The Expressive Power of Multi-Parent Creation in Monotonic Access Control Models *

Paul Ammann Richard Lipton Ravi Sandhu

Submitted to the Journal of Computer Security, October 1995†‡

Abstract

Formal demonstration of equivalence or nonequivalence of different security models helps identify the fundamental constructs and principles in such models. In this paper, we demonstrate the nonequivalence of two monotonic access control models that differ only in the creation operation for new subjects and/or objects; in particular, we show that single-parent creation is less expressive than multi-parent creation. The nature of the proof indicates that this result will apply to any monotonic access control model. The nonequivalence proof is carried out on an abstract access control model, following which the results are interpreted in standard formulations. In particular, we apply the results to demonstrate nonequivalence of the Schematic Protection Model (SPM) and the Extended Schematic Protection Model (ESPM). We also show how the results apply to the typed access matrix model (TAM), which is an extension of the well known access matrix model formalized by Harrison, Ruzzo, and Ullman (HRU). The results in this paper offer theoretical justification for regarding single-parent and multi-parent creation as fundamentally different operations in a monotonic context. The paper also demonstrates that in nonmonotonic models, multi-parent creation can be reduced to single-parent creation, thereby neutralizing the difference in expressive power.

*This work was partially supported by the National Science Foundation under grant CCR-9202270 and the National Security Agency under contract MDA904-92-C-5141.

†Author affiliations: Paul Ammann and Ravi S. Sandhu, Center for Secure Information Systems, George Mason University, Fairfax, VA 22030. Richard Lipton, Department of Computer Science, Princeton University, Princeton, NJ 08544.

‡An earlier version of this paper appeared in the Computer Security Foundations Workshop [ALS92].

1 Introduction

Access control models provide a formal expression for security policies concerning shared resources in multi-user computer systems. Access control models must be sufficiently expressive to state policies of practical interest. Balanced against the need for sufficient expressive power is the safety question, that is, can a given subject ever acquire a given right to some resource in the system. Since the safety question is undecidable even for relatively simple models, it is desirable to find primitive operations that are minimal in the sense that they are theoretically necessary to allow the expression of certain policies. The main result of this paper is a demonstration that two primitive operations, namely single-parent creation and multi-parent creation, differ in fundamental expressive power in any monotonic access control model. We elaborate the result in the context of two standard access control model formulations, namely the access matrix model [HRU76, Lam71, San92b] and the schematic protection model [San88].

We begin with a brief description of the access matrix model. The *protection state* of a system is described by a matrix that has a row and column for each subject and a column for each object. Subject X has access privilege r to subject or object Y iff r is recorded in the $[X,\ Y]$ cell of the matrix. In practice, the matrix is typically sparse, and compact representations such as access control lists (by column) or capabilities (by row) are used.

Harrison, Ruzzo, and Ullman [HRU76] provided a formalism to specify evolution of the protection state. In the access matrix model of Harrison, Ruzzo, and Ullman, to which we refer as HRU, specifications to change the protection state are in the form of commands. Each command has a fixed list of formal parameters, a conditional test, and a list of primitive operations. When a command is invoked, the conditional, or precondition, is evaluated with respect to the actual parameters. The conditional is a conjunction of tests for the presence (but not the absence) of rights in various cells of the access matrix. If the conditional evaluates to true, the primitive operations are carried out; otherwise no change to the protection state occurs. Primitive operations can enter a right into a cell of the matrix, delete a right from a cell of the matrix, create a new subject or object, or destroy an existing subject or object.

A set of commands is collectively known as a *scheme*, and the HRU model is the set of all possible schemes. Sometimes it is convenient to consider schemes that do not delete rights once they have been granted or destroy existing subjects and objects. Such schemes are called *monotonic*. The monotonic HRU model is the set of monotonic HRU schemes.

The safety question for access control was first formulated in the context of the Harrison, Ruzzo, and Ullman access matrix formalization. In monotonic models, once the precondition for a operation is satisfied with respect to a given set of existing subjects, no evolution of the protection state can cause the precondition to become false. Hence the restriction to monotonic models eliminates backtracking and simplifies safety analysis.* However, the monotonic restriction is insufficient to make safety in HRU tractable; both HRU and monotonic HRU have undecidable safety [HRU76, HR78].

The search for tractable safety analysis led to proposals for a number of 'take-grant' models, most of which are summarized in [Lan81, LS77, Sny81]. However, a substantial gap in expressive power exists between these models and HRU. Sandhu's Schematic Protection Model, denoted here as SPM [San88], was developed to fill this gap in expressive power while sustaining efficient safety analysis. With the exception of HRU, the various models referenced above are all subsumed by SPM [San88, San92a]. Although SPM has undecidable safety in general [San92c], for a wide variety of cases of practical interest, it has polynomial safety [San88].

