Journal of Computer Security 1 (1992), 335–383
IOS Press

THE EXTENDED SCHEMATIC PROTECTION MODEL

Paul E. Ammann and Ravi S. Sandhu
Center for Secure Information Systems and
Department of Information and Software Systems Engineering
George Mason University
Fairfax, VA 22030
USA

Abstract

Access control models provide a formalism and framework for specifying control over access to information and other resources in multi-user computer systems. Useful access control models must balance expressive power with the decidability and complexity of safety analysis (*i.e.* the determination of whether or not a given subject can ever acquire access to a given object). The access matrix model as formalized by Harrison, Ruzzo, and Ullman (HRU) has very broad expressive power. Unfortunately, HRU also has extremely weak safety properties. Safety is undecidable for most policies of practical interest, even in the monotonic version of HRU (which only allows revocation which is itself reversible). Remarkably, an alternate formulation of monotonic HRU yields strong safety properties. This alternate formulation is called the Extended Schematic Protection Model (ESPM). ESPM is derived from the Schematic Protection Model (SPM) by extending the creation operation to allow multiple parents for a child, as opposed to the conventional create operation of SPM which has a single parent for a child. Despite its equivalence to monotonic HRU, ESPM retains tractable safety analysis for a large class of protection schemes that are of practical interest. In this paper we first show that ESPM is formally equivalent in expressive power to monotonic HRU. Then we give a complete analysis of the safety properties of ESPM for acyclic can-create relations with attenuating loops (i.e., can-create relations which are acyclic except for certain cycles of length one).

1. Introduction, Background and Motivation

The need for access controls arises in any computer system that provides for controlled sharing of information and other resources among multiple users. Access control models (or protection models) provide a formalism and framework for specifying, analyzing and implementing security policies in multi-user systems. These models are typically defined in terms of the well-known abstractions of subjects, objects and access rights, with which we assume the reader is familiar.

In this section, we first give a brief review of access control models, emphasizing the fact that the conventional black and white distinction between mandatory and

discretionary access controls is inadequate. We then argue that models based on propagation of access rights actually transcend this distinction. This leads us to a discussion of the safety problem of determining whether or not a given subject can ever acquire access to a given object.

There is an essential conflict between the expressive power of an access control model and tractability of safety analysis. The principal contribution of this paper is the formal demonstration that very general expressive power and strong safety properties are simultaneously achieved by the Extended Schematic Protection Model (ESPM).

ESPM is derived from the Schematic Protection Model (SPM) [38] by extending the creation operation to allow multiple parents for a child, as opposed to the conventional create operation of SPM which has a single parent for a child. We establish the general expressive power of ESPM by showing its formal equivalence to the monotonic access-matrix model as formalized by Harrison, Ruzzo, and Ullman [15, 16]. Despite this equivalence, ESPM retains tractable safety analysis for a large class of protection schemes that are of practical interest. These results are established and elaborated in the main body of the paper.

1.1. *Access Control Models*

The first access control models to be proposed [13, 18] were completely discretionary, in that the creator of an object was vested with absolute freedom regarding who may or may not access the object. The vulnerabilities of such completely discretionary access controls (DAC) to Trojan Horse attacks is well known [12, for instance]. This vulnerability led to reliance upon the so-called mandatory access controls (MAC)† of the Bell and LaPadula model (BLP) [4]. Since then, the MAC/DAC distinction has served as a basic principle for computer security. For instance, it has been embodied in the TCSEC [11] (popularly known as the Orange Book).

In recent times, it has become increasingly clear that useful access control models must go beyond the traditional MAC/DAC distinction. Indeed, opinion on this matter has changed so rapidly that what would have been considered heresy a few years ago is now being accepted without controversy. There are several major lines of argument that have together resulted in bringing about this rapid conversion of opinion. We enumerate these below.

(1) *Arguments based on secrecy policies.* As might be expected, these arguments have come from the military sector [14, 28]. In the main, they consist of the demonstration that there are document-handling policies in the military—such as ORCON (originator control) and NOFORN (no foreign)—which cannot be readily expressed in BLP and are indeed not quite MAC or DAC in the conventional sense. An abstract description of this requirement was earlier given by Millen [33].

† The TCSEC [11] defines MAC as "A means of restricting access to objects based on sensitivity (as represented by a label) of the information contained in the object and the formal authorization (*i.e.* clearance) of subjects to access information of such sensitivity".

(2) *Arguments based on integrity policies.* The black and white MAC/DAC distinction of BLP was carried over to integrity by Biba [5]. An early attempt by Lipner [23] to apply Biba's model showed the need for additional controls on program execution. Boebert and Kain [7] pointed out limitations of Biba's MAC and proposed the type-enforcement controls of LOCK. Attempts by Lee [21] and Schockley [47] to implement the Clark and Wilson "commercial" integrity policy [8] within the framework of Biba demonstrated that either additional "mandatory" controls must be enforced, or the policy must be emasculated by requiring certain aspects of it to be statically specified. Clark and Wilson [9] have described a notion of mandatory controls which is derived from their model. Collectively, the papers cited here make a compelling case that the BLP and Biba notions of MAC are simply too limited for integrity policies.

(3) *Arguments based on a more general notion of MAC.* Sandhu [43, 45] has given alternate definitions of "mandatory" which show that the conventional BLP notion of MAC is but one special case of the general notion of access controls based on properties of subjects and objects rather than their identities. In the military non-disclosure context, these properties turn out to be best expressed as partially-ordered labels. In other contexts, these properties are more naturally obtained in other ways. For instance, the data type of an object determines what operations can be executed on that object. Sandhu argues that the real issue we need to focus on is whether or not the properties on which our "mandatory" access controls are based are static or dynamic. He also argues that attempts to define a notion of "mandatory" controls as something which lie "between" label-based mandatory and discretionary controls should be dropped in favor of definitions, such as his, which treat label-based mandatory controls as a special case of general "mandatory" controls. Similar arguments have been made by other authors [1, 20, 34].

(4) *Arguments based on foundational inadequacies of BLP.* The foundational inadequacies of the BLP notion of mandatory controls were demonstrated in a series of papers by McLean [29, 30, 31, 32]. The thrust of McLean's argument is that if labels are considered to be changeable then the state-invariant properties of BLP can be preserved by changing labels, which in effect automatically downgrades information on demand. He shows that the Basic Security Theorem of BLP can be proved for such an obviously insecure system. The main lesson from McLean's work for our purpose here is that we must attend to the dynamic aspects of access control.

We now draw attention to a line of research in access control models based on the idea of controlling the propagation of access rights. The basic concept is that the access-matrix is used not only to control access of subjects to objects, but also to control the transport of access rights from one subject to another. The original access matrix proposal of Lampson [18] contained a particular set of rules for this purpose. These rules amount to giving the owner of an object complete discretion regarding rights in the column corresponding to that object. Graham and Denning [13] proposed a variety of copy flags by which this discretionary ability of the owner could be granted to other subjects in various degrees. It became clear that many different rules could be proposed for such purposes, and that no one set of rules

could be argued to be the single universal policy which everyone should implement in their systems. This led Harrison, Ruzzo and Ullman [15] to develop an access control model, commonly called HRU, in which complex policies for propagation of access rights could be easily stated. The BLP model is known to be a special case of HRU [36], as is the completely discretionary access matrix of [13, 18]. Therefore HRU does indeed transcend the traditional MAC/DAC distinction and could readily handle many of the concerns enumerated above. The problem with HRU, however, is that it has weak safety properties, which we explain next. Before doing so, we wish to emphasize and reiterate that our contribution in this paper is the demonstration of a model, *viz.* ESPM, that is formally equivalent in expressive power to monotonic HRU, and yet has strong safety properties.

1.2. *The safety problem*

As noted above, access control models not only specify the control of access to resources by means of access rights, but also specify the propagation of such rights. The safety problem for access control models is the determination of whether or not a given subject can ever acquire access to a given resource. Thus protection models must satisfy two conflicting requirements:

(1) The need for expressive power sufficient to conveniently describe security policies of practical interest.

(2) The need for tractable analysis of the safety problem.

1.2.1. *Weak safety properties of HRU*

The most general access control model, the access matrix model of Harrison, Ruzzo, and Ullman [15] (HRU), has broad expressive power; unfortunately, it also has weak safety properties. Harrison and Ruzzo [16] proved the following demarcation for the decidability of safety analysis in HRU:

1) Safety is undecidable for *bi-conditional* schemes; i.e., the condition part of every command has at most two terms. In practical terms this means that there is no general algorithm for deciding safety for systems which test two cells of the access matrix.

2) Safety is decidable for *mono-conditional* schemes; i.e., the condition part of every command has at most one term.†

Mono-conditional systems are very restrictive and can accommodate only the simplest security policies. At the same time, models such as take-grant [17], which require bi-conditional commands, do have efficient safety analysis. Thus the demarcation of decidable safety in HRU is too pessimistic.

A restriction on expressive power that can have substantial benefits for safety analysis is that of monotonicity: monotonic models do not allow the deletion of access privileges. It must be noted that a strictly monotonic model is too restrictive to be of much practical use, since the ability to delete access privileges is an important requirement. We are really interested in models which can be reduced

† This result has only been shown for monotonic schemes and for nonmonotonic schemes with "Delete" but not "Destroy". The decidability for the general nonmonotonic case with "Destroy" remains open [16].

to monotonic models for purpose of safety analysis. In particular, we can ignore deletion of an access privilege P whenever the deletion can itself be undone by regranting P. This is by far the most common form of revocation and it is indeed fortunate that such revocation can be ignored for the purposes of worst case safety analysis.

Since monotonic models do not permit the deletion of access privileges, backtracking in analysis can be avoided. However, monotonicity by itself is insufficient for tractable safety analysis. The monotonic version of the access matrix model of Harrison, Ruzzo, and Ullman [15] (HRU) retains broad expressive power; unfortunately, despite its monotonicity, it also retains the weak demarcation of safety cited above. Safety analysis remains undecidable even for monotonic bi-conditional HRU schemes [16].

1.2.2. *Strong safety properties of SPM*

In response to the relatively weak safety properties of HRU, a variety of models† with more desirable safety properties have been proposed [6, 17, 25, 26, 27, 48]. However, a substantial gap in expressive power exists between these models and HRU. Sandhu's Schematic Protection Model (SPM) [38] was developed to fill this gap in expressive power while sustaining efficient safety analysis. The various models referenced above are all subsumed by SPM [39, 46]. SPM has remarkably strong safety properties and has been shown to represent a wide variety of cases of practical interest.

Despite SPM's demonstrated expressive power [38, 39, 46], attempts to show the equivalence of SPM to monotonic HRU have so far failed. We now have strong reason [2, 3, 24] to believe that SPM is in fact theoretically less expressive than HRU, although formal demonstration of this fact remains an important open question.

For the purpose of demonstrating theoretical equivalence, and also for practical reasons, SPM has been extended in a natural way. The single parent creation operation in SPM has been redefined to allow multiple parents for a child. With the joint creation operation, the new model, called ESPM, is shown here to be precisely equivalent to monotonic HRU.

The question naturally arises as how safety analysis is affected by the increase in expressive power provided by the joint creation operation. In this paper, we present a safety analysis algorithm for a restricted subclass of ESPM schemes, namely those schemes in which the create structure is acyclic with the possible exception of attenuating loops (particular kinds of cycles of length one). The algorithm given here is based upon the SPM algorithm presented in [38]. However, the machinery of [38] is insufficient for ESPM schemes. Here we demonstrate that the inclusion of a joint creation operation in the ESPM model still permits tractable safety analysis for many cases of practical interest. We also offer guidelines for how to employ the joint creation operation so as to minimize the computational effort of the safety analysis.

† These models generally include the kind of revocation in which the revocation itself can be undone. They are monotonic in the technical sense of being reducible to monotonic operations for purpose of safety analysis.

1.3. *Outline of the Paper*

The organization of the rest of this paper is as follows. In section 2 we provide motivation for including a joint creation operation in an access control model by describing the natural way in which it applies to well-known protection problems. In section 3, we formally describe ESPM, which is SPM extended with the joint creation operation, and give example implementations of the problems enumerated in section 2. In sections 4 and 5, we demonstrate the equivalence of monotonic HRU and ESPM. Section 4 gives a straightforward simulation of ESPM in monotonic HRU. Section 5 gives the far more intricate simulation of monotonic HRU in ESPM. In section 6, we present a safety analysis algorithm for ESPM schemes with acyclic attenuating loops (i.e., schemes whose can-create relations are acyclic except for certain cycles of length one). The computational complexity of analyzing such schemes is given, and it is shown that the complexity is feasible for many cases of practical interest. Section 7 contains a brief discussion, and section 8 concludes the paper.

2. Motivation for joint creation

In this section we motivate the utility of a joint create operation from a practical point of view by showing how this operation naturally supports a range of protection policies that have been proposed in the literature.

Our first example is that of *mutual suspicion*. This was one of the earliest protection problems identified in the literature [13]. The problem arises whenever two users, say A and B, who do not trust each other have to cooperate in achieving some task. The task requires that B has the ability to exercise a subset of A's privileges, and vice versa. The standard solution to this problem [13] is as follows.

(i) A creates a subject A'; similarly, B creates a subject B'.

(ii) A gives A' the privileges that B needs; similarly, B gives B' the privileges that A needs.

(iii) A gives B' the indirect privilege for A'; similarly, B gives A' the indirect privilege for B'.

At this point A' and B' act as agents for A and B in achieving their cooperative objective. The idea is that A' can indirectly exercise the privileges of B' and vice versa.

This solution depends critically on indirect privileges and as such requires a new concept and mechanism for its implementation. This concept is not particularly easy to formalize; for example, we must consider whether or not the indirect privilege can itself be indirectly exercised. The solution is correct only with the specific assumption that chaining of indirection is disallowed. It is clear that weak restrictions on the indirect privilege have disconcerting implications for access review and safety analysis, but strong restrictions amount to building more policy into our model than we really wish. Moreover, the algorithm described above does not give B any unique access to A', since A is free to send indirect privileges for A' to some other subject C.

On the other hand, the joint create operation provides an ideal solution for the mutual suspicion problem. A and B can jointly create a subject C such that A and B become the joint owners of C. Any pattern of communication from A and B to C and from C to A and B can then be specified. In the case of the mutual suspicion problem, A and B can be allowed to contribute privileges to C freely, but must be restricted in their ability to take privileges from C.

It is important to appreciate that once the joint create operation has occurred, the restrictions needed to solve the mutual suspicion problem can all be stated in terms of the operation that copies privileges from one subject to another. Since the copy operation is one of the fundamental operations that any access control model must support, the required ability to specify restrictions on the copy operation is independent of the mutual suspicion problem and joint creation.

Our second application of joint creation is in solving the well-known *protected subsystem problem* [10, 22]. In this problem we again have two parties A and B, where A is a user and B is a service that A wishes to use for some purpose. We model the invocation of B by A as the joint creation of subject C by A and B. Note that B is a passive participant in this act, while A and C are active subjects. Our requirements are as follows:

(1) A can only give data to C and receive results from C. In particular, A cannot obtain the rights to directly modify the internal data structures of C (i.e., we have information hiding in the sense of data abstraction).

(2) C can obtain data and code from B.

This differs from our mutual suspicion problem only in regard to what C can obtain from A and B. That is, we no longer have a symmetry between A and B. Given the ability to restrict the copy operation, it is a simple matter to specify these constraints in a sufficiently general access control model.

The *confinement problem* as originally formulated by Lampson [19] gives us our next application. This problem is actually a particularly stringent variation of the protected subsystem problem with the following added requirement.

(3) C cannot leak to anyone the data given it by A.

Since the code executed by C can only be obtained from B, A is threatened by Trojan Horses in B's code that might leak A's confidential data. To solve this problem, we need to ensure that C cannot write to any object other than its internal data structures and objects provided by A.†

From the joint creates perspective, the three examples above are all variations of the same requirement. The joint child C of A and B is restricted with respect to privileges it can obtain from A or B as well as privileges A or B can obtain from it. In general, the restrictions applied to parent A are different from those applied to parent B. A key point is that these restrictions are specified in terms of the copy operation to move privileges from one subject to another. The facility to specify such restrictions on the copy operation should certainly be available in any access control model that claims generality.

In these examples, the critical role of joint creation lies in binding C to its parents A and B at the moment of creation. The question naturally arises whether

† Of course, we also need a covert channel analysis to achieve a high level of assurance. In other words, we need to make sure that not only the explicit write operations but also the implicit ones have been accounted for.

or not joint creation can be reduced to more primitive operations. The reduction given in section 5 of monotonic HRU to ESPM strongly indicates that joint creation is fundamentally more powerful than the usual single-parent creation employed in access-control models. To appreciate why this happens consider an attempt to mimic the effect of joint creation of C by A and B as follows.

(1) Let A create C (using the usual single-parent creation). This binds A and C in a unique manner.

