The Complexity of Discretionary Access Control

Stephen Dranger, Robert H. Sloan*, and Jon A. Solworth

Dept. of Computer Science
University of Illinois at Chicago
`stdrange@alumni.uchicago.edu; sloan|solworth@uic.edu`

Abstract. A recent paper presented an access control scheme for discretionary access controls with a decidable safety problem. This paper deals with the complexity analysis of that access control, and finds it to be, in its worst cases, PSPACE-complete, but polynomial time for practical cases. The PSPACE-hardness reduction uses the theory of succinct problems in a more general manner than circuit representation.

1 Introduction

In a computer system, *access controls* restrict subjects (users and/or processes) to performing only those operations on objects (e.g., files) for which they are authorized. For each such operation, the access controls either allow or disallow that operation to be performed. In *Discretionary Access Controls (DACs)*, each *object* has an owner who exercises primary control over the object. DACs are oldest and most widely used class of access controls, the access controls for both Windows and UNIX are DAC. The Unix DAC, for example, has the well known three primitive permissions read, write, and execute.

In the late 1970s, Harrison, Ruzzo, and Ullman (HRU) introduced a seemingly very simple general-purpose language for Discretionary Access Control (DAC). In spite of the simplicity of the HRU language, a *safety* property with parameters a specific permission p, subject s, and object o:

$$\text{“Always } s \text{ does not have permission } p \text{ for } o\text{”} \tag{1}$$

is undecidable [3].

Recently Solworth and Sloan gave a group-based mechanism for designing DACs for which the safety problem (Equation 1) *is* decidable [10], and showed that that mechanism is expressive enough to construct any particular DAC from a taxonomy of DACs given by Osborne et al. (OSM) [8].

That group-based access control scheme was the first general access control model proved both to have a decidable safety property and to be capable of implementing the full range of DAC models. From HRU's work in the 1970s though the early 2000s, general access control models were published that have both decidable (but relatively weak) and undecidable (but more expressive) variants. This includes HRU, Sandhu's 1992 Typed Access Model (TAM) [9], and

* Partially supported by NSF grant No. CCF-0431059.

Koch et al.'s 2002 graph-based model [4, 5]. In each of these cases, decidability is obtained by requiring a type of monotonicity: an operation can add or remove privileges but not both. Thus, for example, changing a user's group membership is not permitted in the decidable version of those access control schemes, since changing groups typically both adds and removes privileges.

In this paper we consider the precise complexity of the safety problem in the Solworth and Sloan access control scheme. We show that for the implementation of any DAC in Osborne et al.'s taxonomy, the safety problem can be decided in polynomial time, and that for the mechanism in its full generality, the safety problem is PSPACE complete. The proof of PSPACE hardness may have some interest in its own right. As we mention very briefly in Section 6, it generalizes the theory of succinct graph problems [2].

In the next section we quickly sketch the group-based access control model of [10] (full details are given in that paper). Section 3 describes the sliding marker net, a new marked graph model, which is the appropriate abstraction of the key part of the group-based access control system. Section 4 gives the polynomial-time result, and Section 5 gives the PSPACE-completeness result. We conclude in Section 6.

2 Review of the access control scheme

Solworth and Sloan's group-based access control model is a general purpose scheme that allows one to describe a wide variety of particular access control systems. It corresponds to what they called "Layer 1" [10], and to what Li and Tripunitara [6] call an *access control scheme.* An access control scheme has a set of states and family of transition functions, and possible permissions. A particular access control system specifies a particular transition function, and the specific permissions, and typically narrows the set of states as well.

In all particular access control systems in the Solworth-Sloan scheme, processes derive authority to perform operations from the *user* on whose behalf they execute. Every *object*—or entity that can be accessed by a process—has a *label* that (indirectly) defines the *privilege* (also called *permission* or *right*) that various users have to perform operations on the object. (A file is the typical example of an object.) Objects are disjoint from users.

In defining a particular system, a fixed constant number of privileges is chosen (e.g., read, write, and execute). Privileges map labels to *groups* of users; the mapping is fixed when the label is created, although the membership of the group is *not* fixed. Protection is at the granularity of labels.

The group mechanism is the novel part of the scheme. A *group set* is a collection of one or more *groups*; a group is a set of users.[1] Every group set has a set of users, a set of group tags, and a set of pairs, $\langle u, t \rangle$ where u is a user ID and t is a group tag, which determine group membership.

[1] What we describe in this section are "*native* groups and group sets" in [10]; to implement a particular DAC policy one typically uses more than one of these native groups to implement one group in the specified policy.

2.1 Formal description of Solworth-Sloan scheme

In this subsection, for completeness, we give a fairly detailed description of the relevant parts of the Solworth-Sloan access control scheme. However, the reader can probably follow the main points of this paper without reading these details.