In SPM, each invocation of the operation to create a new subject or object is associated with a single existing subject. The existing subject is called the *parent* of the creation operation and the new subject or object is called the *child*. An explicit identification of parents and children is lacking in the HRU model, although, as we note later, HRU commands can be analyzed to reveal this information.

Although SPM includes only single-parent creation, multi-parent creation is a desirable operation for many practical applications. For example, the mutual suspicion problem, the protected subsystem problem, and the confinement problem, have solutions that can be naturally implemented with multi-parent creation [AS90]. ORCON, or originator control

*Although safety analysis may dictate treating a model as monotonic, the implementation need not be strictly monotonic. For example, the nonmonotonic action of revoking an access right can be accommodated in a monotonic safety analysis if the revoked right may be regranted. In such cases, the revocation can be ignored from the perspective of safety analysis, but nonetheless implemented in the actual system [San88].

problems, also have natural multi-parent solutions [San92b]. Role-based separation of duties, in which joint authorization by multiple subjects is commonly required, also fits naturally into a multi-parent framework [AS90].

To accommodate multi-parent creation, the Extended Schematic Protection Model, denoted here as ESPM [AS90, AS92], was introduced. SPM and ESPM differ only in that EPSM has multi-parent creation whereas SPM does not; otherwise the models are identical. The tractability of safety analysis for SPM extends to ESPM [AS90, AS92].

Monotonic HRU and ESPM were shown to have equivalent expressive power in [AS92]. The expressive power of SPM was not formally addressed by the proof, but the demonstration of monotonic HRU-ESPM equivalence strongly suggested that single-parent creation is less powerful than multi-parent creation in monotonic models. In this paper we formally address the expressive power of single-parent vs. multi-parent creation. We show that in monotonic access control models, single-parent creation is strictly less expressive than multi-parent creation. An immediate corollary is that SPM is less expressive than ESPM, and hence monotonic HRU. In nonmonotonic models, multi-parent creation can be simulated by single-parent creation along with certain nonmonotonic operations.

The organization of the paper is as follows. In section 2 we define an abstract, graph-oriented access control model. The model allows the description of schemes that correspond to single and multi-parent creation. In section 3 we define a notion of simulation, and we derive comparative results for single and multi-parent schemes. We show that single-parent creation is less expressive than multi-parent creation in monotonic models, but that the two operations are equivalent in nonmonotonic models. In section 4 we relate the results from the abstract model to standard security models. Section 5 summarizes the paper.

2 Graph Model for Access Control

In this section we present an abstract formalization of the notion of an access control model. Consideration of specific models is postponed until section 4; the intent here is to provide a formalism free of unnecessary detail as a basis for the theorems in the next section. We formalize an access control model as an abstract state machine. The abstract machine has two major components, namely a set of allowable states and a set of operations to transit between states.

We begin with a description of allowable states. The state for the abstract machine is a directed graph, which we call a *protection graph*. Nodes in a protection graph correspond to either subjects or objects; no distinction between the two is made here. Edges in a protection graph correspond to access rights. If there is an edge from node A to node B, then the subject corresponding to node A has some abstract right to the subject or object corresponding to node B.

In realistic access control models, it is useful to distinguish between different classes of subjects and/or objects. We accommodate this distinction, which corresponds to the notion of protection type, by allowing nodes to be typed. The type of a node is determined when the node is first created, and cannot be changed afterwards, that is, none of the operations defined for the abstract machine are allowed to change a node's type. Similarly, we accommodate the need to distinguish between different access rights by allowing edges to be typed as well.[†] Several edges may exist between the same pair of nodes as long as the edges are all of different types. Types on edges are also static and cannot be altered. Formal notation for denoting node and arc types is introduced later in this section.

The static graph model described so far is equivalent to the common access matrix, minus any commands to change state, and thus is sufficiently general to represent any given protection state of an access control model.

To complete the description of the abstract machine, we need to consider operations to change the state. Operations which merely observe the state are not important to our analysis, and so are ignored. Our primary goal is to show that multi-parent creation cannot be simulated with single-parent creation in monotonic access control models, and so we impose the restriction in our model that no operation can delete an existing node or edge in a protection graph or change its type. At the end of the next section we show the secondary result that nonmonotonic models can simulate multi-parent creation with single-

[†]In the literature, access rights are usually not described as being typed. In particular, the notion of protection type is quite useful for analyzing subjects and objects, but no such role has been identified for treating access rights as typed objects. For our purposes here, however, it is simpler to treat edges and nodes in the same way, and so we consider both to be typed.

parent creation. Accordingly, a limited form of nonmonotonicity is introduced at that point.

