# The Expressive Power Of Multi-Parent Creation In Monotonic Access Control Models

Paul Ammann† Richard Lipton‡ Ravi S. Sandhu†

†Center for Secure Information Systems, George Mason University, Fairfax, VA
‡Department of Computer Science, Princeton University, and MITL, Princeton, NJ

## 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 in monotonic models. 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. 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 - [13]) and the Extended Schematic Protection Model (ESPM - [1]). We also show how the results apply to the typed access matrix model (TAM - [15]), which is an extension of the well known access matrix model of Harrison, Ruzzo, and Ullman (HRU - [8]). The results in this paper offer theoretical justification for regarding single-parent and multi-parent creation as fundamentally different operations in a monotonic context.

## 1. Introduction

Access control models provide a formal expression for security policies concerning shared resources in multi-user computer systems. Access control models are typically expressed in terms of subjects, objects, and access rights, concepts with which we assume the reader is familiar. 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, *i.e.* can a given subject ever acquire a given right to some other subject or object. 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 necessary to allow the expression of certain policies. In this paper we examine two primitive operations, namely single-parent creation and multi-parent creation, and demonstrate that the latter is fundamentally more expressive than the former.

The safety question for access control was first formulated in the context of the access matrix model of Harrison, Ruzzo, and Ullman, denoted here as HRU [8]. Although HRU has broad expressive power, for most interesting schemes it has undecidable safety analysis. The search for tractable safety analysis led to proposals for a number of models [4, 9, 10, 11, 12, 16]. However, a substantial gap in expressive power exists between these models and HRU. Sandhu's Schematic Protection Model, denoted here as SPM [13], 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 [13, 14]. SPM has remarkably strong safety properties and has been shown to represent a wide variety of cases of practical interest.

In SPM the creation operation for new subjects and objects employs only a single parent. However, multi-parent creation is a desirable operation many practical applications. For example, variations on the mutual suspicion problem, such as the protected subsystem and confinement problems, have solutions that are naturally implemented with multi-parent creation [1]. ORCON, or originator control problems, also have natural multi-parent solutions [15]. Separation of duties, in which the joint authorization by multiple subjects is commonly required, also fits naturally into a multi-parent framework [1].

To accommodate multi-parent creation, the Extended Schematic Protection Model, denoted here as ESPM [1, 3], 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 [2].



0-8186-2850-2/92 $3.00 © 1992 IEEE

Many proposed access control models are monotonic, including SPM, ESPM, and restricted versions of HRU and Sandhu's Typed Access Matrix (TAM - [15]), an extension of HRU that incorporates the notion of protection types. In monotonic models, subjects, objects, and access rights may not be destroyed once they have been created. † The advantage of monotonic models is that the safety analysis need not use backtracking, in that any given change to the protection state does not exclude another change from taking place. 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.

Monotonic HRU and ESPM were shown to have equivalent expressive power in [3]. The expressive power of SPM was not formally addressed by the proof, but the demonstration of HRU-ESPM equivalence strongly suggested that single-parent creation is theoretically less powerful than multi-parent creation in monotonic models. In this paper we formally demonstrate that in monotonic access control models, multi-parent creation is strictly more powerful than single-parent creation. An immediate corollary is that SPM is less expressive than ESPM, and hence monotonic HRU and monotonic TAM.

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 the 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.

† An additional constraint is that preconditions for operations by which the state evolves may not be expressed with the negation operator. The constraint on negation applies not only to monotonic models, but also to some nonmonotonic ones as well, such as nonmonotonic HRU. Although negation can express useful security policies, such as mutually exclusive access rights, its introduction complicates the analysis process severely.

## 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 data type. As usual, the abstract data type 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 data type 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, *i.e.* none of the operations defined for the abstract data type 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 data type, 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

‡ 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.

