The NTree: A Two Dimension Partial Order for Protection Groups

RAVINDERPAL S. SANDHU
The Ohio State University

The benefits of providing access control with groups of users rather than with individuals as the unit of granularity are well known. These benefits are enhanced if the groups are organized in a subgroup partial order. A class of such partial orders, called ntrees, is defined by using a forest of rooted trees or inverted rooted trees as basic partial orders and combining these by refinement. Refinement explodes an existing group into a partially ordered ntree of new groups while maintaining the same relationship between each new group and the nonexploded groups that the exploded group had. Examples are discussed to show the practical significance of ntrees and the refinement operation. It is shown that ntrees can be represented by assigning a pair of integers called *lr-values* to each group so that g is a subgroup of h if and only if l[g] ≤ l[h] and r[g] ≤ r[h]. Refinement allows a complex ntree to be developed incrementally in a top-down manner and is useful for the initial definition of an ntree as well as for subsequent modifications. To make the latter use of refinement practical, a method is presented for assigning lr-values to the new groups introduced by refinement so lr-values assigned to nonexploded groups need not be changed. It is also shown how to guarantee that the lr-values of the exploded group will get assigned to one of the new groups.

Categories and Subject Descriptors: C.0 [**Computer Systems Organization**]: General—*systems specification methodology*; C.2.0 [**Computer-Communication Networks**]: General—*security and protection*; D.2.0 [**Software Engineering**]: General—*protection mechanisms*; D.4.6 [**Operating Systems**]: Security and Protection—*access controls; security kernels*; H.1.0 [**Models and Principles**]: General; H.2.0 [**Database Management**]: General—*security, integrity, and protection*; K.6.m [**Management of Computing and Information Systems**]: Miscellaneous—*security*

General Terms: Design, Management, Security, Theory

Additional Key Words and Phrases: Access control lists, authorization, hierarchies, partial orders, protection groups.

1. INTRODUCTION

The ability to share files and other resources among the users of a system has obvious benefits. It is convenient for both the users and the system administrators to have the facility to specify access based on groups of users as a unit. Membership in a group is presumably determined by the need to share resources and information, so the group provides a suitable unit for an individual user's access decisions. A user can make a file available to an entire group without

Author's Address: Department of Computer and Information Science, The Ohio State University, Columbus, OH 43210.

having to explicitly provide access to every member. Similarly, a file's availability can be revoked from a group without explicitly revoking each member's access. Also, new users can be made members of appropriate groups, thereby obtaining access to a number of files and resources. Some systems, such as the popular UNIX [11], allow for access control only in terms of groups. Even the more sophisticated systems, such as Multics [12], which have provision for specifying access at the level of individual users, recognize the advantages of protection groups and provide facilities for specifying access in terms of groups.

In practice, it is often desirable that groups bear some relationship to each other. For instance, consider a project divided into several independent tasks assigned to different teams. We can define a group for each task team so its members have common access to resources relevant to the task. Since some resources may pertain to the entire project, we can define a project group such that members of the individual task groups are thereby also members of the project group. The project-wide resources are then made explicitly available to the project group alone. This is certainly more convenient than having to explicitly make such resources available to every task group, even if it were possible to do so. It is also more convenient than explicitly making every member of a task group a member of the project group. By allowing membership in a group to automatically imply membership in some other groups, we can reduce the number of explicit access decisions that need to be made by the users, as well as reduce the number of groups to which a user must explicitly belong.

Let G be a set of groups and let $g \preceq h$ signify that group g is a *subgroup* of group h, in the sense that every member of g is thereby also a member of h. Note that members of g have more privileges than members of h. We require that the subgroup relation is a partial ordering of G, i.e., $\preceq$ is reflexive, transitive, and asymmetric. The reflexive property is obviously required since every member of g is already a member of g. Transitivity is certainly an intuitive and reasonable assumption and perhaps even inevitable. After all, if $g \preceq h$ and $h \preceq k$, then every member of g is a member of h and so should also be a member of k. Once the reflexive and transitive requirements are accepted, the asymmetric requirement merely eliminates redundancy by excluding groups that would otherwise be equivalent. If g is a proper subgroup of h we write $g \prec h$; that is, $g \preceq h$ and $g \neq h$.

We say a user is a *direct* member of g if the user is explicitly designated as a member of g and thereby is an *indirect* member of every h such that $g \prec h$. The intention is that a user will be a direct member of a small number of unrelated groups, perhaps just one, but will thereby obtain indirect membership in a larger number of groups. We say that a file or other resource is *explicitly available* to group g if the access control information associated with the file (perhaps in an access-control list) makes explicit mention of group g. The file is thereby *implicitly available* to every proper subgroup h of g. Again the intention is that a file will be explicitly available to a small number of unrelated groups, but implicitly available to a larger number of groups.

Consider again the project example mentioned earlier. Let t_1 and t_2 be task groups and p the project group. If there is no group organization we have essentially two methods for solving the protection and sharing problem. We can

make the project resources explicitly available to both t_1 and t_2 and ignore p. We say this solution is based on *resource and file assignment.* The other solution is to explicitly make every member of t_1 and every member of t_2 a member of p. We say this solution is based on *user assignment.* Both solutions have the undesirable property of introducing redundancy. In the latter case, whenever a new member is added to a task team we will need to also add him or her to the project group. In the former case, suppose a new task group t_3 is created. The protection information associated with all project files will need to be modified to explicitly mention t_3. There is also the possibility of keeping all project files in a single directory and treating the protection attributes of the files as a property of the directory, so the protection information to be modified is in one place. This approach shifts the redundancy problem from files to directories, which may be more manageable, at the cost of lost flexibility in assigning protection attributes of individual files.

On the other hand, if a subgroup relation can be defined, we can specify that task groups are subgroups of the project group and the redundancy problem disappears. For this reason many contemporary systems offer a facility for defining group organization. It appears that two basic approaches have been used thus far. One approach, which we call *explicit group organization,* is to enumerate the subgroups of a group. TOPS-20 is a well known example of this approach. To define a group we need to enumerate the user identifiers of the direct members anyway, and it is reasonable to also enumerate the identifiers of subgroups. If only the immediate subgroups are explicitly enumerated and transitivity is handled by the access control mechanism by following the chains of subgroups, we have a low redundancy solution. However, the mechanism may have to chase several chains of subgroups before arriving at an access decision. If all subgroups are explicitly enumerated we simplify the access control mechanism but reintroduce redundancy.

The second approach for implementing a subgroup relation is based on *implicit group organization* and requires that groups be named so the group organization is reflected in the group names [12]. In our project example we might name the task groups as p/t_1 and p/t_2 while the project group is named as p/p. By use of wildcards we can identify all three groups by p/*, for instance. The access control information for project files can contain explicit mention of p/*, meaning that access is available to all groups that match this pattern. This is a low-redundancy solution, since task group p/t_3 can be created and its members will immediately have access to the project files. Also, a new member of a task group need only be assigned to that task group. The biggest drawback of this approach is the need to set up group names with great care so the wildcard facility can accommodate the group organization. Another drawback is that the users must keep the naming conventions in mind when defining the protection attributes of their files. We feel this is an unreasonable burden for the users. Moreover, we may need several fields in the group names to achieve this effect. For instance, in a rooted tree hierarchy we would need a different field for each level of the tree.

In this paper we propose a class of partial orders, called ntrees, which have a very simple implicit representation and yet appear to cover a large variety of practical situations. We show that ntrees can be represented by assigning a pair

of integers called *lr-values* to each group so that $g \prec h$ if and only if $l[g] < l[h]$ and $r[g] < r[h]$. We also show that new groups can be introduced in ntrees in certain natural ways without affecting the lr-values of existing groups.

NTrees are a method for implicit group organization that can be efficiently implemented and that directly solve a large class of common protection problems. By their very nature, complex ntrees can be incrementally constructed in a top-down manner, using simple ntrees at each step. We are well aware that ntrees do not solve all practical problems directly. Subgroup hierarchies outside the direct scope of ntrees must be handled by an explicit group organization in addition to the implicit ntree structure, or by user or resource assignment. We recommend that deviations from an ntree be handled by explicit group organization or by user assignment rather than by file assignment, if possible. This puts the responsibility for the subgroup organization on the security administrator, where it properly belongs.

To summarize our position, in general, protection problems must be solved by a combination of implicit and explicit group organization and user and resource assignment. Solutions based on resource assignment are undesirable since they place excessive burden on the user. It is no doubt desirable that flexibility in file assignment be available to the users, but it is inappropriate to expect users to understand complicated conventions for solving their protection and sharing requirements. NTrees are a powerful technique for implicit group organization that support many commonly occurring hierarchies directly. The issue of mapping an arbitrary partial order into an ntree by additional explicit group organization and user assignment is an important one, but it is outside the scope of this paper. These issues cannot even be addressed until the implicit group organization is defined. In this sense, the implicit group organization problem is more fundamental, and is the one discussed in this paper.

