April, 1983

DESIGN AND ANALYSIS OF PROTECTION SCHEMES BASED ON THE SEND-RECEIVE TRANSPORT MECHANISM

Ravinderpal Singh Sandhu

DCS-TR-130

Department of Computer Science
Rutgers University
New Brunswick, NJ 08903

DESIGN AND ANALYSIS OF PROTECTION SCHEMES BASED ON THE SEND-RECEIVE TRANSPORT MECHANISM

by

RAVINDERPAL SINGH SANDHU

A thesis submitted to
The Graduate School
of
Rutgers, The State University of New Jersey,
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
Graduate Program in Computer Science

Written under the direction of
Professor Naftaly Minsky
and approved by

N. Minsky

[illegible]

Ann Yasuhara

[illegible]

M. C. Paull

New Brunswick, New Jersey
May 1983

ABSTRACT OF THE THESIS

Design and Analysis of Protection Schemes
Based on the Send-Receive Transport Mechanism
by RAVINDERPAL SINGH SANDHU
Thesis Director: Professor Naftaly Minsky

In a protection mechanism based on authorization, the ability of a **subject** (i.e., a user or a process) to operate on the system is determined by **privileges** in its **domain**. A mechanism for transport of privileges must accommodate a variety of policies, while permitting analysis of the privileges which a given subject might obtain. The **send-receive transport mechanism** was designed by Minsky with these objectives in mind. In this mechanism, a transport operation is explicitly authorized at both the source and destination, and the authorization is selective with respect to which privileges can be transported.

Here we study a restricted version of this mechanism. Under our restrictions a protected system is designed in two stages. Firstly, a **protection scheme** is defined by specifying the values of certain parameters, which determine the static component of every subject's domain. Secondly, the **initial state** is defined by specifying the dynamic component of every subject's domain. This state then evolves as permitted by the protection scheme.

We formulate the **flow-analysis problem** which is concerned with determining a bound on the authorization for transport of privileges, given a protection scheme and an initial state. We develop techniques for deriving and improving the desired bound. The major complication in doing so is the create operation, which permits the protection state to evolve in an unbounded manner. We investigate conditions which enable us to ignore the create operation. We also investigate conditions under which the initial authorization for transport of privileges remains invariant in every derived state.

We study additional analysis issues in the context of sub-classes of our design framework. The questions raised in such detailed analysis depend on the structure of these sub-classes.

ACKNOWLEDGMENTS

I take this opportunity to thank my advisor, Professor Naftaly Minsky, for his support and encouragement throughout my studies at Rutgers University. Naftaly's inspiring ideas and critical comments have strongly influenced the technical content as well as the actual presentation of this thesis.

I wish to give special thanks to Professor Ann Yasuhara for her critical study of my work. Her comments have contributed immensely to the clarity and correctness of the presentation.

I wish to thank the other members of my committee: Professors Alex Borgida, Abe Lockman and Marv Paull, for their insightful comments and time in reading the thesis.

Finally, I wish to thank my colleagues and friends at Rutgers University for an intellectually exciting environment which made my stay at Rutgers very enjoyable.

This thesis is dedicated to my wife Jaspreet for her patient and cheerful support during a demanding period of our lives.

TABLE OF CONTENTS

CHAPTER 1

THE PROBLEM AND ITS BACKGROUND

1.1. PROTECTION MECHANISMS

The discipline of **protection**[1] in computer systems is concerned with constraining the operations which a user (or a process) may invoke on various components of the system. The need for such constraints arises from the sharing of information and resources among multiple users.

A standard viewpoint is to regard a computer system as consisting of a set of **subjects** and a set of **objects**, denoted by SUB and OBJ respectively. The idea is that subjects model active entities such as users and processes, whereas objects model passive entities such as text files. We assume that these two sets are disjoint[2] and use the term **entity** to refer to either a subject or an object.

Every subject can autonomously invoke **operations** in a system. As examples of typical operations we have the following.

- A user reads a text file.
- A process wakes-up another process.
- A process creates a file and authorizes another process to write into this file.
- A user creates two processes and allows these processes to share files with each other.

---

[1]Protection is but one aspect of the general problem of security. For a classification of the problems in data security see Denning and Denning [2] or Denning [3].

[2]Some authors define subjects to be a subset of objects but we find our approach more convenient.

In a protected system, only those operations for which the subject has explicit permission will actually be executed. In a privilege based approach, the permission to invoke operations is represented by a set of **privileges** possessed by the subject and called its **domain**. The ability of a subject to invoke operations is then a function of the privileges in the subject's domain. The set of all such domains is called the **protection state** of the system.

A critical aspect of protection is that certain operations change the protection state. As instances of such operations, we have the last two examples enumerated above. These operations are themselves authorized by the protection state. The provision of such operations raises a host of challenging problems, some of which we investigate in this thesis.

There are then two major issues involved in the design of a **protection mechanism** based on privileges, viz.,

1. The question of what is a privilege and exactly how a privilege is represented.

2. The question of **dynamics**, by which we mean the available set of operations for changing the protection state and exactly how these operations are authorized.

In this thesis we are primarily concerned with one element of such dynamics, which we call the **transport of privileges**. This is that portion of the mechanism which controls movement of privileges from the domain of one subject to the domain of another. We will only concern ourselves with other aspects of a protection mechanism to the extent that they influence this movement. In particular, we ignore the issue of destruction or revocation[3] of privileges.

There are two fundamentally conflicting goals in the design of a transport mechanism:

- The mechanism must accommodate the rich variety of policies encountered in the real world.

- The mechanism must be analyzable so that we can determine the set of privileges a given subject might obtain.

---

[3]By no means are we saying that these issues are not important. Indeed, an implementation of our proposed mechanism will need to address these issues. However, in our analysis we finesse these issues by adopting a conservative worst-case scenario throughout.

The first objective calls for generality. Unfortunately, excessive generality makes analysis questions undecidable or intractable [6]. The problem then is to balance these conflicting goals.

In this thesis, we investigate a particular approach which caters to both these requirements. This approach is based on the dynamics of the operation-control mechanism of Minsky [12, 13, 14]. Before considering the specifics of our approach, let us first discuss what is meant by a privilege and how privileges are represented.

1.1.1. The Nature of Privileges

The capability approach for representing privileges in a subject's domain has been very popular in the context of operating systems[4]. In this approach, authorization for a subject A to invoke operations is expressed by a set of ordered pairs called **capabilities** or **tickets**. Each capability consists of two parts as follows.