We allow for three types of operations:

1. Initial state operations.

2. Node creation operations.

3. Edge adding operations.

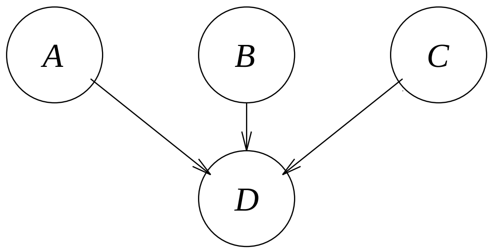
Initial state operations provide initial states for the protection graph. We allow one or more such operations, which require no prior state and produce a statically specified initial state.

A node-creation operation adds one or more new nodes, and some number of edges (possibly zero) to the protection graph. Creation operations are classified by parentage. The number of parent nodes is defined by the number of parameter nodes that exist prior to the invocation of the operation. Creation operations are also classified by the number of child nodes, which is the total number of parameter nodes in the operation less the number of parents. However, an operation's number of children is less important than its number of parents, and so we often omit the child classification when describing an operation. In section 4, we discuss multi-child operations in more detail in the context of the HRU model.

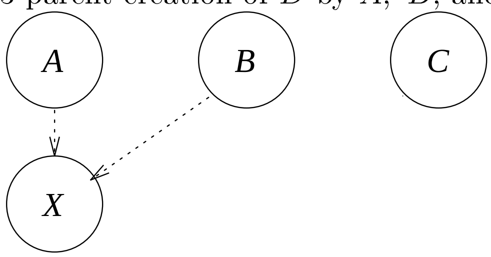
For example, a single-parent creation operation might produce a single new node in the protection graph and create an edge from the parent node to the new node. As another example, a double-parent creation operation might produce two child nodes in the protection graph and install one edge between the two parents and and another edge between the two children.

Depending on the number of node and edge types, there may be many permutations of these operations, such that the type of each node and edge is taken into account. For example, a complete description of the single-parent creation operation above would specify that the parent be of type t_1, the child be of type t_2, and that edge from parent to child be of type t_3.

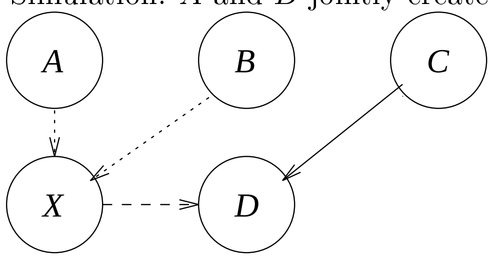
Multi-parent creation can be simulated with double-parent creation, a result shown in the context of the ESPM model in [AS92]. Figure 1 shows a construction for triple-parent



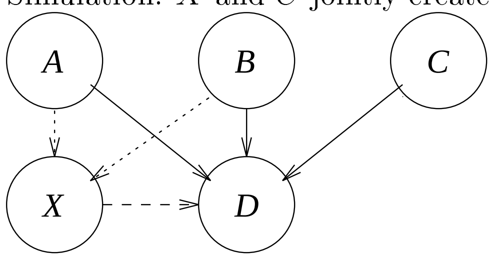
a) 3-parent creation of D by A, B, and C



b) Simulation: A and B jointly create X



c) Simulation: X and C jointly create D



d) Simulation: Edges $A \to D$ and $B \to D$ added

Figure 1: Simulation of 3-Parent Creation with 2-Parent Creation

(single-child) creation in the graph model used in this paper. A simple extension to the construction applies to larger numbers of parents.

Figure 1(a) shows a three-parent creation of D by A, B, and C. Figure 1(b-d) shows the simulation via double-parent creation. In figure 1(b), A and B jointly create X, a node of a type used only in the construction. A and B are tied to X with edges that are also of a type special to the construction. In figure 1(c), X and C jointly create D. The edge $C \rightarrow D$ is of the final desired type. The edge $X \rightarrow D$ is of a type used only in the simulation. In figure 1(d), the edges $A \rightarrow D$ and $B \rightarrow D$ are added by edge adding operations that rely on the presence of the $A \rightarrow X$, $X \rightarrow D$ and $B \rightarrow X$, $X \rightarrow D$ edges, respectively.[‡]

The theorems in the next section are all proved in terms of single-parent vs. double-parent creation. The equivalence of double-parent creation and multi-parent creation permits us to avoid direct consideration of multi-parent creation, but still generalize the results to include multi-parent creation.

Edge adding operations (one may think of such operations as zero-child operations) do not create new nodes. Thus all nodes must be created either by initial state operations or by node creation operations. In addition, if any operation ever becomes applicable, then it must remain applicable. In the abstract machine, if the precondition for an operation is ever satisfied for a given set of inputs, then it is forever satisfied for that set of inputs. Note that our notion of monotonicity specifically excludes tests for the absence of edges.

Definition: A *scheme* is a complete abstract machine definition. Specifically, a scheme defines finite sets of node types, edge types, initial states operations, node creation operations, and edge adding operations.