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 alter or delete an existing node or edge in a protection graph. At the end of the next section we show the secondary result that nonmonotonic models can simulate multi-parent creation with single-parent creation. Accordingly, a limited form of nonmonotonicity will be 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 a single new node, and possibly one or more edges that terminate at the new node, to the protection graph. Creation operations are classified by the number of parent nodes that participate. Typically, a creation operation ties each parent node to child node with one or more edges, which we refer to here as *parent edges.* † The presence of a single parent edge is sufficient for edge adding operations to install additional parent edges and/or edges from the child to the parent. To simplify our discussion, we hereafter assume, unless otherwise indicated, that node creation operations install a single parent edge from each parent to the child.

For example, a single-parent creation operation produces a single new node in the protection graph and creates a parent edge from the specified parent node to the new node. Similarly a double-parent creation operation produces a single new node in the protection graph and creates two parent edges, one from each specified parent to the new node.

Depending on the number of node and edge types, there may be many permutations of these operations, such that the types of the parent nodes, the child nodes, and the parent edges, are all taken into account. For example, a double-parent creation operation may state that one parent be of type $t_1$, the other parent be of type $t_2$, that the resulting child be of type $t_3$, and that both parent edges be of type $t_4$.

† While the inclusion of parent edges in node creation operations is optional, it is noted that if such edges are omitted, then the parent plays no special role with respect to the child.

It turns out that multi-parent creation can be simulated with double-parent creation, a result shown in the context of the ESPM model in [3]. ‡ We invoke this result to avoid direct consideration of node creation with more than two parents. Accordingly, the theorems in the next section are all proved in terms of single-parent vs. double-parent creation. By the theorem in [3], however, the results generalize to single-parent vs. multi-parent creation.

Operations that add edges are subject to two constraints. Such operations:

(1) may not create new nodes.

(2) must be monotonic.

The first constraint specifies that all nodes must be created either by the initial state operation or by node creation operations. The second constraint specifies that if an operation ever becomes applicable, then it must remain applicable. In the abstract data type view, 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.

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

Aside: Since the structure we have described is an abstract data type, schemes are easily captured with formal specification notations for abstract data types. For example, abstract model specifications such as Z [17] can be used to represent schemes directly.

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 $ET(S)$. We designate the state at some specified time $t$ by $G(S, t)$. At $t_0$, the initial time in a scheme, the state $G(S,t_0)$ corresponds to the result of one of the initial state operations.

Schemes are classified according to the largest number of parents that can participate in a creation operation. Thus all creation operations in a single-parent scheme have exactly one parent. Creation operations in a double-

‡ The simulation of multi-parent creation with double-parent creation requires the assumption that a single node may redundantly function as more than one parent in a multi-parent creation operation. Motivation for this assumption may be found in [3].

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.

We adopt a strong notion of correspondence between schemes. We require the simulating scheme, B, to maintain as part of its state a subgraph that corresponds exactly to the entire state for scheme A. We implement this requirement as follows. First, the set of node (edge) types in the simulation scheme must be a superset of the set of node (edge) types in the original scheme, *i.e.* $NT(A) \subseteq NT(B)$ and $ET(A) \subseteq ET(B)$. Second, any node (edge) in the simulation that is of the same type as a node (edge) in the original must actually correspond to a node (edge) in the original. Thus scheme B cannot contain extraneous nodes (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.

At any time $t$ when B is successfully simulating A, $G(A, t) \subseteq G(B, t)$ and, if we only consider nodes (edges) in $NT(A)$ ($ET(A)$), then the two graphs are the same. We formalize the notion as follows:

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 (edges) not in $NT(A)$ ($ET(A)$).

As an aside, we discuss briefly the decision to require that the graphs in question be identical rather than simply isomorphic. The use of the more general construct, *i.e.* isomorphism, does not add to the generality of the simulation, but it does complicate the discussion. Specifically, we disallow mappings in which the correspondence of a node (edge) in the simulation to a node (edge) in the original changes from one state to the next. The correspondence is set when a node (edge) in the simulation is created and does not change thereafter.