1. The first part uniquely identifies some subject or object in the system.
2. The second part consists of a set of **rights**.

The domain of a subject then consists of a set of capabilities. For a particular subject A, we denote this set by dom(A). Possession of the capability B/x by subject A, authorizes A to perform certain operations on the entity B as determined by the set of rights x carried by the capability. Objects do not possess capabilities[5]. As an example, consider the following situation.

```
dom(A) = {D/ru, B/w}
dom(B) = {E/r,  B/s}
```

[4]The notion of a capability was first proposed by Dennis and Van Horn [4] and has an intuitive appeal. It has been incorporated directly in the addressing mechanisms of a computer, in systems such as HYDRA [1, 19] and CAP [18]. It has also been embedded in linguistic structures such as the capability managers of Kieburtz and Silberschatz [8].

[5]Throughout this thesis we will assume that objects do not possess privileges. There are then two aspects to the subject-object distinction, viz., that subjects are active while objects are passive, and that subjects possess privileges while objects do not. There are some problems in coupling both aspects. In particular, we cannot classify an entity which is passive but does possess privileges. We will see in section 2.4 how such entities can be accommodated in our approach. Intuitively, these entities are modeled as subjects whose capacity for autonomous action is negated by appropriate adjustment of their privileges.

Here, subject A is authorized to read and update file D and to wake-up process B. Process B itself is a subject, which is authorized to read file E and to put itself to sleep.

There are two critical assumptions which underlie a capability based approach, as follows.

1. All addressing in the system is done via capabilities.
2. Capabilities are **unforgeable** in that they cannot be constructed at will by a subject.

The second assumption raises the issue of dynamics which we will shortly illustrate by means of the take-grant mechanism.

In his operation-control mechanism, Minsky [12, 13, 14] introduced another kind of privilege called an **activator**. The motivation for doing so, is to express policies encountered in information systems. The general form of an activator is

$$\textbf{can-do}\ OP(\underline{p}) \longrightarrow \underline{q}\ \textbf{if}\ F(\underline{p},\underline{q})$$

where

OP: is the name of some operation
$\underline{p}$: is a vector of input patterns
$\underline{q}$: is a vector of output patterns
F: is a predicate called the qualifier

Each input pattern has the form p_i:[t/R], where t is a type and R is a set of rights. Similarly, each output pattern has the form q_j:[t/R]. We say that a ticket T/R' **matches** the pattern t/R if the entity T is of type[6] t and if R' is a subset of R. The symbols p_i and q_j merely serve as references for statement and evaluation of the predicate F.

Possession of an activator by a subject A, authorizes A to invoke the named operation. Successful execution of the operation depends on the following conditions.

1. Subject A must provide tickets which match the input patterns of the activator. The corresponding entities are then the actual arguments of the operation.

[6] In the context of objects, the notion of type corresponds to the well known notion of a data type. In the context of subjects, the type distinguishes various categories of subjects, e.g., systems programmers from ordinary users. The notion of subject types is critical to our work and is discussed at length in chapter 2.

2. The predicate F must evaluate to true on the actual arguments and outputs of the operation.

3. The tickets for the outputs of the operation must match the output patterns of the activator.

If the operation is successfully executed tickets for the actual outputs are then placed in the domain of A.

We now motivate the utility of activators by means of an example. Consider a situation where a number of users access documents maintained in a library. Assume that each document and each user has a security classification. The policy regarding access to documents is that every user A can read documents whose classification is less than or equal to the classification of A. This policy can be very simply expressed by placing the following activator in the domain of every user A.

can-do GET-READ-TICKET → q:[*doc*/read] **if** class(q) ≤ class(A)

Then A can obtain tickets with the read right for all documents of the designated classification. This activator has a special form in that there are no input patterns[7]. Such activators are called **demand** activators, and the activator above is equivalently represented as follows.

can-demand q:[*doc*/read] **if** class(q) ≤ class(A)

This example is one instance of a general class of policies called **value-based policies**, where the access to objects (or more generally, to entities) is stated in terms of properties of objects and subjects.

Now consider implementing the same policy by means of capabilities alone. We discuss two different approaches to doing so. In the first approach, we explicitly place read tickets for all document of the appropriate classification in the domain of every user. The immediate objection to doing so, is that the domain of a user then contains a large number of capabilities, many of which may never be used. Even if we accept this situation, there is an additional problem in maintaining the policy as new documents (or new users) are introduced. Whenever a new document is created, tickets for this document must be placed in the domain of every user with a higher classification. Doing so involves a change in a large number of domains.

[7]Of course, there has to be some means by which A can specify the particular document for which the read ticket is desired. However, this identification is not effected via a ticket.

This is contrary to a fundamental principle of systems design which, in the present context, is stated as follows.

> A "small" change in the system state should result in a "small" change in the protection state.

Or alternately, with a somewhat different emphasis, this principle is stated as follows.

> Any "frequently" occurring event must result in a "small" change in the protection state.

In short, it is simply unacceptable for a minor and frequent event such as creation of a document to result in a system-wide change in every user's domain. So much for the first approach.

The second approach, and the one which an experienced systems designer would suggest, relies on mediation by a distinguished subject designated as (say) the security officer. Every user is allowed to obtain read tickets from this security officer, and it is up to the officer to ensure that the stated policy is enforced. Indeed, this security officer need not be a human being and may well be a system process programmed to enforce this policy. But then, the fact that users can obtain read tickets from this subject amounts to users possessing the demand activator above; and the fact that the security officer is trusted (or programmed) to enforce the stated policy is reflected in the qualifier of this activator. So the second approach is really no different from the activator approach, and amounts to a particular implementation of the demand activator.

This illustration of the utility of activators is based on a fairly simple example. Minsky [12, 13, 14] discusses a variety of more sophisticated policies which are relatively easy to enforce with activators, but are cumbersome or impossible to enforce using capabilities alone.

1.1.2. The Take-Grant Mechanism

We now illustrate the specification and analysis of a protection mechanism in the context of the **take-grant mechanism**[8]. This mechanism was proposed by Jones,

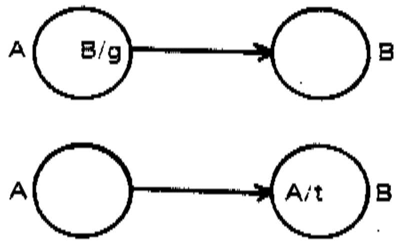
[8]There are actually two versions of the take-grant mechanism. We discuss the simpler version where all subjects are assumed to be active entities. In the enhanced version there is a provision for including subjects which are passive. This enhancement does not significantly alter the properties we discuss here.