2. TWO DIMENSION PARTIAL ORDERS

We begin by reviewing the mathematical basis for ntrees and their implicit representation by lr-values. Partial orderings are conventionally depicted by Hasse diagrams as shown in Figure 1, for instance. The partial order represented by a Hasse diagram is obtained by directing the edges downwards; for example, from a to b in Figure 1(a), indicating that a is a subgroup of b, and taking the transitive and reflexive closure of the resulting directed graph.

Every partial order P on a set of elements G can be extended to a linear ordering of G, by the familiar procedure of topological sorting. In general there will be more than one linear extension of P. Let $\Gamma(P)$ be the collection of all linear extensions of P. The intersection of linear orderings $L_1, L_2, \ldots, L_k$ is defined as the set of ordered pairs $\{(u, v) \mid (u, v) \in L_1 \wedge (u, v) \in L_2 \cdots \wedge (u, v) \in L_k\}$. A *realizer* of P is a subset of $\Gamma(P)$ whose intersection equals P. It is easy to see that $\Gamma(P)$ is a realizer of P, so a realizer always exists. The *dimension* of a partial order P, written as dim(P), is the size of the smallest realizer of P [3]. A partial order has dimension one if and only if it is a linear ordering. The partial order of Figure 1(a) has a size two realizer consisting of the linear orderings abcd and acbd, so its dimension is two. Similarly, the partial order of Figure 1(b) has dimension two because it has a realizer abcd and cadb. The partial order of Figure 1(c) has dimension three. The results of dimension theory cited in this

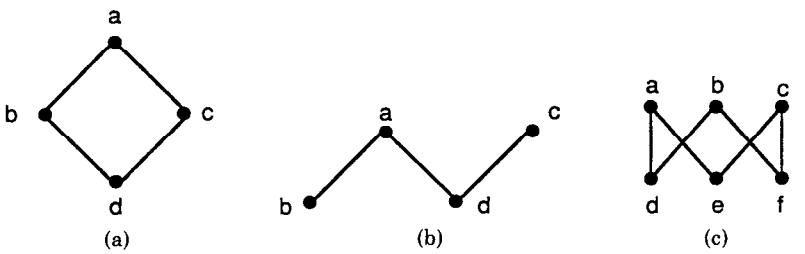


Fig. 1. Hasse diagrams.

paper are all available in the books by Fishburn [4] and Golumbic [5], often with simpler proofs than the original source.

In this paper we restrict the subgroup relation $\preceq$ to be a partial order with dimension two or less. We now motivate this restriction. If $\dim(\preceq) \leq k$ we can represent the subgroup partial ordering on a set of groups G by assigning a k-tuple of integers to each group in G as follows. Let $L_1, L_2, \ldots, L_k$ be a realizer of $\preceq$. The i^{th} component of the k-tuple assigned to g is the position of g in the linear ordering L_i. It follows that $g \preceq h$ if and only if each component of the k-tuple assigned to g is less than or equal to the corresponding component of the k-tuple assigned to h. We can then determine whether one group is a subgroup of another by a component-wise comparison of fixed size tuples of integers. This representation will be useful if k is much smaller than $|G|$ (the size of G). It is known that for every set G of size six or more, there exists a partial ordering on G with dimension $\lfloor 1/2 \mid G \mid \rfloor$ [7]. If we fix the upper bound on $\dim(\preceq)$ at some small value while permitting a large number of protection groups, it follows that we cannot allow $\preceq$ to be an arbitrary partial order. An upper bound of one amounts to assuming that $\preceq$ is a linear ordering, which is not very useful. So the smallest useful upper bound on $\dim(\preceq)$ is two.

It turns out that the class of partial orders with dimension less than or equal to two includes several cases of practical importance. A partial order whose Hasse diagram is a rooted tree, as in Figure 2(a) for instance, has dimension two. A size two realizer for a rooted tree is easily computed by a left-to-right preorder traversal L and a right-to-left preorder traversal R as demonstrated in Figure 2. This tree can then be represented by assigning the pair of integers $\langle 1, 1 \rangle$ to a, $\langle 2, 6 \rangle$ to b, and so on. By a component-wise comparison of these pairs we can determine for instance that $a \prec g$, whereas c and g are incomparable. Similarly, a partial order whose Hasse diagram is an inverted rooted tree, as in Figure 2(b) for example, has dimension two. A size two realizer for an inverted rooted tree is easily computed by reversing the size two realizer for the corresponding rooted tree. A proof of these observations follows.

THEOREM 1. *A partial order whose Hasse diagram is a rooted tree or an inverted rooted tree has a realizer of size two.*

PROOF. By the above discussion it suffices to consider the case of a rooted tree. Let T be a rooted tree, with left-to-right preorder traversal L and right-to-left preorder traversal R. If $(u, v) \in T$, then clearly u precedes v in both L and R, so $(u, v) \in L \cap R$. If $(u, v) \notin T$, without loss of generality, let the path in the

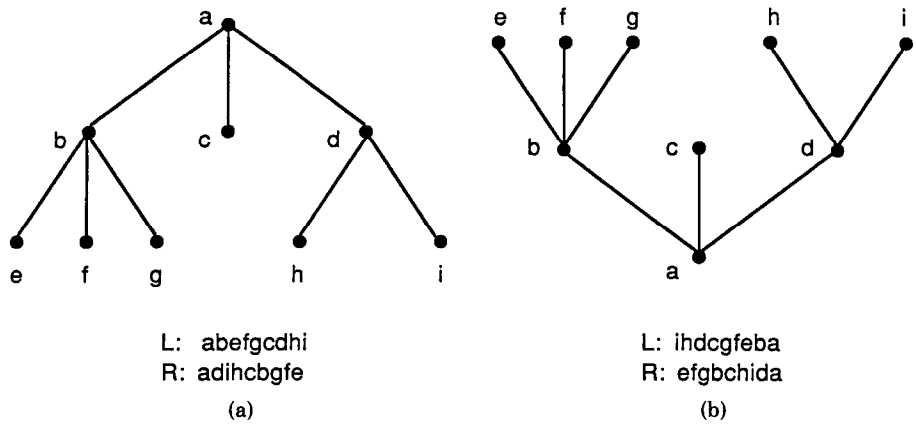


Fig. 2. A tree and an inverted tree.

tree from the root to u be to the left of the path from the root to v. But then u precedes v in L and follows v in R, so $(u, v) \notin L \cap R$. □

More generally, a partial order whose Hasse diagram consists of a forest of mutually disjoint trees and inverted trees has a realizer of size two, as will be proved shortly. The partial orders of Figures 1(a) and 1(b) are examples of dimension two partial orders with Hasse diagrams other than trees.

There are partial orders of practical importance that have dimension greater than two. For instance, in Figure 1(c) the a, b, and c groups might represent project teams while the d, e, and f groups represent different categories of resources. Figure 1(c) is a particular case of the more general situation where $p_1, \ldots, p_n$ are project groups, $r_1, \ldots, r_n$ are resource groups, each project group needs resources in all but one of the resource groups, and it so happens that project p_i needs resources in groups r_j, $j \neq i$. The resulting subgroup relation is $\{(p_i, r_j) \mid i \neq j\}$. For $n \geq 3$ the dimension of this partial order is n [3]. Another situation of practical importance with possibly high-dimension partial orders arises in military security policies [1, 2, 9]. Let S be the set of compartments whose subsets determine the categories and let 2^S be the power set of S, that is, the set of all subsets of S. The dimension of the set inclusion partial order on 2^S is $|S|$ [8]. Since this partial order can be represented using $|S|$ bits for each subset of S, the dimension approach is clearly not useful for this case.

Although some of the theory we develop (Section 4.1 in particular) is applicable to partial orders of arbitrary dimension, we are skeptical about whether the use of an upper bound such as three or four will provide substantially greater benefit than our proposal of limiting the dimension to two. There will remain practical cases where the need for a higher dimension can be argued. As discussed in the Introduction, our objective is to develop an implicit group organization that can be efficiently implemented and that directly solves a large class of commonly occurring situations. We are proposing the class of ntrees for this purpose, which are a proper subset of the two-dimension partial orders. We reiterate that ntrees do not solve all practical requirements for protection groups directly and

deviations from the ntree organization must be handled by additional explicit group organization or by user assignment.

A major advantage of two-dimension partial orders is that their theory is well understood and quite simple. One of the significant results is that two-dimension partial orders can be efficiently recognized and a size two realizer can be efficiently computed by the following procedure. The incomparability graph of a partial order has an undirected edge connecting g and h if g and h are incomparable. A partial order has dimension less than or equal to two if and only if its incomparability graph is transitively orientable [3]. A polynomial algorithm for recognizing transitively orientable graphs and computing the orientation exists [10], from which a size two realizer is easily obtained. An efficient method is also known for determining all possible realizers of size two [5]. For dimension three and higher, it is not even known whether we can efficiently determine the dimension of a partial order without enumerating all possible realizers [5].