(2) Establish the B to C binding by copying privileges for C to B and/or vice versa.

The problem with this approach is that the means to achieve step 2 inevitably implies that a similar binding can also be established between C and some other subject D. In other words it appears impossible to uniquely bind C to B without introducing some new mechanism, such as joint create, for this purpose. As is shown in section 7, there are mechanisms other then joint create that can produce the desired effect. However, joint creation appears to be the most natural choice.

Our final application of joint creation stems from the *separation of duties* concept of Clark and Wilson [8]. Joint creation turns out to be a particularly effective way of specifying separation of duties with respect to creation of new users. As noted in [35, 40] separation of duties is often best expressed in terms of roles such as manager, security-officer, clerk, etc. For simplicity assume that each user has a unique role in the system. The following rules show how joint creation can specify the involvement of distinct users with different roles in the process of enrolling new users in the system.

(1) A manager and a security-officer can jointly create a new clerk.

(2) A senior-manager and a security-officer can jointly create a new manager.

(3) Senior-managers and security-officers can only be created by the system-owner.

In this case, joint creation is more concerned with involving multiple parties in the decision to effect the creation, in contrast to our earlier examples where the focus was on the unique binding between the child and its multiple parents. Note that while the joint creation operation offers an elegant solution for this last example, it is not strictly necessary. For example, the techniques described in [41] can be used to achieve separation of duties within the conventional framework of single-parent creation.

3. The ESPM model

In this section we define a formal model called ESPM (or extended SPM) that includes a joint creation operation. The model is based on Sandhu's Schematic Protection Model [38] or SPM. ESPM differs from SPM precisely in having a multi-parent create operation rather than the conventional single-parent creation of SPM. By way of introduction, we first review the formal definition of SPM, and then describe its extension to ESPM. For the sake of brevity motivational details and examples have been kept to the bare essentials. Further motivation and a variety of more complex examples are given in [38].

3.1. *Review of SPM*

SPM is based on the key principle of protection types, henceforth abbreviated as types. SPM subjects and objects are strongly typed, *i.e.*, the type of an entity

(subject or object) is determined when the entity is created and does not chan thereafter. Types are an abstraction of the intuitive notion of properties that a security relevant. An SPM scheme is, to a large extent, but not exclusively, defin in terms of types. The dynamic privileges in SPM are tickets of the form Y/r whe Y identifies some unique entity and r is a right. The notion of type is extended tickets by defining type(Y/r) to be the ordered pair (type(Y), r). That is the ty of a ticket is determined by the type of entity it addresses and the right symbol carries.

SPM has only two operations for changing the protection state, viz., crea and copy.† These operations are authorized by rules which comprise the schen defined by specifying the following (finite) components.

(1) Disjoint sets of subject types TS and object types TO. Let $T = TS \cup TO$.

(2) A set of rights R. The set of ticket types is thereby $T \times R$.

(3) A can-create function $cc : TS \rightarrow 2^T$.

(4) Create rules of the form $cr_p(u, v) = c/R_1 \cup p/R_2$ and $cr_c(u, v) = c/R_3 \cup p/R$

(5) A collection of link predicates $\{\text{link}_i\}$.

(6) A filter function $f_i : TS \times TS \rightarrow 2^{T \times R}$ for each predicate link_i.

The notation 2^X denotes the power set of X, *i.e.* the set of all subsets of X.

An SPM scheme is itself static and does not change. It constrains the evolutic of the protection state which is changed by create and copy operations. We no discuss these operations in turn.

The Create Operation. Creation is authorized exclusively by types. Subjec of type u can create entities of type v if and only if $v \in cc(u)$. Tickets introduced a the side effect of creation are specified by create-rules. Each create-rule has the tw components shown above, $cr_p(u, v) = c/R_1 \cup p/R_2$ and $cr_c(u, v) = c/R_3 \cup p/R$ where p and c respectively denote parent and child and the R_i are subsets of I When subject U of type u creates entity V of type v, the parent U gets the ticke V/R_1 and U/R_2. The child V similarly gets the tickets V/R_3 and U/R_4. Fo example, *file* $\in$ *cc(user)* authorizes *users* to creates *files*. And cr_p(*user, fil* $= c/\mathbf{rw}$ and cr_c(*user, file*) $= \emptyset$ gives the creator **r** and **w** tickets for the create file.

When an object is created, tickets for that object can only be placed in th parent's domain. The formalism expresses this in a straightforward manner b requiring $cr_p(u, v) = c/R_1$ and $cr_c(u, v) = \emptyset$, when v is an object type. Th situation for subject creation is more complicated. The SPM formalism allows th create rules to place tickets for the child subject in the child's domain and/or th parent's domain. It is obvious that this is a useful feature of the model. The SPI formalism also allows the create rule to place tickets for the parent subject in th child's domain and/or the parent's domain.

It is easy to make a case for placing the parent's tickets in the child's domai For instance, this may be the only way to set up a parent to child link (see Cop Operation) or vice versa as happens in the take-only and grant-only models c

† The original definition of SPM [38] included a third operation called demand that ha since been shown to be redundant [42].

[27]. The placement of tickets for the parent in the parent's own domain is a more subtle issue. It is a theoretical device which allows us to define the notion of an attenuating create rule as one in which each ticket for the child can be traced to the corresponding ticket for its parent. Attenuating create rules play an important role in safety analysis, as will be discussed in section 6. An alternative to this artifice, which would allow us to achieve the same objective, is to let the effect of the create rule depend on the current domain of the parent. This alternative was rejected in SPM because it introduces additional complexity in the notation and in safety analysis.

The Copy Operation. A copy of a ticket can be transferred from one subject to another, leaving the original ticket intact. Permission to copy a ticket Y/r depends in part on possession of the SPM copy flag, c, for that ticket, denoted Y/rc. Possession of Y/rc implies possession of Y/r but not vice versa. It is possible to copy Y/r only, or to copy Y/rc. In the former case, the ticket cannot be further copied, whereas, in the latter case, it may be further copied. Let $\mathrm{dom}(U)$ signify the set of tickets possessed by U. Three independent pieces of authorization are required to copy Y/r from U to V.

(1) $Y/rc \in \mathrm{dom}(U)$, *i.e.*, U must possess Y/rc for copying either Y/rc or Y/r.

(2) There is a link from U to V. Links are established by tickets for U and V in the domains of U and V. A link predicate takes two subjects, say X and Y, as arguments and evaluates to true or false. If true, it establishes a connection from X to Y that can be used to copy tickets from the domain of X to the domain of Y. Its definition is in terms of the presence of some combination of tickets for X and Y in the domains of X and Y. The idea is that the link predicate should be evaluated by examining the domains of the two subjects of concern only with respect to presence of tickets for these two subjects. We emphasize this aspect by saying that link predicates are local. That the definition should depend only on the presence and not the absence of tickets is a well-known principle for protection [37]. As a special case we also allow a link predicate which is always true to be defined.
Formally we have the following definition: the predicate $\mathrm{link}_i(U, V)$ is defined as a conjunction or disjunction, but not negation, of one or more of the following terms for any $r \in R$: $U/r \in \mathrm{dom}(U)$, $U/r \in \mathrm{dom}(V)$, $V/r \in \mathrm{dom}(U)$, $V/r \in \mathrm{dom}(V)$, and **true**. Some examples of link predicates from the literature are given below [25, 27, 38, respectively]:

$$\mathrm{link}_{tg}(U, V) \equiv V/g \in \mathrm{dom}(U) \vee U/t \in \mathrm{dom}(V)$$
$$\mathrm{link}_{t}(U, V) \equiv U/t \in \mathrm{dom}(V)$$
$$\mathrm{link}_{sr}(U, V) \equiv V/s \in \mathrm{dom}(U) \wedge U/r \in \mathrm{dom}(V)$$
$$\mathrm{link}_{u}(U, V) \equiv \mathbf{true}$$

(3) The last condition is defined by the filter functions f_i, one per predicate link_i. The value of $f_i(u, v)$ specifies types of tickets that may be copied from subjects of type u to subjects of type v over link_i. Also f_i determines whether or not the copied ticket can have the copy flag. Example values are $T \times R$, $TO \times R$, and $\emptyset$, respectively authorizing all tickets, object tickets and no tickets to be copied. Selectivity in the copy operation is controlled by the filter functio and specified entirely in terms of types. The filter functions are a powerful too for specifying policies. They impose mandatory controls which are inviolabl and confine the discretionary behavior of individual subjects.

In short Y/r can be copied from U to V iff there exists some link_i such that:

$$Y/rc \in \mathrm{dom}(U) \wedge \mathrm{link}_i(U, V) \wedge y/r \in f_i(u, v)$$

where the types of U, V and Y are respectively u, v and y. To copy Y/rc from U to V, it must also be the case that $y/rc \in f_i(u, v)$.

3.2. *Adding Joint Creation to SPM*

We extend the creation operation in the SPM model above to enable groups of subjects to jointly create new subjects and objects. The practical motivation for such an operation has been discussed earlier in section 2. We call the extended model ESPM, and we refer to the extended creation operation as *joint creation*, or simply *creation* in those cases where no confusion arises. Joint creation includes the SPM creation operation as a special case. In all other respects, ESPM is identical to SPM.

The (joint) can-create function for ESPM is a mapping:

$$cc : TS \times TS \times \ldots \times TS \rightarrow 2^T$$

In ESPM, the domain of cc is an N-tuple of subject types, as opposed to a single subject type in the SPM case. ESPM imposes no bound on the maximum value of N, although for any given scheme this value is, of course, a constant. Further if type constraints are met, we allow a subject to redundantly participate as more than one parent in a joint create operation. The option of forcing the parents to be unique was rejected because it is contrary to the tone of the SPM copy operation where the same subject can participate as the source, destination and reference of the transported ticket. It also seems natural to follow the lead taken in programming languages, where it is the usual practice to allow a single object to replace multiple formal parameters.

It remains to extend the create rules cr_p and cr_c of SPM to describe the distribution of tickets that results from joint creation. We have a variety of choices as to which tickets parents are allowed to acquire as a result of a joint creation operation. The most general choice is to allow the cr_p function to supply a parent with arbitrary tickets, not only for itself and the child, but also for any other parent. This option has the undesirable side effect of duplicating the functionality of the links and filter functions. Since we do not want the joint create operation to be a substitute for the copying of tickets, we restrict the cr_p operation such that a parent X does not acquire tickets of the form Y/r for any other parent Y.

With the above restriction in mind, the most general choice for specifying the distribution of tickets as a result of creation is to give a separate cr_p rule for each parent-child pair and a single cr_c rule that allows the child to acquire a different set of tickets from each parent. Formally, we define N create rules of the form:

$$cr_{p_i}(t_{p_1}, t_{p_2}, \ldots, t_{p_N}, t_c) = c/R_1^i \cup p_i/R_2^i \quad \text{for } i = 1 \ldots N$$

and one rule of the form:

$$cr_c(t_{p_1}, t_{p_2}, \ldots, t_{p_N}, t_c) = c/R_3 \cup p_1/R_4^1 \cup p_2/R_4^2 \cup \ldots \cup p_N/R_4^N .$$

The t_{p_i} are the types of the N parents, and the t_c is the type of the child. In all of the create rules c is the name of the jointly created entity and p_i is the name of the *ith* parent. For the ESPM create rules, note that the sets R_1, R_2, and R_4 from the SPM create rules have each been expanded into N sets, R_1^i, R_2^i, and R_4^i, for $i = 1 \ldots N$. The set R_3 is unchanged.

This completes our definition of ESPM. To summarize, ESPM is identical to SPM except for the create operation, which is extended in ESPM to allow multi-parent creation, thus going beyond the conventional single-parent creation of SPM.

3.3. *Examples*

We now outline ESPM solutions to the example problems posed in section 2.

3.3.1. *Example 1*

For the mutual suspicion problem, A and B have equal roles, and so we consider them to be of the same type; let s be the type of A and B. The joint agent, C, need not be of the same type as A and B; let t be the type of C. We set $cc(s, s) = t$, and define $cr_{p_1}(s, s, t) = cr_{p_2}(s, s, t) = \emptyset$ and $cr_c(s, s, t) = p_1/x \cup p_2/x$, where x is a special right that is used to tie the parents A and B to the child C. We restrict right x such that it cannot be acquired except through creation. Links that require the parent child tie can then be used by A and B to transfer a limited set of privileges to C.

3.3.2. *Example 2*

In the second example of the protected subsystem, the roles of A and B are no longer symmetric. We reflect this by setting the types of A and B to r and s, respectively. We now set $cc(r, s) = t$, and define $cr_{p_1}(r, s, t) = cr_{p_2}(r, s, t) = \emptyset$ and $cr_c(r, s, t) = p_1/x \cup p_2/xy$, where x is a right indicating permission to transmit data values, and y is a right indicating permission to transmit access to code. Now, with proper filters on the links connecting entities of types r, s, and t, A and C can exchange data only; but C can receive data and code from B.

3.3.3. *Example 3*

In the third example of the confinement problem, joint creation is accomplished with the same strategy as for Example 2. The filter functions can prevent C from leaking data to:

(1) Any subject, regardless of type. The filter functions thus prevent any access to $C's$ data.

(2) Subjects of a specific type. The filter functions allow access to certain subjects but deny (direct) access to others. Indirect access can be analyzed by safety analysis, as will be discussed in section 6.

The former case gives us total confinement while the latter gives partial confin ment.

3.3.4. *Example 4*

In the fourth example of separation of duties, a solution using joint creation mig be developed as follows: Suppose that the types c, m, sm, so, and o represent clerk managers, senior-managers, security-officers, and system-owners, respectively. V define the following cc functions:

(1) $cc(m, so) = c$: A manager and a security-officer can jointly create a new cler

(2) $cc(sm, so) = m$: A senior-manager and a security-officer can jointly create new manager.

(3) $cc(o) = \{sm, so\}$: Senior-managers and security-officers can only be create by the system-owner.

4. Reduction of ESPM to HRU

We now turn to a formal evaluation of the expressive power of ESPM. The mos general monotonic protection model to date is the monotonic access matrix mod of Harrison, Ruzzo, and Ullman [15], which we refer to as monotonic HRU. I turns out that ESPM is precisely equivalent to monotonic HRU in its expres sive power. The equivalence result (stated in theorems 1 and 2 below) is estab lished by simulating monotonic HRU in ESPM and vice versa. In this section we carry out the relatively straight-forward task of implementing ESPM in mono tonic HRU. In the next section, we demonstrate the much more difficult revers construction.

4.1. *Implementing ESPM in monotonic HRU*

The key idea here is to encode ESPM types as rights in the HRU simulation. W handle each aspect of the ESPM model in turn. Without loss of generality, w ignore the distinction between subjects and objects in the simulation.

(1) *Types.* These are encoded as rights as described below. The basic idea is that an entity X of type x will have the right x in the $[X, X]$ cell of the access matrix.

(2) *Rights.* The set of rights needed for the simulation is the set R_E from the ESPM scheme augmented with a new right for each ESPM type. Call this new set R_T. In addition, we need to represent the ESPM copy flag. For every right in R_E, define another right to represent that right along with the copy flag. Call this new set R_{Ec}. We adjust the simulation such that whenever a right from R_{Ec} is entered into the access matrix, the corresponding right from R_E is entered as well. Thus, $R_H = R_E \cup R_{Ec} \cup R_T$ is the entire set of rights.