Lipton, and Snyder [7, 10, 17] as a simplified model for the dynamics of capability based mechanisms.

The take-grant mechanism provides two rules for the transport of capabilities from one subject to another, as follows.

> **The Grant Rule**: If subject A possesses the capability B/g, then A is authorized to transfer a copy[9] of <u>any</u> capability from dom(A) to dom(B).
>
> **The Take Rule**: If subject B possesses the capability A/t, then B is authorized to transfer a copy of <u>any</u> capability from dom(A) to dom(B).

We represent these two situations by the diagrams shown below[10]. A subject is depicted by a labeled circle, with relevant capabilities in the domain of the subject shown inside the circle. In both cases, the direction of the edge indicates the direction in which transfer of capabilities is authorized.



We will use this graphical technique, for representing authorization for transport of capabilities, throughout this thesis.

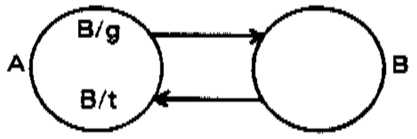
The take-grant mechanism also provides a **create rule** by which new subjects may be introduced in the system. Specifically, any subject A is authorized to create another subject B resulting in the following situation.

$$B/t \in \text{dom}(A)$$
$$B/g \in \text{dom}(A)$$

That is, immediately after the create operation there is authorization for transport of capabilities from A to B and vice versa. In terms of a diagram, the situation after subject A creates subject B is as follows.

[9] In this thesis we do not consider rules in which the transfer of a capability is effected by removing it from one domain before placing it in another.

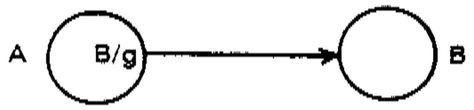
[10] Our presentation here is somewhat different from that of Jones, Lipton and Snyder [7, 10, 17], in order to be consistent with the rest of the thesis. The notation we use was suggested by Lockman and Minsky [11].



The take, grant, and create rules define the dynamics of the take-grant mechanism. There is no provision for revocation of capabilities. Thus once a subject acquires a capability it cannot be removed from the domain of the subject. In this sense the take-grant mechanism is **monotonic**. All the mechanisms we study in this thesis are monotonic[11].

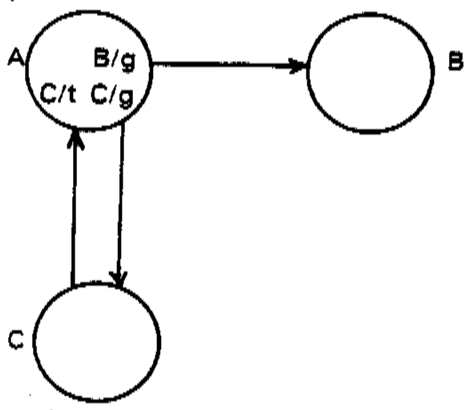
A major simplification in the take-grant mechanism is the lack of any selectivity in the transport operation. Either every capability can be transported from dom(A) to dom(B) or none can be transported. This is a serious limitation, since we might want to restrict the kinds of capabilities which can be transported from one subject to another. But granted this simplification, the take-grant rules seem quite natural and almost self-evident. Indeed, these rules do model mechanisms which have been proposed and implemented.

Now let us see how a system which uses the take-grant mechanism may evolve. Specifically, consider a situation where there is authorization for direct transport of capabilities from subject A to subject B. Under the take-grant rules this authorization can be in one of two forms. Consider first the case where the authorization is at the source of the transport operation, so that we have the following situation.

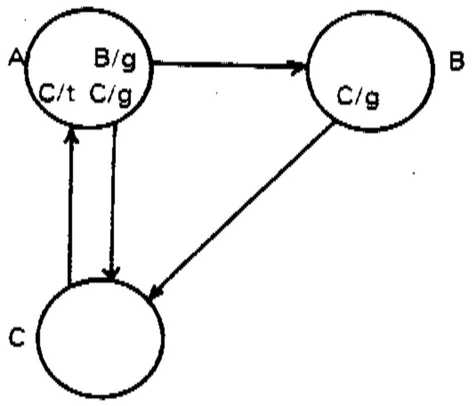


By the create rule, subject A is authorized to create a new subject C resulting in the situation shown below.

[11] In the presence of a facility for revocation of privileges, the monotonic assumption is interpreted as stating that no revocation actually takes place. This is consistent with a worst-case scenario, which we adopt throughout this thesis.

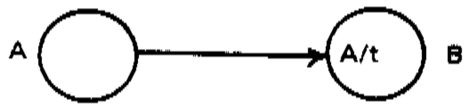


Then the C/g capability can be moved to the domain of B resulting in the following protection state.

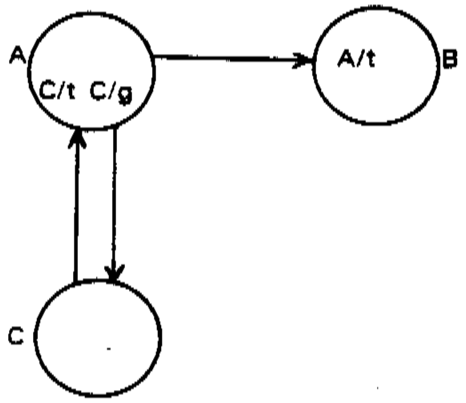


Now capabilities can be moved from dom(B) to dom(A) via subject C. Hence, the initial transport of capabilities from A to B has been reversed by establishing an indirect channel for transport of capabilities from B to A.

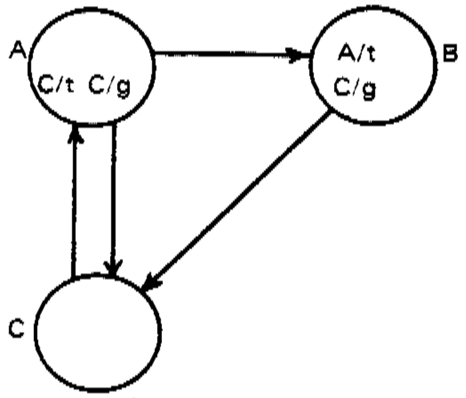
A similar phenomena arises if the initial authorization for direct transport of capabilities from A to B is at the destination of the transport operation, i.e.,



As before, subject A can create a new subject C resulting in the situation shown below.



But then, the C/g capability can be moved to the domain of B establishing the following state.