The rest of the paper is organized as follows. In Section 3 we review the operation of refinement by which a node in a Hasse diagram is replaced by another Hasse diagram. The significant property is that refinement allows us to develop complex Hasse diagrams in a top-down fashion without increasing the dimension. This leads us to define the class of partial orders called ntrees that are constructed by refinement using a forest of rooted trees or inverted rooted trees at each step so the dimension is no greater than two. We discuss examples to show the practical importance of ntrees for protection groups. In Section 4 we discuss how the subgroup partial order can continue to be developed by refinement even after the system has been in operation for a while. The important consideration is that this should require only an incremental change in representation of the subgroup partial ordering. The representation developed in Section 4.1 is actually applicable to partial orders of any dimension, while the representation of Section 4.2 applies only to ntrees. Section 5 concludes the paper.

3. NTREES

A fundamental result of dimension theory allows us to construct new partial orders from existing ones without increasing the dimension [6]. Let P and Q be partial orders on disjoint sets G and H, respectively. Consider some $u \in G$. The *refinement* of u in P into Q is the partial order P′ on the set $(G - \{u\}) \cup H$ formed by the union of the following sets of ordered pairs.

(1) $\{(x, x') \mid (x, x') \in P \quad \text{for all } x, x' \in G - \{u\}\}$
(2) $\{(x, y) \mid (x, u) \in P \quad \text{for all } x \in G - \{u\}, y \in H\}$
(3) $\{(y, x) \mid (u, x) \in P \quad \text{for all } x \in G - \{u\}, y \in H\}$
(4) $\{(y, y') \mid (y, y') \in Q \quad \text{for all } (y, y') \in H\}$

Figure 3(c) shows the result of refining d in the partial order of Figure 3(a) into the partial order of Figure 3(b).

Informally, the refinement of u in P into Q is the partial order whose Hasse diagram is obtained by substituting Q's Hasse diagram in place of u in P's Hasse diagram. We say that u is the group that is *refined* or *exploded* and that Q is the

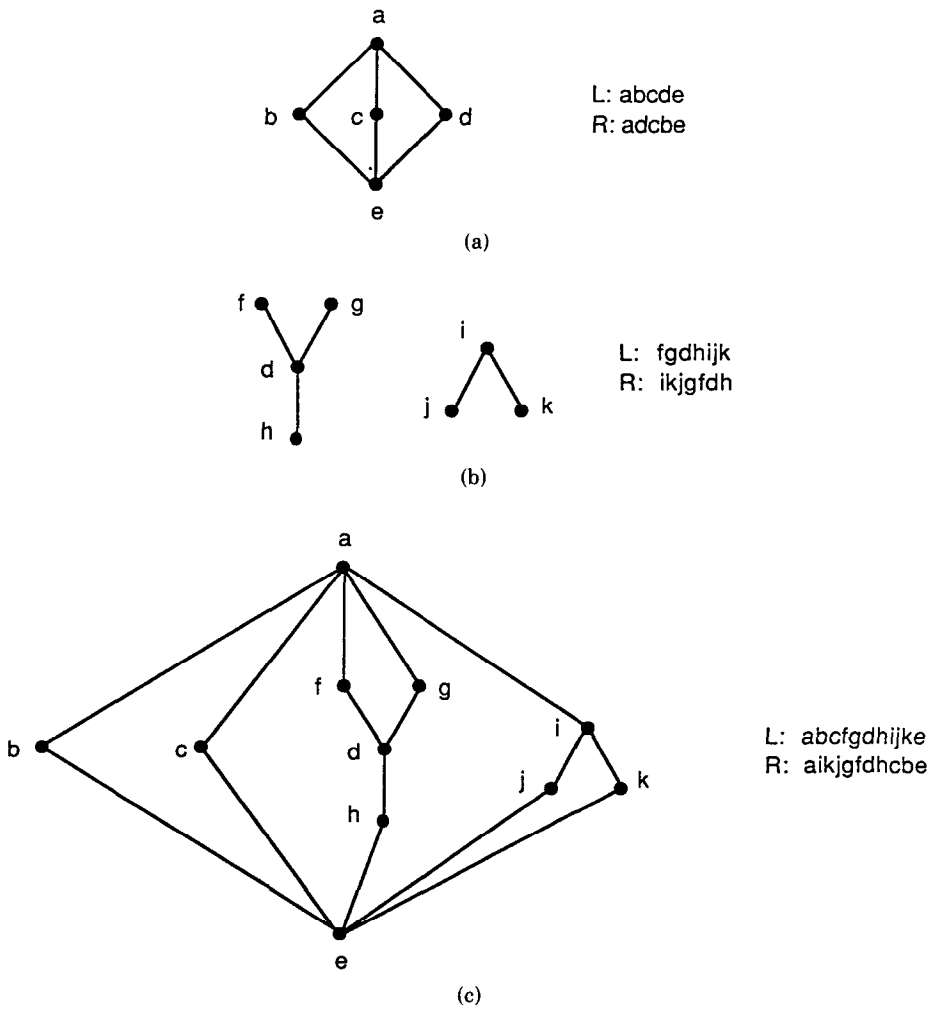


Fig. 3. Refinement.

refining partial order. We think of refinement as exploding an existing group into a partially ordered set of new groups while maintaining the same relationship between the new groups and other previously existing groups that the exploded group had. Refinement is a natural method for incrementally developing more detail in a top-down manner. It is particularly important because of the following result.

THEOREM 2. [6] *Let* P *and* Q *be partial orders on disjoint sets* G *and* H, *respectively. Let* $u \in G$. *If* P′ *is the refinement of* u *in* P *into* Q, *then* $\dim(P') = \max\{\dim(P), \dim(Q)\}$.

PROOF. Let $d = \dim(P')$ and $e = \max\{\dim(P), \dim(Q)\}$. By assumption there are realizers $L_1, \ldots, L_e$ and $J_1, \ldots, J_e$ for P and Q, respectively. For $i = 1 \ldots e$, refine u in J_i into L_i, obtaining the linear ordering J_i'. Then $J_1', \ldots, J_e'$ is a

realizer for P′. So d ≤ e. Conversely, by assumption, there is a realizer $M_1, \ldots, M_d$ for P′. Select an arbitrary member v of H. In every M_i replace v by u and drop all other elements of H to obtain the linear ordering M_i'. By definition, v has the same relationship in P′ to the elements of G − {u} that u does in P. So $M_1', \ldots, M_d'$ is a realizer for P and e ≤ d. □

Clearly, if P and Q have dimension less than or equal to two so does P′. It is now easy to prove our earlier claim regarding a forest of mutually disjoint trees and inverted trees.

COROLLARY 3. *A partial order whose Hasse diagram consists of a forest of mutually disjoint rooted trees and inverted rooted trees has a realizer of size two.*

PROOF. An empty partial order, where all distinct elements are pairwise incomparable, has a size two realizer obtained by any linear ordering of the elements and its reverse. A forest of rooted trees and inverted trees can be obtained by refining the elements of an empty partial order one at a time into a rooted tree or inverted rooted tree, as appropriate. The corollary follows from Theorems 1 and 2. □

From Hirugachi's theorem it follows that so long as we confine ourselves to partial orders of dimension less than or equal to two, we can repeatedly apply the refinement operation to generate new partial orders whose dimension will not exceed two. We call this the *successive refinement procedure.* The simplest dimension two partial orders are the rooted tree and inverted rooted tree. These represent important relationships between groups that have practical applications. Rather than allowing arbitrary dimension two partial orders in the process of successive refinement, we propose that only trees and inverted trees be used. This leads us to the following definition.

Definition 4.

(1) A partial order whose Hasse diagram is a forest of mutually disjoint rooted trees and inverted rooted trees is an *ntree.*
(2) A partial order obtained by refining a node in an ntree into another ntree is an *ntree.*
(3) Nothing else is an *ntree.*

The n in the name ntree is intended as a mnemonic both for inverted and for nested in the sense of refinement. Clearly, the dimension of an ntree is less than or equal to two. The partial orders of Figures 1(a), 2, and 3 are ntrees, whereas the partial orders of Figure 1(b) and 1(c) are not.

To illustrate the usefulness of ntrees in a practical context, consider a project divided into three independent tasks with each task assigned to a team. We can define groups t_1, t_2, and t_3 for the tasks and a group s for the project supervisors related as in Figure 4(a), so the supervisors are members of each task team but not vice versa. This allows the information and resources, such as working documents for each task group, to be kept separate and inaccessible from other task groups while a supervisor can access all of these. Alternatively, we can define a single group p related to the task groups as shown in Figure 4(b). With this structure the task teams can share information and resources of common interest

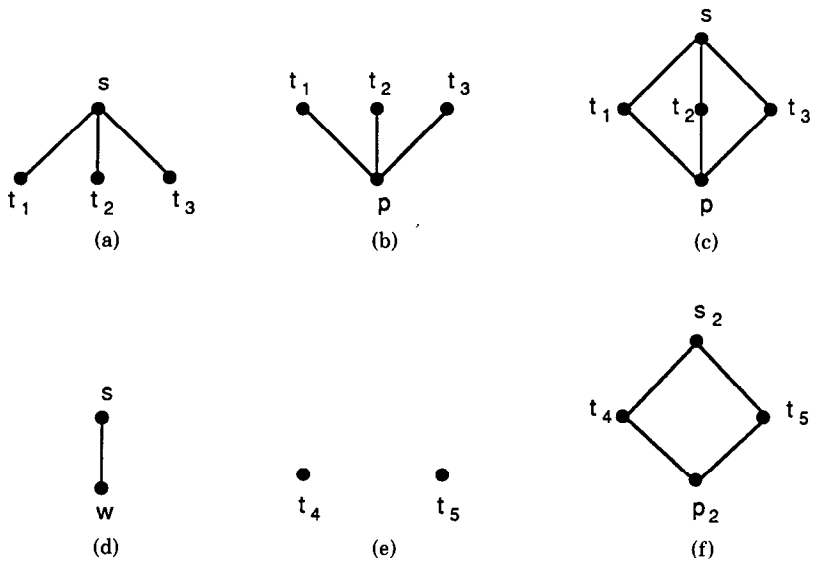


Fig. 4. NTrees for a project.

