Logic Web Language. *Located at http://www.ruleml.org/fol*, 2005.
5. D. F. C. Brewer and M. J. Nash. The chinese wall security policy. In *IEEE Symposium on Security and Privacy*. IEEE, 1989.
6. G. Edjlali, A. Acharya, and V. Chaudhary. History-based access control for mobile code. In *ACM Conference on Computer and Communications Security*, pages 38–48, 1998.

7. P. W. L. Fong. Access control by tracking shallow execution history. In *IEEE Symposium on Security and Privacy*, pages 43–55. IEEE Computer Society, 2004.
8. K. Havelund and G. Rosu. Synthesizing Monitors for Safety Properties. In *TACAS*, volume 2280 of *LNCS*. Springer, 2002.
9. A. Jøsang, R. Ismail, and C. Boyd. A survey of trust and reputation systems for online service provision. *Decision Support Systems*, 43(2):618–644, 2007.
10. D. Kroening and O. Strichman. *Decision Procedures*. Springer, 2008.
11. K. Krukow, M. Nielsen, and V. Sassone. A framework for concrete reputation-systems with applications to history-based access control. In *ACM Conf. Comp. and Commun. Sec.*, 2005.
12. K. Krukow, M. Nielsen, and V. Sassone. A logical framework for reputation systems and history based access control. Journal of Computer Security. To appear, 2008.
13. Y. Matiyasevich. *Hilbert's 10th Problem*. MIT Press, Cambridge, 1993.
14. A. Pnueli. The temporal logic of programs. In *Proc. FOCS-77*, pages 46–57, 1977.
15. M. Roger and J. Goubault-Larrecq. Log auditing through model-checking. In *CSFW*, pages 220–234. IEEE, 2001.
16. M. Sipser. *Introduction to the Theory of Computation*. Intl. Thomson Publishing, 1996.
17. G. Winskel and M. Nielsen. Models for concurrency. In *Handbook of logic in computer science (vol. 4): semantic modelling*. Oxford University Press, 1995.