Formally, in the Solworth-Sloan scheme, every state (in any possible system) is a tuple with the following (many) components:

- The set U of current users
- The set O of current objects
- The set L of current labels of objects
- A map $\ell : O \to L$ giving the label of every object
- The set GS of current group sets
- The set G of current groups
- The set of privileges, and for each privilege, a map $p : L \to G$ telling which group of users has that privilege
- A map $\mathrm{gs} : G \to GS$ that determines the group set of each group
- The set T^G of current group tags, and a map $f : T^G \to GS$ that gives the group set of each group tag.
- The set $GL \subset U \times T^G$ of all user–group tag pairs. Each ordered pair $\langle u, t\rangle$ can be thought of as a "group label" on a notional group object.
  Each user has at most one group tag for each group set at any time; that is, if both $\langle u, t\rangle, \langle u, t'\rangle \in GL$ then $f(t) \neq f(t')$. The current set of users of a given group set is exactly the set of users that have a an group tag associated with the group set; that is, the users of groups set gs are exactly $\{u : \exists t : f(t) = gs \text{ and } \langle u, t\rangle \in GL\}$.
- A relation $r \subset: T^G \times G$ relating group tags to groups. User u is a member of group g iff there is a $t \in T^G$ such that $\langle u, t\rangle \in GL$ and $r(t, g)$.
  We require that group tags give membership only in groups in the tags group set; that is, that if $r(t, g)$, we have $f(t) = \mathrm{gs}(g)$.
- A set of triples $rules \subset T^G \times T^G \times G$. Each triple $\langle t, t', g\rangle$ gives a group label relabel rule of the form $Relabel(t, t') = g$, which means that any member of group g can, for any u change group label $\langle u, t\rangle$ to $\langle u, t'\rangle$.
  A relabel rule must be inside of a single group set; that is, if $\langle t, t', g\rangle \in rules$, then $f(t) = f(t')$. However, we do *not* require that the group administering the relabel be in the group set; that is, we do not require $\mathrm{gs}(g) = f(t)$.
- The current set $N \subset T^G$ of *new user group tags*. When a new user u_n is added to the system, for each $t \in N$ a new group label $\langle u_n, t\rangle$ is created.

2.2 Key points about groups and group sets

In this paper, we are not concerned with any state transitions that add new groups or group sets to a system. We are concerned exclusively with those transitions that change the membership of existing groups. There are two types of transitions that change group membership: the addition of new users, and the application of group tag relabel rules to change the user–group tag pairs. These transition rules are the same in any system using the Solworth-Sloan access control scheme—what varies from system to system are the groups and group sets that can be created. In obtaining the hardness results in this paper, we assume that arbitrary groups and groups sets can be created.

Roughly speaking (see the previous subsection for more details), every system is comprised of a set of users U, a set of objects O, a set of labels L, a mapping $\ell : O \rightarrow L$, and group sets, groups, and permissions. We describe group sets in detail shortly, but each group set has some (unique) groups, and the set of all groups in all groups sets is denoted G. Now for each primitive privilege (e.g., read, write) we have a map $p : L \rightarrow G$ specifying the group of users that have that permission for objects with a particular label.

Each group set includes the following components:

1. Its set of groups. Every group belongs to exactly one group set.
2. Its set of group tags. Every group tag is associated with exactly one group set.
3. The current group tag–user pairs.
4. An optional new user tag.
5. Its set of users.[2]
6. The relation of the group set's group tags to its groups.
7. Group tag relabel rules.

All components are finite, and all components except the group tag–user pairing are specified and fixed for the life of the group set when the group set is created. An initial group tag–user pairing is specified when the group set is created, but it evolves over time.

Each user in the group set has exactly one group tag at any time. The group-tag–user pairs are referred to as *group labels*, and are thought of as labels on notional objects. Thus, if group tag G is associated with groups g_1 and g_3, and user u has group tag G in group set $\mathcal{G}$, then (at that time) u is a member of groups g_1 and g_3, and not a member of any other group of $\mathcal{G}$ (From this information alone, we cannot tell which groups of *other* group sets u is in.)

The definition of the group set tells which users are initially in the group set, and the group tag for each user. If $\mathcal{G}$ has new user tag G_n, then all users added to the system are added to group set $\mathcal{G}$ with tag G_n.

For any two group tags G_1, G_2 in group set gs, a group relabel permission $\mathit{Relabel}(G_1, G_2) = g$ may be defined at gs's creation that enables any member of group g to change the group tag of any user u in $\mathcal{G}$ from G_1 to G_2. Group g may be either in gs or an existing group in another group set.

2.3 Safety problem in this setting

In this setting, the safety problem of Equation 1 becomes, "Can user s become a member of group $g = p(\ell(o))$?", where $l = \ell(o)$ is the label of object o and $p(l)$ is the group having permission p for label l. This in turn becomes the question,

[2] Formally, the set of users is induced by the set of group tag–user pairs.

"for each group tag G associated with group g in group set gs, could user u be paired with tag G?"

3 Abstract model of problem

To model the safety problem for this access control system, we introduce the *sliding marker net* model. Group tags form the vertices of a digraph, with the edges corresponding to the existence of a relabel rule. Markers on the vertices represent users. Thus k markers in a vertex representing group tag G correspond to k users having a user–group-tag pair with G. Each edge is *governed* by a set of vertices—corresponding to the tags for the group that has the relabel permission for that edge. A marker can be moved across an edge only if at least one vertex in the set of vertices governing that edge contains a marker. A set of new marker vertices that can have markers added to them models the new user tag mechanism for adding new users. More formally:

Definition 1. *A* **sliding marker net** *is a 5-tuple,* (V, E, A, M_0, N)*, where* V *and* E *are the vertices and edges of a directed graph,* A *is a function* $A : E \to 2^V$*, mapping each edge to a vertex set,* $M_0 : V \to \mathbb{N}$ *is the* initial marking*, and* $N \subseteq V$*. (*N *represents the group of new users.) We say vertex set* $A(e)$ governs *edge* e *(if* $|A(e)| = 1$ *we may say a vertex governs an edge). The underlying digraph is called the sliding marker graph.*

The marking of a sliding marker net is its state. For marking M we say that vertex v is *nonempty* if $M(v) > 0$ and *empty* if $M(v) = 0$.