In the discussion below, it is useful to be able to identify the various parts of a scheme for reference purposes. Thus we proceed as follows for a scheme S. We designate the set of node types by $NT(S)$ and the set of edge types by $ET(S)$.

Schemes are classified according to the largest number of parents in a creation operation. For example, all creation operations in a single-parent scheme have exactly one parent.

[‡]Note that the simulation of multi-parent creation with double-parent creation requires that a single node be allowed to redundantly function as more than one parent in a multi-parent creation operation. Thus, for example, a triple-parent creation with A, A, and B as parents must be allowed. Most access control models have this property, including HRU, ESPM, and TAM [San92b].

Creation operations in a double-parent scheme may have either one or two parents, and so on.

We discuss common properties of schemes properties by introducing the notion of a model.

Definition: A *model* is a set of schemes.

Models are classified according to the schemes that make up the model. Thus a single-parent model contains single-parent schemes, a double-parent model contains single-parent and double-parent schemes, and so on. In addition, models whose schemes include only monotonic operations are called monotonic models.

This completes the definition of a family of abstract access control models. We now turn to examining the differences between models with different number of parents.

3 Non Equivalence Results

In this section we first define the notion of simulation, and then derive comparative results about single-parent models and double-parent models.

3.1 Definition Of Simulation

We eventually wish to decide if one model is as expressive as another. To do this, we need to formalize the notion that one scheme simulates another. In the discussion that follows, the scheme that is being simulated, denoted as the *original scheme* is represented by scheme A. The scheme implementing the simulation, denoted as the *simulation scheme*, is represented by scheme B.

An important aspect of our notion of simulation is that it is defined from the perspective of safety analysis. In particular, we arrange matters such that the simulation scheme B can leak one of the rights from scheme A (that is, create an edge of a type defined in scheme A) if and only if scheme A can leak that same right. Thus the definition of simulation presented below differs from the more common notion used for other purposes.

We adopt the following notion of correspondence between schemes. We require the simulating scheme B to maintain as part of its state a subgraph that is the entire state for scheme

A. We implement this requirement as follows. First, the sets of node and edge types in the simulation scheme must be supersets of the sets of node and edge types, respectively, in the original scheme, that is, $NT(A) \subseteq NT(B)$ and $ET(A) \subseteq ET(B)$.[§] Second, any actual node or edge in the simulation of the same type as a node or edge in the original must actually correspond to a node or edge in the original. Thus scheme B cannot contain extraneous nodes or edges of the types defined for scheme A. In general, however, scheme B can contain nodes and edges of other types for use as auxiliary structures for the simulation.

When B reaches a state that matches a state of A, then the two graphs are the same with respect to nodes in $NT(A)$ and edges in $ET(A)$. Formally:

Definition: A state in scheme A, an original scheme, and a state in scheme B, a simulation scheme, *correspond* iff the graph defining state in scheme A is identical to the subgraph obtained by taking the state in scheme B and discarding all nodes and edges not in $NT(A)$ or $ET(A)$.

We disallow mappings in which the correspondence of a node or edge in the simulation to a node or edge in the original changes as the simulation proceeds. The correspondence is set when a node or edge in the simulation is created and does not change thereafter.

Since the simulation might employ a series of operations to accomplish what the original scheme does in one operation, exact correspondence, as defined above, is too limiting with respect to the simulating scheme. For example, in the double-parent simulation of three-parent creation illustrated earlier in fig. 1, the simulation created the desired node and edges piecemeal in three separate operations. The important characteristic of intermediate states is that there be sequence of operations that leads to a corresponding state.

Two other properties complete the definition of simulation:

Definition: Scheme B *simulates* scheme A iff the following conditions hold:

1. For every state a reachable by scheme A, there exists some state b reachable by scheme B such that a and b correspond.

2. For every state b reachable by scheme B, either:

[§]It is certainly possible to allow the node and edge types in A to be named differently in B, but for our purposes, renaming nodes and edges merely complicates the notation without tangible benefit.

(a) The state a that corresponds to b is reachable by scheme A, or

(b) There exists some successor state b' of b such that the state a' that corresponds to b' is reachable by scheme A.

The definition of simulation given above is adequate for monotonic models, but not for nonmonotonic models. The reason is that in the nonmonotonic case it is necessary to prohibit the simulating scheme B from leaking rights from $ET(A)$ in an intermediate state where A cannot reach the corresponding state, and then deleting these rights before arriving in a state where A can reach a corresponding state. We do not develop a definition of simulation adequate for nonmonotonic models, such as the definition in [Gan95], since our main result is for monotonic models.

Finally, we formalize the notion of expressive power.

Definition: Model Y is *less expressive than* model X iff there exists at least one scheme A in model X that cannot be simulated by any scheme B in model Y.