Once again, the initial transport of capabilities from A to B has been reversed by establishing an indirect channel for transport of capabilities from B to A.

Every initial state can then evolve to a state where the subjects are partitioned into disjoint classes, such that there is no transport of capabilities between subjects in different classes, while within each class all subjects can transport capabilities, either directly or indirectly, to each other[12]. Of course, such a state will actually be realized only if the subjects cooperate in doing so. However, from a worst-case viewpoint, such a state is feasible[13]. This property of **symmetric transport** sharply limits the (worst-case) policies enforceable by the take-grant mechanism.

[12]Except for some extreme circumstances, the situation is similar in the enhanced version of the take-grant mechanism which incorporates passive subjects.

[13]We adopt a worst-case viewpoint throughout this thesis, so that for us a protection mechanism provides security to the extent that it rules out the feasibility of realizing "undesirable" or "unsafe" protection states.

For the take-grant mechanism the question of whether an arbitrary protection state can be realized from a given initial state is efficiently decidable. The basic reason for this efficiency is the symmetric transport property, which unfortunately also limits the policies enforceable by this mechanism.

Lockman and Minsky [11] investigated the reason for this phenomena of symmetric transport in the take-grant mechanism. They considered the **grant-only** and **take-only** variations of this mechanism. Inspite of the apparent duality of these two cases, it turns out that the symmetric transport property applies to the grant-only mechanism but not to the take-only mechanism[14].

Perhaps, the major lesson to be drawn from these results is that there is a definite need for careful analysis in determining the extent to which a protection mechanism is capable of enforcing security policies. Of course, the take-grant mechanism is simplistic in its major assumption of non-selective transport. Nonetheless, it does model mechanisms which have been proposed and implemented. The symmetric transport property is thereby more disturbing, since it exposes a basic flaw in these mechanisms.

1.1.3. Selectivity and Locality

Now that we have illustrated the specification and analysis of a protection mechanism, let us return to the question of what are the desirable features of a protection mechanism. The fundamental objective is to provide generality and analyzability.

The take-grant mechanism allows for efficient analysis. However, it fails to meet the objective of generality in any way. This mechanism is capable of enforcing exactly one policy where the subjects are classified into equivalence classes; such that every pair of subjects in the same class can transport any capability to each other, while it is not possible to transport capabilities between subjects in different classes.

At the other extreme, we have the work of Harrison, Russo, and Ullman [6].

[14]The take-only mechanism has a rather different property, which we will explain in section 1.2.4.

They defined a grammar for specifying the dynamics of a capability based protection mechanism[15]. This grammar is capable of specifying a large number of policies encountered in practice. Unfortunately, as they demonstrated, the generality of their approach makes analysis issues undecidable or intractable. Thus, we must be careful to balance these two conflicting goals of generality and analyzability.

Now, the objectives of generality and analyzability are not sufficiently well-defined to provide constructive guidelines for the design of a transport mechanism. Following Minsky [15], we adopt two design principles to provide such guidelines.

First consider the issue of generality. The subjects and objects in a given system have various attributes associated with them, as well as various relations among them. These attributes and relations correspond to attributes and relations which exist in the external world. Policies for distribution and acquisition of privileges are often formulated with reference to such attributes and relations. We discussed one such example in section 1.1.1, as an instance of the general class of value-based policies. In that example, the policy for accessing an object was stated in terms of the relative values of a subject's and object's classification.

As another example consider the following policy. Objects O_1 through O_n are documents internal to the compiler design group. Subjects A_1 through A_m are members of this group, and subject A_0 is the boss. Capabilities for an object internal to the group may be released to subjects outside the group, only if A_0 approves. The policy is an instance of the general class of **cooperative policies**. Here, subjects outside the design group can obtain tickets for internal documents only if the boss approves.

Mechanisms such as the take-grant mechanism cannot support these policies, due to lack of any selectivity in the transport operation. This leads us to the following principle.

> **Principle of Selectivity**: The rules which determine how a protection state can be changed should be selective with respect to the kinds of privileges to which they apply.

The exact manner in which we propose to allow for such selectivity will be discussed shortly.

---

[15] Their formulation is in terms of the access control matrix [5, 9]. However, it can be equivalently viewed as a formulation within the capability approach.

Another serious flaw in the take-grant mechanism is that the control it provides over the transport of privileges is inherently non-local, in the sense that the movement of privileges into a given domain or out of it can be authorized solely by privileges residing outside the domain. For example, for capabilities to flow into the domain of A it suffices for some subject B to have A/g in its domain. This is inconsistent with the concept of modularization, which requires that the designer have some control over privileges which can flow into or out of a particular domain. Moreover, this control should be independent of the rest of the system in which the domain is embedded. This leads us to the following principle.

> **Principle of Locality**: The movement of privileges into a given domain and out of it should be subject to authorization by privileges already in this domain.

A major consequence of adopting this principle is that it is then possible to make assertions, about privileges which might flow into or out of a given domain, which are (largely) independent of a particular protection state. This ability to make assertions, which are based on a small number of domains embedded in a larger system, is called **local analyzability** (see Minsky [15]). In this thesis we are primarily interested in questions of **global analysis** where the assertions are based on the entire system. Somewhat fortituously, as demonstrated by Minsky [15], the principle of locality has significant global effects on the flow of privileges.

1.2. THE SEND-RECEIVE TRANSPORT MECHANISM

The send-receive transport mechanism[16] was proposed by Minsky [15] as one mechanism which embodies both principles of selectivity and locality. Recall that, in section 1.1.1, we identified two kinds of privileges called tickets and activators. In the send-receive transport mechanism we assume that the distribution of activators is static while the distribution of tickets is dynamic[17]. Thus the only privileges which are transportable are tickets.

In our discussion of the take-grant mechanism, we were able to suppress

---

[16]Our discussion of this mechanism here is somewhat different in its details from that of Minsky [15], so as to be consistent with the rest of the thesis.

[17]In the more general version of operation-control the distribution of activators is also dynamic. In this thesis we restrict ourselves to the simplified version.

consideration of tickets for objects. Indeed, we were only concerned with tickets with the take and grant rights, since the distribution of these tickets determines the transport properties of a protection state. For the same reason, in presenting the send-receive mechanism we will confine our attention to tickets which control the transport operation.

In the take-grant mechanism there are three ways by which a subject A can obtain a ticket.

1. A may be given the ticket in the initial state.

2. A may obtain the ticket by a transport operation from the domain of some other subject B.

3. A may obtain the ticket as a result of a create operation.

In addition to these three ways for obtaining a ticket, in the send-receive mechanism we allow for a ticket to be obtained by a demand operation[18].