(3) *Single Parent Creation.* Consider the cc function for each parent type t_p. Recall that the definitions of the create rules are: $cr_p(t_p, t_c) = c/R_1 \cup p/R_2$ and $cr_c(t_p, t_c) = c/R_3 \cup p/R_4$. For each t_c that t_p can create, define an HRU

command as follows:

HRU_{t_p,t_c}(X ,Y)
if $t_p \in$[X,X] **then**
create Y
enter t_c in $[Y,Y]$
/* for each r in R_1 */
enter r **in** $[X,Y]$
/* for each r in R_2 */
enter r **in** $[X,X]$
/* for each r in R_3 */
enter r **in** $[Y,Y]$
/* for each r in R_4 */
enter r **in** $[Y,X]$
end

(4) *Multiple Parent Creation.* Since an N-parent create can be implemented with 2-parent creates (as is formally shown in section 4.2 below), it is sufficient to consider only 2-parent creation operations. Recall that the definitions of the create rules for 2-parent creation are: $cr_{p_1}(t_{p_1}, t_{p_2}, t_c) = c/R_1^1 \cup p_1/R_2^1$, $cr_{p_2}(t_{p_1}, t_{p_2}, t_c) = c/R_1^2 \cup p_2/R_2^2$, and $cr_c(t_{p_1}, t_{p_2}, t_c) = c/R_3 \cup p_1/R_4^1 \cup p_2/R_4^2$. Consider the cc function for each type pair t_{p_1}, t_{p_2}. For each t_c that the pair can jointly create, define an HRU command as follows:

$HRU_{t_{p_1},t_{p_2},t_c}(X,Y,Z)$
if $t_{p_1} \in [X,X] \wedge t_{p_2} \in [Y,Y]$ **then**
create Z
enter t_c in $[Z,Z]$
/* for each r in R_1^1 */
enter r **in** $[X,Z]$
/* for each r in R_1^2 */
enter r **in** $[Y,Z]$
/* for each r in R_2^1 */
enter r **in** $[X,X]$
/* for each r in R_2^2 */
enter r **in** $[Y,Y]$
/* for each r in R_3 */
enter r **in** $[Z,Z]$
/* for each r in R_4^1 */
enter r **in** $[Z,X]$
/* for each r in R_4^2 */
enter r **in** $[Z,Y]$
end

(5) *Links and filters.* First, translate the predicate, $p_i(X,Y)$, associated with each link$_i$ into a form suitable for inclusion in the conditional part of an HRU command. This is easily done by rewriting each expression of the form $Y/r \in$

dom (X) as $r \in [X,Y]$. Second, for the corresponding filter function, $f_i(t_x, t_y$ for each ticket type $t_z/r_j \in f_i(t_x,t_y)$ define the following HRU command:

$HRU_{\text{link}_i,t_z,r_j}(X,Y,Z)$
if $p_i(X,Y) \wedge t_x \in [X,X] \wedge t_y \in [Y,Y] \wedge t_z \in [Z,Z] \wedge r_j c \in [X,Z]$ **then**
enter r_j in $[Y,Z]$
end

To allow for the copy flag in the filter function, define another HRU comman for each ticket type $t_z/r_j c \in f_i(t_x,t_y)$:

$HRU_{\text{link}_i,t_z,r_j c}(X,Y,Z)$
if $p_i(X,Y) \wedge t_x \in [X,X] \wedge t_y \in [Y,Y] \wedge t_z \in [Z,Z] \wedge r_j c \in [X,Z]$ **then**
enter $r_j c$ in $[Y,Z]$
enter r_j in $[Y,Z]$
end

The initial state for the simulation is created as follows:

(1) A row and column of the access matrix is created for each ESPM entity. L the type of a given entity X be x. The right x is entered into the $[X,X]$ ce of the access matrix.

(2) For every ticket P_Y/r that a entity P_X holds, enter r into the access matr cell $[P_X, P_Y]$.

The correctness of the construction is almost self-evident because there a simple one-to-one mapping from ESPM commands to HRU commands in th simulation. A formal statement of correctness is given below.

Theorem 1. *An ESPM subject X can hold a ticket of the form Y/r (or Y/rc) the original ESPM system iff the access matrix cell $[X,Y]$ can contain r (or rc) the original HRU system.*

PROOF. The proof follows from the construction. If X acquires the ticket Y/r (Y/rc) then it is possible for r (or rc) to appear in the $[X,Y]$ cell of the acce matrix by mimicking each ESPM command by the corresponding HRU comman Similarly, if r (or rc) $\in [X,Y]$ then it is possible for X to acquire the ticket Y/r (Y/rc) by mimicking each HRU command by the corresponding HRU command.

4.2. *Equivalence of N-parent and 2-parent Joint Creates*

As we demonstrate below, any multiple-parent creation can be implemented by set of 2-parent creations and the introduction of a fixed number of additional type rights, links and filter functions. Thus the N-parent creation operation technical does not provide more expressive power than 2-parent creation; the N-parent for of the operation can be simply regarded as a desirable convenience. In the curre context, the construction below demonstrates that simulating 2-parent ESPM HRU is sufficiently general to show that N-parent ESPM can be simulated in HR

We illustrate a construction for implementing N-parent creation with 2-pare creation in fig. 1. For simplicity, we show the special case in which the pare types and child types are all distinct. The construction can be extended to t

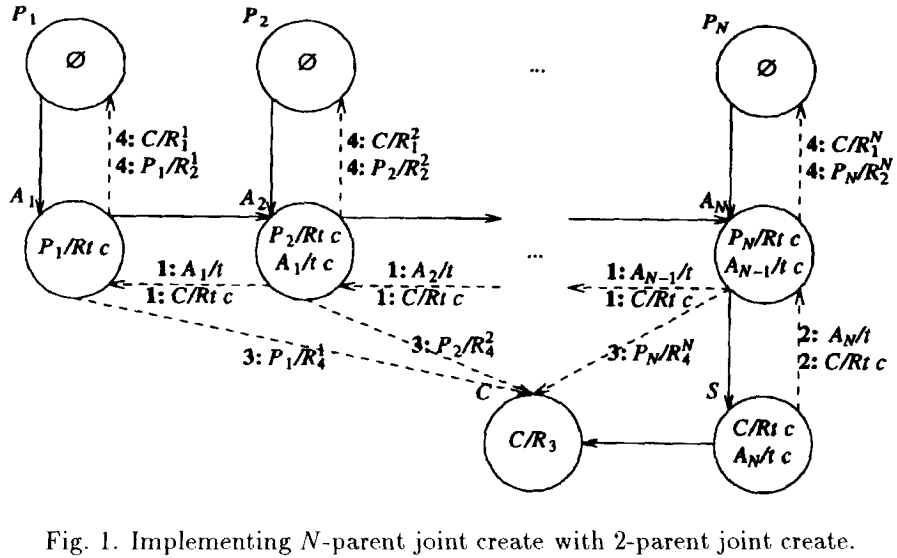


Fig. 1. Implementing N-parent joint create with 2-parent joint create.

general case in which the parent and child types are not all distinct, but the result is substantially more intricate and tedious to describe. The key aspect of the simulation is the manner in which 2-parent creates are chained together to simulate creation with an arbitrary number of parents. In addition the simulation requires detailed bookkeeping to ensure that creation tickets are properly distributed.

In fig. 1, entities are represented by circles. The name of each entity appears to the upper left of the entity, and the list inside the entity describes the tickets that the entity acquires as a result of creation. Solid arrows point from parents to children, and dashed arrows indicate the copying of the specified tickets. Labels on the copied ticket indicate the link used to copy that ticket.

Suppose we are implementing the N-parent creation operation specified by the can-create function, $cc(p_1, p_2, \ldots, p_N) = Z$, where Z is a subset of T, and the create rules are of the form given above. Consider a particular child type $c \in Z$. As per our discussion above, we are assuming that the p_i and c are all distinct.

Our construction requires that an additional $N + 1$ types be defined. We name the types $a_1, a_2, \ldots, a_N$ and s. The construction also requires an additional right, t, that is used to tie parents and children together. In the construction, each entity A_i of type a_i acts as an agent for the parent P_i. Each A_i eventually copies the tickets P_i/R_4^i to the child C. The entity S of type s acts as an agent for the child and eventually distributes the tickets C/R_1^i to the parents, P_i. The tickets C/R_3 are deposited directly into the child as a result of the create operation, but, to avoid inconsistency, the tickets P_i/R_2^i must be withheld from the parents until the entire joint create simulation is successful. To this end, the P_i/R_2^i tickets are stored in the A_i agents until the ticket A_i/t is returned to A_i by its child A_{i+1}, thereby indicating successful creation of the child entity.

As can be seen from the solid arrows in fig. 1, the replacement can-create functions are as follows:†

$$
\begin{aligned}
&cc(p_1) = a_1 \\
&cc(p_{i+1}, a_i) = a_{i+1}, \quad i < N \\
&cc(a_N) = s \\
&cc(s) = c
\end{aligned}
$$

The create rules are:

$$
\begin{aligned}
&cr_p(p_1, a_1) = \emptyset && cr_c(p_1, a_1) = p/Rtc \\
&cr_{p_1}(p_{i+1}, a_i, a_{i+1}) = \emptyset && cr_c(p_{i+1}, a_i, a_{i+1}) = p_1/Rtc \cup p_2/tc \\
&cr_{p_2}(p_{i+1}, a_i, a_{i+1}) = \emptyset && \\
&cr_p(a_N, s) = \emptyset && cr_c(a_N, s) = p/tc \\
&cr_p(s, c) = c/Rtc && cr_c(s, c) = c/R_3
\end{aligned}
$$

The creation rules arrange directly for the tickets c/R_3 to accrue to the child. Links and filter functions are required to deliver the tickets c/R_1^i and p_i/R_2^i to the *ith* parent and p_i/R_4^i to the child. These links are indirectly activated by the creation of the child entity.

For those links that require activation, the activation is represented by $A/t \in \mathrm{dom}\,(A)$, where A is an agent. The right t is also used to identify an agent's parent; $A_1/t \in \mathrm{dom}\,(A_2)$ indicates that A_1 is a parent of A_2. The links and filters are as follows:‡

$$
\begin{aligned}
&\mathrm{link}_1(A_i, A_{i-1}) = A_{i-1}/t \in \mathrm{dom}\,(A_i) \wedge A_i/t \in \mathrm{dom}\,(A_i) && f_1(a_i, a_{i-1}) = a_{i-1}/t \cup c/Rtc \\
&\mathrm{link}_2(S, A_N) = A_N/t \in \mathrm{dom}\,(S) && f_2(s, a_N) = a_N/t \cup c/Rtc \\
&\mathrm{link}_3(A_i, C) = C/t \in \mathrm{dom}\,(A_i) && f_3(a_i, c) = p_i/R_4^i \\
&\mathrm{link}_4(A_i, P_i) = P_i/t \in \mathrm{dom}\,(A_i) \wedge A_i/t \in \mathrm{dom}\,(A_i) && f_4(a_i, p_i) = c/R_1^i \cup p_i/R_2^i
\end{aligned}
$$

In the 2-parent simulation, the parents and child receive exactly the tickets that the N-parent operation specifies. In addition, the simulation arranges that no tickets are delivered to the parents unless the creation of the child is successful. Further note that the child cannot be created if any parent is absent. At worst, extra agents of the type a_i will be created, but these agents cannot interact with any other ESPM entities, and are therefore harmless. Thus the simulation disallows "partial" creation operations in the sense that nothing is effectively achieved unless the entire operation is permitted and carried out.

† Technically, the definition of $cc(p_1)$ is incomplete since entities of type p_1 may well be able to create entities of type other than a_1. Thus the definition of the can-create function for p_1 should be read as being augmented with a_1, not replaced by it.

‡ Note that these are templates for links. link_1 is a template for $N - 1$ links, $i \in 1 \ldots N - 1$. link_3 and link_4 are each templates for N links, $i \in 1 \ldots N$. Also, in the definitions of the filter functions, we adopt the convention that only nonempty f_i are explicitly given.

5. Reduction of HRU to ESPM

We now complete the evaluation of the expressive power of ESPM begun in the previous section. Below we show how an arbitrary protection scheme implemented in HRU can be simulated by an equivalent ESPM scheme. This simulation is inherently complicated because of the very local and incremental nature of the primitive operations of ESPM. A single HRU command has the potential to test and alter a large number of cells of the access matrix. These cells are not constrained to fall into any regular pattern. In ESPM on the other hand a link predicate can only test the domains of the two subjects at the end points of the link, and only for rights for the source and destination subjects. In other words, the ESPM link predicate can test exactly four cells of the access matrix. These are the [S,S], [S,D], [D,S] and [D,D] cells where S is the source subject and D the destination subject. Moreover the ESPM copy command can modify only one cell of the access matrix. Similar remarks apply to creation in ESPM vis-a-vis creation in HRU. It therefore takes considerable effort to simulate the wide-ranging tests and changes effected by an HRU command in terms of the very local and incremental tests and changes of ESPM commands.

In this section we first give the formal definition of the simulation. As the construction proceeds, we establish various lemmas showing what each piece of the construction achieves in the simulation. Once the construction is completed, we have the necessary lemmas to show overall correctness. At the end of the section we supply an example to illustrate the construction.

5.1. *Overview*

There are three basic problems in simulating monotonic HRU in ESPM. They correspond to the various parts of the HRU command:

(1) *Parameter List Generation.* HRU commands are invoked with a particular set of entities as parameters. For a valid simulation, it must be possible to manipulate exactly the set of entities that correspond to any possible HRU parameter list. Thus one task for any simulation of HRU is to mimic the gathering of arbitrary existing HRU entities into ordered parameter lists of the proper size. We conjecture that SPM is unable to provide this grouping operation; hence, in comparing the expressive power of SPM to ESPM, this is the key stage in the simulation of HRU. Clearly, the joint creation operation of ESPM is ideally suited for the task.

(2) *Validating the Conditional.* The basic process of an HRU command is to permit the various Create and Enter operations only if certain cells in the access matrix contain specified rights. A mechanism is required to simulate the evaluation of the term corresponding to each cell. Another mechanism is required for combining the values of the individual terms into an overall evaluation of the conditional.

(3) *Implementing Primitive HRU Operations.* Simulating "Enter" operations is straightforward in ESPM; it is easy to arrange that the ticket simulating the right in question be copied only if the conditional evaluates to $TRUE$. However, "Create" operations are another matter. ESPM does not contain a conditional creation operation. Therefore, in showing that an ESPM scheme can

simulate an HRU scheme, we must simulate conditional creation. Simulating conditional creation can be accomplished in various ways; we do it by augmenting the HRU scheme with an additional right that indicates that an entity is "alive". Thus, even though we cannot conditionally control the creation of ESPM entities, we can conditionally control the presence of tickets indicating liveness. We must also ensure that entities not marked as being alive do not participate in changing the protection state. Again, various options are available; we choose to augment the conditional expression in an HRU command to ensure that all relevant entities hold a "live" ticket. Since we are simulating monotonic HRU, we need not address the HRU "Delete" and "Destroy" operations.

5.2. *The monotonic HRU access matrix model*

We now formally define the monotonic access matrix model that we simulate. We denote the set of HRU rights by R_H. We consider I commands, each denoted HRU_i, $i \in 1 \ldots I$,† structured as follows.

$HRU_i(P_1, \ldots, P_{J_i}, P_{J_{i+1}}, \ldots, P_{J_i+M_i})$
if $T_1^i \wedge T_2^i \wedge \ldots \wedge T_{K_i}^i$ **then**
C_1^i
C_2^i
$\ldots$
$C_{M_i}^i$
E_1^i
E_2^i
$\ldots$
$E_{N_i}^i$
end

in which:

(1) The P_j, where $j \in 1 \ldots J_i$, are formal parameters representing existing HRU entities. The P_j, where $j \in J_{i+1} \ldots J_i + M_i$, are the names of entities that may be created by the HRU command. Note that this ordering of existing entities before to-be-created entities in the parameter list entails no loss of generality

(2) The T_k^i, where $k \in 1 \ldots K_i$, are terms of the form "$r \in [P_X, P_Y]$", where X and Y are in the set $1 \ldots J_i$. The absence of disjunctions in the conditional entails no loss of generality since disjunctions can be simulated with multiple HRU commands. Note, however, that negation is disallowed in HRU. We define a function, $\mathbf{t}_r^i(k) : 1 \ldots K_i \rightarrow R_H$, that describes which HRU right appears in the term T_k^i and functions, $\mathbf{t}_X^i(k) : 1 \ldots K_i \rightarrow 1 \ldots J_i$ and $\mathbf{t}_Y^i(k) : 1 \ldots K_i \rightarrow 1 \ldots J_i$, that describe, respectively, the positions of the formal parameters that assume the roles of P_X and P_Y in term T_k^i.

(3) The C_m^i, where $m \in 1 \ldots M_i$, are HRU primitives of the form "Create P_X" where $X = J_i + m$. Since we consider only the monotonic access matrix model

† The notation $x \ldots y$ is shorthand for the set, $\{i : N | x \leq i \leq y\}$, where N is the set of integers.

there is no "Delete P_X" operation, and we are free to order the HRU primitives so that all "Create" operations precede all "Enter" operations. We define a function, $\mathbf{c}^i_X(m) : 1 \ldots M_i \rightarrow J_{i+1} \ldots J_i + M_i$, that describes the position of the formal parameter X in C^i_m.

(4) The E^i_n, where $n \in 1 \ldots N_i$, are HRU primitives of the form "Enter right r in $[P_X, P_Y]$". X and Y are in the range $1 \ldots J_i + M_i$. We define a function, $\mathbf{e}^i_r(n) : 1 \ldots N_i \rightarrow R_H$, that describes which HRU right r appears in the the enter command E^i_n, and functions, $\mathbf{e}^i_X(n) : 1 \ldots N_i \rightarrow 1 \ldots J_i + M_i$ and $\mathbf{e}^i_Y(n) : 1 \ldots N_i \rightarrow 1 \ldots J_i + M_i$, that describe, respectively, the positions of the formal parameters that assume the roles of P_X and P_Y in term E^i_n.

For an illustration of these definitions, the reader is referred to the example given in Section 5.6.

5.3. *ESPM scheme definition*

Below, we define the types, rights, can-create function, creation rules, and links and filter functions required in the simulation of a given HRU scheme in ESPM. The description of the links and filter functions is quite detailed and intricate. For clarity, the presentation is broken up into discussions of the links required to achieve various subgoals. Each subgoal is stated as a lemma.