Given a marking M of a sliding marker net, we can obtain a new marking M', or make a *move*, in two ways. One is by "sliding a marker" over edge $(u, v) \in E$. This move requires that $M(u) > 0$ and that there is a $w \in A(u, v)$ s.t. $M(w) > 0$ (i.e., $A(u, v)$ has at least one nonempty vertex in marking M), and decreases $M(u)$ by 1 and increases $M(v)$ by 1. The other kind of move is to add 1 to $M(v)$ for every $v \in N$, which models adding new users.

The Vertex set nonemptiness problem for sliding marker nets is as follows: Given a sliding marker net (V, E, A, M_0, N) and a vertex set S, is there a sequence of moves that yields a new marking in which at least one $v \in S$ is nonempty? This has the same complexity as Vertex nonemptiness (ignoring a multiplicative factor of at most $|V|$), whether a particular $v \in V$ can be made nonempty, and we work with this slightly simpler problem. To the best of our knowledge, sliding marker net Vertex nonemptiness is a novel graph problem.

We are in fact analyzing the complexity of a slightly different problem than the analog of the stated safety problem (1). The sliding marker net analog of the safety problem in this context is, "Can a *specific, named marker* reach vertex v?" However, this problem and Vertex nonemptiness are both PSPACE-complete. A slightly messier version of Theorem 1's construction would show this problem is in PSPACE as well. In the other direction, there is a simple reduction of Vertex nonemptiness to the specific-marker version: Given a sliding marker net and vertex v^*, add two new vertices v_1 and v_2 with an edge

(v_1, v_2), and add the specific named marker to v_1 in the initial marking. Finally set $A((v_1, v_2)) = \{v^*\}$. The VERTEX NONEMPTINESS instance has a solution iff the named marker can reach v^*.

4 The OSM DACs can be decided in polynomial time

The group mechanism of Sloan and Solworth can be used to implement some intricate access control policies that go beyond what are conventionally considered DAC systems. To obtain the PSPACE hardness result we show in Section 5, we construct an artificial access control system that certainly goes beyond a normal reasonable DAC system. In this section we briefly argue that the safety question for conventional DAC systems can be answered in polynomial time.

The result needed for sliding marker nets is that VERTEX NONEMPTINESS can be decided quickly for an sliding marker net with a simple structure. We say vertices $U \subset V$ of sliding marker net (V, E, A, M_0, N) form an *isolated component* iff both (1) there is no directed edge between U and $\bar{U}$ in the underlying graph and (2) every edge e within U has $A(e) \subseteq U$.

Theorem 1. *If the underlying digraph of sliding marker net* (V, E, A, M_0, N) *consists of isolated components where each component has only a constant number of vertices, then* VERTEX NONEMPTINESS *can be solved in linear time.*

Proof. We need consider only the component with the vertex in question; let its size be c. To decide its nonemptiness, first, add c markers to every vertex in the set N. If any vertex in the component has more than c markers, reduce its number of markers to c. Then exhaustively construct the portion of the state space obtained only by sliding markers in the component. The required bookkeeping can be done in time linear in the size of the sliding marker net.

Now OSM gave a taxonomy intended to include all reasonable DAC systems. In all of them, the owner of an object can grant or revoke ordinary permissions (e.g., read, write, and execute) to the object. The various OSM DACs allow the following variations: (1) Whether or not a owner of an object can give it away (change ownership). (2) Whether the owner of an object can delegate the right to grant ordinary permissions to other users. The case where the owner can delegate is called liberal DAC, and it has a number of variations: (a) who can revoke ordinary permissions, and (b) whether the delegation propagates. OSM propose no propagation ("one-level grant"), that an owner can make either a grant that cannot be propagated, or a grant that can be propagated once ("two-level grant"), and arbitrary propagation (multi-level grant).

Solworth and Sloan [10] sketched the implementation of any of these DAC mechanisms on top of their group mechanism. Objects are protected at the granularity of *labels*. For each label l and each ordinary permission, two group sets are used, one for the ordinary permission and one for the administration. The ordinary permission group set has two group tags, and the administrative group set has only two tags in all cases except for liberal DAC with 2-level grant, when it has three tags. In all cases, all group tag relabel permissions within those two group sets are held by members of the administrative group set. The group tags correspond to vertices. Relabeling of a users group tag occurs only among group tags inside one group set. The sets $A((u,v))$ correspond to the users allowed to relabel an arbitrary user's group tag from u to v. Thus the construction gives isolated components of 4 vertices for each OSM DAC except for 5 vertices for liberal DAC with two-level grant.

Thus we have sketched the argument for:

Corollary 1. *The safety problem can be decided in polynomial time for all the OSM DACs.*

Remark: The extension to OSM to allow n-level grants for any constant n is also decidable.

5 General problem is PSPACE complete

In this section we will show that sliding marker net VERTEX NONEMPTINESS is PSPACE complete.