We have discussed part of the simulation, namely the notion of correspondence. Two other properties are required. The complete definition is as follows:

Definition: Under the definition of correspondence above, scheme B *simulates* scheme A iff the following conditions hold:

(1) If scheme A can reach a given state, scheme B can reach a corresponding state.

(2) If scheme B can reach a given state, scheme A can reach a corresponding state.

Finally, we formalize the notion of expressive power.

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.

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

### 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 fig. 1.

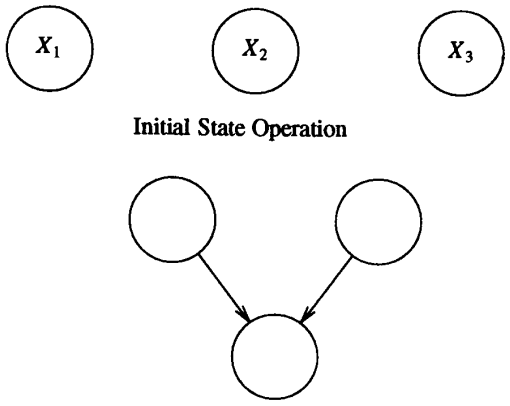



**Fig. 1. Operations In Scheme A.**

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

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 parent edges $X_1 \rightarrow Y_1$, $X_2 \rightarrow Y_1$, as shown in fig. 2.

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 $Y_1$. Without loss of generality, suppose $X_1$ is the parent and suppose that the operation adds a single parent 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 edge $X_3 \rightarrow Y_1$, which corresponds to an unreachable state in scheme A. The argument is summarized in fig. 2. □

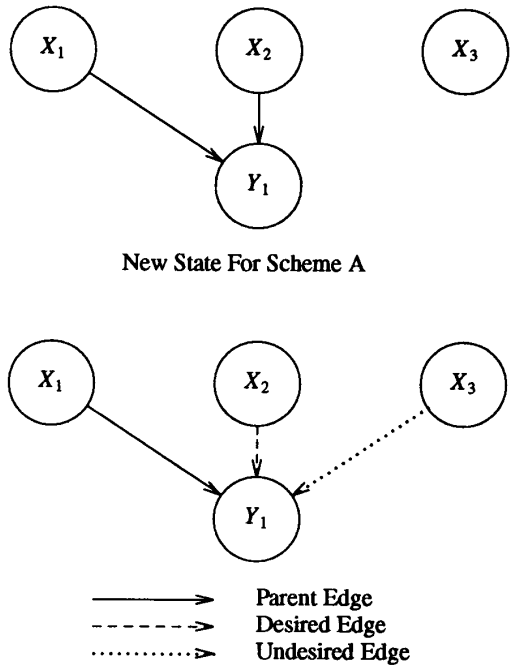



**Fig. 2. Identical Initial States: Simulation By B Fails**

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 $X_1 \rightarrow Y'_1$ and $X_2 \rightarrow Y'_1$ that arises 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 $X_1 \rightarrow Y_1$ and $X_2 \rightarrow Y_1$ by referring to the existing edges $X_1 \rightarrow Y'_1$ and $X_2 \rightarrow Y'_1$. By exploiting the structure encoded in the initial state of the simulation, scheme B avoids adding the edge $X_3 \rightarrow Y'_1$, 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.

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

Proof We prove Theorem 1, using the definition of *more expressive than* given earlier, by showing that the two-parent scheme A cannot be simulated by any monotonic, single-parent scheme B.

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.

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 $X_1 \rightarrow Y_1$ and $X_2 \rightarrow Y_1$ 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 $X_1 \rightarrow Y_1$ and $X_2 \rightarrow Y_1$.

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 results must correspond to reachable states in scheme A. Thus a reachable state in scheme B is for $W$ 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. The construction is illustrated in fig. 3. □

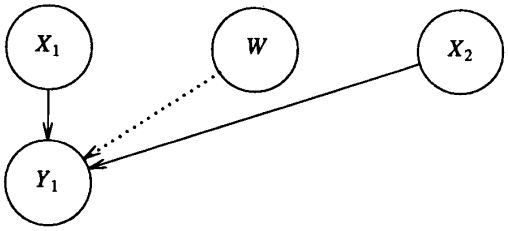
### 3.3. Non Monotonic Operations

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.