(for instance, the final design produced by a task team) while keeping working documents and such within each task group. Finally, the tree and inverted tree are not only useful by themselves but can occur together as in Figure 4(c). The partial order of Figure 4(c) is an ntree. One method of constructing this ntree by refinement is to begin with two groups, s for supervisors and w for workers, as in Figure 4(d). Then refine w into the three task groups and project group of Figure 4(b) to obtain the ntree of Figure 4(c).

The ntree of Figure 4(c) embodies three important aspects of a protection policy:

(1) *Separation*: The three task groups t_1, t_2, and t_3 are pairwise incomparable with respect to the subgroup ordering.
(2) *Sharing*: The three separate task groups are all subgroups of a common group p that allows sharing of information and resources.
(3) *Oversight*: The three separate task groups all have s as a common subgroup to facilitate oversight and coordination.

Independent groups that are pairwise incomparable provide support only for separation. A tree supports separation and oversight while an inverted tree supports separation and sharing. The ntree supports all three aspects. Moreover, since ntrees can be nested by refinement, these three basic policy aspects are available at every level of detail. For instance, if it turns out that task t_3 of Figure 4(c) should really be two distinct tasks, we can refine it into t_4 and t_5 of Figure 4(e). Alternately, if t_3 is complex enough to justify treating it as a subproject we may refine it into the partial order of Figure 4(f). Note that the separation between incomparable groups in an ntree is not absolute but is intended to model the logical situation. For instance, task teams t_1 and t_2 may

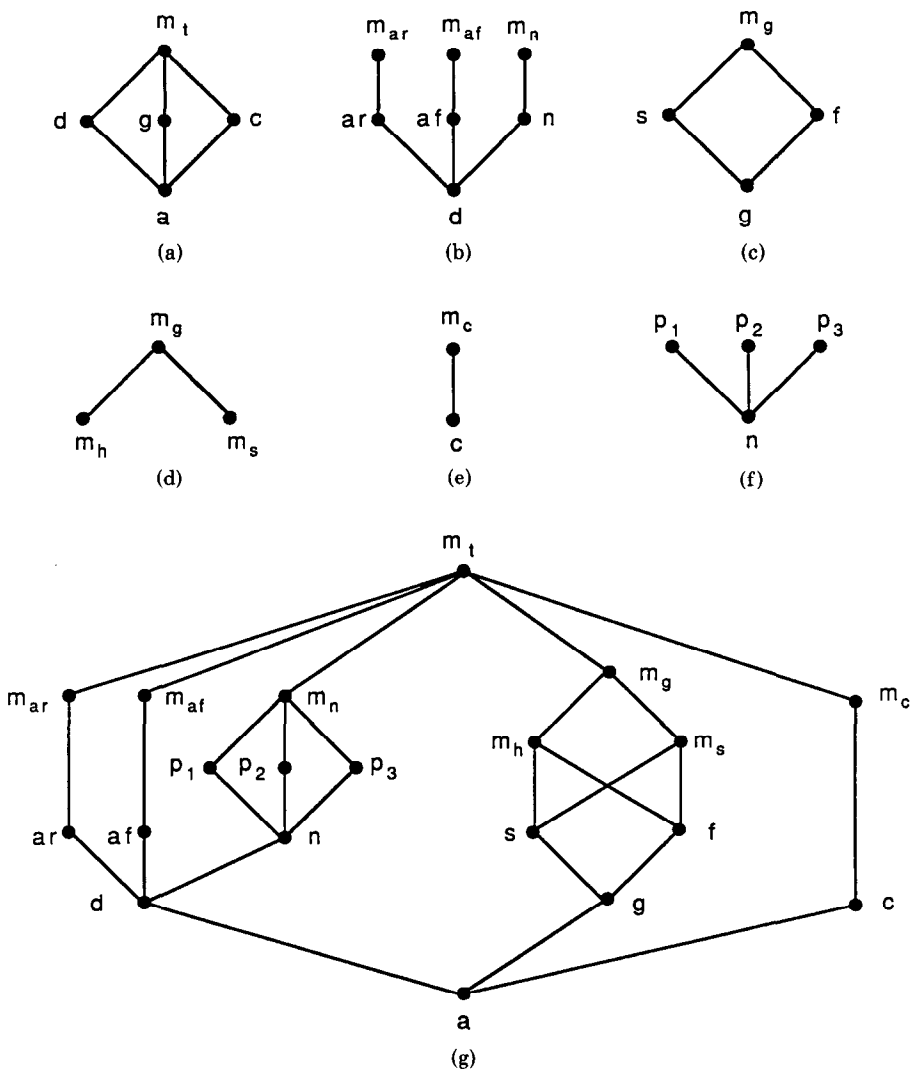


Fig. 5. A sequence of refinements.

actually have some common members. A user who is a member of both teams will have to be explicitly made a direct member of t_1 and of t_2. The task groups are logically separate in that membership in one does not imply membership in the other, and as such are incomparable in the ntree.

Repeated application of refinement allows us to construct large and complex ntrees from simple ones while making policy decisions incrementally. This facilitates top-down design of an organization's subgroup structure. A sequence of refinements is illustrated in Figures 5(a) through 5(f) with the end result shown in Figure 5(g). At the top level of design, we begin with the subgroup ordering of Figure 5(a), which shows m_t for top management overseeing three

area groups d, g, and c for defense, government, and commercial, respectively. These three independent area groups are subgroups of the all-inclusive group a. This ntree is isomorphic to the ntree of Figure 4(c) but occurs at a high level in the organization. The three area groups d, g, and c in Figure 4(a) are then refined independently into different ntrees. The defense group d is refined into the ntree of Figure 5(b) with three subareas ar, af, and n for army, air force, and navy, respectively, each with its own management group. It is convenient and useful to allow the name of the exploded group to occur as the name of one of the groups in the refining ntree. Formally, we can think of this as a *renaming* of one of these groups after refinement has been done. Figure 5(b) indicates that the name d will be used after refinement for a group to facilitate sharing among the three subareas. Proceeding in this manner, the government group g is refined into the ntree of Figure 5(c) with two subareas s and f for state and federal, respectively, with a management group m_g for oversight and g retained as a group for sharing. Then m_g is refined into the ntree of Figure 5(d) with two management subareas m_h for hardware and m_s for software with m_g itself being retained for oversight of these two management subareas. The commercial group is refined into the ntree of Figure 5(e) with a management group m_c and c being retained. Finally, to illustrate how individual projects can be factored in, we show the navy group n in Figure 5(b) being refined into the ntree of Figure 5(f) with three project groups p_1, p_2, and p_3, and n being retained as a group for sharing among the three projects.

The sequence of refinements outlined above results in the ntree of Figure 5(g). There is ample opportunity for further refinement of this structure. For instance, ar, af, s, f, and c can be refined into projects as was done for n. Or the project groups p_1, p_2, and p_3 can be refined into tasks and subprojects along the lines of Figure 4. The successive refinement approach allows policy decisions to be made incrementally and independently by different people in the organization. For example, top management need only be concerned about the high-level structure of Figure 5(a) and leave it to lower-level managers to decide the refinement of the d, g, and c groups. The persons responsible for refining d need not be concerned about the refinement of g or c. Similarly, the refinement of a project group can be the responsibility of the project manager independent of what other project managers may do. Thus policy outlines can be decided at a high level while details are determined at appropriate lower levels.

It is possible that successive refinement, or any other method for developing the subgroup partial order, may result in some groups that are unlikely to be used. For instance, it may turn out on further consideration that group g in the ntree of Figure 5(g) is not required because the s and f groups can achieve all the sharing needed by means of group a; or, perhaps the m_h and m_s groups turn out to be unnecessary. In such cases we can drop these unneeded groups while inducing the subgroup partial ordering on the remaining groups as shown in Figure 6. It turns out that inducing an ntree on a subset of the groups results in another ntree.