5.1 Vertex nonemptiness is in PSPACE

Theorem 2. *Sliding marker net* VERTEX NONEMPTINESS $\in$ *PSPACE.*

Proof. We argue VERTEX NONEMPTINESS is in PSPACE = NPSPACE. Consider net (V, E, A, M_0, N). Vertices having more than one, as opposed to exactly one, marker affect the emptiness/nonemptiness of vertices in reachable markings only because some of those "extra" markers could later move, creating additional nonempty vertices. Thus we can treat markings with more than $|V|$ markers in a vertex as if they had only $|V|$ in that vertex. Hence we may modify M_0 to a marking M_0' by first adding $|V|$ markers to every vertex in N ("new users"), and then reducing the marking of any vertex u with more than $|V|$ markers to $|V|$.

Thus we have at most $|V|^2$ markers. So the total number of distinct markings is less than the number of ways we can fit $|V|^2$ indistinguishable markers into $|V|+1$ containers, or $\binom{|V|^2+|V|}{|V|^2} < (|V|^2+|V|)^{|V|}$. (We add another container to represent the markers that are not on the sliding marker net yet.) To solve the problem nondeterministically, we simply choose a legal move, increment a counter, and repeat. This requires $O\left(|V|\log|V| + \log\left(\left(|V|^2+|V|\right)^{|V|}\right)\right) = O(|V|\log|V|)$ space: $O(|V|\log|V|)$ to keep track of where the markers are (i.e., we need to store $|V|+1$ integers in the range of 0 to $|V|^2$) and $O\left(\log\left(\left(|V|^2+|V|\right)^{|V|}\right)\right)$ to keep to keep track of the binary counter.

5.2 Vertex nonemptiness is PSPACE hard

Our reduction for PSPACE hardness is loosely inspired by succinct representations of graphs [2, 7]. In that theory, a graph is represented by a (sometimes) exponentially smaller circuit describing the graph. Here, the state space of the sliding marker net is exponentially bigger than the sliding marker net itself.

Theorem 3. *Sliding marker net* VERTEX NONEMPTINESS *is PSPACE-hard.*

We will reduce QUANTIFIED BOOLEAN FORMULA (QBF) to VERTEX NONEMPTINESS. The reduction is rather involved; we give it here, and then sketch the argument for its correctness.

Reduction

Throughout let n be the number of variables in the given QBF

$$\psi = Q_1 x_1 Q_2 x_2 \cdots Q_n x_n \phi(x_1, \ldots, x_n) \tag{2}$$

where each Q_i is either $\exists$ or $\forall$, and ϕ is some 3CNF formula.

We implicitly use but do not construct a "QBF graph" of ψ that has a path from a designated source to a designated sink iff ψ is satisfiable. This graph is adopted from [7]; here we construct a digraph. We construct a sliding marker net such that it has designated vertex that can become nonempty iff the QBF graph has a source to sink path. We will rely on:

Proposition 1 ([7]). *A QBF is satisfiable if and only if its QBF graph has a path from s to t.*

QBF graph The QBF graph is a digraph with source s and sink t; every vertex is either "s-side" or "t-side." Each vertex corresponds to a partial assignment to the n Boolean variables. A vertex's *level* m, for $0 \leq m \leq n$, tells how many variables are set. The only level 0 vertices are s and t. For $1 \leq m \leq n$, in a level m vertex, the first m variables are set. We give each vertex a $2n + 1$ bit label: First, n bits for the level m in (padded) unary; then 1 bit for s or t; then m bits for the setting of the first m variables; and lastly $(n - m)$ 0s if on the s side, or 1s if on the t side.

The edges are (roughly) as follows. For $m < n$, each s-side level $(m - 1)$ vertex has one or two level m successors on the s-side: two successors setting x_m to each of 0 and 1 if Q_m is $\exists$, and one successor setting $x_m = 0$ if Q_m is $\forall$. For each (complete) variable assignment a that satisfies ϕ, there is an edge from the level n s-side to the level n t-side vertex with assignment a. For $0 < m \leq n$, the edges out of each t-side vertex with level m are as follows. If $Q_m = \exists$ or the assignment to $x_m = 1$, then the vertex has a level $m - 1$ t-side successor with the same assignment to the first $m - 1$ variables and with $x_m = 1$ (though the level $m - 1$ means that this is not considered part of the partial assignment). If