Type definitions

The entities used in the ESPM simulation can be grouped into various categories. Each category is used to mimic part of the evaluation of an HRU command. The categories are defined below.

(1) *Entities that mimic HRU entities.* We call these entities **proxies**. The simulation arranges that a **proxy** P_X can hold a ticket of the form P_Y/r for some HRU right r iff the HRU access matrix cell $[X, Y]$ can contain r. The single **proxy** type is p.

(2) *Entities to represent existing* **proxies** *in each possible parameter position of a single HRU command.* We call these entities **agents**. The number of types of **agents** is $J_{\max}$, which is the maximum value over the J_i, where $i \in 1 \ldots I$. The set of **agent** types is $\{a_j | j \in 1 \ldots J_{\max}\}$.

(3) *Entities to represent the collection of J_i existing HRU entities in the HRU_i command.* We call these entities **validators**. The creation of **validators** is the step that requires the joint creation operation. The **validators** assume a coordinating role in the simulation; they are responsible first for overseeing the simulation of HRU conditional evaluation, and then for enabling the simulation of "Create" and "Enter" primitives. There are I types of **validators**, one for each HRU command, HRU_i. The set of **validator** types is $\{v^i | i \in 1 \ldots I\}$.

(4) *Entities to collectively determine the truth of the entire conditional expression in an HRU command by examining each conjunct of the conditional in turn.* We call these entities **terms**. For each command HRU_i, there are K_i types of **terms**. The set of **term** types is $\{t^i_k | k \in 1 \ldots K_i, i \in 1 \ldots I\}$.

(5) *Entities to implement the HRU primitive* "Create P_X". We call these entities **creates**. For each command HRU_i, there are M_i types of **creates**. The set of **create** types is $\{c^i_m | m \in 1 \ldots M_i,\ i \in 1 \ldots I\}$.

(6) *Entities to implement the HRU primitive* "Enter r in $[P_X, P_Y]$". We call these entities **enters**. For each command HRU_i, there are N_i types of **enters**. The set of **creates** types is $\{e^i_n | n \in 1 \ldots N_i, i \in 1 \ldots I\}$.

In summary, the types required are $TS = \{p, a_j, v^i, t^i_k, c^i_m, e^i_n\}$ where the subscripts and superscripts have the ranges indicated above. Without loss of generality, we may ignore HRU objects and only consider HRU subjects, and thus set $TO = \emptyset$.

Rights

The set of rights, R, in the ESPM model is the union of two subsets:

(1) R_H, *the set of rights in the HRU model.* We include in this set an activation right a which marks a **proxy** as being "alive". Recall that the HRU conditional is augmented with tests that ensure that each participating entity is alive, *i.e.*, $a \in [P_X, P_X]$ for each parameter P_X.

(2) R_E, *the set of rights used in the simulation.* $R_E = \{x, y, t\}$ contains a right x that marks a **proxy** (or some **agent** acting on behalf of a **proxy**) for the role of P_X in considering an access matrix cell $[P_X, P_Y]$. Similarly, the right y marks a **proxy** (or **agent**) to take the P_Y role. In addition, the ESPM rights include a verification right, t. In the construction, t plays two roles. First t is used to indicate validity as follows: If $U/t \in \mathrm{dom}(U)$ for some ESPM entity U, then the part of the simulation that U corresponds to can be considered to have evaluated to **true**. Typically, links will be enabled by $U/t \in \mathrm{dom}(U)$. Second, the right t is used as a tie between ancestors and descendants of entities. In this case, links will be enabled by $U/t \in \mathrm{dom}(V)$ for some parent or child V of U.

Can-create function

The creation relations among the ESPM types is shown in fig. 2. Arrows in fig. 2 point from parent types to child types. Formally:

$$
\begin{aligned}
&cc(p) = \{a_j | j \in 1 \ldots J_{\max}\} \\
&cc(a_1, a_2, \ldots, a_{J_i}) = \{v^i\}, \text{ for } i \in 1 \ldots I \\
&cc(v^i) = \{t^i_1\} \cup \{c^i_m | m \in 1 \ldots M_i\} \cup \{e^i_n | n \in 1 \ldots N_i\}, \text{ for } i \in 1 \ldots I \\
&cc(t^i_k) = \{t^i_{k+1}\}, \text{ for } k < K_i \text{ and } i \in 1 \ldots I \\
&cc(c^i_m) = \{p\}, \text{ for } m \in 1 \ldots M_i \text{ and } i \in 1 \ldots I.
\end{aligned}
$$

Note that the joint create operation is only required for the construction of the **validators**. Also note that the structure is cyclic; hence safety is, in general, outside the cases known to be decidable for ESPM schemes built via the construction. This attribute is consistent with the weak safety properties of HRU. Observe that **proxies**, which are the ESPM entities that model HRU entities, can only be created by **creates**, which are the ESPM entities that correspond to HRU "Create" operations.

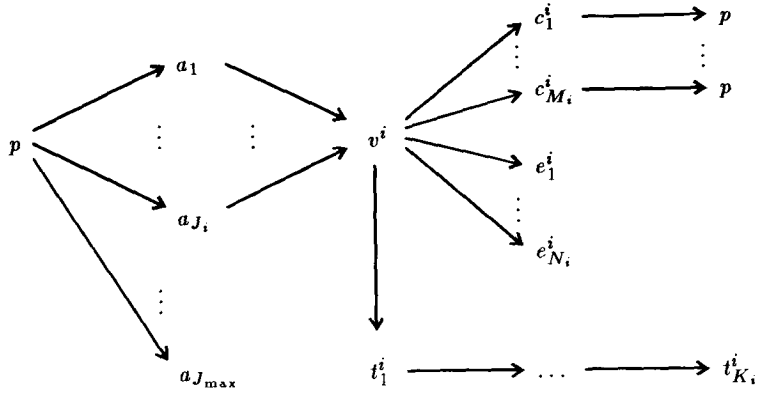


Fig. 2. Create and joint create graph

Creation rules

The create and joint create rules are:

$cr_p(p, a_j) = \emptyset$	$cr_c(p, a_j) = p/R_H \cup \{x, y\}c$
$cr_{p_j}(a_1, a_2, \ldots, a_{J_i}, v^i) = \emptyset,\ j \in 1 \ldots J_i$	$cr_c(a_1, a_2, \ldots, a_{J_i}, v^i) = \bigcup_{j \in 1 \ldots J_i} p_j/xyc$
$cr_p(v^i, t^i_1) = c/tc$	$cr_c(v^i, t^i_1) = p/tc$
$cr_p(t^i_k, t^i_{k+1}) = c/t\ c$	$cr_c(t^i_k, t^i_{k+1}) = \emptyset$
$cr_p(v^i, c^i_m) = c/xy\ c$	$cr_c(v^i, c^i_m) = \emptyset$
$cr_p(v^i, e^i_n) = \emptyset$	$cr_c(v^i, e^i_n) = p/t$
$cr_p(c^i_m, p) = c/R_H \cup \{x, y\}\ c$	$cr_c(c^i_m, p) = \emptyset$

Note that **agents** act as ticket sources for their **proxies**. Each **agent** has **proxy** tickets for all of the HRU rights plus the x and y marker rights. For newly created entities, the appropriate **create** entity serves as the ticket source. The distribution of tickets from creation is illustrated in fig. 3. Entities in fig. 3 are shown by circles, and solid arrows point from parents to children. Tickets acquired through creation are shown inside the relevant circle.

Links and filters

For evaluation of the conditional, we wish for a ticket V/tc (which starts out in the T_1 child of a **validator** V) to wend its way from one **term** to the next until T_{K_i} returns it to V iff the entire conditional is true. The links allow T_k to copy the ticket $V/t\ c$ on to its child T_{k+1} only if the conjunct that T_k represents has evaluated to true, a state which is represented by $T_k/t \in \mathrm{dom}\,(T_k)$. Entity T_{K_i} copies V/t back to V if the all of the conjuncts have evaluated to true. Once V acquires the V/t ticket, it activates its **enters**, which proceed to mark new **proxies** as existing and pass appropriate tickets to the proper **proxies**.

There are various subproblems in implementing this scheme. We describe each of the subproblems in turn and supply the links necessary to solve that subproblem. Each subproblem is illustrated with a figure. In the figures, entities are represented

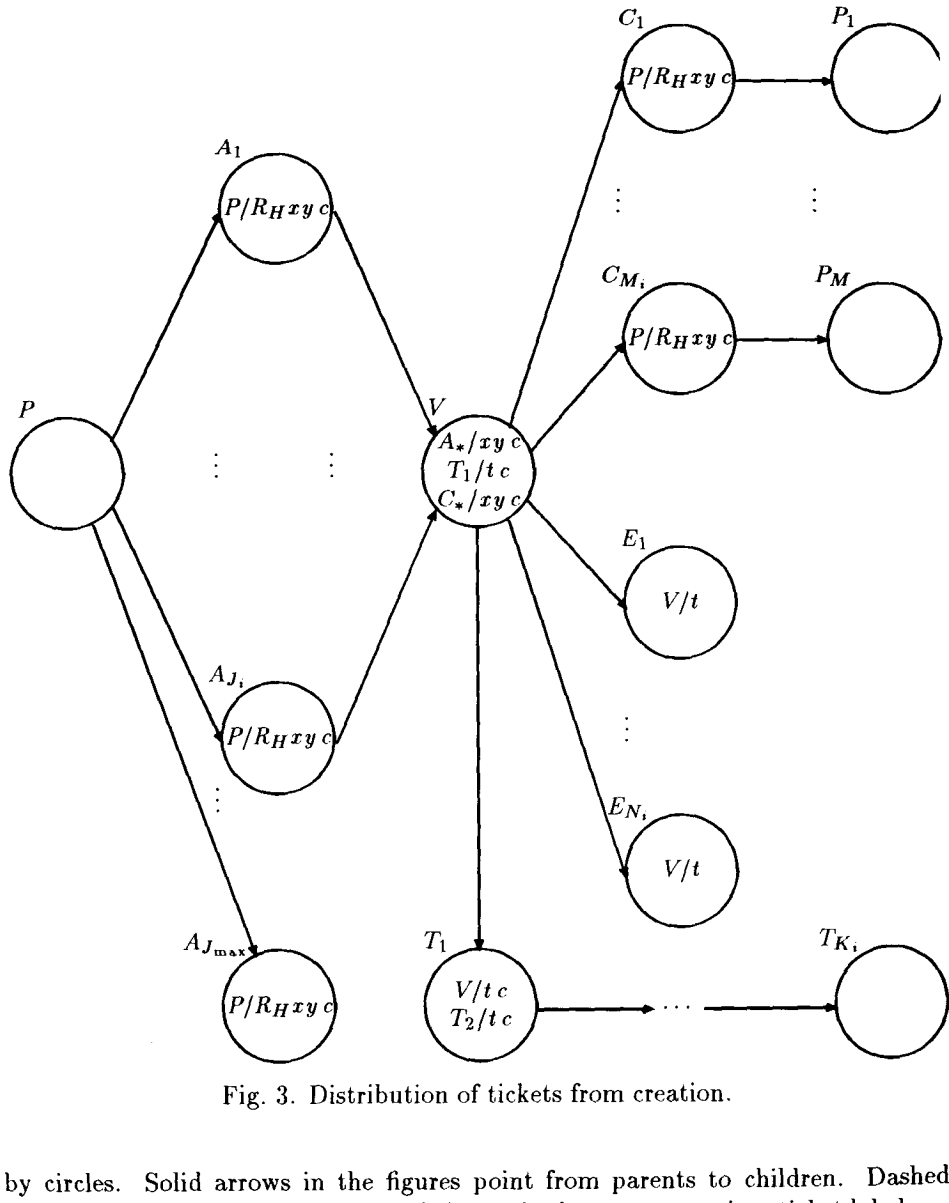


Fig. 3. Distribution of tickets from creation.

by circles. Solid arrows in the figures point from parents to children. Dashed arrows show the flow of the indicated ticket via the copy operation; ticket labels on a dashed arrow are prefixed with the number of the relevant link. Thus "**1:** AX/x" means that link $_1$ is used to transfer the ticket A_X/x.† Relevant tickets that are in the domain of an entity prior to the operation under discussion are shown inside the corresponding circle. For each subproblem, we state and prove a lemma to establish that the subproblem has been solved.

† Note that each link link $_i$ is in fact a template for a set of links, and not a single link. The same comment, of course, applies to filters f_i.

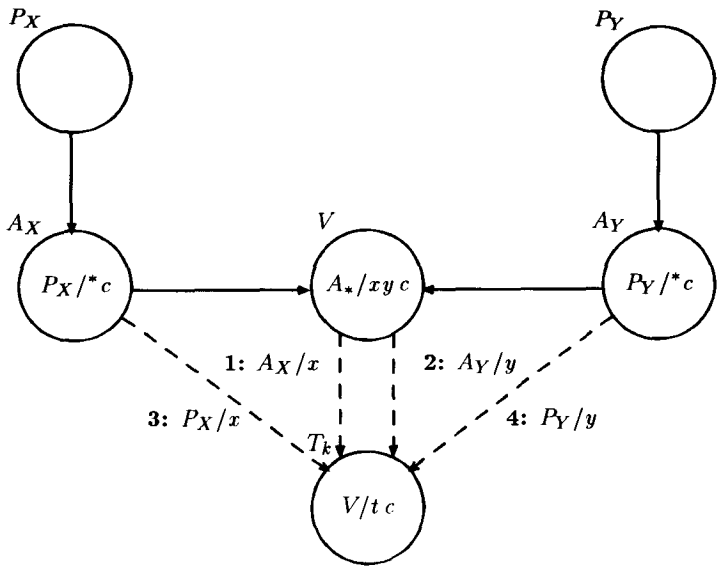


Fig. 4. Passing parameter tickets to T_k.

Passing parameters to terms

The first subproblem is for a **term** to determine which **proxies** assume the roles of P_X and P_Y in simulating the access matrix cell $[P_X, P_Y]$. Fig. 4 illustrates a solution to this problem.

In fig. 4, V acquires the tickets $A_j/xy\ c$, $j \in 1 \ldots J_i$, during the joint create operation. The first task is to transfer the proper tickets A_X/x and A_Y/y to T_k. The values of X and Y are determined by the formal parameters used in the kth conjunct of the conditional in the HRU command. Recall that this information is recorded in the functions $\mathbf{t}^i_X(k)$ and $\mathbf{t}^i_Y(k)$ (see Section 5.2). The next task is for T_k to obtain P_X/x and P_Y/y from A_X and A_Y, respectively. The formal definitions of these parameter passing links are given below.

$$\begin{array}{ll}
\text{link}_1(V, T_k) = V/t \in \text{dom}\,(T_k) & f_1(v^i, t^i_k) = a_{\mathbf{t}^i_X(k)}/x \\
\text{link}_2(V, T_k) = V/t \in \text{dom}\,(T_k) & f_2(v^i, t^i_k) = a_{\mathbf{t}^i_Y(k)}/y \\
\text{link}_3(A_j, T_k) = A_j/x \in \text{dom}\,(T_k) & f_3(a_j, t^i_k) = p/x \\
\text{link}_4(A_j, T_k) = A_j/y \in \text{dom}\,(T_k) & f_4(a_j, t^i_k) = p/y
\end{array}$$

This fragment of the ESPM scheme establishes the following property.

Lemma 5.1. **term** T_k *receives tickets* P_X/x *and* P_Y/y *iff in the simulated command,* HRU_i*, the* k*th term in the conditional tests the access matrix cell* $[P_X, P_Y]$.

PROOF. The "if" part of the proof is directly established by the construction above. For the "only if" part, note that the functions $\mathbf{t}^i_X(k)$ and $\mathbf{t}^i_Y(k)$ restrict the filter functions to tickets from the desired **agents**. By inspection of the rest of the scheme, it is clear that this is the only way for a term T_k to receive tickets for **proxies**.

In subsequent lemmas, the "only if" part of the proof consists of the last two sentences of this proof, appropriately modified to reflect the assertion of the lemma. For sake of brevity we only state the "if" part of the proof in these lemmas.