Formally, let P be a partial order on G and let G′ be any subset of G. The partial order P′ obtained by *inducing* P on G′ is $\{(u, v) \mid (u, v) \in P \wedge u, v \in G'\}$. By restricting the linear orderings in a realizer of P to the elements of G′ we

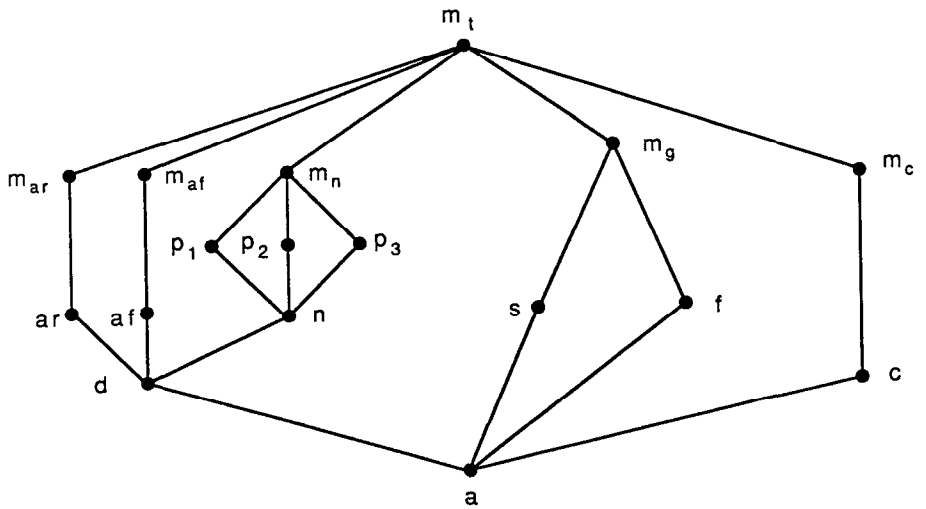


Fig. 6. An induced ntree.

obtain a realizer for P′. So dim(P′) is less than or equal to dim(P). For ntrees we can make a stronger statement, as follows.

THEOREM 5. *Let* $\preceq$ *be an ntree on the set of groups* G *and let* $G' \subseteq G$. *The partial order* $\preceq'$ *obtained by inducing* $\preceq$ *on* G′ *is an ntree.*

PROOF. From the definition of ntree it is evident that any ntree can be obtained by a refinement sequence that at each step refines a group into a forest of disjoint rooted trees and inverted rooted trees (rather than into an arbitrary ntree). Let $\Sigma = \sigma_1\sigma_2 \ldots \sigma_n$ be such a refinement sequence the end result of which is $\preceq$. Assume, without loss of generality, that there is no renaming during the refinement steps. Each step σ_i in this sequence refines some group h_i into a forest F_i of rooted and inverted rooted trees on a set of groups H_i. Let $G'' = G' - G$; that is, G″ is the set of groups removed when inducing $\preceq$ on G′. Note that G″ cannot include groups that are refined at any of the refinement steps of Σ. Obtain F_i^* by inducing F_i on the set of groups $H_i - G''$. Clearly F_i^* is a forest of rooted and inverted rooted trees. For each σ_i in Σ define the refinement step σ_i^* as the refinement of h_i into F_i^*. Let $\preceq^*$ be the ntree that results from the modified refinement sequence $\Sigma^* = \sigma_1^*\,\sigma_2^* \ldots \sigma_n^*$.

We show that $\preceq' = \preceq^*$. If $u \preceq' v$ there is a refinement step σ_i in Σ that refines some h_i into a forest F_i with $(u, v) \in F_i$. The corresponding refinement step σ_i^* in Σ^* refines h_i into F_i^* with $(u, v) \in F_i^*$. So $u \preceq^* v$. Conversely, if $u \preceq^* v$ there is a refinement step σ_i^* in Σ^* that refines some h_i into a forest F_i^* with $(u, v) \in F_i^*$. But then the corresponding refinement step σ_i in Σ refines h_i into a forest F_i that includes (u, v). So $u \preceq' v$. □

Because of this result we can eliminate unneeded groups at any stage in the successive refinement procedure by inducing the current ntree on the subset of

groups that we wish to retain, thereby obtaining another ntree. This is useful both during the initial design of an ntree as well as during the life of a system. For instance, when tasks and projects get completed the corresponding groups can be removed.

Another consequence of Theorem 5 is that the result of inducing $\preceq$ on a subset of groups that are of interest to a particular user is an ntree. Recall that a user is a direct member of g if the user is explicitly designated as a member of g and thereby is an indirect member of every h such that $g \preceq h$. Let G' be the set of groups of which a given user is a direct or indirect member. The subgroup relation induced on G' is an ntree, so the groups to which a user belongs form an ntree. As another example let G' be the set of groups that are comparable with respect to $\preceq$ with one or more of the groups to which a user belongs directly. That is G' is the set of groups with which this user can share resources. Again the subgroup relation induced on G' is an ntree. In this sense the user's perception of the subgroup relation will always be an ntree.

4. DYNAMIC REFINEMENT

It is not possible to correctly anticipate all protection needs in advance and a facility for adding new groups to an ntree is definitely desirable. For example, as new projects are undertaken we can introduce new groups for these; or, perhaps, new task groups need to be introduced as the project life cycle matures. The mechanism for adding new groups will be greatly simplified if we can do so by assigning lr-values to the new groups while leaving the lr-values of existing groups unchanged. In this section we present two methods for doing so. To achieve this effect we propose to limit additions to an ntree to the refinement of an existing group. We will shortly demonstrate that this is a reasonable restriction although at the cost of requiring some additional care and foresight at each refinement step.

By *dynamic refinement* we mean that a group is refined after the system has been in operation for a while. The two methods for dealing with dynamic refinement differ in that the first and simpler method cannot guarantee that the group with the name of the exploded group gets the same lr-values as the exploded group had. The second method is able to guarantee this property at the cost of some additional bookkeeping.

A dynamic refinement step presents two important questions regarding its immediate effect. First, what is the status of resources that were explicitly available to the exploded group? An obvious suggestion is that each resource should now become available to one or more of the groups in the refining ntree. The second question is what happens to members of the exploded group? By definition of refinement, indirect members of the exploded group become indirect members of each group in the refining ntree. For the direct members, an obvious suggestion is to assign each one as a direct member of one or more of the groups in the refining ntree.

We propose a simple default answer to both questions by insisting that one of the groups in the ntree that replaces the exploded group should have the same name as the exploded group. With this rule we can take the view that the exploded group continues to exist after refinement. This provides a reasonable default for both questions. That is, the resources explicitly available to the

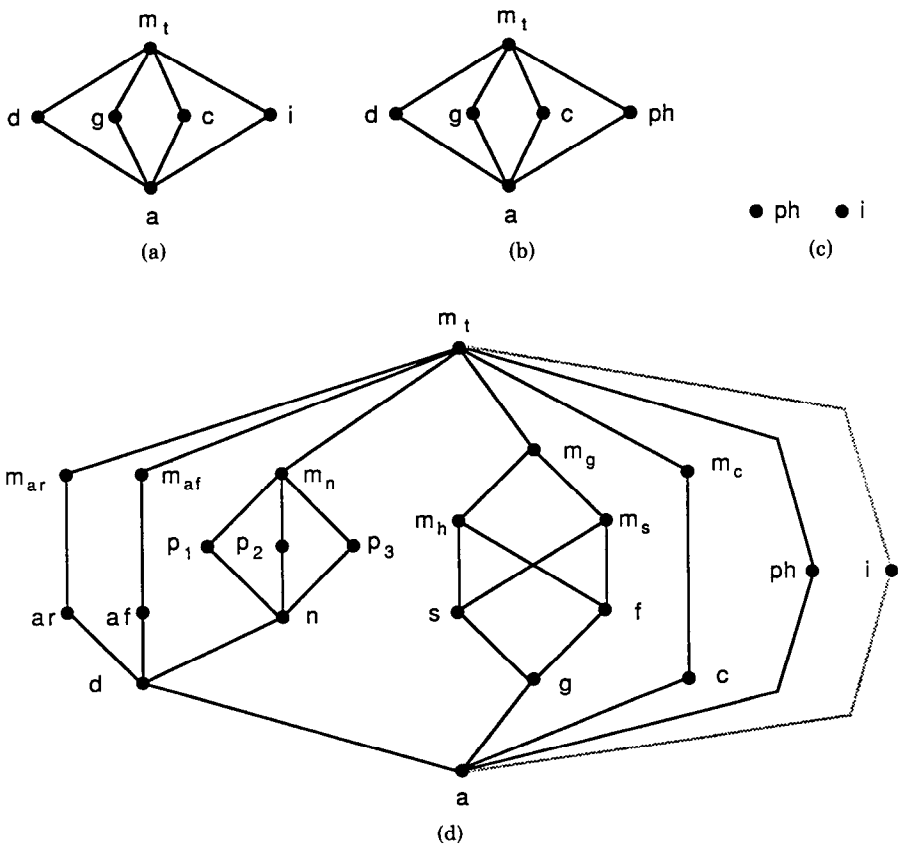


Fig. 7. Place holders for dynamic refinement.

exploded group continue to be explicitly available to the group in the refining ntree that has the same name as the exploded group. Similarly, the direct members of the exploded group become direct members of the group in the refining ntree that has the same name as the exploded group. The other groups introduced by refinement have no direct members and no resources explicitly available to begin with.