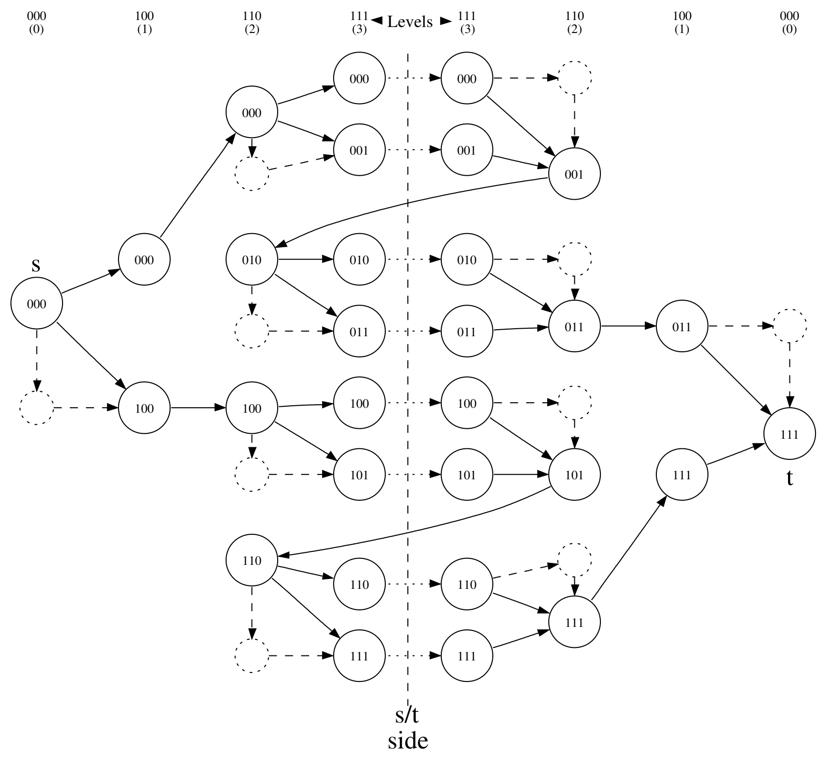

Fig. 1. QBF graph for $\exists x_1 \forall x_2 \exists x_3 \phi$. The vertex labels shown are the values of x_i, written $x_1 x_2 x_3$. (Full vertex label doesn't fit in this diagram.) Vertex levels are shown above in both unary and decimal. The line through the center of the graph divides the s and t sides. The dotted horizontal lines through that line represent edges that exist exactly when ϕ is true for that particular value of $x_1 x_2 x_3$. Dashed vertices and dashed edges show the minor modification in our construction. Notice that unset variables are always 0 on the s side and 1 on the t side.

$Q_m = \forall$ and $x_m = 0$, then the vertex has a level m s-side successor with the same assignment except $x_m = 1$.

We now make one modification, so that there are no edges where both the level and the truth setting changes. For existential variables on the s side, for the successor that corresponds to setting the variable to 1, we use two vertices—the immediate successor has a label with the level *not* increased and the truth setting changed to 1, and that vertex has one successor with the level increased. Similarly, for the t side we decrease the level, then change the truth setting to 0 when we unset an existential variable. See Figure 1.

Sliding marker net construction We do not construct the QBF graph. We do construct a sliding marker net of size polynomial in n that has certain special markings corresponding to vertices of ψ's QBF graph. Our construction will allow it to move from one special marking to another if and only if there is a corresponding edge in the QBF graph. From there, we will show this construction therefore allows a designated vertex v^* to be marked if and only if there is an s-to-t path in the QBF graph.

The sliding marker net has two isolated vertices, *empty* (with no marker) and *full* (with one marker). We lay out the remaining vertices in rows within $3n+2$ numbered columns: first n *level columns*, then one *s/t column*, then n *truth (assignment) columns*, then n *ON/OFF columns*, and lastly one *final column*. We denote the ith column by C_i. All columns except the s/t column have $3n+2$ vertices. (The number of vertices in the s/t column depends on the number of clauses in the formula.) The initial marking puts one marker in each column, in the top-most vertex. Each column has both directed edges between every pair of (vertically) adjacent vertices, except the final column which has only the down directed edge. There are no other edges. We refer to edges from vertex m to $m+1$ in a column as *down (side)* edges, and edges from vertex $m+1$ to m as *up (side)* edges. See Figure 2 for the layout of the sliding marker graph.

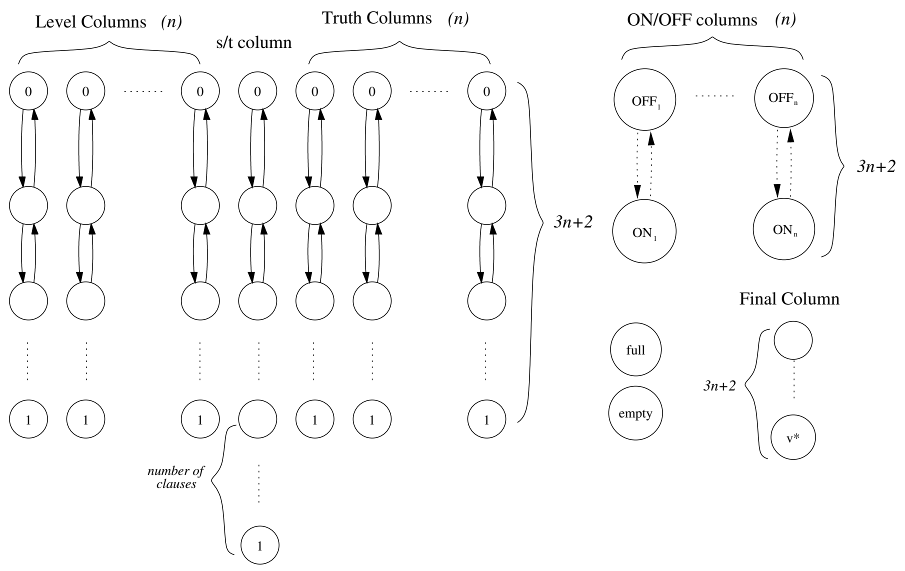

Fig. 2. Overall layout of constructed sliding marker graph for n-variable formula.

The top-most and bottom-most rows play a special role. We refer to the top row of the ON/OFF columns as the OFF row and the bottom row as the ON row. For the other columns we refer to the top row as the 0 row and the bottom row as the 1 row; we also speak of ON, OFF, 0, and 1 vertices. (It may help to remember that QBF graph we are modeling starts at 0 and ends at 1.) The notation 0_m (respectively 1_m) stands for the top (respectively bottom) vertex of column C_m; $0/1_m$ denotes the vertex set$\{0_m, 1_m\}$. Somewhat inconsistently,

OFF_i (respectively ON_i) stands for the OFF (respectively ON) vertex of the ith ON/OFF column, which is column C_{2n+1+i}. When clear from context we also use the term for a vertex for the singleton set containing that vertex.