Before getting into the details of the send-receive mechanism, we first define some basic concepts and conventions. We then discuss the transport, demand and create operations. As a special case of the send-receive mechanism, Minsky [15] defined and analyzed the uniform send-receive mechanism. We conclude this section by reviewing Minsky's analysis.

1.2.1. Tickets and the Copy Flag

A ticket consists of an address and a set of right symbols. The address uniquely identifies some entity in the system, and the right symbols authorize protected operations on the entity being addressed. For the sake of our formalism, we assume that every ticket carries exactly one right symbol. This is a matter of convenience for our exposition, and we do not suggest that tickets must indeed be implemented with only a single right. Under our restriction, a ticket with multiple right symbols is modeled as a set of tickets, each of which addresses the same entity and carries a single right. The underlying assumption here, is that two tickets A/x and A/y for the same entity A are equivalent to a single ticket A/xy which

[18]This operation was introduced in section 1.1.1, where we showed why this operation is useful.

carries both right symbols. This assumption can be made without any loss of generality[19].

We assume that every right symbol comes in two variations with and without the **copy flag**, denoted by the symbol c. The interpretation of the copy flag is that a ticket A/xc is potentially transportable[20], whereas the ticket A/x cannot be transported under any circumstances. At the same time, for all purposes other than the transport operation, the rights x and xc are identical. The copy flag allows us to incorporate tickets which are inherently non-transportable. Such a facility is frequently provided in a protection mechanism. It serves to provide a gross selectivity with respect to the transport operation[21]. We have the following convention.

```
   x - denotes a right symbol without the copy flag
  xc - denotes a right symbol with the copy flag
 x:c - denotes a right symbol with or without the copy flag
```

We assume that

$$A/xc \in \mathrm{dom}(B) \rightarrow A/x \in \mathrm{dom}(B)$$

Due to the role of the copy flag, this assumption does not entail any loss of generality. When the symbol x:c occurs at two places in a single context, it is intended that the interpretation at both places be the same. Thus the statement

$$[A/x{:}c \in \mathrm{dom}(B) \rightarrow A/x{:}c \in \mathrm{dom}(B) \cup \mathrm{dom}(C)]$$

has the connotation that

$$[A/x \in \mathrm{dom}(B) \rightarrow A/x \in \mathrm{dom}(B) \cup \mathrm{dom}(C)]$$
$$\wedge$$
$$[A/xc \in \mathrm{dom}(B) \rightarrow A/xc \in \mathrm{dom}(B) \cup \mathrm{dom}(C)]$$

This **colon convention** will often permit us to write a single statement instead of two separate statements. We refer to a ticket which carries the copy flag as being a **copiable** or **transportable** ticket.

---

[19]Indeed, if A/xy is not equivalent to A/x and A/y we can introduce another symbol, say z, to represent the combination xy, so that A/xy is equivalent to A/z.

[20]In the sense that a copy, A/xc or A/x, can be transported provided all other conditions for the transport operation are satisfied.

[21]The send-receive transport mechanism provides for further selectivity in a manner to be shortly discussed.

1.2.2. The Transport Operation

The principle of locality requires that every movement of a ticket from, say, dom(A) to dom(B) should be authorized by the current content of both these domains. By definition, the domain of a given subject has jurisdiction only over the activity of this particular subject. It follows that the movement of a ticket from A to B should involve two operations, viz.,

1. Some kind of **send** operation to be carried out by A, and which is authorized by privileges in the domain of A.

2. Some kind of **receive** operation to be carried out by B, and which is be authorized by privileges in the domain of B.

We call this the **send-receive protocol** [15]. There are actually two aspects to this protocol. Firstly, we have the authorization aspect, which is the requirement that every transport be authorized by the domains of both the source and the destination subject. Secondly, we have the volitional aspect that both the source and destination subjects actively participate in each movement. In this thesis we will insist only on the authorization aspect of the protocol. The reason is that we adopt a worst-case viewpoint in our analysis, so that everything which can happen will indeed happen. We can then accommodate a variety of volitional assumptions about the subjects involved in a transport operation[22]. In order to emphasize the irrelevance of volitional issues, we will treat a matching pair of send and receive operations as a single transport operation and, for the most part, will not be concerned with the question of whether the sender or the receiver initiates this operation.

Now let us consider the details of the send and receive operations. For the send operation we need to identify the ticket which is being sent and to identify the destination subject. For the latter purpose we define the **send right**, denoted by the symbol s. The send operation then has the following form.

---

[22]In particular, we can model the volitional aspect of the take-grant mechanism by assuming that every subject in the system is a willing sender and receiver. For instance, if A performs a receive operation to obtain a privilege from B, we assume that B will indeed perform the corresponding send operation (provided B is authorized to do so). Then the receive operation is volitionally equivalent to the take operation, although the two operations are quite different with respect to the authorization aspect. For both operations, A does not need the explicit consent of B to obtain a ticket from B. However, the transfer will not take place under the send-receive protocol unless B has appropriate authorization. There is a similar volitional equivalence between the send and grant operations.

send `C/x:c` **to** `B/s`

where C/x:c is a ticket to be sent and B/s is a **send ticket** for the destination subject B. The effect of this operation is that a copy of the ticket is sent to B, leaving the original ticket intact in the domain of the sender.

For a subject to perform the send operation, in addition to the tickets C/xc and B/s, he needs an activator which allows invocation of the send operation. Such an activator is called a **send-activator** and has the following form.

can-send `p:[`*c*`/x:c]` **to** `s:[`*b*`/s]` **if** `Q(p,s)`

Now consider a subject A who has this activator in his domain. By virtue of possessing this activator, A is authorized to perform a send operation

send `C/x:c` **to** `B/s`

provided the following conditions are satisfied.

1. The ticket C/xc is in the domain of A.
2. The entity C is of type[23] *c*.
3. The ticket B/s is in the domain of A.
4. The subject B is of type *b*.
5. The predicate Q evaluates to true when evaluated with respect to C and B.

The copy of the ticket C/xc which is actually transported, may or may not carry the copy flag, depending on whether the pattern *c*/x:c does or does not include the copy flag.

The counterpart of the send operation is the receive operation which has the form

receive `C/x:c` **from** `A/r`

where C/x:c is a ticket to be received, A/r is a **receive ticket** for the source subject A, and r is the **receive right**. The effect of this operation is to place the ticket C/x:c in the domain of the receiver.