Dynamic refinement requires some foresight and planning to be truly effective. Since additions to the existing ntree are limited to those achievable by refinement of an existing group, it is necessary to anticipate the need for future groups at each refinement step. Consider the refinement sequence of Figure 5. Suppose we wish to add a fourth area group i for international as shown in Figure 7(a) while retaining all other refinement steps of Figure 5. If the refinement sequence of Figure 5 has already been carried out it is too late to do so, since the groups d, g, and c of Figure 5(a) have all been refined during this sequence. Refining any of the groups in Figure 5(g) will not provide the desired result.

We can anticipate the need for a group such as i in the future by reserving a *place-holder* at an appropriate step in the refinement sequence. In this case we would begin the refinement sequence of Figure 5 with the ntree of Figure 7(b)

rather than Figure 5(a). The purpose of the place-holder group ph is precisely to anticipate the need for a group such as i in the future. The refinement sequence of Figure 5 modified in this manner will result in the ntree shown by bold lines in Figure 7(d). This differs from the ntree of Figure 5(g) only by the place-holder ph. The international group i can now be introduced by the dynamic refinement of ph in Figure 7(d) into the ntree of Figure 7(c). This will result in i being introduced into the ntree of Figure 7(d) as shown by the light-colored lines. Additional area groups can be introduced by further dynamic refinement of ph in this manner. Note that if the ph group is eliminated from Figure 7(d), the resulting partial order is, of course, an ntree. The point is that this ntree cannot be constructed from the ntree of Figure 5(g) by refinement. However, if we have anticipated the need for additional area groups in the future by introducing ph as a place-holder, this ntree can be constructed by refining the place-holder group and then simply ignoring ph.

We can similarly modify the other refinement steps of Figure 5 to include place-holders. For instance, when we refine n into the ntree of Figure 5(f) we should include a place-holder to anticipate the need for more than three project groups in the future. We see the role of place-holders as another advantage of the successive refinement approach to developing a complex ntree. Along with making the policy decisions regarding the current subgroup structure incrementally, we can introduce place-holders incrementally to anticipate future needs of the organization.

4.1 Quota Offset Numbering

We now turn to the all-important question of how dynamic refinement affects the representation of the subgroup partial order. Consider an ntree on the set of groups G with realizer L, R of size two. The technique suggested in Section 2 for representing the ntree is to assign a pair of integers l[g], r[g] to each group $g \in G$ where l[g] and r[g] are the positions of g in L and R, respectively, so that $g \preceq h$ if and only if $l[g] \leq l[h]$ and $r[g] \leq r[h]$. We refer to these numbers individually as the *l-values* and *r-values* and jointly as the *lr-values*. If g is dynamically refined we must of course assign lr-values to each group that results from g's explosion. It is also necessary to assign new l-values and r-values to each nonexploded group that occurs after g, respectively, in L and in R. This may involve a large fraction of the nonexploded groups and is likely to be cumbersome. Moreover, as a general principle of system design, an operation involving group g should not require a mechanism to deal with groups that have been left unchanged.

We present a technique for assigning lr-values to the groups in an ntree so that after a dynamic refinement step there is no need to change the lr-values of nonexploded groups. The technique actually works for partial orders of any dimension, but since our objective is to use it for ntrees we state it in terms of ntrees. In Section 4.2 we modify the technique to guarantee that one of the groups in the refining ntree will be assigned the same lr-values that the exploded group had. This will extend our earlier rule that one of the groups in the refining ntree should have the same name as the exploded group, to the property that the lr-values of this group remain unchanged after refinement. This modification is

applicable only to ntrees and does not apply to arbitrary two-dimension partial orders.

The basic idea is to assign a *quota* q[g] to each group g. The quota is a positive nonzero integer specifying the maximum number of groups that g can be refined into, be it in a single step or by a sequence of refinements. Since g already exists, the minimum value of q[g] is 1. The maximum number of groups that can ever exist is the sum of the quotas of currently existing groups. The use of quotas for this purpose is no different in principle than the use of quotas for many other resource allocation decisions in an operating system, such as quotas on disk space, main memory, processor time, and so on. It appears to be a desirable facility for the system administrators in any case, as a means of limiting the number of groups that can be created at different places in an organizational hierarchy.

By assigning a quota to each group we can determine how many integers to allocate for groups that may result from future refinement steps. Let L and R be linear orderings that comprise a size two realizer for the subgroup partial ordering. The l-value of a group g is obtained by adding one to the sum of the quotas of all groups that precede g in L. Similarly, the r-value of a group g is obtained by adding one to the sum of the quotas of all groups that precede g in R. When a group g is exploded as part of the refinement operation we partition g's quota among the groups in the refining ntree. Any partition whose sum equals q[g] is acceptable. The l-values and r-values of groups in the refining ntree are then computed in the same way as for the initial assignment, except that we begin with l[g] and r[g], respectively, rather than beginning with 1. A formal statement of this method is given in Figure 8. The method consists of an initial numbering to be used when an ntree is first set up, and an incremental numbering to be used when a group is exploded. Incremental numbering is identical to initial numbering, except that we start with the lr-values of the exploded group rather than 1.

The lr-values assigned by the initial numbering to the ntree of Figure 3(a) are shown in Figure 9(a) for the quotas listed there. Consider the refinement of d into the ntree of Figure 3(b). Let each of the new groups in Figure 3(b) be allocated a quota of 6 each, with d retaining a quota of 24 out of its original quota of 60. We then obtain the lr-values of Figure 9(b) for the resulting ntree of Figure 3(c). The lr-values of nonexploded groups are of course unchanged. The lr-values assigned to groups introduced by refinement are determined by the structure of the refining ntree and the allocation of the quota for further refinement. The lr-values of Figure 9(b) can be computed either by the initial numbering of Figure 9(a) followed by the incremental numbering for the refinement of d, or directly by the initial numbering of the ntree of Figure 3(c) with the specified quotas.

The key idea in the quota offset method is that l-values and r-values in the range [l[g], l[g] + q[g] − 1] and [r[g], r[g] + q[g] − 1], respectively, are reserved for groups to be introduced by a future refinement of g. We now prove the quota offset method is correct.

THEOREM 6. *The quota offset numbering method of Algorithm* 1 *is correct.*

Initial numbering

```
next := 1;
while L is non-empty do
  x := head(L);
  l[x] := next;
  next := next + q[x];
  L := tail(L);
end;
```

```
next := 1;
while R is non-empty do
  x := head(R);
  r[x] := next;
  next := next + q[x];
  R := tail(R);
end;
```

Incremental numbering on explosion of group g

```
next := l[g];
while L is non-empty do
  x := head(L);
  l[x] := next;
  next := next + q[x];
  L := tail(L);
end;
```

```
next := r[g];
while R is non-empty do
  x := head(R);
  r[x] := next;
  next := next + q[x];
  R := tail(R);
end;
```

Fig. 8. Algorithm 1: Quota offset numbering method.

PROOF. To prove the correctness of this method we show that after any sequence of refinements the following are true.

(1) $l[g] \leq l[h] \wedge r[g] \leq r[h] \Leftrightarrow g \preceq h$

(2) No group other than g itself has an l-value in the range $[l[g], l[g] + q[g] - 1]$ or an r-value in the range $[r[g], r[g] + q[g] - 1]$.

The first property guarantees that the current ntree is correctly represented by the lr-values. The second guarantees that g can continue to be refined within the limits of its quota. We prove these properties by induction on the number of refinement operations. For the basis case, let this number be 0, so lr-values are assigned by the initial numbering. The two properties are obviously true since increasing l-values and r-values are assigned to successive groups in L and R, respectively, and next is incremented by q[x] on every iteration. Assume both properties are true after n refinement operations and let u be refined in the $n + 1^{st}$ refinement step into the ntree N, so groups in N are assigned lr-values by the incremental numbering and lr-values of nonexploded groups are unchanged. Let q[u], l[u], and r[u] be the quota and the lr-values of u *before* this refinement. By definition, the sum of the quotas of groups in N equals q[u]. It follows that the l-values and r-values assigned to groups in N are, respectively, in the range $[l[u], l[u] + q[u] - 1]$, and $[r[u], r[u] + q[u] - 1]$. Since next is incremented by q[x] on every iteration, the second property is true for groups in N. Therefore, by induction hypothesis, the second property is true for all groups after the incremental numbering. After the $n + 1^{st}$ refinement the first property continues to be true for the nonexploded groups since their lr-values are unchanged. Let v be a group in the refining ntree N and w be a nonexploded group. Since the lr-values of w are outside the range $[l[u], l[u] + q[u] - 1]$, and $[r[u], r[u] + q[u] - 1]$ while the lr-values of v are in this range, the relation

	q	l	r
a	5	1	1
b	15	6	81
c	15	21	66
d	60	36	6
e	5	96	96

(a) L: abcde, R: adcbe

	q	l	r
a	5	1	1
b	15	6	81
c	15	21	66
d	24	48	36
e	5	96	96
f	6	36	30
g	6	42	24
h	6	72	60
i	6	78	6
j	6	84	18
k	6	90	12

(b) L: abcfgdhijke, R: aikjgfdhcbe

Fig. 9. Quota offset numbering for Figure 3.

between the lr-values of w and v is exactly the same as the relation between the lr-values of w and the old lr-values of u. So the lr-values assigned to v by the incremental numbering correctly represent the subgroup relation between the nonexploded groups and the groups in the refining ntree N. It remains to show that the lr-values assigned to groups in N correctly represents the subgroup relation between these groups. This follows, since increasing l-values and r-values are assigned to successive groups in L and in R, respectively. □

To summarize, the quota offset numbering method allows us to assign lr-values in such a way that on refining group g we need only compute lr-values for groups in the refining ntree while the lr-values of nonexploded groups remain unchanged. The method also provides a simple mechanism for enforcing a quota on the number of groups that may arise from refining an existing group as well as for allocating this quota for further refinement.