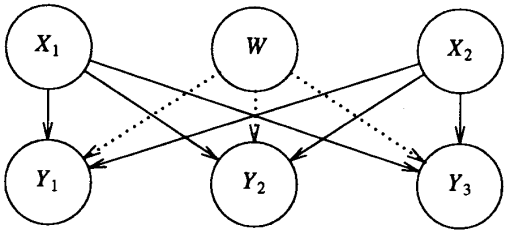
We define a limited form of nonmonotonicity for edge adding operations by allowing the destruction of edges in a graph.

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

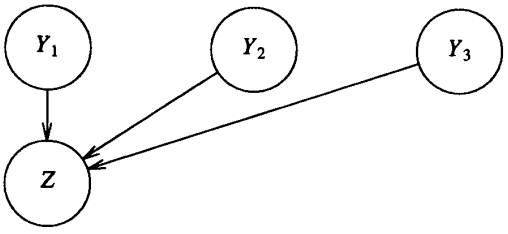
Definition: A scheme is *nonmonotonic* if it includes any nonmonotonic operation.




Scheme B Simulates $X_1$ and $X_2$ Create $Y_1$




Scheme B Reaches Acceptable Corresponding State




Scheme B Reaches Noncorresponding State

**Fig. 3. Arbitrary Initial State: Simulation By B Fails**

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

Theorem 2 Nonmonotonic single-parent models are as expressive as monotonic multi-parent models.

Proof We prove theorem 2 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.

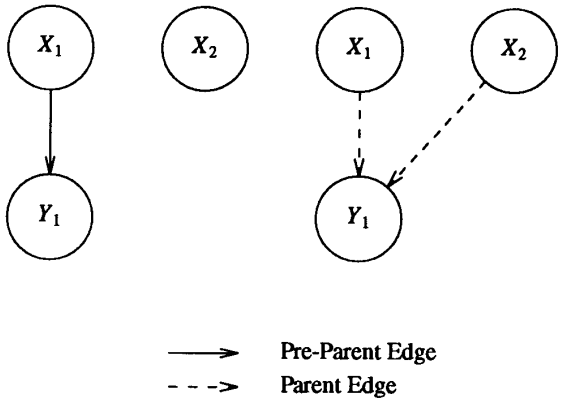



**Fig. 4. Nonmonotonic Simulation $X_1, X_2$ Create $Y_1$.**

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 parent edge. The operation also adds a parent 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 a parent edge to the child. Fig. 4 illustrates the construction. □

### 3.4. Discussion

The two theorems 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 [1, 15] that multi-parent creation is a natural and obvious choice to implement a variety of access control policies, such as mutual suspicion, originator control, and separation of duties. Such observations lend only informal support to the conjecture that multi-parent creation is more expressive than single-parent creation. To date, formal support of such a position has been lacking.

In the previous section it was formally demonstrated that multi-parent creation is more expressive than single-parent creation in the context of a monotonic, abstract graph model. 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 certain HRU operations as either single-parent or multi-parent. Define a single-parent HRU creation operation to be an operation that has two arguments and creates a single new subject or object. One argument is the parent and the other is the new child. Define a multi-parent HRU creation operation to be an operation that has $N>2$ arguments and creates a single new subject or object.[†] $N-1$ arguments are the parents and the remaining argument is the child.

No loss of expressive power is incurred by restricting attention to HRU schemes where all subject and object creation occurs in operations that can be classified as described above. One verification of this assertion is in the proof that ESPM is equivalent to monotonic HRU [3]. In [3], the creation operations used in simulating ESPM with HRU were either multi-parent or single-parent.