Passing parameters to enters

Similar processing is needed for passing parameters to each **enter**. Some additional complexity is introduced since new **proxies** as well as existing **proxies** can appear in an HRU "Enter" command, and these two cases need to be treated slightly differently. It turns out that for **enters**, P_Y/y is not needed in the domain of E_n since the agent ticket A_j/y is sufficient. Thus only P_X/x is copied to E_n. The two cases where P_X is an existing **proxy** and a newly created **proxy** are illustrated in fig. 5. We have the following definitions for the parameter passing links for **enters**.

$$\begin{array}{ll}
\text{link}_5(V, E_n) = V/t \in \text{dom}\,(E_n) & f_5(v^i, e^i_n) = \mathbf{e}^i_X(n) \le J_i? \\
\quad \wedge V/t \in \text{dom}\,(V) & \quad a_{\mathbf{e}^i_X(n)}/x : c^i_{\mathbf{e}^i_X(n)}/x^\dagger \\
\text{link}_6(V, E_n) = V/t \in \text{dom}\,(E_n) & f_6(v^i, e^i_n) = \mathbf{e}^i_Y(n) \le J_i? \\
\quad \wedge V/t \in \text{dom}\,(V) & \quad a_{\mathbf{e}^i_Y(n)}/y : c^i_{\mathbf{e}^i_Y(n)}/y \\
\text{link}_7(A_j, E_n) = A_j/x \in \text{dom}\,(E_n) & f_7(a_j, e^i_n) = p/x \\
\text{link}_8(C_m, E_n) = C_m/x \in \text{dom}\,(E_n) & f_8(c^i_m, e^i_n) = p/x
\end{array}$$

Note that link_5 and link_6 remain disabled until V acquires the ticket V/t, which ensures that the entire conditional has evaluated to **true**. This gives us the following property.

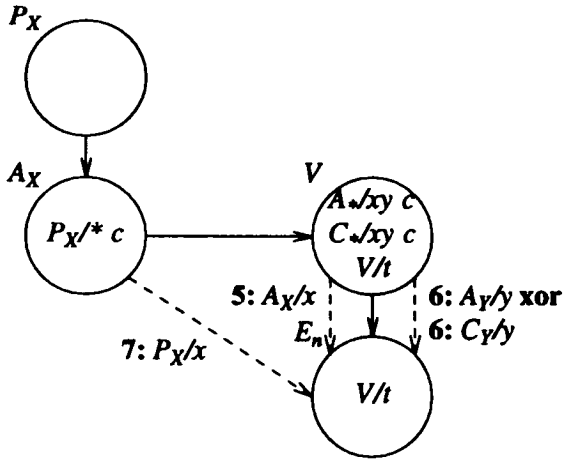
Lemma 5.2. **enter** E_n *receives tickets* P_X/x *and* A_Y/y *(*C_Y/y *if* $Y > J_i$*) iff in the simulated command,* HRU_i*, the* n*th Enter is into the access matrix cell* $[P_X, P_Y]$ *and the conditional has evaluated to* **true**.

PROOF. The distribution of P_X/x and A_Y/y (or C_Y/y) in the "if" part of the proof is shown by the above construction. That the distribution only occurs if the conditional is **true** is ensured by the tests in link_5 and link_6 on the predicate $V/t \in \text{dom}\,(V)$. The predicate $V/t \in \text{dom}\,(V)$ holds iff the entire conditional evaluates to **true** (see Lemma 5.4 below).

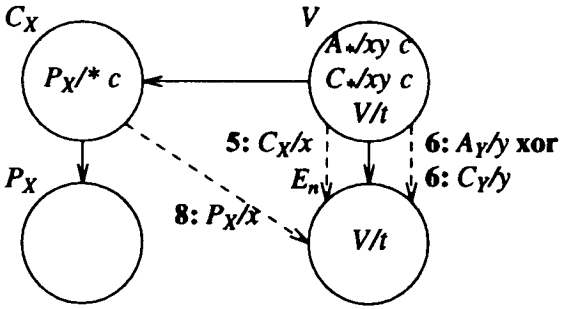
Evaluating terms

Another subproblem is for a **term** to determine the truth or falseness of the corresponding conjunct. Fig. 6 illustrates a solution to this subproblem. In fig. 6, the task is to determine if a specific **proxy** P_X holds a ticket P_Y/r for some specific HRU right r. In fig. 6a, this property is discovered by joint cooperation between

† The conditional expression notation $b?e_1 : e_2$ follows C semantics; *i.e.* if b evaluates to **true**, then the entire expression has value e_1, otherwise it has value e_2. Note that in this context the expression has the flavor of a macro expansion; the resulting ESPM scheme is still static.



a) P_X is an existing proxy

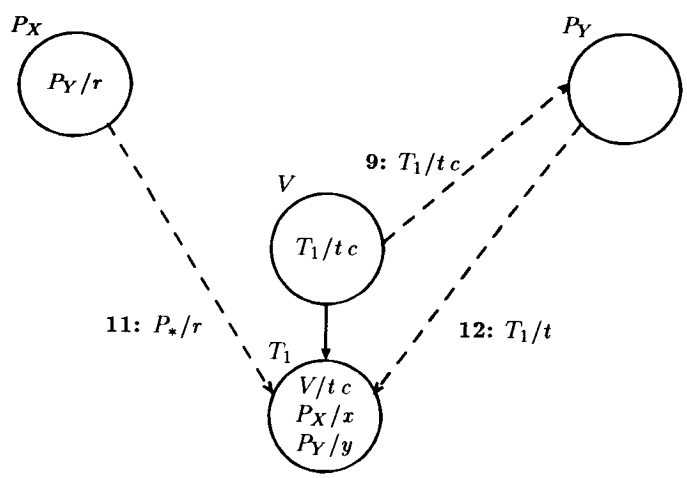


b) P_X is a newly created proxy

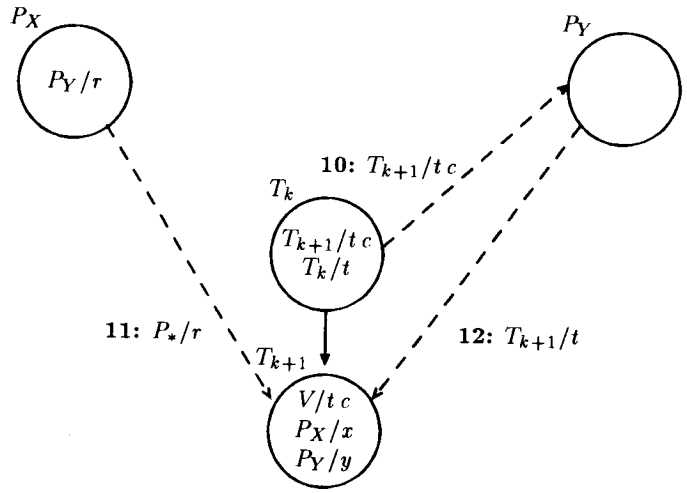
Fig. 5. Passing parameter tickets to E_n.

the **validator** V and its child T_1. In fig. 6b, the joint cooperation is similarly between the **term** T_k and its child T_{k+1}. It suffices to explain the latter case. First T_k copies verification tickets $T_{k+1}/t\ c$ to *all* **proxies**. T_{k+1} then receives tickets of the form P_*/r from **proxy** P_X for the specific right r determined by the function $\mathbf{t}_r^i(k+1)$ (see Section 5.2). Finally, T_{k+1} receives its verification ticket T_{k+1}/t from P_Y iff the ticket P_Y/r has been copied from P_X to T_{k+1}. The links are defined as follows:

$$\begin{aligned}
&\text{link}_9(V,P) = \textbf{true} && f_9(v^i,p) = t_1^i/t\ c\\
&\text{link}_{10}(T_k,P) = \textbf{true} && f_{10}(t_k^i,p) = k < K_i\,?\ t_{k+1}^i/t\ c : \emptyset\\
&\text{link}_{11}(P,T_k) = P/x \in \text{dom}\,(T_k) && f_{11}(p,t_k^i) = p/\mathbf{t}_r^i(k)\\
&\text{link}_{12}(P,T_k) = P/y \in \text{dom}\,(T_k) && f_{12}(p,t_k^i) = t_k^i/t\\
&\qquad \wedge P/\mathbf{t}_r^i(k) \in \text{dom}\,(T_k)
\end{aligned}$$



a) Term T_1.



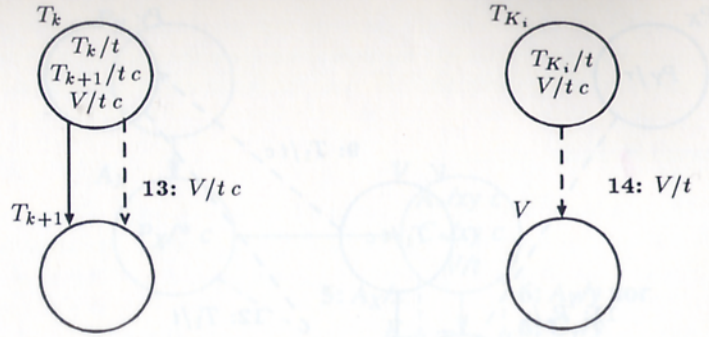
b) Term T_{k+1}.

Fig. 6. Evaluating terms in the conditional.

We have the following lemma.

Lemma 5.3. term T_k *receives the ticket* T_k/t *iff in the simulated command,* HRU_i*, the* k*th term in the conditional,* $\mathbf{t}_r^i(k) \in [P_X, P_Y]$*, evaluates to* **true** *and all previous terms (i.e. terms* $1 \ldots k-1$ *for* $k > 2$*) also evaluate to* **true**.

PROOF. The acquisition of T_k/t by T_k is shown by construction. The requirement that all prior terms evaluate to **true** is shown by inductively noting that T_k/t is not released by T_{k-1} until T_{k-1} has itself received T_{k-1}/t, *i.e.* until T_{k-1} has evaluated to **true**.



a) Typical step in evaluation b) Last step in evaluation

Fig. 7. Evaluating the entire conditional.

Evaluating the conditional

In addition to having each successive **term** evaluate whether $P_Y/r \in \mathrm{dom}\,(P_X)$, it is necessary to accumulate these results to determine the value of the entire conditional. This information is tracked by the V/t ticket, which passes each successive **term** as it evaluates to *true*. The process is illustrated in fig. 7. Part a) of fig. 7 illustrates the process in the middle of the conditional; part b) illustrates the process at the end. If a term T_k acquires the ticket T_k/t, it is free to copy $V/t\,c$ on to its child T_{k+1}, or, if $k = K_i$, copy V/t to V. This is achieved by the following link definitions.

$$\text{link}_{13}(T_k, T_{k+1}) = T_k/t \in \mathrm{dom}\,(T_k) \wedge T_{k+1}/t \in \mathrm{dom}\,(T_k) \qquad f_{13}(t^i_k, t^i_{k+1}) = v^i/t\,c$$
$$\text{link}_{14}(T_{K_i}, V) = T_{K_i}/t \in \mathrm{dom}\,(T_{K_i}) \wedge V/t \in \mathrm{dom}\,(T_{K_i}) \qquad f_{14}(t^i_{K_i}, v^i) = v^i/t$$

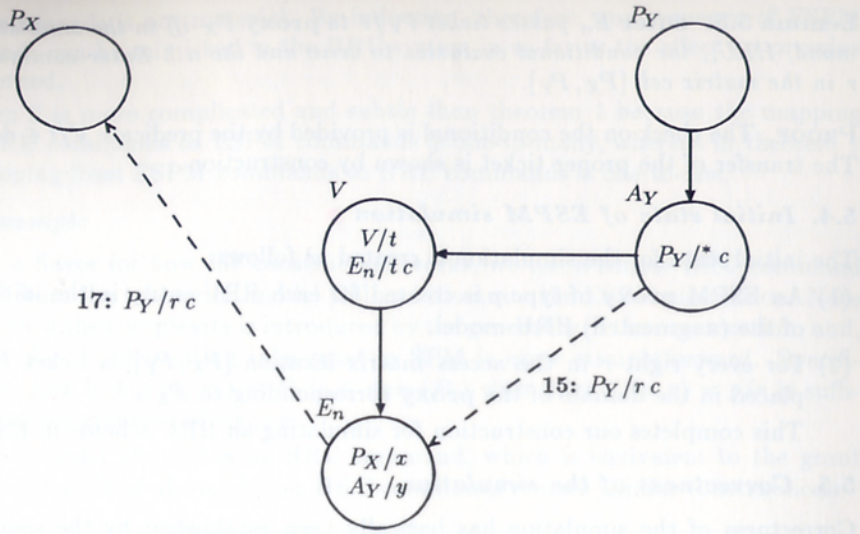
These link predicates give us the following property.

Lemma 5.4. validator V *receives the ticket* V/t *iff in the simulated command,* HRU_i*, the entire conditional evaluates to* **true**.

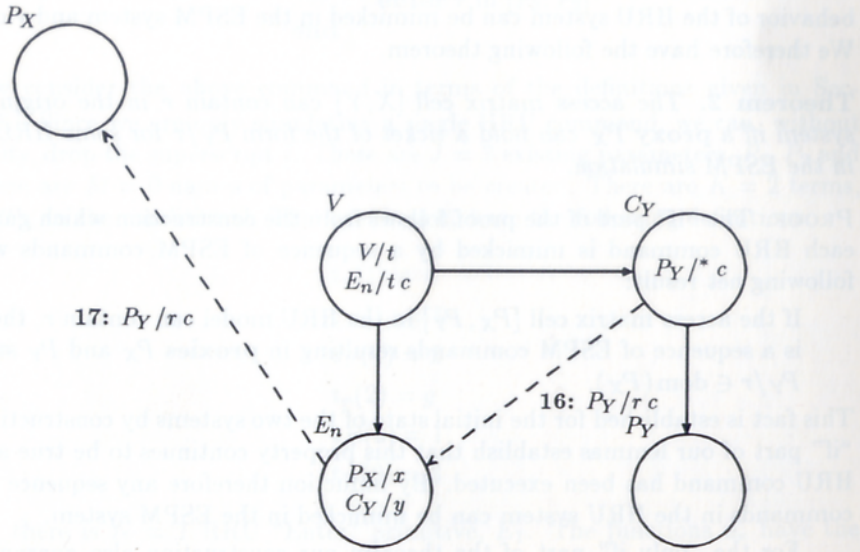
PROOF. V/t can only pass from one **term** to the next if the **term** evaluates to **true**. If each **term** evaluates to **true**, then the entire conditional evaluates to **true**. The mechanics of delivering V/t to V are shown by construction.

Simulating the enter command

The final subproblem is for an **enter** to pass a ticket giving a **proxy** a particular right to some other **proxy**. The task is for an **enter** to obtain an ticket $P_Y/r\,c$ and transmit it on to P_X. Since A_Y has all possible tickets for P_Y for existing **proxies**, (C_Y holds the corresponding tickets for new **proxies**), all that is required is for the links and filters from A_Y (or C_Y) to E_n to be restrictive enough to obtain exactly the correct ticket. E_n then forwards the ticket on to P_X. Fig. 8a illustrates a solution to this subproblem for the case in which the P_Y in the *Enter* command corresponds to an existing **proxy**. Fig. 8b illustrates the case in which P_Y corresponds to a newly created **proxy**. Activation of these links is taken care



a) Existing P_Y



b) Newly created P_Y.

Fig. 8. Entering P_Y/r into P_X.

of in the parameter passing links (see fig. 5), which are active only if the entire conditional has evaluated to **true**. The definitions are:

$$\text{link}_{15}(A_j, E_n) = A_j/y \in \mathrm{dom}\,(E_n) \qquad f_{15}(a_j, e^i_n) = \mathrm{e}^i_Y(n) \le J_i?p/\mathrm{e}^i_r(n)c : \emptyset$$
$$\text{link}_{16}(C_m, E_n) = C_m/y \in \mathrm{dom}\,(E_n) \qquad f_{16}(c^i_m, e^i_n) = \mathrm{e}^i_Y(n) \le J_i?\emptyset : p/\mathrm{e}^i_r(n)c$$
$$\text{link}_{17}(E_n, P) = P/x \in \mathrm{dom}\,(E_n) \qquad f_{17}(e^i_n, p) = p/\mathrm{e}^i_r(n)c$$

We have the following property.

Lemma 5.5. **enter** E_n *passes ticket* P_Y/r *to* **proxy** P_X *iff in the simulated command,* HRU_i*, the conditional evaluates to* **true** *and the nth Enter command puts* r *in the matrix cell* $[P_X, P_Y]$.

PROOF. The check on the conditional is provided by the predicate $V/t \in \mathrm{dom}\,(V)$. The transfer of the proper ticket is shown by construction.

5.4. *Initial state of ESPM simulation*

The initial state for the simulation is created as follows:

(1) An ESPM **proxy** of type p is created for each HRU entity in the initial state of the (augmented) HRU model.

(2) For every right r in the access matrix location $[P_X, P_Y]$, a ticket P_Y/rc is placed in the domain of the **proxy** corresponding to P_X.

This completes our construction for simulating an HRU scheme in ESPM.

5.5. *Correctness of the simulation*

Correctness of the simulation has basically been established by the sequence of lemmas established above. We need to show that the original HRU system and its ESPM simulation constructed above are equivalent in the following sense: the behavior of the HRU system can be mimicked in the ESPM system and vice versa. We therefore have the following theorem.