Definition: Model Y is *as expressive as* model X iff the following holds: For every scheme A in model X, there exists a scheme B in model Y such that scheme B can simulate scheme A.

Definition: Model X is *equivalent* to model Y iff model X is as expressive as model Y and model Y is as expressive as model X.

3.2 A Nonequivalence Theorem

Before we state and prove the various results below, we describe the scheme A that is used in the proofs. Scheme A has exactly one type of node and one type of edge. There is a single initial state operation that produces an initial state for scheme A with 3 nodes: X_1, X_2, and X_3, and no edges. Scheme A has a double-parent creation operation. The double-parent creation operation creates a child node and introduces an edge from each parent to the child. Scheme A is illustrated in figure 2.

Eventually we wish to show that monotonic single-parent models are less expressive than monotonic multi-parent models. Let us begin, however, with a simpler result to illustrate the technique used in the proof.

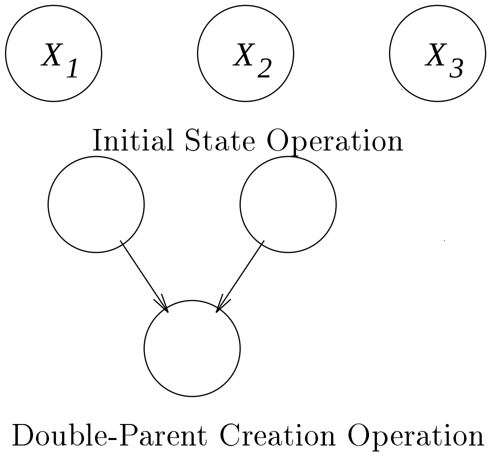
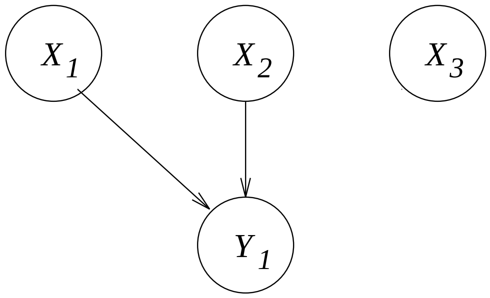


Figure 2: Operations in Schema A

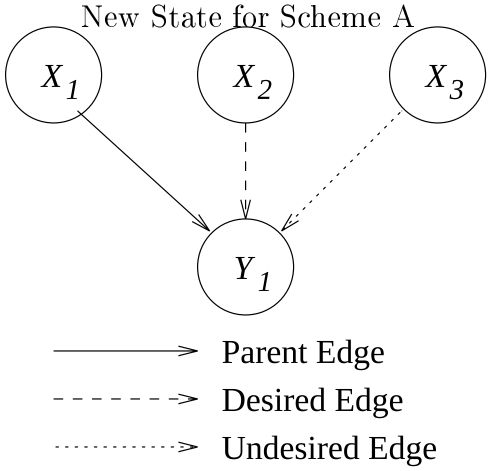
Lemma 1 There is no single-parent scheme B that can simulate scheme A if the initial state for B is identical to the initial state for A.

Proof: The restriction that the initial states be identical means that scheme B cannot encode any information in extra nodes or edges in the initial state. Scheme B cannot simulate the following creation operation in scheme A: scheme A creates a new node, Y_1, using the double-parent creation operation with X_1 and X_2 as parents. The resulting graph has nodes X_1, X_2, X_3, Y_1, and edges $X_1 \rightarrow Y_1$, $X_2 \rightarrow Y_1$, as shown in figure 3.

Let us examine the possible actions of scheme B to see why the simulation fails. Scheme B must use a single-parent creation to create Y_1. Some node must be the parent, so suppose X_1 is the parent and suppose that the single-parent creation adds a single edge $X_1 \rightarrow Y_1$. Scheme B must eventually use some monotonic edge adding operation to introduce the edge $X_2 \rightarrow Y_1$. Since X_2 and X_3 have no distinguishing characteristics, we can mimic the operations used to introduce the edge $X_2 \rightarrow Y_1$ to similarly introduce the edge $X_3 \rightarrow Y_1$. But since scheme A has no edge adding operations, it is clear that Y_1 cannot have an in degree of 3 in any state of scheme A. Therefore, the simulation is broken. In summary, scheme B cannot introduce node Y_1 and edges $X_1 \rightarrow Y_1$ and $X_2 \rightarrow Y_1$ without also allowing



New State for Scheme A



Noncorresponding B State Via Single-Parent Creation

Figure 3: Identical Initial States: Simulation by B Fails

edge $X_3 \to Y_1$, which corresponds to an unreachable state in scheme A. The argument is summarized in figure 3. $\square$

The argument captured by Lemma 1 is insufficient to show in general that a single-parent scheme B cannot simulate scheme A, because scheme B may anticipate the creation of Y_1 with an encoding in its initial state. To see one possible way in which this encoding might be done, suppose that there is a node, Y_1', and edges $X_1 \to Y_1'$ and $X_2 \to Y_1'$ that arise from an initial state operation in scheme B. The simulation is free to use such nodes and edges as long as they are of separate types from the types in scheme A. Scheme B can use single-parent creation with parent Y_1' to create child Y_1. Scheme B can further use monotonic edge-adding operations to create edges $X_1 \to Y_1$ and $X_2 \to Y_1$ by referring to the existing edges $X_1 \to Y_1'$ and $X_2 \to Y_1'$. By exploiting the structure encoded in the initial state of the simulation, scheme B avoids adding the edge $X_3 \to Y_1'$, and thus the simulation is correct on this step.