Thus the abstract graph model can be mapped to a subset of HRU that is equivalent in expressive power to full HRU. Generalizations of HRU, such as TAM, Sandhu's Typed Access Matrix Model [15], are also covered by our analysis. Operations in TAM can also be classified to to distinguish single and multi-parent creation. Finally, both HRU and TAM can accommodate nonmonotonic formulations.

In summary, the results of the previous section apply to HRU and TAM as follows. Multi-parent creation can be defined easily in monotonic HRU and TAM and is more expressive than single-parent creation. In nonmonotonic HRU and TAM, multi-parent creation can be simulated by nonmonotonic operations.

## 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 nonmonotonic 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, multi-parent creation is strictly more expressive than single-parent creation. In nonmonotonic HRU and TAM, multi-parent creation can

[†] If desired, the addition of parent rights to cells in the matrix can be restricted so as to adhere to the node creation operations described for the abstract graph model. In addition, other varieties of operations can also be included in the classification, but we do not consider such operations here.

be simulated with nonmonotonic operations.

## References


[1] Ammann, P.E. and Sandhu, R.S., "Extending the Creation Operation in the Schematic Protection Model", *Proceedings Sixth Annual Computer Security Application Conference*, Tucson, AZ (1990).

[2] Ammann, P.E. and Sandhu, R.S., "Safety Analysis For The Extended Schematic Protection Model", *Proceedings of the 1991 IEEE Symposium Research in Security and Privacy*, Oakland, CA, May, 1991.

[3] Ammann, P.E. and Sandhu, R.S., "The Extended Schematic Protection Model", *The Journal Of Computer Security*, to appear.

[5] Bell, D.E. and LaPadula, L.J., "Secure Computer Systems: Unified Exposition and Multics Interpretation", *Mitre Technical Report MTR-2997*, Bedford, MA (1975).

[4] Bishop, M. and Snyder, L., "The Transfer of Information and Authority in a Protection System", *7th ACM Symposium on Operating Systems Principles*, 45-54 (1979).

[6] Department of Defense National Computer Security Center, *Department of Defense Trusted Computer Systems Evaluation Criteria*, DoD 5200.28-STD, (1985).

[7] Graham, G.S. and Denning, P.J., "Protection - Principles and Practice", *AFIPS Spring Joint Computer Conference*, 40:417-429 (1972).

[8] Harrison, M.H., Ruzzo, W.L. and Ullman, J.D., "Protection in Operating Systems", *CACM*, 19(8):461-471 (1976).

[9] Jones, A.K., Lipton, R.J. and Snyder, L., "A Linear Time Algorithm for Deciding Security", *17th IEEE Symposium on the Foundations of Computer Science*, 337-366 (1976).

[10] Lipton, R.J. and Budd, T.A., "On Classes of Protection Systems", In *Foundations of Secure Computations*, DeMillo, R.A., Dobkin, D.P., Jones, A.K. and Lipton, R.J. (Editors), Academic Press (1978).

[11] Lipton, R.J. and Snyder, L., "A Linear Time Algorithm for Deciding Subject Security", *JACM*, 24(3):455-464 (1977).

[12] Lockman, A. and Minsky, N., "Unidirectional Transport of Rights and Take-Grant Control", *IEEE Transactions on Software Engineering*, SE-8(6):597-604 (1982).

[13] Sandhu, R.S., "The Schematic Protection Model: Its Definition and Analysis for Acyclic Attenuating Schemes", *JACM*, 35(2):404-432 (1988).

[14] Sandhu, R.S., "Expressive Power of the Schematic Protection Model", *The Journal Of Computer Security*, to appear.

[15] Sandhu, R.S., "The Typed Access Matrix Model", to appear in *Proceedings of the 1992 IEEE Symposium Research in Security and Privacy*, Oakland, CA, May, 1992.

[16] Snyder, L. "Formal Models of Capability-Based Protection Systems", *IEEE Transactions on Computers*, C-30(3):172-181 (1981).

[17] Spivey J.M., *The Z Notation: A Reference Manual*, Prentice Hall, 1989.