The receive operation is authorized by a **receive-activator** with the following general form.

[23]In the context of objects, the notion of type corresponds to the well known notion of a data type. In the context of subjects, the type distinguishes various categories of subjects, e.g., systems programmers from ordinary users. We will discuss the notion of type in further detail in chapter 2.

can-receive p:[*c*/x:c] **from** s:[*a*/r] **if** Q(p,s)

By virtue of possessing this activator, subject B is authorized to perform a receive operation

receive C/x:c **from** A/r

provided the following conditions are satisfied.

1. The entity C is of type *c*.
2. The ticket A/r is in the domain of B.
3. The subject A is of type *a*.
4. The predicate Q evaluates to true when evaluated with respect to C and A.

Here again, the copy of the ticket C/xc which is actually transported, may or may not carry the copy flag, depending on whether the pattern *c*/x:c does or does not include the copy flag. As an example, consider the following situation.

1. A and B are subjects of type *sub*.
2. The ticket B/s is in the domain of A.
3. The ticket A/r is in the domain of B.
4. $D_1,...,D_n$ are objects of type *doc*. The right symbol u authorizes an update operation on these objects.
5. A possesses the ticket D_1/uc.
6. A possesses the single send-activator shown below.

   can-send d:[*doc*/u] **to** s:[*sub*/s] **if** class(d) < class(s)

7. B possesses the single receive-activator shown below.

   can-receive d:[*doc*/u] **from** s:[*sub*/r] **if** dept(s) = 'Finance'

Then, by virtue of possessing the send-activator above, whose patterns match the tickets D_1/u and B/s, A can invoke the send operation

send D_1/u **to** B/s

This operation will be successful only if class(D_1) is less than class(B). Similarly, by virtue of possessing the receive-activator above, whose patterns match the tickets D_1/u and A/r, B can invoke the corresponding receive operation

receive D_1/u **from** A/r

This operation will be successful only if dept(A) is 'Finance'. If both these operations are successfully executed, then the ticket D_1/u is placed in the domain of B.

By inspection of the two activators above, the following facts can be easily deduced, where D_i denotes an object of type *doc*.

1. It is not possible to transfer a ticket D_i/uc from dom(A) to dom(B).

2. It is possible to transfer a ticket D_1/u from dom(A) to dom(B) only if dept(A) is 'Finance'. Moreover, this is possible only if class(D_1) is less than class(B).

Observe that there are many alternative pairs of send and receive activators which will implement the same policy for transport of update tickets, for objects of type *doc*, from A to B. Examples of such alternatives are shown below.

1. A possesses the single activator

   can-send d:[*doc*/uc] **to** s:[*sub*/s] **if** class(d) < class(s)

   and B possesses the single activator

   can-receive d:[*doc*/u] **from** s:[*sub*/r] **if** dept(s) = 'Finance'

   Here A can send copiable tickets to B. However, B is still limited to receiving non-copiable tickets.

2. A possesses the single activator

   can-send d:[*doc*/uc] **to** s:[*sub*/s] **if** true

   and B possesses the single activator

   ```
   can-receive d:[doc/u] from s:[sub/r] if dept(s) = 'Finance' ∧
                                            class(d) < class(B)
   ```

   Here, the restriction on the class has been transferred to the receive-activator of B.

3. A possesses the single activator

   can-send d:[*doc*/u] **to** s:[*sub*/s] **if** class(d) < class(s)

   and B possesses the single activator

   can-receive d:[*doc*/uc] **from** s:[*sub*/r] **if** dept(s) = 'Finance'

   Here B can receive copiable tickets from A. However, A cannot send such tickets.

Of course, these alternative formulations of the same policy for transport of update tickets for objects of type *doc* from A to B, have different implications with respect to the policy for transport of such tickets from A to other subjects, as well as to B from other subjects.

Let us now return to considering the general form of the send and receive activators. The fact that we treat the transport operation as consisting of a matching pair of send and receive operations, is a direct consequence of the principle of locality. We provide for selectivity in this transport mechanism in two ways.

1. We distinguish transportable and non-transportable tickets by means of the copy flag.

2. We distinguish the kinds of tickets which can be transported in terms of the patterns and qualifiers of the send and receive activators.

At the same time, we introduce only a single send right s, and a single receive right r (along with the corresponding copiable versions sc and rc). Since these rights control the transport operation, we call them the **transport rights**. Correspondingly, tickets which carry one of the s, r, sc or rc rights are called **transport tickets**. All other right symbols are called **inert rights** and are treated as uninterpreted symbols. Tickets which carry such rights are called **inert tickets**.

Now, within the send-receive protocol, it is possible to introduce any number of send rights $s_1,...,s_n$, and any number of receive rights $r_1,...,r_m$. This facility would allow us to specify selectivity along another dimension as, for instance, if A possesses the following activators.

can-send d:[*doc*/u] **to** s:[*sub*/s_1] **if** class(d) = 1
can-send d:[*doc*/u] **to** s:[*sub*/s_2] **if** class(d) = 2

Here a destination subject B can be addressed by either of two tickets, B/s_1 or B/s_2. The corresponding activators differ in the kinds of tickets which can be sent. In the former case the class has to be 1, whereas in the latter case the class must be 2. The decision not to provide such selectivity, in terms of different send and receive rights, is quite deliberate. The motivation is to keep the concept of a ticket as simple as possible. Every additional right symbol we introduce places an additional burden on the addressing mechanism. Since the send and receive activators are capable of specifying fine distinctions with respect to the tickets which match their patterns, there is no pressing need to introduce additional right symbols for this purpose.

1.2.3. The Demand and Create Operations

The demand operation allows a subject to obtain a ticket simply by demanding it. It is authorized by a demand-activator which has the following form.

can-demand p:[*t*/x:c] **if** Q(p)

A subject who possesses this activator can obtain a ticket T/x:c for an entity of type *t*, provided the predicate Q(T) is true. In section 1.1.1, we illustrated the utility of this operation for obtaining tickets for objects. This operation is also useful in obtaining tickets for subjects. For example, consider the following policy.

> Every pair of subjects, A and B, in the same department must be able to transport tickets to each other (as determined by the send and receive activators of A and B).

This policy is easily implemented by providing every subject A with the following demand-activators, where we assume that all subjects are instances of the single type *sub*.

can-demand p:[*sub*/s] **if** dept(p) = dept(A)
can-demand p:[*sub*/r] **if** dept(p) = dept(A)

Every pair of subjects in the same department can then obtain send and receive tickets, for each other, simply by demanding them. Here again, this policy is an instance of the general class of value-based policies.