A marking of this sliding marker net is a ***QBF marking*** if every marker is in the top or a bottom row of its column. A QBF marking corresponds to a QBF vertex whose label is obtained from the bits of the level, s/t, and truth columns of the sliding marker net. See Figure 3 for an example of a QBF marking.

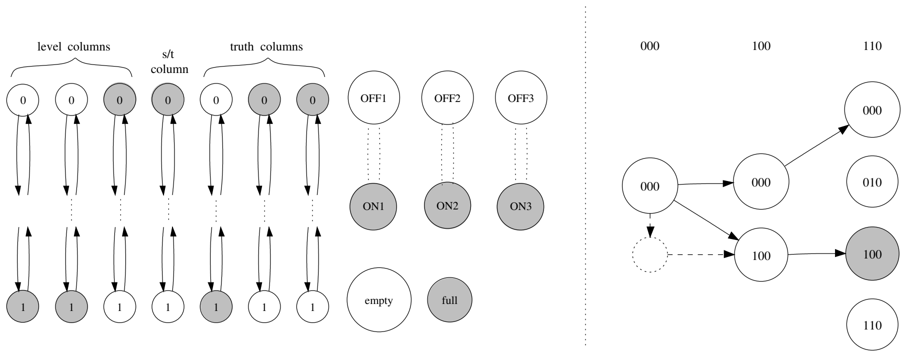

Fig. 3. Example of a marking of the sliding marker net that corresponds to a vertex in the QBF graph. In this figure, the shaded circles in the sliding marker net represent vertices with markers on them and the shaded circle in the QBF graph represents the vertex that the sliding marker net marking corresponds to.

We give some more notation; then describe the governing sets for the edges. For simplicity's sake, we call an edge (in either direction) between rows r and $r+1$ the "rth edge." (Row 1 is the top row.) Besides vertex sets mentioned already, we need $n-1$ additional OFF vertex sets: $\{OFF_1, OFF_2\}, \dots, \{OFF_1, \dots, OFF_n\}$. We refer to these vertex sets as $OFF_{1\dots 2}, \dots, OFF_{1\dots n}$.

There is a correspondence between the rth edge of an arbitrary column (excluding the final column) and the rth column for $1 \leq r \leq 3n+1$. First, for any column C_m, we can speak of the first n *edges* of column C_m as being the "level edges" of C_m. Similarly, edge $n+1$ is the "s/t edge", and then we have n "truth" edges followed by n ON/OFF edges. Second, if the rth edge of column C_m is not governed by *empty*, *full*, or some OFF group, then it must be governed by a vertex set contained in the corresponding column C_r.

That is, for any column C_m, if the rth edge of C_m for $1 \leq r \leq 3n+1$ is not governed by one of *empty*, *full*, or some OFF group, then the *rth edge of column* C_m (between rows r and $r+1$) *must be governed by one of the vertex sets* 0_r, 1_r, $0/1_r$ *of column* r. Thus we often use the shorthand of saying that edge e has $A(e) = 0$ (or 1 or 0/1 or ON) without specifying the column for the set $A(e)$, because $A(e)$'s column is determined by e's row.

We now specify the governing vertex set of each edge. First, every ON/OFF edge is governed by ON unless otherwise stated.

The level columns Our goal is that a marking with a marker in 1_i for $i \leq m$ and in 0_i in the remaining $n-m$ level columns should correspond to a QBF graph vertex with level m.

Level column C_m, down edges: The first $m-1$ level edges are governed by 1; the mth by *full*, and the remaining $n-m$ level edges by 0. That is, the rth down edge in *column* C_m is governed by the 1 vertex of *column* C_r for $r < m$, and by the 0 vertex of *column* C_r for $r > m$. This forces the level columns to move their markers from the 0 to the 1 row in order—C_m can move its marker down the first n rows from its 0 vertex only if $C_1, \ldots, C_{m-1}$ all have markers in their 1 vertices, and $C_{m+1}, \ldots, C_n$ all have markers in their 0 vertices.

The s/t down edge is governed by 0 (s), insuring that we can move down this edge only there is a marker in the s/t column corresponding to the s side of the QBF graph.

The n truth edges depend on Q_m. If $Q_m = \forall$, the xth truth edge is governed by the 0/1 vertex set $1 \leq x \leq m-1$, and by the 0 vertex set for $m \leq x \leq n$. This corresponds to allowing either setting of the first $m-1$ Boolean variables and requiring the remaining variables to be 0 in the universal case. The case $Q_m = \exists$ is the same *except* that the mth truth edge is governed by the 0/1 rather than the 0 vertex set, because for existential quantifiers, m Boolean variables would have been set already.

Level column C_m, up edges: These are the same as the down edges with two exceptions. First, the s/t edge is governed by 1 (t), corresponding to being on the t side of the QBF graph for decreasing of level. Second, those truth edges that must be governed by the 0 vertex for the down edges must instead be governed by the 1 the vertex. This is because instead of making sure variables have been set, we must make sure variables have been unset.