There are many possible encodings that can anticipate the double-parent creation operations by nodes in the original scheme. However, the trick of encoding these possibilities into the initial state cannot always be applied, as is shown in the proof of the main theorem below. Informally, we wish to show that multi-parent creation adds expressive power unavailable in single-parent models. Formally, we show:

Theorem 1 Monotonic single-parent models are less expressive than monotonic multi-parent models.

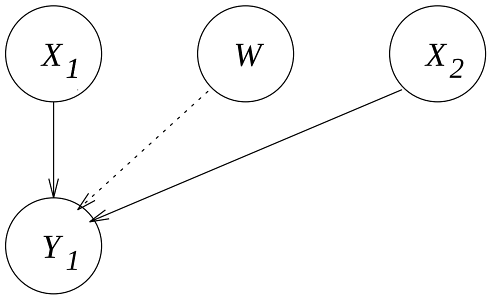
Proof: We prove the theorem by showing that the two-parent scheme A cannot be simulated by any monotonic single-parent scheme B. Thus, no matter which schemes a monotonic single-parent model includes, it can not be as expressive as a multi-parent model.

The proof again proceeds by contradiction, but the details are more involved than in Lemma 1. We first show that if a single-parent scheme B can simulate the double-parent scheme A, then scheme B must have certain properties. We use these properties to show that scheme B can reach a state that corresponds to an unreachable state in scheme A. The construction is illustrated in figure 4.

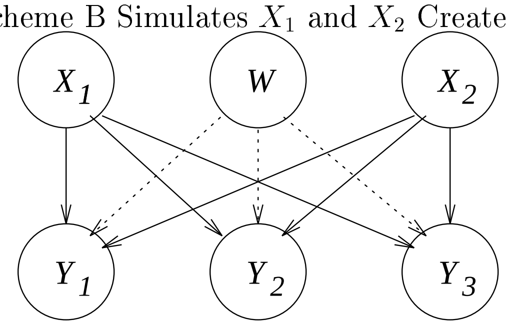
Consider a candidate scheme B that is claimed to be able to simulate scheme A. Let scheme A perform the following creation: X_1 and X_2 produce child Y_1 and edges $X_1 \to Y_1$ and $X_2 \to Y_1$ with double-parent creation. In scheme B, there must be some node in the simulation, call it W, that performs a single-parent creation of Y_1. It does not matter whether W is X_1, X_2, or some other node. Further, the simulation must arrange the introduction of the edges $X_1 \to Y_1$ and $X_2 \to Y_1$.

The key observation in the proof is that the single-parent creation operation in scheme B may be invoked repeatedly with W as the parent, and that the resulting states must correspond to reachable states in scheme A.¶ Thus a reachable state in scheme B is for W

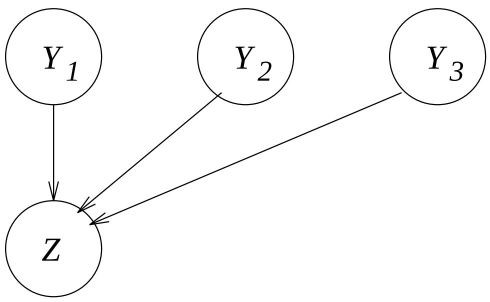
¶Technically, the reachability of corresponding states in A applies to either the resulting states of scheme B or to successors of these states.