Theorem 2. *The access matrix cell* $[X, Y]$ *can contain* r *in the original HRU system iff a* **proxy** P_X *can hold a ticket of the form* P_Y/r *for some HRU right* r *in the ESPM simulation.*

PROOF. The "if" part of the proof follows from the construction which guarantees each HRU command is mimicked by a sequence of ESPM commands with the following net result:

> If the access matrix cell $[P_X, P_Y]$ in the HRU model can contain r, then there is a sequence of ESPM commands resulting in **proxies** P_X and P_Y such that $P_Y/r \in \mathrm{dom}\,(P_X)$.

This fact is established for the initial state of the two systems by construction. The "if" part of our lemmas establish that this property continues to be true after one HRU command has been executed. By induction therefore any sequence of HRU commands in the HRU system can be mimicked in the ESPM system.

For the "only if" part of the theorem our construction also guarantees the converse property.

> If there are **proxies** P_X and P_Y such that $r \in R_H$ and $P_Y/r \in \mathrm{dom}\,(P_X)$ then there is a sequence of HRU commands by which the access matrix cell $[X, Y]$ contains r.

Again, this fact is true for the initial state of the two systems by construction. Now consider a sequence of ESPM commands that enters P_Y/r in $\mathrm{dom}\,(P_X)$. The "only if" part of our lemmas establish that this can happen only if there is an HRU command whose condition evaluates to true and whose body includes the statement **enter** r in $[X, Y]$. This HRU command can therefore be executed in the HRU system. Since the systems are monotonic, the sequence of execution of the

HRU commands is not material. By induction, therefore, any sequence of ESPM commands can be mimicked in the HRU system in so far as the effect on **proxies** is concerned.

Theorem 2 is more complicated and subtle than theorem 1 because the mapping from HRU commands to ESPM commands is one-to-many, whereas in theorem 1 the mapping from ESPM commands to HRU commands is one-to-one.

5.6. *Example*

To give a flavor for how the construction works, we use a simple HRU command adapted from the take-grant model [25] and illustrate the ESPM simulation. Note that much of the complexity is introduced by the generality of the construction, and in particular, implementing take-grant in SPM is quite straightforward. Specifically, the link $\mathrm{link}_g(P_1, P_2) \equiv P_2/g \in \mathrm{dom}\,(P_1)$ with filter $f_g(p, p) = p/r$ is sufficient to allow granting of right r.

We consider the following HRU command, which is equivalent to the grant command described above. Other HRU commands require similar constructions.

$$
\begin{aligned}
&GRANT(P_1, P_2, P_3)\\
&\quad \textbf{if } r \in [P_1, P_2] \wedge g \in [P_1, P_3] \textbf{ then}\\
&\quad\quad \textbf{enter } r \text{ in } [P_3, P_2]\\
&\quad \textbf{end}
\end{aligned}
$$

We now consider the above command in terms of the definitions given in Section 5.2.† Since we are only simulating a single HRU command, we can, without ambiguity, drop the superscript i. There are $J = 3$ existing parameters, P_1, P_2 and P_3. There are $M = 0$ names of parameters to be created. There are $K = 2$ terms T_1 and T_2, in the conditional. The functions $\mathbf{t}_*$ have the following definitions:

$$
\begin{aligned}
\mathbf{t}_r(1) &= r\\
\mathbf{t}_X(1) &= 1\\
\mathbf{t}_Y(1) &= 2\\
\mathbf{t}_r(2) &= g\\
\mathbf{t}_X(2) &= 1\\
\mathbf{t}_Y(2) &= 3
\end{aligned}
$$

Finally, there is $N = 1$ HRU "Enter" primitive, E_1. The functions $\mathbf{e}_*$ have the following definitions:

$$
\begin{aligned}
\mathbf{e}_r(1) &= r\\
\mathbf{e}_X(1) &= 3\\
\mathbf{e}_Y(1) &= 2
\end{aligned}
$$

According to the construction, the ESPM scheme requires the following sets of rights: $R = \{g, r, x, y, t\}$. The g right is, of course, the grant right, and r is a

† Since no entities are being created, we can, for simplicity, ignore the activation right a. When complete, the augmented conditional reads: **if** $r \in [P_1, P_2] \wedge g \in [P_1, P_3] \wedge a \in [P_1, P_1] \wedge a \in [P_2, P_2] \wedge a \in [P_3, P_3]$ **then**...

generic HRU right. The other rights are the special rights used by the simulation. The types needed are: $T = \{p, a_1, a_2, a_3, v, t_1, t_2, e\}$.

We briefly recap the role of each type. A **proxy** of type p represents an HRU entity. An **agent** of type a_j represents **proxies** in parameter position j, $j \in 1 \ldots 3$. A **validator** of type v represents collections of triples of **proxies**. A **term** of type t_1 evaluates $r \in [P_1, P_2]$, and a **term** of type t_2 evaluates $g \in [P_2, P_3]$. Finally, an **enter** of type e implements "Enter g in $[P_2, P_3]$". Since there are no HRU *create* primitives, we do not need any **creates** of type c.

The creation relations among these types are:

$$cc(p) = \{a_1, a_2, a_3\}$$
$$cc(a_1, a_2, a_3) = \{v\}$$
$$cc(v) = \{t_1, e\}$$
$$cc(t_1) = \{t_2\}$$

The create and joint create rules are:

$cr_p(p, a_j) = \emptyset$	$cr_c(p, a_j) = p/\{g, r, x, y\}c,\ j \in 1..3$
$cr_{p_j}(a_1, a_2, a_3, v) = \emptyset,\ j \in 1 \ldots 3$	$cr_c(a_1, a_2, a_3, v) = \bigcup_{j \in 1 \ldots 3} p_j/xyc$
$cr_p(v, t_1) = c/t\ c$	$cr_c(v, t_1) = p/t\ c$
$cr_p(t_1, t_2) = c/t\ c$	$cr_c(t_1, t_2) = \emptyset$
$cr_p(v, e) = \emptyset$	$cr_c(v, e) = p/t$

The following links and filter functions are required. Recall that the link definitions given for link_i in the construction are simply templates for a variety of actual links. For simplicity, we do not separately identify links below; only the template name is given.

$\text{link}_1(V, T_1) = V/t \in \text{dom}(T_1)$	$f_1(v, t_1) = a_1/x$
$\text{link}_1(V, T_2) = V/t \in \text{dom}(T_2)$	$f_1(v, t_2) = a_1/x$
$\text{link}_2(V, T_1) = V/t \in \text{dom}(T_1)$	$f_2(v, t_1) = a_2/y$
$\text{link}_2(V, T_2) = V/t \in \text{dom}(T_2)$	$f_2(v, t_2) = a_3/y$
$\text{link}_3(A_1, T_1) = A_1/x \in \text{dom}(T_1)$	$f_3(a_1, t_1) = p/x$
$\text{link}_3(A_1, T_2) = A_1/x \in \text{dom}(T_2)$	$f_3(a_1, t_2) = p/x$
$\text{link}_4(A_2, T_1) = A_2/y \in \text{dom}(T_1)$	$f_4(a_2, t_1) = p/y$
$\text{link}_4(A_3, T_2) = A_3/y \in \text{dom}(T_2)$	$f_4(a_3, t_2) = p/y$
$\text{link}_5(V, E) = V/t \in \text{dom}(E) \wedge V/t \in \text{dom}(V)$	$f_5(v, e) = a_3/x$
$\text{link}_6(V, E) = V/t \in \text{dom}(E) \wedge V/t \in \text{dom}(V)$	$f_6(v, e) = a_2/y$
$\text{link}_7(A_3, E) = A_3/x \in \text{dom}(E)$	$f_7(a_3, e) = p/x$
$\text{link}_9(V, P) = \textbf{true}$	$f_9(v, p) = t_1/tc$
$\text{link}_{10}(T_1, P) = \textbf{true}$	$f_{10}(t_1, p) = t_2/tc$
$\text{link}_{11}(P, T_1) = P/x \in \text{dom}(T_1)$	$f_{11}(p, t_1) = p/r$
$\text{link}_{11}(P, T_2) = P/x \in \text{dom}(T_2)$	$f_{11}(p, t_2) = p/g$
$\text{link}_{12}(P, T_1) = P/y \in \text{dom}(T_1) \wedge P/r \in \text{dom}(T_1)$	$f_{12}(p, t_1) = t_1/t$
$\text{link}_{12}(P, T_2) = P/y \in \text{dom}(T_2) \wedge P/g \in \text{dom}(T_2)$	$f_{12}(p, t_2) = t_2/t$
$\text{link}_{13}(T_1, T_2) = T_1/t \in \text{dom}(T_1) \wedge T_2/t \in \text{dom}(T_1)$	$f_{13}(t_1, t_2) = v/t\ c$
$\text{link}_{14}(T_2, V) = T_2/t \in \text{dom}(T_2) \wedge V/t \in \text{dom}(T_2)$	$f_{14}(t_2, v) = v/t$
$\text{link}_{15}(A_2, E) = A_2/y \in \text{dom}(E)$	$f_{15}(a_2, e) = p/r\ c$
$\text{link}_{17}(E, P) = P/x \in \text{dom}(E)$	$f_{17}(e, p) = p/r\ c$

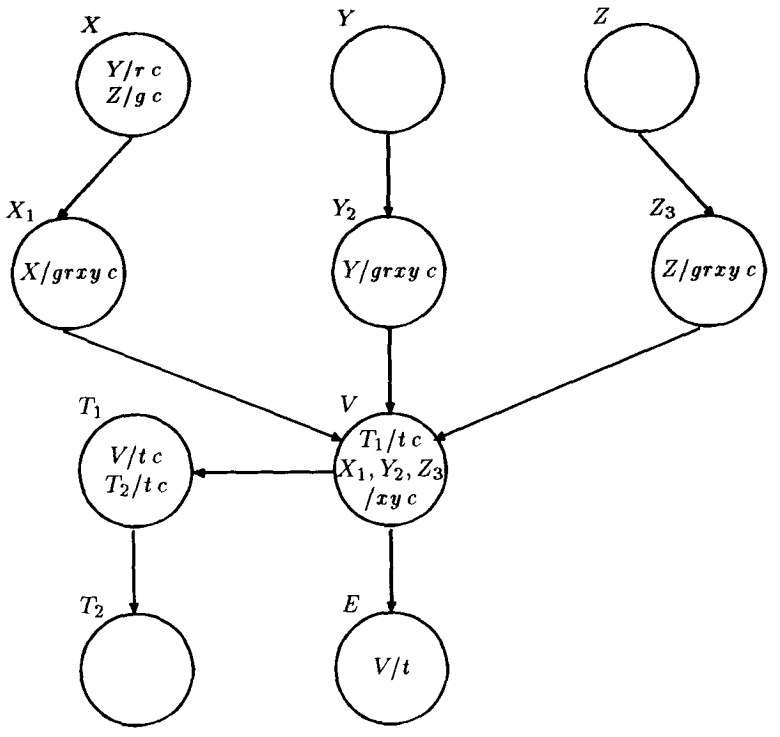


Fig. 9. Example creation of entities with initial tickets.

As an example, consider the following initial state for the HRU model: Assume there are three entities, $\{X, Y, Z\}$, and that the cell $[X, Y]$ contains r and the cell $[X, Z]$ contain g. Executing the HRU command GRANT (X,Y,Z) results in the cell $[Z, Y]$ acquiring right r.

The corresponding initial state for the ESPM simulation is the same initial three entities, X, Y, and Z, where the type of each entity is p. In addition, the tickets $Y/r\ c$ and $Z/g\ c$ are in the domain of X. Through the creation of entities and the exercise of the links defined above, it is possible for Z to acquire the ticket $Y/r\ c$. A sketch of the major events in this process is shown in below. The entities resulting from this sequence of relations is shown in fig. 9.

A possible log of the seven required creations is:

X creates an **agent** X_1 of type a_1

Y creates an **agent** Y_2 of type a_2

Z creates an **agent** Z_3 of type a_3

X_1, Y_2, and Z_3 jointly create a **validator** V of type v

V **creates a term** T_1 of type t_1

V **creates an enter** E of type e

T_1 **creates a term** T_2 of type t_2

A possible log of the 24 required copy operations is:

link_1 : V copies X_1/x to T_1
link_2 : V copies Y_2/y to T_1
link_3 : X_1 copies X/x to T_1
link_4 : Y_2 copies Y/y to T_1
link_9 : V copies T_1/tc to X, Y, and Z
link_{11} : X copies Y/r to T_1
link_{12} : Y copies T_1/t to T_1 /* term T_1 evaluates to true */
link_{10} : T_1 copies T_2/tc to X, Y, and Z
link_{13} : T_1 copies V/tc to T_2
link_1 : V copies X_1/x to T_2
link_2 : V copies Z_3/y to T_2
link_3 : X_1 copies X/x to T_2
link_4 : Z_3 copies Z/y to T_2
link_{11} : X copies Z/g to T_2
link_{12} : Z copies T_2/t to T_2 /* term T_2 evaluates to true */
link_{14} : T_2 copies V/t to V /* the entire conditional evaluates to true */
link_5 : V copies Z_3/x to E
link_6 : V copies Y_2/y to E
link_{15} : Y_2 copies Y/rc to E
link_{17} : E copies Y/rc to Z /* enter E_1 is implemented */

6. Safety analysis of espm

We now turn to analysis of the safety question, *i.e.* the determination of whether a given subject can ever acquire access to a given resource. In terms of a particular ESPM scheme, we wish to know if a given entity can ever acquire a particular ticket. The fact that safety analysis is undecidable for *arbitrary* ESPM schemes is immediate from either of the following two observations:

(1) As has been demonstrated above, ESPM is equivalent in expressive power to HRU, and safety is undecidable for HRU [15, 16].

(2) ESPM is a generalization of SPM, and safety is known to be undecidable for arbitrary SPM schemes [44].

However, relatively minor restrictions (from a practical point of view) on ESPM schemes permit decidable and indeed tractable safety analysis. The safety analysis of ESPM presented here is modeled on Sandhu's treatment of SPM in [38]. Although there are significant differences in the details, it turns out that, as in SPM [38], the tractability of safety analysis for ESPM given here is determined by the creation operation.

At a broad level, we adopt the following strategy for safety analysis:

(1) Starting with the given initial state, first create as many subjects as are necessary to account for the worst-case behavior of the system with respect to propagation of access rights. Call this the *canonical state* of the system.

(2) Given the canonical state, perform all copy operations until the state does not change any further. Call this the *maximal state* of the system.

(3) A specific safety question such as, "Can subject X obtain right r for object Y?" is then answered by looking at the maximal state and seeing whether or not X actually possesses the ticket Y/r in this worst-case state.

The second step in this procedure is guaranteed to terminate because the canonical state has a finite number of subjects, objects, and rights, and therefore the copy operations will eventually be unable to propagate any new privileges. The problem lies in the first step, where we need some criteria to determine when all the necessary create operations have occurred. In other words, we need to be able to recognize a canonical state. The undecidability result of [44] shows that it is impossible, in general, to recognize a canonical state. However, there are reasonable restrictions on the can-create function of SPM which make construction of the canonical state straightforward [38]. In particular, if can-create has no cycles or only has the so-called attenuating cycles of length one, the canonical state can be constructed by an operation called *unfolding*. These restrictions on can-create are eminently reasonable, as evidenced by the fact that no practical policy to date has required non-attenuating cycles in can-create [38, 39, 46].

Before we apply the general approach given in [38] to ESPM, we first review and summarize the analysis from [38] that is required as background. Section 6.1 presents the general concepts of maximal flow and maximal states employed in [38]. Section 6.2 reviews the proof that maximal states are finite and always exist, even though they may not be recognizable.

The remaining sections describe how the analysis of [38] is modified and augmented to apply to ESPM. In section 6.3 we define the analog of SPM's acyclic attenuating create operations in ESPM. This is a straightforward generalization and the restriction makes it possible to compute maximal states. Section 6.4 defines the analog of SPM's surrogate function in ESPM which we call the ID function. This requires some care because of the multi-parent creation in ESPM. Section 6.5 defines the construction of a canonical state in ESPM by unfolding. Unlike in SPM we have to be careful regarding the sequence of creates to demonstrate that the unfolding algorithm terminates for acyclic attenuating creates. Section 6.6 gives a proof that the given construction answers the safety question, and section 6.7 discusses the complexity of safety analysis including some observations on how the complexity can be kept manageable in practice.

6.1. *Safety analysis from the maximal state*

In this section we develop the terminology and concepts required for analyzing the safety of ESPM schemes. A change in state caused by a single copy or create operation is called a *transition*. A transition is *legal* provided there is proper authorization for the operation causing it. A *history* is a sequence of legal transitions. Histories are denoted by upper case letters and states by lower case letters or special symbols. Unless otherwise mentioned, a history is applied to the initial state. Any state that can be derived by a history is *derivable*.