The s/t column The s/t column is special in this construction: it has a different number of vertices (and edges). This column is used to control the edges between the s and t sides. There is a path from s to t if the Boolean formula ϕ is true for the particular variables; edges from t back to s are needed to handle universally quantified variables.

The s/t column, C_{n+1}, starts with $3n+2$ vertices with $3n+1$ edges just like the level columns. The s/t column's down level edges are governed by 1, since we must be at level n to evaluate the Boolean formula. The s/t down edge is governed by *full.* The truth edges are governed by the 0/1 vertex set. The governing vertex sets for the up edges of this s/t column are the same as for the down direction except that edges governed by ON on the down side are governed by $OFF_{1 \ldots n}$ on the up side.

Going downwards from vertex $3n+2$, this column should "compute" whether ϕ is true on the truth setting encoded in the truth columns in the current marking (i.e., Boolean x_i is 1 (respectively 0) if the 1 (respectively 0) vertex of the ith truth column is marked). We add one additional vertex to the column for each clause of ϕ. To govern the down edge for each of these these additional vertices, we use a vertex set of three vertices (from the truth columns) for the clause. If the clause contains x_i (respectively $\bar{x}_i$), then the vertex set contains truth column i's 1 (respectively 0) vertex.

The corresponding up edges are all governed by the *full*.

The truth columns Column C_{n+1+m} is the mth of the n truth columns.

Truth column m, down edges: The first $m-1$ level edges are governed by 1. Edge m is governed by 0_m if $Q_m = \exists$ and 1_m if $Q_m = \forall$. (If x_m is existential, our QBF graph sets $x_m = 1$ immediately *before* incrementing the level to m.) The remaining $n-m$ level edges are governed by 0. The s/t edge is governed by 0/1 if $Q_m = \exists$ and 1 if $Q_m = \forall$.

The first $m-1$ truth edges are governed by 0/1, and the mth by *full*. The last $n-m$ truth edges are governed by the 0/1 vertex set if $Q_m = \exists$, and the 1 vertex if $Q_m = \forall$.

If $Q_m = \forall$, the mth ON/OFF edge is governed by OFF_m.

Truth column m, up edges: The first $m-1$ level edges are governed by 0/1; the remaining level edges by 0. This is to allow x_m to revert to false if a universal x_i for $i < m$ needs to *reset* itself. The s/t edge is governed by 1 (t) to make sure we are on the t side of the graph.

The truth edges are governed by 0/1 except that the mth truth edge is governed by *full*. The first $m-1$ ON/OFF edges are all governed by $OFF_{1\ldots m-1}$. (All the truth edges of the first truth *column* are governed by *empty*.)

ON/OFF columns The n ON/OFF columns are designed to be ON/OFF switches. Intuitively they are used to *turn off* the sliding marker net—put it into a QBF marking not corresponding to an QBF graph vertex. This is used when we have two QBF vertices u and v that differ in more than one area of information, which happens when we need to test the second value of a universally quantified variable. The governing vertex sets are assigned so that we can move ON/OFF switch m from ON to OFF only at level m.

ON/OFF column m up edges: We first give the rules for the up edges. The first m level edges are governed by 1 and the remaining level edges by 0. This ensures that we can move a marker down over these edges only at level m. The mth level is the point in the graph where we must retest the x_m with the other Boolean value if $Q_m = \forall$.

The s/t edge is governed by 1 (t), because we must be on the t side of the graph. The first $m-1$ truth edges are governed by 0/1; the mth truth edge by 0; the remaining $n-m$ truth edges by 1. The mth ON/OFF edge is governed by *full*.

ON/OFF column m down edges: If $Q_m = \exists$, then *empty* is the governing group for every edge of the mth ON/OFF column. Thus x_m can never be turned on after being turned off.

Otherwise ($Q_m = \forall$), the level edges are the same as for the down edges of this column, and the s/t edge is governed by 0 (s), because to turn back on, we must be on the s side of the QBF graph. The first $m-1$ truth edges are governed by 0/1, the mth truth edge by 1, and the remaining $n-m$ truth edges by 0. The mth ON/OFF edge is governed by *full.*

The final column The final column has $3n+2$ vertices; its bottom vertex the special designated vertex v^*.

The final column's first n edges are level edges governed by the 1 vertex, its next n edges behave as a second set of level edges, governed by 0, its next n edges behave as truth edges, governed by 1, and its last edge as an s/t edge governed by t.

This ensures that the rest of the net first enters a marking with level n, and then later enters a marking with level 0. Then we make sure that we have the right Boolean values for the end and we are on the correct side.

Sketch of why reduction is correct

Let $G(\psi)$ be the QBF graph of ψ of the form of Equation (2), and let SM be the constructed sliding marker net. Then SM is of size $O(n^2 + |\phi|)$, and can be constructed in polynomial time.

By Proposition 1, ψ is satisfiable if and only if $G(\psi)$ has an s to t path. Thus we need to argue that a marker can reach the bottom vertex of the final row of SM if and only $G(\psi)$ has an s to t path.

A QBF marking is a ***QBF ON marking*** if all the markers in the ON/OFF columns are in ON. The point of our construction is that each QBF ON marking of the sliding marker net corresponds to a vertex of the QBF graph whose level, s/t, and truth labels match the marking of the level, s/t, and truth-setting columns of the sliding marker net. In this correspondence, the initial marking of SM is a QBF ON marking that corresponds to the vertex s of $G(\psi)$.