Scheme B Simulates X_1 and X_2 Create Y_1



Scheme B Reaches Acceptable Corresponding State



Scheme B Reaches Noncorresponding State

Figure 4: Arbitrary Initial States: Simulation by B Fails

to create two more children, Y_2 and Y_3. The monotonic edge adding operations used to produce edges $X_1 \rightarrow Y_1$ and $X_2 \rightarrow Y_1$ can also be mimicked to produce edges $X_1 \rightarrow Y_2$, $X_2 \rightarrow Y_2$, $X_1 \rightarrow Y_3$, and $X_2 \rightarrow Y_3$. So far, scheme B is in good shape, in that scheme A can certainly reach the state in which X_1 and X_2 have produced three children, Y_1 , Y_2 , and Y_3 with double-parent creation. The difficulty is that Y_1, Y_2, and Y_3 are indistinguishable in scheme B. Any edge adding operation in scheme B that can add an edge terminating at Y_1 can also be duplicated to add a similar edge that terminates at Y_2 or Y_3. Also, any edge adding operation in scheme B that can add an edge originating at Y_1 can also be duplicated to add a similar edge originating at Y_2 or Y_3. We exploit this fact to break the simulation. In scheme A, let Y_1 and Y_2 produce child Z with double-parent creation. In the simulation, Z must be produced with single-parent creation. No matter which node in the simulation is the single-parent of Z, an edge adding operation must be invoked to add at least one of the edges $Y_1 \rightarrow Z$ or $Y_2 \rightarrow Z$. This edge adding operation can also be duplicated to introduce the edge $Y_3 \rightarrow Z$. But then the simulation has reached a state that does not correspond to a reachable state in scheme A, because Z has in degree 3, which no node produced by A has.

□

3.3 Non Monotonic Operations

We now illustrate that the nonequivalence of single-parent and double-parent creation schemes shown in Theorem 1 does not hold in the presence of nonmonotonic operations.

We define a limited form of nonmonotonicity for edge adding operations by allowing the destruction of edges in a graph.

Definition: An edge adding operation is *nonmonotonic* if it destroys an existing edge.

Definition: A scheme is *nonmonotonic* if it includes any nonmonotonic operation.

Definition: A model is *nonmonotonic* if it includes a nonmonotonic scheme.

Note that changing the type of an edge, which is one way of viewing the transformation discussed below, is also nonmonotonic, since it amounts to destroying an existing edge and adding a new one.

Observation 1 Nonmonotonic single-parent models are as expressive as monotonic multi-parent models.

We establish Observation 1 by construction. We exhibit a single-parent creation operation and nonmonotonic edge-adding operation that achieves the same state as a monotonic double-parent creation operation.

Again we denote scheme A as the original and scheme B as the simulation. Consider a double-parent creation operation in scheme A. Scheme B simulates the operation by performing single parent creation of the new node with a special type of edge called a *pre-parent* edge. Scheme B then invokes a nonmonotonic edge adding operation that has as inputs the original parent, the second parent, and the child. The edge adding operation deletes the pre-parent edge and replaces it with a permanent edge. The operation also adds a edge between the second parent and the child. Since the operation destroys the pre-parent edge, there is no possibility of a third node acquiring an edge to the child via the nonmonotonic operation. Figure 5 illustrates the construction.

3.4 Discussion

The two results given above outline the extent to which single-parent creation and multi-parent creation differ. In monotonic schemes, the two operations have different expressive power. In non-monotonic schemes, other nonmonotonic operations can simulate multi-parent creation. Nonetheless, the results offer a compelling case for considering multi-parent creation as a fundamental operation in access control models.

4 Applying The Results To Standard Models

It is observed in [AS90, San92b] that multi-parent creation is a natural and obvious choice to implement a variety of access control policies, such as protected subsystem, confinement, mutual suspicion, originator control, and some cases of separation of duties. Such observations lend only informal support to the conjecture that single-parent creation is less expressive than multi-parent creation. To date, formal support of such a position has been lacking.

In the previous section it was formally demonstrated that single-parent creation is less expressive than multi-parent creation in the context of a monotonic, abstract graph model. The nature of the proof indicates that the results apply to any monotonic access control

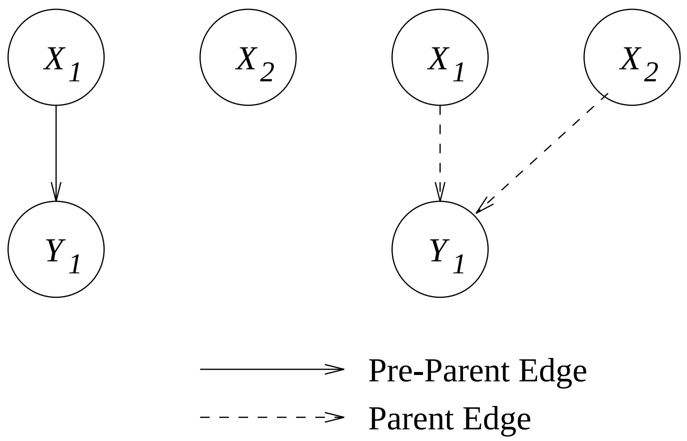


Figure 5: Nonmonotonic Simulation

model for which subject creation is defined and for which the number of parents in a creation operation can be enumerated. In this section we carry out the direct but necessary task of extending the results to standard access control models.

4.1 Application To SPM and ESPM

SPM and ESPM are monotonic access control models whose only difference is that SPM has single-parent creation whereas ESPM has multi-parent creation. Other operations in either model can be directly mapped to edge-adding operations in the abstract graph model. Thus this paper gives a formal demonstration that SPM and ESPM have different expressive power.