In analysis, we are interested in functions and relations which depend on the state, e.g., dom and link_i. When appropriate, we qualify these with a superscript to identify the state, e.g., dom^h and link_i^h identify the context as state h. The initial state is identified by the superscript 0. The set of subjects and entities in state h are respectively denoted by SUB^h and ENT^h. Both dom and link_i exhibi

a monotonic property because of the absence of revocation and deletion, i.e., if g is derived from h then $\text{link}_i^h \subseteq \text{link}_i^g$ and $\forall A \in SUB^h$, $\text{dom}^h(A) \subseteq \text{dom}^g(A)$. Because the functions and relations used in analysis depend on the presence rather than absence of tickets in domains, this monotonic property extends to all functions and relations we consider.

The *flow* function expresses the authorization for copying tickets from one subject to another in a given state, accounting for indirect as well as direct copying. For every pair of subjects, its value is a set of ticket types determined by the state and scheme. Its definition is based on the following notion: There is a $path^h$ from A to B provided either one of the following conditions is true.

(1) In state h, there is an enabled link directly from A to B. E.g. $\text{link}_i^h(A, B)$.
(2) In state h, there is a directed sequence of enabled links indirectly connecting A to B.

In the former case we say the path is *single link*, whereas in the latter case the path is *multi-link* and *traverses* the intermediate entities.

Consider a multi-link path from A to B which traverses $C_1, C_2, \ldots, C_n$. Let $Y/xc \in \text{dom}(A)$. Y/xc can be copied from A to B using this path provided Y/xc can be copied across each link in the path. Further, Y/x can be copied from A to B using this path provided Y/xc can be copied across each link in the path from A to C_n and Y/x copied from C_n to B; that is the copy flag must be copied on all except the last link. This leads to the following definition: the *capacity* of a $path^h$ from A to B is as follows, where the types of A and B are respectively a and b.

(1) The capacity of a single-link path is $f_i(a, b)$.
(2) The capacity of a multi-link path is the intersection of the f_i over each intermediate link. In addition, the copy flag must be present in at least all but the last link.

Note that only the types of entities involved in this definition are significant, not their specific identities.

The concept of capacity readies us to define the flow function: For every state h define the *flow* function $flow^h : SUB^h \times SUB^h \rightarrow 2^{T \times R}$ by $flow^h(A, B)$ equal to the union of the capacity of all paths in state h from A to B. By convention $flow^h(A, A)$ is $T \times R$. Computation of $flow^h$ is straightforward in principle and of polynomial complexity in $|SUB^h|$. The operation is basically one of transitive closure.

The fundamental issue in analysis is to predict behavior of the flow function. This is especially so since create operations are authorized solely by the scheme, whereas copy is authorized by both the scheme and the distribution of tickets. Because $flow^h$ is monotonic, for a given pair of subjects it can only increase in derived states. From this fact, we show in section 6.2 that there exists a derivable state with the maximum value of $flow^h$ between all subjects in SUB^0. We call such a state a *maximal state*.

Let $flow^*$ denote the flow function in a maximal state. By definition, $flow^*$ specifies the ticket types that can be copied from A to B in the worst case, either directly, or indirectly via some other subjects. We can then easily determine if a specific ticket can be copied from A to B. The safety problem [15] poses the question whether or not it is possible to have a derivable state with $Y/x : c$ in

$\text{dom}(B)$.† The $flow^*$ function allows us to rewrite this question: is there any subject A who possesses Y/xc in the initial state and $type(Y)/x : c \in flow^*(A, B)$?

There typically are an unbounded number of maximal states. For example a system can continue to evolve indefinitely by creation of new subjects. The important point is that this can no longer increase the flow between A and B. As for SPM, the fundamental analysis question for ESPM is to compute a maximal state. Before we consider this problem we first show that maximal states exist.

6.2. *Existence and computation of maximal states*

The concept of maximal state is defined in terms of the initial set of subjects. To focus on changes in *flow* with respect to subjects in SUB^0, we introduce the following notions of reducibility and equivalence. A derivable state h is *reducible* to a derivable state g written $h \leq g$ if and only if $\forall A, B \in SUB^0 : flow^h(A, B) \subseteq flow^g(A, B)$.

For a given system two derivable states h and g are *equivalent*, written $h \equiv g$ if and only if $h \leq g$ and $g \leq h$. Because of its focus on the initial set of subjects, this equivalence relation partitions the derivable states into a finite collection of equivalence classes. For every pair of subjects in SUB^0, *flow* can take on at most $|2^{T \times R}|$ distinct values. Hence there are at most $|SUB^0|^2 * |2^{T \times R}|$ distinct equivalence classes, which is clearly finite.

We are now ready to formalize the notion of maximal state. For a given system m is a *maximal state* if and only if m is derivable and for every derivable state h $h \leq m$. Clearly, all maximal states are equivalent. The flow function in a maximal state completely defines the potential for copying tickets between subjects present in the initial state.

Proving the existence of maximal states is relatively straightforward [38]. Unfortunately, the proof is non-constructive and thereby does not provide a method for computing maximal states. In general, it is necessary to create new subjects to to derive a maximal state from the initial state. The problem is to determine which new subjects need to be created. In the general case, this is an undecidable problem for SPM, and hence also for ESPM. We do have exact solutions in several special cases of practical interest, the most important one of which we discuss below.

6.3. *Acyclic attenuating creation for ESPM*

In the safety analysis given for SPM in [38], the create function is restricted to be acyclic except for certain cycles of length one, or loops. The loops that are allowed are for create operations with *attenuating* create rules. Attenuating create rules specify that for those tickets acquired as a result of a create operation, the tickets acquired by the child are a subset of those acquired by the parent. The idea is that the parent can then simulate any possible action of the child, and thus the creation of the child may be ignored.

† For cases in which the role of the copy flag is not important, we use the notation $Y/x : c$ as an abbreviation for either Y/x or $Y/x\,c$, but not both. Thus in a particular context either all indicated tickets have the copy flag, or none does.

For the analysis of the joint create operation in ESPM, we similarly restrict the multiple-parent create function to be acyclic except for attenuating create loops. The restriction that at least one parent in a multiple-parent, loop-create operation must be able to simulate the child effectively means that, in general, child tickets may not be distributed to either the child or its parents, and that no parent tickets may be distributed to the child. Thus for creation operations in which the type v of the child is the same as the type u_i of one or more parents, we have the general rule:

$$cr_{p_i}(u_1, u_2, \ldots, u_N, v) = p_i/R_2^i \text{ for } i \in 1..N$$
$$cr_c(u_1, u_2, \ldots, u_N, v) = \emptyset.$$

Note that a parent can only use an attenuating loop create operation to increase the set of tickets that it holds for itself.

However, the general rule given above is too pessimistic in two important situations. First, for single-parent creation, these rules are relaxed, as was done in [38], so that the parent's tickets need only be a superset of the child's tickets. Second, for multi-parent creation, *one* of the parents with type matching the child may be treated in a similar manner; the child may receive tickets for itself and that one parent if that parent receives a superset of these tickets.† If multiple parents in an attenuating loop creation were allowed to receive tickets for the child, no single parent would be able to simulate the actions of the child. For this reason, attenuating create rules do not allow such a situation.

6.4. *The ID function*

To realize the strategy outlined at the beginning of this section, we need to provide a sufficiently rich set of canonical entities and show how to map actual ESPM entities onto canonical entities. In [38], a special function, called the *surrogate* function, is introduced to provide correspondence between canonical SPM entities and entities in arbitrary SPM histories. The surrogate function turns out to be inadequate for ESPM schemes because, except for *individual* entities in the initial state, the surrogate function is based strictly on the notion of type. For joint create, we need to capture the notion of grouping entities together. We therefore define a function, which we call the ID function, which assigns a canonical name or identification to every ESPM entity. This name effectively summarizes the ancestry of each entity and maps it to the canonical ESPM entity which simulates that entity.

The ID function is recursively defined below. Consider an entity V of type v. If V is not in the initial state, it is assumed to have parent(s) $U_1 \ldots U_N$ of types $u_1 \ldots u_N$, respectively. There are three cases to consider:

(1) If V is in the initial state then $ID(V) = V$. The ID of an entity in the initial state is simply the name of that entity.

(2) If $v \neq u_i$ for all $i \in 1 \ldots N$ then $ID(V) = C_v(ID(U_1), \ldots, ID(U_N))$. The ID of an entity that is strictly below all of its parents in the create/joint-create graph is simply a grouping of the ID's of the child's parents. The group is tagged with the child's type. Note that, since there are a finite number of types, we may effectively regard the C_v as constants.

(3) If $v = u_i$ for some $i \in 1 \ldots N$ then $ID(V) = ID(U_i)$ for the smallest i for which $v = u_i$. Since any child produced by an attenuating create rule can be simulated by at least one of the parents, the ID of an entity that has been created with an attenuating loop create rule is the ID of a parent of the matching type. Since there may be multiple parents with the same type as the child, we simply define the ID function to map to the first such parent.

† Without loss of generality, we may assume that the parent which is allowed to receive tickets for the child is the first appropriately-typed parent listed in the parameters of the can-create function. The assumption simplifies the ID function's definition, given below.

We define one canonical entity for each element in the range of the ID function. Thus questions about the correspondence between ESPM entities and canonical entities reduce to questions about the ID function. Our first task is to show that there are a finite number of canonical entities for acyclic attenuating schemes, *i.e.* that the range of ID is finite:

Lemma 6.1. *Given any acyclic attenuating ESPM scheme, the range of the ID function is a finite set.*

PROOF. Consider each of the three cases. Clearly, case 1 presents no difficulty since the case represents the base case of the recursion and there are a finite number of entities in the initial state. In a depth-first evaluation, Case 2 cannot be applied more often than there are types in the ESPM scheme due to the acyclic structure of the create graph. Since the tree structure introduced by case 2 has a bounded number of children at each node and a finite depth, the number of entries in the tree must be finite. Finally, case 3 does not alter the value of the ID function, so it may be invoked an arbitrary number of times without affecting the function's value. Since cases 1 and 2 are the only rules that can be used to generate distinct names, and since each case can only be applied a finite number of times, the range of the ID function is finite.

6.5. *The unfolding algorithm*

Our goal is to unfold an initial state and have the result be the complete collection of canonical entities, one entity for each element in the range of the ID function. As a first step, we define cc', to be the acyclic part of the function cc. Now, if we simply apply arbitrary rules from cc' first to entities in an initial state and then to entities in each resulting state, it is not at all clear whether the unfolding operation terminates. However, there is an order for applying the creation rules in cc' such that the unfolding process is guaranteed to terminate. The order depends upon the type of entity produced by a create operation, but not upon the parent type(s).

To develop the correct ordering, we reformulate the can-create function cc' as a relation. For each tuple of types $(u_1, u_2, \ldots, u_N)$ in the domain of cc', we replace the mapping $cc'(u_1, u_2, \ldots, u_N) = S$, where $S = \{s_1, s_2, \ldots, s_M\}$ is a subset of T, with the relation, $\{((u_1, u_2, \ldots, u_N), s_1), ((u_1, u_2, \ldots, u_N), s_2), \ldots, ((u_1, u_2, \ldots, u_N), s_M)\}$. Note that in each ordered pair in the relation, the abscissa, or first element, is the tuple of parent types and the ordinate, or second element, is a single child type. We refer to an ordered pair in this relation as a create-tuple.

Now we (partially) order the create-tuples based on the ordinate (i.e. the child type). That is, for every pair of types s and t, if s precedes t in the creation graph

Types:
$T = \{x, y, z\}$
Create Rules:
$cc(x, y) = \{y, z\}$
$cc(x) = \{x, y\}$
Acyclic Portion Of Create Rules:
$cc'(x, y) = z$
$cc'(x) = y$
Ordered Create Tuples from cc':
$((x), y)$
$((x, y), z)$
Initial State:
$\{X_1, X_2, Y_1\}$
Canonical State and ID function values:

X_1	$ID(X_1) = X_1$
X_2	$ID(X_2) = X_2$
Y_1	$ID(Y_1) = Y_1$
Y_2	$ID(Y_2) = C_y(ID(X_1)) = C_y(X_1)$
Y_3	$ID(Y_3) = C_y(ID(X_2)) = C_y(X_2)$
Z_1	$ID(Z_1) = C_z(ID(X_1), (ID(Y_1)) = C_z(X_1, Y_1)$
Z_2	$ID(Z_2) = C_z(ID(X_1), (ID(Y_2)) = C_z(X_1, C_y(X_1))$
Z_3	$ID(Z_3) = C_z(ID(X_1), (ID(Y_3)) = C_z(X_1, C_y(X_2))$
Z_4	$ID(Z_4) = C_z(ID(X_2), (ID(Y_1)) = C_z(X_2, Y_1)$
Z_5	$ID(Z_5) = C_z(ID(X_2), (ID(Y_2)) = C_z(X_2, C_y(X_1))$
Z_6	$ID(Z_6) = C_z(ID(X_2), (ID(Y_3)) = C_z(X_2, C_y(X_2))$

Fig. 10. Example of unfolding an initial state.

then every create-tuple with s as an ordinate must precede every create-tuple with t as an ordinate. We may then make the key observation, which is guaranteed by the acyclic structure of cc', that in the ordered list of create-tuples, every create-tuple that employs type t as a parent follows all of the create-tuples that produce type t as a child. The observation allows us to consider a create-tuple once during the unfolding process and be sure that the create-tuple will not be subsequently "re-enabled" with a new set of parents as a result of some later creation operation.

The *unfolding algorithm* to build the canonical state is as follows:

(1) Put the entities from the initial state into the canonical state.

(2) Reformulate the can-create function as create tuples as described above. Order the create tuples by ordinate (*i.e.* the child type) consistently with the partial order imposed by the creation graph. (Note that in general there are many possible orderings).

(3) Proceed down the ordered list of create-tuples and apply each create-tuple once to each possible tuple of parent entities in the canonical state. Place the resulting entity from each application of a create-tuple into the canonical state.

The unfolding algorithm is illustrated with an example in fig. 10. In fig. 10, entities are represented by subscripted upper case letters; the type of a given entity is the same letter in lower case.

We now establish two key results:

Lemma 6.2. *The unfolding algorithm terminates.*

PROOF. That the unfolding algorithm terminates can be seen by noting that no application of a particular create-tuple can result in either that create-tuple, or any other create-tuple considered before it, being applicable to a previously unconsidered tuple of entities. Thus for each create-tuple, there are a fixed number of applications possible. Since each create-tuple is considered only once, the procedure terminates.

Lemma 6.3. *The ID function applied to the unfolded state is a bijection.*

PROOF. The proof has two parts. First we show that the ID function produces a different value for each element in the unfolded state, *i.e.* that it is 1 to 1, or injective. Next we show that the range of the ID function when applied only to the unfolded state is, in fact, the entire range of the ID function, *i.e.* that it is onto, or surjective.

The first part of the proof proceeds by induction. Since the names of entities in the initial state are unique, the ID function is clearly injective when applied to the initial state. For the inductive hypothesis, assume that the ID function is injective when applied to entities in a partially unfolded state and consider the next application of the current create-tuple. The value of the ID function for the new entity must be constructed by applying case 2 of the definition of the ID function. Since each create tuple is considered once for each possible, unique tuple of parents, the value of the ID function, which lists both the identities of the parent entities and the type of the child entity, must differ from the ID's of all other entities in the partially unfolded state.

For second part of the proof, consider the ID function applied to some arbitrary entity. The value of the ID function is constructed by recursive applications of the the three rules that define the ID function. Since rule 3 maps the ID of a child to the ID of one of the child's parents, applications of rule 3 are not represented in the range of the ID function. Thus the range of the ID function is entirely determined by rules 1 and 2, both of which are exhaustively considered in the unfolding process.

Finally, we consider the attenuating loop portion of cc. We allow a single application of each attenuating rule to each possible entity or set of entities in the unfolded state. The resulting entities are *not* placed into the canonical state, since the IDs for these entities are already present. The effect is to supply each canonical parent with all tickets that can be acquired as a result of creation.

6.6. *Safety analysis for ESPM schemes*

We are now ready to prove the central result of this section. We show that every history for a given system can be simulated by a history without create operations applied to the fully unfolded state. Care has been taken with the definition of the ID function so that the following theorem and proof can directly parallel the presentation of theorem 17 in [38].

Theorem 3. *For an acyclic attenuating ESPM scheme, for every history H which derives h from the initial state there exists a history G, without create operations,*

which derives g from the fully unfolded state u such that:

$$\forall A, B \in SUB^h : flow^h(A, B) \subseteq flow^g(ID(A), ID(B))$$

PROOF. We may assume without loss of generality that H is in canonical form, i.e., all create operations precede all copy operations. G is obtained from H by replacing the individual transitions of H as follows while preserving the relative order.

(1) Ignore all create operations.

(2) Replace "copy $A/x : c$ from B to C" by "copy $ID(A)/x : c$ from $ID(B)$ to $ID(C)$."

We first establish the following assertions.

(1) Every transition in G is legal.

(2) $A/x : c \in \text{dom}^h(B) \Rightarrow ID(A)/x : c \in \text{dom}^g(ID(B))$.