Next consider the create operation, by means of which new objects and subjects are introduced in a system. There are two issues which must be addressed in specifying this operation. The first issue is the authorization required for a create operation. In the send-receive mechanism this authorization is stated by a **create-activator** which has the following general form.

can-create p:[*t*] **if** Q(p)

Possession of this activator authorizes authorizes creation of entities of type *t* which satisfy the predicate Q. For example, let subject A possesses the following activator.

can-create p:[*sub*] **if** dept(p) = dept(A)

Then A is authorized to create a subject A' which is in the same department as that of A. Such activators provide selectivity with respect to the create operation.

The second issue is the precise semantics of the create operation. For instance, in the above example we need to specify exactly what activators and tickets the created subject A' is given immediately after it has been created. For the moment, we do not define precisely how this specification is to be made. However, we recognize the need to restrict the impact of a create operation in the following way.

> **Principle of Local Creates**: The immediate result of a create operation where subject A creates a subject A' should only involve the domains of A and A'. Similarly, the immediate result of a create operation where subject A creates an object O should only involve the domain of A.

This principle is a special case of the more general principle, introduced on page 6, viz., that a "small" or "frequently" occurring event should result in a "small" change in the protection state. Creation of subjects and objects is precisely such an event.

1.2.4. The Uniform Send-Receive Mechanism

As a special case of the send-receive transport mechanism Minsky [15] defined the **uniform send-receive mechanism**. In this mechanism there is no selectivity in the transport of tickets which carry transport rights. In this respect it is similar to the take-grant mechanism. The uniform send-receive mechanism is defined by imposing the following restrictions on the send-receive transport mechanism.

1. All tickets carry the copy flag.

2. All subjects are instances of a single subject type *sub*.

3. Every subject possesses the following activators.

   can-send [*sub*/sc] **to** [*sub*/s]
   can-send [*sub*/rc] **to** [*sub*/s]
   can-receive [*sub*/sc] **from** [*sub*/r]
   can-receive [*sub*/rc] **from** [*sub*/r]

   Note that there are no qualifiers on these activators.

4. Every subject is authorized to create new subjects.

5. The result of a create operation where subject A creates subject B is that the following tickets are placed in the indicated domains.

   $B/sc,\ B/rc \in \mathrm{dom}(A)$
   $A/sc,\ A/rc \in \mathrm{dom}(B)$

   Thus immediately after the create operation there is an authorization for transport of tickets from A to B and vice versa.

The transport of transport tickets is then governed by the following rule.

The Send-Receive Rule: If subject A possesses the ticket B/s and subject B possesses the ticket A/r then a copy of <u>any</u> transport ticket can be transported from dom(A) to dom(B).

The name uniform distinguishes this mechanism from the general case where the transport of transport tickets is selective.

The uniform send-receive mechanism is of interest for several reasons. Firstly, it is a simple version of the send-receive transport mechanism and is thus analyzable. Secondly, it has the same flavor as the take-grant mechanism and allows for a comparative analysis, thereby providing insight into the consequences of adopting the principle of locality. Finally, it serves as a theoretical tool for an approximate analysis of the send-receive mechanism. That is, assume the send and receive activators are less permissive than in the uniform case. Then, for a given initial distribution of tickets, if a particular configuration of the protection state cannot be realized in the uniform case it certainly cannot be realized in the less permissive case.

Before reviewing the analysis of the uniform mechanism, we introduce some notation which will be used throughout the thesis. First consider the domain of a given subject. In the operation-control mechanism, there are two kinds of privileges in a subject's domain, viz., tickets and activators. Since there is no provision for transport of activators in the send-receive mechanism, this aspect of a subject's domain is determined when the subject is created and cannot change thereafter. However, the set of tickets in a subject's domain does change as additional tickets are acquired by the subject. We identify the latter set as follows.

Definition 1.1: Given a particular protection state k, $dom^k(A)$ denotes the set of tickets in the domain of subject A in state k. ■

This idea of fixing the context by a superscript will be used for all state dependent sets, functions and relations. If the context is not relevant we simply omit the superscript. We denote the initial state by the superscript 0 so that, for example, $dom^0(A)$ is the set of tickets possessed by subject A in the initial state. Next we define the following relations.

Definition 1.2: For every protection state k, define the associated binary relations

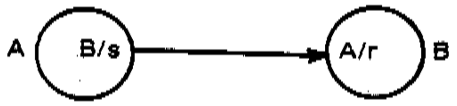
$$\mathtt{slink}^k,\ \mathtt{rlink}^k,\ \mathtt{link}^k \subseteq \mathrm{SUB}^k \times \mathrm{SUB}^k$$

as follows

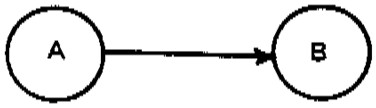
$$\begin{aligned} \mathtt{slink}^k(A,B) &\leftrightarrow B/s \in \mathrm{dom}^k(A) \\ \mathtt{rlink}^k(A,B) &\leftrightarrow A/r \in \mathrm{dom}^k(B) \\ \mathtt{link}^k(A,B) &\leftrightarrow \mathtt{slink}^k(A,B) \wedge \mathtt{rlink}^k(A,B) \end{aligned}$$

■

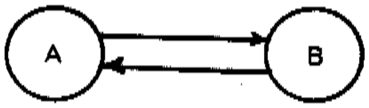
We depict the existence of a link from A to B as follows.



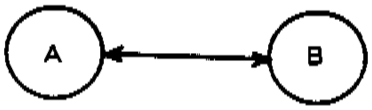
In general, we show a subject A as a circle labeled by the letter A with relevant tickets in the domain of A listed inside the circle. We frequently omit explicit mention of the tickets which establish a link and, on such occasions, we place the label inside the circle, as for example



In such cases, the tickets required to establish the depicted links are implicitly asserted to be present in appropriate domains. If there is a link in both directions we show it as



or as



The existence of a link from A to B indicates the possibility of direct transport of tickets from A to B. The actual tickets which can be transported are determined by dom(A), the send activators of A and the receive activators of B. In the special case of the uniform send-receive mechanism, existence of a link from A to B suffices to authorize transport of transport tickets from dom(A) to dom(B).

The following relation expresses the possibility of an indirect transport of tickets from one subject to another.