To show that an s to t path in $G(\psi)$ implies that a marker can reach the bottom vertex of the final column of SM, it suffices to show that if there is a directed edge from vertex u to vertex v in $G(\psi)$, then from a QBF ON marking corresponding to the label of u, there is a series of legal moves to a QBF ON marking corresponding to the label of v. In fact, this can be done without moving any of the markers in the ON/OFF columns in all cases except when u is on the t side and v is on the s side. In that case, then exactly one ON/OFF column has to have its marker move from the ON to the OFF column and then back again, as well as markers moving in at least one truth column, as SM passes through one or more QBF markings.

Secondly, we have to show that for any QBF ON marking of SM that corresponds to the label of a vertex v that is in $G(\psi)$, then for every QBF ON marking reachable directly (without any intervening QBF ON marking) in SM, there is a corresponding successor of v in $G(\psi)$.

The following technical lemma says that we can always restrict our attention in SM to moves that move one marker from one end of a column all the way to the other end, without any moves in other columns in between, and would be used in a formal proof of both directions.

Lemma 1. *If the constructed sliding marker net can go from QBF marking M_1 to QBF marking M_2, then it can do so with moves such that a marker always moves from one top/bottom vertex to the opposite top/bottom vertex before any other marker moves at all.*

Proof. Given any QBF marking M and any two columns C_c and C_d, if the marker in C_c moves, and stops at row x in between column C_c's top and bottom vertices, then the marker in column C_d cannot move from one to the other top/bottom vertex because (exactly) one edge of column C_d is governed either by a vertex set consisting of one or two top/bottom vertex vertices of column C_c or by *empty*. Therefore, if the marker in column C_c moves, then C_d's marker cannot move until the marker in column C_c reaches a top/bottom vertex (and thus sliding marker net reaches a QBF marking again).

If the marker in c begins to move, then the most any other column d can do until c reaches a top/bottom vertex is to move only part of the way from top/bottom vertex of its column to the other. Therefore, when c's marker finally reaches one top/bottom vertex of the column, the graph is not in a QBF marking; however, the marker in column d could have reached the same vertex had it waited until c "finished" its move.

6 Concluding remarks

In this paper, we have determined the complexity of the safety problem for the group-based access control system of [10]. For Osborne et al.'s exhaustive catalog of DAC systems [8], it is polynomial time. In general, it is PSPACE-complete.

This in fact creates an important open problem for practical access control systems, because there are numerous access control policies that have aspects of discretionary access control, but are not purely discretionary. The group-based mechanism appears to be powerful enough to implement any of the access control policies discussed in the literature, discretionary, mandatory, or otherwise. In considering any particular such policy, say, Chinese Wall [1], we have been able to make some sort of particular argument similar to Section 4 that that policy's implementation gives a safety problem that can be decided in polynomial time. However, we would like to find a characterization of some group structure that could implement any of these more exotic access control policies in addition to DAC policies, while still having a polynomial-time safety problem.

In the area of complexity theory, we think the time may be ripe to revisit the area of succinct graph representations, taking a much broader view of representations, and, simultaneously, a view that is more tied to application areas. In the original theory [2], a graph is represented by an exponentially smaller circuit that recognizes the graph. Here "recognize" meant that the circuit returns 1 for those binary numbers that encode an edge of the graph. In this paper we showed an application from computer access controls where a group membership system could represent, through its state space, an exponentially larger graph. Perhaps similar results can be obtained for various structures used in contemporary artificial intelligence, such as Bayes nets (belief nets).

References

1. D. F. C. Brewer and M. J. Nash. The Chinese Wall security policy. In *Proc. IEEE Symp. Security and Privacy*, pages 206–214, 1989.
2. Hana Galperin and Avi Wigderson. Succinct representations of graphs. *Information and Control*, 56:183–198, 1983.
3. Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. Protection in operating systems. *Communications of the ACM (CACM)*, 19(8):461–471, 1976.
4. Manuel Koch, Luigi V. Mancini, and Francesco Parisi-Presicce. Decidability of safety in graph-based models for access control. In *Proc. European Symp. Research in Computer Security (ESORICS)*, pages 229–243. LNCS, Springer-Verlag, 2002.
5. Manuel Koch, Luigi V. Mancini, and Francesco Parisi-Presicce. A graph-based formalism for RBAC. *ACM Transactions on Information and System Security (TISSEC)*, 5(3):332–365, 2002.
6. Ninghui Li and Mahesh V. Tripunitara. Security analysis in role-based access control. In *Proc. of ACM Symposium on Access Control Models and Technologies (SACMAT)*, 2004.
7. Antonio Lozano and José L. Balcazár. The complexity of graph problems for succinctly represented graphs. In *Proc. of Graph-Theoretic Concepts in Computer Science*, volume 441 of *Lecture Notes in Computer Science*, pages 277–285, 1990.
8. Sylvia Osborn, Ravi Sandhu, and Qamar Munawer. Configuring role-based access control to enforce mandatory and discretionary access control policies. *ACM Transactions on Information and System Security (TISSEC)*, 3(2):85–106, 2000.
9. Ravi S. Sandhu. The typed access matrix model. In *Proc. IEEE Symp. Security and Privacy*, pages 122–136, 1992.
10. Jon A. Solworth and Robert H. Sloan. A layered design of discretionary access controls with decidable properties. In *Proc. IEEE Symp. Security and Privacy*, pages 56–67, 2004.