4.2 Application to Access Matrix Models

In the monotonic HRU access matrix model there is no explicit separation between creation operations and operations that add rights to cells in the matrix. A given HRU command may create any number of new subjects and objects and enter arbitrary values into new and existing cells.

However, it is not difficult to classify monotonic HRU operations as to the number of parents and children involved. Define a single-parent HRU creation operation to be an operation that has $N > 1$ arguments, of which exactly 1 exists prior to the command and the remaining $N - 1$, if the command is successful, are created by the command. Define a multi-parent HRU creation operation to be an operation that has $N > 2$ arguments, of which at least two exist prior to the command and at least one does not. In either case, the number of children is simply the number of arguments minus the number of parents. All other monotonic operations may be classified as edge adding operations.

Thus the abstract graph model can be mapped directly to monotonic HRU. Generalizations of HRU, such as TAM, Sandhu's Typed Access Matrix Model [San92b], are also covered.

4.3 Multi-Child vs. Single-Child Operations

Unlike the difference between multi-parent and single-parent creation operations, multi-child and single-child creation operations share equivalent expressive power, even in monotonic models. One verification of this assertion is the simulation of ESPM by monotonic HRU in [AS92]. In that construction, the monotonic HRU operations used to simulate ESPM are all single-child. Another verification is the reverse process of simulating monotonic HRU with ESPM. Since all operations in ESPM are single child, and since ESPM is equivalent in expressive power to monotonic HRU, multi-child creation does not add to expressive power.

5 Conclusion

In this paper we have presented an abstract framework for comparing different access control models. We have used the framework to demonstrate that single-parent creation is less expressive than multi-parent creation in monotonic access control models. Although non-monotonic models can simulate multi-parent creation, the results from monotonic models argue for considering multi-parent creation as a fundamental primitive operation.

The results from the abstract framework are incorporated into access control models from the literature. We have applied the results in this paper to show that the Schematic Protection Model (SPM), which has single-parent creation, is less expressive than the Extended Schematic Protection Model (ESPM), which has multi-parent creation. We have also applied the results to the access matrix model of Harrison, Ruzzo and Ullman (HRU) and to Sandhu's Typed Access Matrix Model (TAM), a generalization of HRU that incorporates the notion of protection types. Without loss of expressive power, HRU and TAM may be formulated so as to classify operations as multi-parent or single-parent. In monotonic HRU and TAM, single-parent creation is strictly less expressive than multi-parent creation. In nonmonotonic HRU and TAM, multi-parent creation can be simulated with nonmonotonic operations.

References

[AS90] P.E. Ammann and R.S. Sandhu. Extending the creation operation in the schematic protection model. In *Sixth Annual Computer Security Application Conference*, pages 340–348, Tucson, AZ, December 1990.

[AS92] P.E. Ammann and R.S. Sandhu. The extended schematic protection model. *The Journal Of Computer Security*, 1(3&4):335–384, 1992.

[Gan95] S. Ganta. *The Expressive Power of Access Control Models.* PhD thesis, George Mason University, Fairfax, VA 22030, 1995.

[HR78] M.H. Harrison and W.L. Ruzzo. Monotonic protection systems. In DeMillo et al, editor, *Foundations of Secure Computations*, pages 337–365. Academic Press, 1978.

[HRU76] M.H. Harrison, W.L. Ruzzo, and J.D. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, 1976.

[Lam71] B.W. Lampson. Protection. In *5th Princeton Symposium on Information Science and Systems*, pages 437–443, 1971. Reprinted in *ACM Operating Systems Review* 8(1):18–24, 1974.

[Lan81] C.E. Landwehr. Formal models for computer security. *ACM Computing Surveys*, 13(3):247–278, 1981.

[LS77] R.J. Lipton and L. Snyder. A linear time algorithm for deciding subject security. *Journal of the ACM*, 24(3):455–464, 1977.

[San88] R.S. Sandhu. The schematic protection model: Its definition and analysis for acyclic attenuating schemes. *Journal of the ACM*, 35(2):404–432, April 1988.

[San92a] R.S. Sandhu. Expressive power of the schematic protection model. *The Journal Of Computer Security*, 1(1):59–98, 1992.

[San92b] R.S. Sandhu. The typed access matrix model. In *Proceedings IEEE Computer Society Symposium on Research In Security and Privacy*, pages 122–136, Oakland, CA, May 1992.

[San92c] R.S. Sandhu. Undecidability of safety for the schematic protection model with cyclic creates. *Journal of Computer and System Sciences*, 44(1):141–159, February 1992.

[Sny81] L. Snyder. Formal models of capability-based protection systems. *IEEE Transactions on Computers*, C-30(3):172–181, 1981.