(3) For every i, $\text{link}_i^h(A, B) \Rightarrow \text{link}_i^g(ID(A), ID(B))$.

Assertion 3 follows directly from assertion 2, and is crucial to the second part of the proof. Assertions 1 and 2 are proved by induction on the number of copy operations in H.

Basis Case

Let there be no copy operations in H, so that H consists of creates while G is empty.

Assertion 1

Since there are no transitions in G, assertion 1 is trivially satisfied.

Assertion 2

Without copy operations, there are only two ways by which $A/x : c$ can appear in $\text{dom}^h(B)$: either the ticket is present in the initial state, or it is introduced as a side effect of creation. If $A/x : c \in \text{dom}^0(B)$, then $ID(A) = A$ and $ID(B) = B$ so assertion 2 is trivially true. If $A/x : c \in \text{dom}^h(B)$ because of a create operation in H, assertion 2 follows from the construction of the canonical state.

Induction Step

Assume assertions 1 and 2 are true for every history with k copy operations, and consider a history H with $k+1$ copy operations. H consists of an initial sequence H' with k copy operations followed by a single copy operation. Let h' be the state derived by H'. Let G' correspond to H'. By induction hypothesis and assertion 1, G' is a history (i.e. every transition in G' is legal). Let g' be the state derived by G' applied to the unfolded state. Let the final operation of H be "copy $A/x : c$ from B to C." By construction, the final operation of G is "copy $ID(A)/x : c$ from $ID(B)$ to $ID(C)$."

Assertion 1

For the final operation of H to be legal, the following conditions must be true for some i.

(1) $A/xc \in \text{dom}^{h'}(B)$

(2) $\text{link}_i^{h'}(B, C)$

(3) $type(A/x : c) \in f_i(type(B), type(C))$

By the induction hypothesis and assertion 2 it follows that the first two conditions above respectively imply that

(1) $ID(A)/xc \in \text{dom}^{g'}(ID(B))$

(2) $\text{link}_i^{g'}(ID(B), ID(C))$

Since ID preserves types, it follows from the third condition above that

(3) $type(ID(A)/x : c) \in f_i(type(ID(B), type(ID(C)))$.

So the three conditions required to authorize the final operation of G are tru in state g', and the final operation in G is legal.

Assertion 2

h differs from h' at most by $A/x : c \in \text{dom}^h(C)$. By construction, th final operation of G ensures that $ID(A)/x : c \in \text{dom}^g(ID(C))$. Thi completes the induction step.

It remains to prove that $\forall A, B \in SUB^h : flow^h(A, B) \subseteq flow^g(ID(A), ID(B))$ We do so by showing that for every $path^h$ from A to B, there is a $path^g$ from $ID(A)$ to $ID(B)$ with the same capacity as the $path^h$ from A to B. The proof is by induction on the number of links. For the basis case, consider a $path^h$ from A to B of length 1, that is $\text{link}_i^h(A, B)$. By assertion 3, we have $\text{link}_i^g(ID(A), ID(B))$ Since ID preserves types, the basis case is true. Assume the hypothesis is true for every $path^h$ of length k, and consider a $path^h$ from A to B of length $k+1$ Then there is some C with a $path^h$ from A to C of length k and $\text{link}_j^h(C, B)$. By the induction hypothesis, there is a $path^g$ from $ID(A)$ to $ID(C)$ with the same capacity as the $path^h$ from A to C. By assertion 3 we have $\text{link}_j^g(ID(C), ID(B))$ Since ID preserves types, it follows there is a $path^g$ from $ID(A)$ to $ID(B)$ with the same capacity as the $path^h$ from A to B.

The essence of the theorem is that all histories applied to the initial state can be simulated by histories without create operations applied to the fully unfolded state u. Let $\#u$ be the no-creates maximal state which results from u as the initial state. We have the following corollary.

Corollary. *For a system with an acyclic attenuating scheme, $\#u$ is a maximal state.*

PROOF. From the theorem and definition of $\#u$, for every history H which derives state h from the initial state

$$\forall A, B \in SUB^h : flow^h(A, B) \subseteq flow^{\#u}(ID(A), ID(B)).$$

In particular, $\forall A, B \in SUB^0 : flow^h(A, B) \subseteq flow^{\#u}(A, B)$, because for such subjects $ID(A) = A$ and $ID(B) = B$.

6.7. *Complexity of safety analysis*

We now present the safety algorithm's cost in computational complexity and offer guidelines on how to minimize costs in practical applications. As is the case for SPM, the time required to construct the canonical state for ESPM is exponential in

the number of types if the *cc* relation is dense. This may not be a serious obstacle if the scheme employs relatively few types.

On the other hand, the required time is also multiply exponential in the number of parents of the joint creation operator. The reason is that each application of an N-parent create operation that produces a specific child type, v, results in on the order of I^N new entities being added in the worst case to the canonical state, where I is the number of entities in the canonical state before any entities of type v are produced. For each case in which the child of a joint create can participate as a parent in another joint create (excluding attenuating loops) this expansion is repeated.

Let x be the number of create-tuples derived from cc', N_i be the number of parents in the creation operation corresponding to the *ith* create-tuple for $i \in 1 \ldots x$, and I be the size of the initial state. Note that N_i may well equal 1, which corresponds to using the SPM create operation. The application of the first create-tuple results in a canonical state whose size, I_1, is on the order of $I_1 = O(I + I^{N_1})$. The application of the second create-tuple results in $I_2 = O(I_1 + I_1^{N_2})$. This process continues up to the complete canonical state I_x, $I_x = O(I_{x-1} + I_{x-1}^{N_x})$.

Clearly, the joint create operation needs to be used with great care to keep the analysis feasible. Some simple rules are:

(1) Use single parent creation where possible.

(2) Do not use a value for N that is any larger than necessary, *ie* keep the number of parents in joint creates as small as possible.

(3) Minimize the opportunities for descendants of a joint creation operation to participate in further creation operations.

(4) Avoid dense *cc* functions.

These appear to be reasonable guidelines which can be easily achieved in practice.

7. Discussion

Below we briefly enumerate issues that are raised by the demonstration of equivalence between ESPM and HRU and the safety algorithm presented in the previous section.

7.1. *Equivalence of ESPM/SPM*

We have argued that from a theoretical point of view, joint creation appears to confer additional expressive power not available with the traditional single-parent creation. In particular, we have shown that extending the create operation of SPM [38] in this manner gives us a model, ESPM, which is equivalent to the monotonic HRU model [16]. We conjecture that SPM is strictly weaker than ESPM; however we offer no direct proof. The relationship between the expressive power of ESPM and SPM remains an important open question.

7.2. *Ease of expression*

Whether or not joint creation does possess a fundamental expressive power lacking in the SPM model, joint creation clearly offers, as was shown in section 3, an easier and more natural way to express a variety of protection schemes. From a practical point of view, ease of expression is at least as important power of expression. Fortunately, as was shown in section 6, the additional expressive ease of joint creation does not have an undue cost on safety analysis, provided care is exercised in its use.

7.3. *Other extensions*

It is worth noting that there are a variety of alternate mechanisms that could be used in place of joint creation to extend SPM sufficiently to make it equivalent to monotonic HRU. Two such mechanisms are countdown tickets, a mechanism by which a ticket can only be used a fixed number of times, and use-once tickets, the limiting case of countdown tickets.

To simulate grouping N entities together with countdown tickets, a child is created with an N-countdown ticket. The child can then distribute this ticket to N entities, each of which can now act as if it is an actual (joint) parent of the child entity. Use-once tickets can simulate countdown tickets in the same manner in which 2-parent creation can simulate N-parent creation, a result which was shown in section 4.2.

Countdown tickets have the undesirable property of being non-monotonic; operations which are possible in a given state may not be possible in a subsequent state. Thus the analysis of schemes with countdown tickets is potentially forced to use a backtracking algorithm. In addition, the notion of joint creation appears to capture a natural operation, whereas countdown tickets appear to be more of an implementation mechanism.

7.4. *Safety in ESPM versus monotonic HRU*

Why is it that these two models are equivalent in expressive power yet have significantly different safety properties? We believe the main reason is due to the typing structure built into ESPM (which it inherited from SPM). The only means of distinguishing one row or column from another in the HRU model is by the contents of the individual cells. In (E)SPM (*i.e.*, SPM or ESPM) on the other hand rows and columns are distinguished by type as well as by the rights in individual cells. This behavior can be mimicked in HRU by encoding the type information into rights as shown in Section 4. In doing so, the simulation looses the distinction between types and rights. There is therefore no simple statement of what acyclic type-based creation means in HRU. In future work it would be interesting to look for models other than (E)SPM which have a type notion built into their primitive operations and are equivalent to (E)SPM. This would corroborate our conjecture that typing is fundamental to safe and expressive access control models.

8. Conclusions

The challenge with access control models is to provide adequate expressive power without sacrificing safety analysis. To date, models with broad expressive power, *eg* HRU, exhibit weak safety properties, and models with desirable safety properties exhibit less expressive power than HRU. In this paper we have established the remarkable result that by extending SPM with a joint creation operation, the

resulting model, called ESPM, simultaneously enjoys the expressive power of monotonic HRU and retains the strong safety properties of SPM. ESPM is therefore, in effect, an alternate formulation of HRU with strong safety properties.

We have given a complete proof of equivalence between monotonic HRU and ESPM. This is the first demonstration of the equivalence of two powerful access-control models which are clearly very different in their definitions. Additional results showing formal equivalence of radically different access-control models would advance our understanding of computer security. It is also important to obtain equivalence results for the non-monotonic case.

We have also presented a safety analysis algorithm for ESPM schemes with acyclic attenuating loops in the create structure, and we have given the computational complexity of the safety algorithm. Our analysis shows that safety is tractable for many cases of practical interest, provided some care is used in applying the create operations of ESPM. Guidelines have been discussed in section 6.1. The analysis is closely related to the safety analysis of SPM given in [38]. However, significant extensions are required to make the technique of [38] work for ESPM.

We are left with the important and difficult open problem regarding the relative power of ESPM and SPM. In other words, does joint creation offer a fundamental expressive power not available by single parent creation? This is not merely a significant theoretical question but also one of considerable practical interest to designers and implementors of operating systems and database management systems. We conjecture that SPM is less expressive than ESPM, and therefore less expressive than monotonic HRU.

Acknowledgment

The authors take this opportunity to thank Howard Stainer and Sylvan Pinsky for their support and encouragement, making this work possible. The authors also thank several anonymous referees whose insightful comments have considerably improved the clarity and correctness of the presentation. The authors have also benefited from conversations with Richard Lipton on the meaning of simulations between models.

References

[1] Abrams, M., Eggers, K., LaPadula, L., Olson I., "A Generalized Framework for Access Control: An Informal Description", *Proceedings Thirteenth National Computer Security Conference*, Washington, DC, October, (1990).

[2] 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).

[3] Ammann, P.E. and Sandhu, R.S., "Safety Analysis For The Extended Schematic Protection Model", *IEEE Computer Society Symposium On Research in Security and Privacy*, Oakland, CA, May, (1991).

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

[5] Biba, K.J., "Integrity Considerations for Secure Computer Systems", *Mitre Technical Report MTR-3153*, Bedford, MA, (1977).

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

[7] Boebert, W.E. and Kain, R.Y., "A Practical Alternative to Hierarchical Integrity Policies", *Proceedings Eighth National Computer Security Conference*, (1985), 18–27.

[8] Clark, D.D. and Wilson, D.R., "A Comparison of Commercial and Military Computer Security Policies", *Proceedings 1987 IEEE Symposium on Security and Privacy*, Oakland, CA, May, (1987).

[9] Clark, D.D. and Wilson, D.R., "Evolution of a Model for Computer Integrity", *Report of the Invitational Workshop on Data Integrity*, NIST Special Publication, (1989), 500–168.

[10] Cohen, E. and Jefferson, D., "Protection in the Hydra Operating System", *5th ACM Symposium on Operating Systems Principles*, (1975), 141–160.

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

[12] Downs, D.D., Rub, J.R., Kung, K.C. and Jordan, C.S., "Issues in Discretionary Access Control", *Proceedings 1985 IEEE Symposium on Security and Privacy*, Oakland, CA, May, (1985).

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

[14] Graubart, R.D., "On the Need for a Third Form of Access Control", *Proceedings Twelfth National Computer Security Conference*, Baltimore, MD, October, (1989).

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

[16] Harrison, M.H. and Ruzzo, W.L., "Monotonic Protection Systems", in *Foundations of Secure Computations* (DeMillo, R.A., Dobkin, D.P., Jones, A.K. and Lipton, R.J., eds.), Academic Press, (1978).

[17] 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*, (1976), 337–366.

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

[19] Lampson, B.W., "A Note on the Confinement Problem", *Communications of ACM* **16** (10), (1973), 613–615.

[20] LaPadula, L.J., "Formal Modeling in a Generalized Framework for Access Control", *Computer Security Foundations Workshop*, (1990), 100–109.

[21] Lee, T.M.P., "Using Mandatory Integrity to Enforce "Commercial" Security", *Proceedings 1988 IEEE Symposium on Security and Privacy*, Oakland, CA, May, (1988).

[22] Linden, T.A., "Operating System Structures to Support Security and Reliable Software."", *ACM Computing Surveys* **8** (4), (1976), 409–445.

[23] Lipner, S.B., "Non-discretionary Controls for Commercial Applications", *Proceedings 1982 IEEE Symposium on Security and Privacy*, Oakland, CA, April, (1982).

[24] Lipton, R.J., Personal communication, April, 1991.

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

[26] 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., eds.), Academic Press, (1978).

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

[28] McCollum, C.J., Messing, J.R. and Notargiacomo, L., "Beyond the Pale of MAC and DAC - Defining New Forms of Access Control", *Proceedings 1990 IEEE Symposium on Research in Security and Privacy*, Oakland, CA, May, (1990).

[29] McLean, J., "A Comment on the 'Basic Security Theorem' of Bell and LaPadula", *Information Processing Letters* **20** (2), (1985), 67–70.

[30] McLean, J., "Reasoning About Security Models", *Proceedings 1987 IEEE Symposium on Security and Privacy*, Oakland, CA, May, (1987).

[31] McLean, J., "The Algebra of Security", *Proceedings 1988 IEEE Symposium on Security and Privacy*, Oakland, CA, May, (1988).

[32] McLean, J., "Specifying and Modeling Computer Security", *IEEE Computer* **23** (1), (1990), 9–16.

[33] Millen, J.K., "A1 Policy Modeling", *Proceedings Seventh National Computer Security Conference*, (1984), 137–145.

[34] Miller, D.V. and Baldwin, R.W., "Access Control by Boolean Expression Evaluation", *Proceedings Fifth Annual Computer Security Applications Conference*, (1988).

[35] Nash, M.J. and Poland, K.R., "Some Conundrums Concerning Separation of Duty", *IEEE Symposium on Security and Privacy*, (1990), 201–207.

[36] Pittelli, P., "The Bell-LaPadula Computer Security Model Represented as a Special Case of the Harrison-Ruzzo-Ullman Model", *Proceedings Tenth National Computer Security Conference*, Baltimore, MD, October, (1987).

[37] Saltzer, J.H. and Schroeder, M.D., "The Protection of Information in Computer Systems", *Proceedings of IEEE* **63** (9), (1975), 1278–1308.

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

[39] Sandhu, R.S., "Expressive Power of the Schematic Protection Model (Extended Abstract)", *Computer Security Foundations Workshop*, (1988), 188–193.

[40] Sandhu, R.S., "Transaction Control Expressions for Separation of Duties", *4th Aerospace Computer Security Applications Conference*, (1988), 282–286.

[41] Sandhu, R.S., "Transformation of Access Rights", *IEEE Symposium on Security and Privacy*, (1989), 259–268.

[42] Sandhu, R.S., "The Demand Operation in the Schematic Protection Model", *Information Processing Letters* **32** (4), (1989), 213–219.

[43] Sandhu, R.S., "Terminology, Criteria and System Architectures for Data Integrity", *Report of the Invitational Workshop on Data Integrity*, NIST Special Publication, (1989), 500–168.

[44] Sandhu, R.S., "Undecidability of the Safety Problem for the Schematic Protection Model with Cyclic Creates", *JCSS* **44** (1) February, (1992).

[45] Sandhu, R.S., "Mandatory Controls for Database Integrity", in *Database Security III: Status and Prospects* (Spooner, D.L. and Landwehr, C., eds.), Elsevier Science Publishers B.V. (North Holland), (1990).

[46] Sandhu, R.S., "Expressive Power of the Schematic Protection Model", *The Journal Of Computer Security* **1** (1), (1992), 59–98.

[47] Shockley, W.R., "Implementing the Clark/Wilson Integrity Policy Using Current Technology", *Proceedings Eleventh National Computer Security Conference*, Baltimore, MD, October, (1988).

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