4.2 Conservative Quota Offset Numbering

The quota offset numbering method has the property that in general the lr-values of the exploded group will no longer be a valid pair of lr-values after refinement. For instance, the pair of lr-values assigned to d in Figure 9(a) does not occur in Figure 9(b). We have earlier argued that naming one of the groups in the refining ntree to have the same name as the exploded group is desirable as a default rule, so that in effect the refined group continues to exist after

refinement. The possibility of inconsistency and integrity problems will be reduced if this group has the same lr-values as the exploded group did. A numbering method that achieves this is said to be *conservative.*

In order to modify the quota offset numbering method to be conservative we need some additional restrictions in the refinement steps. So far we have allowed a group to be refined into an arbitrary ntree, the only restriction being that the number of groups in the refining ntree cannot exceed the exploded group's quota. Now the groups introduced by refinement of a group g can be classified into three disjoint categories: *up-groups*, which are subgroups of g; *down-groups*, of which g is a proper subgroup; and *split-groups*, which are incomparable with g. To facilitate conservative numbering we partition the quota q[g] of each group g into the *up-quota* $q_u[g]$, the *down-quota* $q_d[g]$, and the *split-quota* $q_s[g]$, which specify the maximum number of groups in the corresponding categories. The *total-quota* q[g] is the sum of these three components. Recall that q[g] includes a count of 1 for group g itself. By our definitions, this is counted as part of the up-quota of each group. So if q[g] = 1, which is the minimum value, we have $q_u[g] = 1$ and $q_d[g] = q_s[g] = 0$.

This partitioning of the total-quota gives us the additional information needed to achieve conservative numbering. After refinement of g, g's total-quota is allocated among the groups introduced by refinement. The sum of the total-quotas assigned to the up-groups, down-groups, and split-groups is subtracted from $q_u[g]$, $q_d[g]$, and $q_s[g]$, respectively. For instance, consider the quotas of Figure 10(a) for the ntree of Figure 3(a). When d is refined into the ntree of Figure 3(b), as before let the total-quota of all new groups be 6 each. Since f and g are up-groups, $q_u[d]$ is reduced from 20 to 8. The only down-group is h, so $q_d[d]$ is reduced from 10 to 4. The split groups i, j, and k reduce $q_s[d]$ from 30 to 12. The total-quota of 6 allocated to each new group can be arbitrarily partitioned into the three components, and this partitioning is of no consequence for reduction of d's quotas.

Consider what happens when g is exploded into an ntree with realizer L, R. The up-groups of g must precede g in both L and R while the down-groups must follow g. To ensure that g retains its lr-values after refinement we need information about where the split-groups occur in L and R. Each split-group must either follow g in L and precede g in R or vice versa. To achieve conservative numbering we impose the following restriction on the realizer for the refining ntree.

Definition 7. A realizer L, R is a *left-most realizer* for group g if the split-groups of g all occur after g in L and before g in R.

For example the realizer of Figure 2(a) is left-most for a, b, and e, while the realizer of Figure 2(b) is left-most for i, d, and a. For two-dimension partial orders in general, a left-most realizer may not exist. For instance, consider the partial order of Figure 11 with the size two realizer shown there. The algorithm of [10] shows that the incomparability graph of this partial order has exactly two transitive orientations, leading to two possible realizers: the one shown in Figure 11 and the other obtained by interchanging L and R. By inspection, neither of these is a left-most realizer for c.

	q_u	q_d	q_s	l	r
a	1	4	0	1	1
b	5	5	5	10	90
c	5	5	5	25	75
d	20	30	10	55	55
e	5	0	0	100	100

(a) L: abcde, R: adcbe

	q_u	q_d	q_s	l	r
a	1	4	0	1	1
b	5	5	5	10	90
c	5	5	5	25	75
d	8	4	12	55	55
e	5	0	0	100	100
f	2	2	2	47	43
g	2	2	2	53	37
h	1	5	0	72	60
i	3	3	0	80	18
j	1	2	3	84	31
k	1	5	0	90	22

(b) L: abcfgdhijke, R: aikjgfdhcbe

Fig. 10. Conservative quota offset numbering for Figure 3.

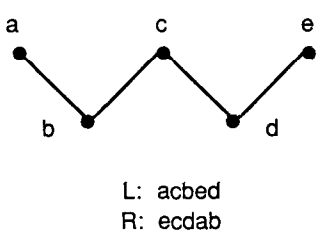


Fig. 11. A partial order with size two realizer.

It turns out that for ntrees a left-most realizer is guaranteed to exist. Every tree and inverted tree has a left-most realizer easily obtained by rearranging the tree so the group of interest ends up being left-most. Since ntrees are constructed from these basic partial orders, we can ensure that at each refinement step we use a left-most realizer for the group of interest. Formally, we have the following result.

THEOREM 8. *For any ntree and any group* g *in the ntree there exists a left-most realizer* L, R *for* g.

PROOF. For any ntree there exists a refinement sequence Σ where at each step the refining ntree is a rooted tree, an inverted rooted tree, or an empty partial

order; that is, all distinct groups are pairwise incomparable. This follows from our observation in the proof of Theorem 5 that every ntree can be constructed by a refinement sequence where each refinement is into a forest of rooted and inverted rooted trees. Such a forest can obviously be constructed by refining the groups in an empty partial order into trees and inverted trees, as appropriate. We prove the theorem by induction on the number of refinement steps in Σ.

For the basis case, let the number of refinement steps be 0 so the ntree is a rooted tree, inverted rooted tree, or is empty. For a rooted tree, a left-most realizer for g is obtained by rearranging the tree so that g is on the left-most path from the root and computing the realizer L, R as the left-to-right and right-to-left preorder traversals, respectivley. For an inverted rooted tree, a left-most realizer for g is obtained by rearranging the tree so that g is on the right-most path to the root, and reversing the left-most realizer for the corresponding rooted tree obtained by the vertical reflection of the inverted tree. Finally, the empty partial order has a left-most realizer for g, since any linear ordering and its reverse are a realizer and we can select g to be the first group in L.

At each refinement step let the realizer L″, R″ for the resulting ntree be obtained by respectively substituting the realizer L′, R′ for the refining ntree in place of the exploded group in the realizer L, R of the existing ntree. Assume the theorem is true for n refinement steps and consider the $n + 1^{st}$ refinement. If g is left-most in L, R and is not exploded in the $n + 1^{st}$ step, it obviously continues to be left-most in L″, R″. If g is exploded in the $n + 1^{st}$ step, by our argument for the basis case, we can construct L′, R′ to be left-most for g. But then g obviously continues to be left-most in L″, R″. □

In conjunction with our discussion above regarding the partial order of Figure 11, this theorem also demonstrates that ntrees are a proper subset of two-dimension partial orders.

It follows from Theorem 8 that if we construct ntrees by limiting ourselves to a forest of rooted and inverted rooted trees at each refinement step, we will always be able to come up with a left-most realizer. This is a reasonable restriction on how to go about doing refinement. The more general question of how to construct a left-most realizer for an arbitrary group in an arbitrary ntree without being given the refinement sequence is beyond the scope of this paper, although we conjecture there is an efficient method for doing so.

The importance of a left-most realizer for the refining ntree is that we then know how many integers to allocate below l[g] for the up-groups and above l[g] for the down-groups and split-groups that may result from future refinement. Similarly, we know how many integers to allocate below r[g] for the up-groups and split-groups and above r[g] for the down-groups that may result from future refinement. As in the quota offset method, we reserve a consecutive range of q[g] l-values and r-values for future refinement of g. These ranges for the l-values and r-values are, respectively, $[l[g] - q_u[g] + 1, l[g] + q_d[g] + q_s[g]]$, and $[r[g] - q_u[g] - q_s[g] + 1, r[g] + q_d[g]]$. If $q_u[g] = 1$ and $q_s[g] = 0$, these ranges are the same as those for the quota offset method.

These observations lead us to the *conservative quota offset numbering method* of Figure 12. As in the quota offset method of Algorithm 1, there is an initial numbering beginning from 1 and an incremental numbering beginning with the

Initial numbering

```
next := 1;
while L is non-empty do
  x := head(L);
  l[x] := next + q_u[x] - 1;
  next := next + q[x];
  L := tail(L);
end;
```

```
next := 1;
while R is non-empty do
  x := head(R);
  r[x] := next + q_u[x] + q_s[x] - 1;
  next := next + q[x];
  R := tail(R);
end;
```

Incremental numbering on explosion of group g

```
next := l[g] - q_u^o[g] + 1;
while L is non-empty do
  x := head(L);
  l[x] := next + q_u[x] - 1;
  next := next + q[x];
  L := tail(L);
end;
```

```
next := r[g] - q_u^o[g] - q_s^o[g] + 1;
while R is non-empty do
  x := head(R);
  r[x] := next + q_u[x] + q_s[x] - 1;
  next := next + q[x];
  R := tail(R);
end;
```

Fig. 12. Algorithm 2: Conservative quota offset numbering method.

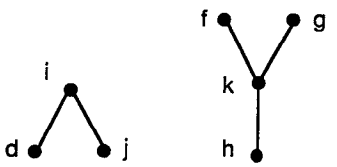
lower end of the range of lr-values reserved for g's refinement. In the incremental numbering the superscript o on the quotas of g indicates that the old values of the quotas before refinement are to be used. The method requires that the realizer for the incremental numbering be left-most for the exploded group g. The lr-values assigned by this method to the ntree of Figure 3(a) are shown in Figure 10(a) for the quotas listed there. Consider the refinement of d into the ntree of Figure 3(b), which fortuitously happens to have a left-most realizer for d shown there. The incremental numbering gives the lr-values of Figure 10(b) for the quotas listed there. Now suppose the groups d and k in Figure 3(b) are interchanged. By inspection we can rearrange this refining ntree so that d is left-most as shown in Figure 13(a) with the realizer shown there. With this left-most realizer and same quotas as earlier, we obtain the lr-values of Figure 13(b).

Note that the conservative quota offset method is equivalent to the quota offset method if $q_u[g] = 1$, $q_d[g] = q[g] - 1$, and $q_s[g] = 0$. It remains to prove the correctness of the conservative quota offset method.

THEOREM 9. *The conservative quota offset numbering method of Algorithm 2 is correct if the realizer for the incremental numbering is left-most for the exploded group.*

PROOF. To prove the correctness of this method we show that after any sequence of refinements the following are true:

(1) $l[g] \leq l[h] \wedge r[g] \leq r[h] \Leftrightarrow g \preceq h$.
(2) No group other than g itself has an l-value in the range $[l[g] - q_u[g] + 1, l[g] + q_d[g] + q_s[g] - 1]$ or an r-value in the range $[r[g] - q_u[g] - q_s[g] + 1, r[g] + q_d[g] - 1]$.



(a)

	q_u	q_d	q_s	l	r
a	1	4	0	1	1
b	5	5	5	10	90
c	5	5	5	25	75
d	14	10	0	55	55
e	5	0	0	100	100
f	2	2	2	37	33
g	2	2	2	43	27
h	1	5	0	72	60
i	3	3	0	80	8
j	1	2	3	84	21
k	1	5	0	90	12

(b) L: abcfgdhijke, R: aikjgfdhcbe

Fig. 13. Rearranging the refining ntree.

(3) The lr-values of all exploded groups are unchanged by the incremental numbering.

As in the proof of Theorem 6, the first property guarantees that the current ntree is correctly represented by the lr-values while the second guarantees that g can continue to be refined within the limits of its quota. The proof of the first two properties is essentially the same as their proof in Theorem 6. The third property gives us conservative numbering on refinement. By assumption, the realizer for the refining ntree used in incremental numbering is left-most for the exploded group g. Let the superscript o denote the lr-values and quotas of g before refinement and the superscript n denote these values after refinement. The groups that precede g in L are the up-groups of g. From the incremental numbering it is evident that

$$l^n[g] = l^o[g] - q_u^o[g] + 1 + \Sigma\, q[x] + q_u^n[g] - 1$$

where x ranges over the up-groups of g, not counting g itself. By the allocation of quotas, we have $q_u^o[g] = \Sigma\, q[x] + q_u^n[g]$, so the l-value of g is unchanged. Similarly, the groups that precede g in R are the up-groups and split-groups of g. From the incremental numbering it is evident that

$$r^n[g] = r^o[g] - q_u^o[g] - q_s^o[g] + 1 + \Sigma\, q[x] + \Sigma\, q[y] + q_u^n[g] + q_s^n[g] - 1$$

where x ranges over the up-groups of g, not counting g itself, and y ranges over the split-groups of g. By the allocation of quotas, we have $q_u^o[g] = \Sigma\, q[x] + q_u^n[g]$ and $q_s^o[g] = \Sigma\, q[y] + q_s^n[g]$, so the r-value of g is unchanged. □

5. CONCLUDING REMARKS

To summarize, we have defined the ntree as a two-dimension partial order suitable for the subgroup relation between protection groups. The ntree is a useful and substantial generalization of the rooted tree and inverted rooted tree partial orders. We have shown how to develop complex ntrees incrementally in a top-down manner by successive refinement. We have proposed a representation for ntrees on the basis of their dimension as partial orders and have argued that the ntree is most likely the only useful application of dimension theory in the context of the subgroup relation, particularly in that using some small integer bigger than two as an upper bound on dimension is not going to be of much additional value. Since ntrees have dimension less than or equal to two, we can assign a pair of integers called lr-values to each group so that g is a subgroup of h if and only if $l[g] \leq l[h]$ and $r[g] \leq r[h]$. Each l-value or r-value requires $\log_2(m)$ bits where m is the maximum number of groups that can ever exist and we can represent the ntree using $2*\log_2(m)$ bits per group. We have shown how to assign lr-values and quotas so that groups can be dynamically refined without changing the lr-values assigned to nonexploded groups. Moreover, we have shown how to guarantee that the lr-values of the exploded group will be assigned to one of the refined groups by partitioning the quota into up, down, and split components.

We have defined the subgroup relation as a reflexive partial order. Clearly, our representation allows us to determine whether a group is a proper subgroup of another group or whether the two groups in question are identical. This allows us to distinguish whether a user who is known to the access-control mechanism as a direct member of group g is thereby a direct or indirect member of group h. The usefulness of ntrees can be enhanced by distinguishing access based on direct membership from access based on indirect membership. This allows a degree of *privacy* in that the direct members of a group can share resources that are inaccessible by indirect members. For instance, the direct members of project team p_1 in the ntree of Figure 7(d) can share files not available to the direct members of m_n or m_t.

It is worth noting that a group with lr-values, both 1, will correspond to a super-group (i.e., a highly privileged group) whose members are thereby members of every group. Because the super-group naturally turns out to have these special lr-values, we can build special rules into the access-control mechanism for these values. For instance, the distinction between direct and indirect membership may be ignored so a direct member of the super-group is considered to be a direct member of every group. Also, note that if m is the maximum number of groups allowed, a group with lr-values both equal to m will correspond to a public-group of which every group is a subgroup. Once again we can build special rules for these extreme values, although it is probably less useful to do so for the public-group as compared to the super-group.

Finally, we note that we have not addressed several important issues, such as how users are made direct members of groups and exactly who is authorized to refine an existing group. There are numerous possibilities here, the simplest and typical one being to centralize such authority in a system administrator. A more interesting alternative would be to decentralize this authority by perhaps nominating some distinguished members of a group to add new members to the group,

and perhaps even allowing this notion of distinguished membership to be inherited via the subgroup partial ordering. These issues are beyond the scope of this paper. We do wish to emphasize that these must be addressed and resolved in an implementation.

ACKNOWLEDGMENT

The author gratefully acknowledges several valuable comments made by the referees, which have improved not only the presentation of the paper but also its substance.

REFERENCES

1. DENNING, D. E. A lattice model of secure information flow. *Commun. ACM 19*, 5 (May 1976), 236–243.
2. DENNING, D. E., AND DENNING, P. J. Data security. *ACM Comput. Surv. 11*, 3 (Sept. 1979), 227–249.
3. DUSHNIK, B., AND MILLER, E. W. Partially ordered sets. *Am. J. Math. 63* (1941), 600–610.
4. FISHBURN, P. C. *Interval Orders and Interval Graphs: A Study of Partially Ordered Sets.* John Wiley & Sons, New York, 1985.
5. GOLUMBIC, M. C. *Algorithmic Graph Theory and Perfect Graphs.* Academic Press, New York, 1980.
6. HIRUGACHI, T. On the dimension of partially ordered sets. *Science Rep. Kanazawa Univ. 1* (1951), 77–94.
7. HIRUGACHI, T. On the dimension of orders. *Science Rep. Kanazawa Univ. 4* (1955), 1–20.
8. KOMM, H. On the dimension of partially ordered sets. *Am. J. Math. 70* (1948), 507–520.
9. LANDWEHR, C. E. Formal models for computer security. *ACM Comput. Surv. 13*, 3 (Sept. 1981), 247–278.
10. PNUELI, A., LEMPEL, A., AND EVEN, S. Transitive orientation of graphs and identification of permutation graphs. *Canadan J. Math. 23* (1971), 160–175.
11. RITCHIE, D. M., AND THOMPSON, K. The UNIX time-sharing system. *Commun. ACM 17*, 7 (July, 1974), 365–375.
12. SALTZER, J. H. Protection and the control of information sharing in MULTICS. *Commun. ACM 17*, 7 (July 1974), 388–402.

Received October 1986; revised June 1987; accepted July 1987