Definition 1.3: For every protection state k, define the associated binary relation

$$\text{path}^k \subseteq \text{SUB}^k \times \text{SUB}^k$$

by $\text{path}^k(A,B)$ if and only if

1. $\text{link}^k(A,B)$

or 2. There exists a sequence of subjects $C_1,\ldots,C_n$ such that

(a) $\text{link}^k(A,C_1)$

(b) $\text{link}^k(C_i,C_{i+1})$ $i=1,\ldots,n-1$

(c) $\text{link}^k(C_n,B)$

■

Here again, for the uniform send-receive mechanism, existence of a path from A to B suffices to authorize an indirect transport of transport tickets from dom(A) to dom(B). We can similarly define the spath^k and rpath^k relations.

For the analysis of the uniform send-receive mechanism the following relation is of particular importance.

Definition 1.4: For a given initial state define the binary relation

$$\text{path}^* \subseteq \text{SUB}^0 \times \text{SUB}^0$$

by $\text{path}^*(A,B)$ if and only if there is a protection state h derived from the initial state, by a sequence of transport, demand and create operations, such that $\text{path}^h(A,B)$. ■

Thus, $\text{path}^*(A,B)$ expresses the feasibility of there ever being a channel for transport of tickets from A to B. For the special case of the uniform mechanism, $\text{path}^*(A,B)$ if and only if transport tickets can indeed be transported from dom(A) to dom(B).

Let us now consider computation of the path^* relation from the initial state. Since the protection state can evolve in a unbounded manner by successive creation of new subjects, it is not immediately apparent how to compute this relation. Indeed, it is not clear whether this relation is computable. For the uniform send-receive mechanism, Minsky proved that the create operation can be ignored in computing the path^* relation, as indicated below.

Definition 1.5: For a given initial state define the binary relation

$$\text{path}'' \subsetneq \text{SUB}^{\circ} \times \text{SUB}^{\circ}$$

by $\text{path}''(A,B)$ if and only if there is a protection state h derived from the initial state without the use of any create operations such that $\text{path}^{h}(A,B)$. ■

Theorem 1.1: For the uniform send-receive mechanism

$$(\forall \langle A,B\rangle \in \text{SUB}^{\circ} \times \text{SUB}^{\circ})\,[\text{path}^{*}(A,B) \leftrightarrow \text{path}''(A,B)]$$

Proof: Minsky [15]. ■

In this sense the uniform mechanism is **create-invariant**. The create-invariant property insures that the computation of the path^{*} relation reduces to a finite problem. We will study a similar property in chapter 4 in the context of a non-uniform mechanism. A critical step in Minsky's proof is the following notion.

Definition 1.6: An initial protection state is said to be a **self-reference state** provided

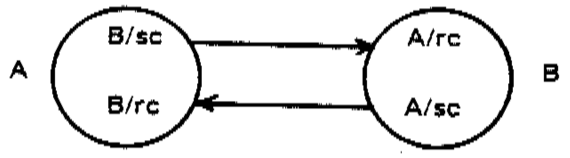
$$(\forall A \in \text{SUB}^{\circ})\,[A/s \in \text{dom}^{\circ}(A) \wedge A/r \in \text{dom}^{\circ}(A)]$$

■

Minsky established the following lemma.

Lemma 1.2: For the uniform send-receive mechanism it can be assumed, without any loss of generality, that the initial state is a self-reference state.

Proof: By the create rule any subject A can create a subject B resulting in the following situation.



But then the tickets A/sc and A/rc can be transported from dom(B) to dom(A). Since the self-reference tickets can always be acquired by any subject in this manner, there is no loss of generality in assuming that these tickets are present in the initial domain of every subject. ■

The notion of a self-reference state is then used to show that the path^{*} relation can be computed without considering the create operation.

Minsky additionally established the following result.

Theorem 1.3: For the uniform send-receive mechanism

$$(\forall \langle A,B\rangle \in \text{SUB}^{\circ} \times \text{SUB}^{\circ})\,[\text{path}^{*}(A,B) \rightarrow \text{rpath}^{\circ}(A,B)]$$

■

This result is particularly useful in the context of an initial state which satisfies the following constraint.

Definition 1.7: An initial state is said to **have no dangling rlinks** provided

$$(\forall\langle A,B\rangle \in SUB^{O} \times SUB^{O})\,[A/r \in dom^{O}(B) \rightarrow B/s \in dom^{O}(A)]$$

■

The corollary below follows immediately.

Corollary 1.3.1: For the uniform send-receive mechanism it is the case that for every initial state with no dangling rlinks

$$(\forall\langle A,B\rangle \in SUB^{O} \times SUB^{O})\,[path^{*}(A,B) \leftrightarrow path^{O}(A,B)]$$

■

We call this the **flow-invariant** property. This property ensures that the path relation set up in the initial state remains invariant in all subsequent states. It is then a trivial matter to compute the path* relation. Note that, even though the path relation cannot change, it is still possible to establish links which did not exist in the initial state. We will study a similar property in chapter 5 in the context of a non-uniform mechanism. As specific examples of an initial state with no dangling rlinks we have the following situations.

1. Every subject in the initial state is given send tickets for every subject, i.e.,

$$(\forall\langle A,B\rangle \in SUB^{O} \times SUB^{O})\,[B/s \in dom^{O}(A)]$$

   This distribution models the take-only mechanism of Lockman and Minsky [11]. It follows from corollary 1.3.1 that the take-only mechanism is flow-invariant. This contrasts with the symmetric transport property exhibited by the take-grant and the grant-only mechanisms (see page 10).

2. The initial state is **balanced** in the following sense

$$(\forall\langle A,B\rangle \in SUB^{O} \times SUB^{O})\,[B/s \in dom^{O}(A) \leftrightarrow A/r \in dom^{O}(B)]$$

   For a balanced state there are no stray tickets so that the initial state is designed in terms of links rather than individual tickets.

1.3. OBJECTIVES AND OUTLINE OF THE THESIS

Our objective in this thesis is to continue investigation of the send-receive transport mechanism. We will study a selective version of this mechanism. This contrasts with earlier analysis of protection mechanisms, viz., the take-grant mechanism of Jones, Lipton, and Snyder [7, 10, 17], its variations due to Lockman and Minsky [11], and the uniform send-receive mechanism of Minsky [15]. In these mechanisms the transport operation is non-selective, at least, with respect to transport of transport tickets.

The general version of the send-receive transport mechanism is too complex to attack straightaway. Hence, we make some simplifying assumptions. These assumptions are stated in chapter 2. Under these assumptions we find it useful to depart from the activator formulation for expressing selectivity in the transport, demand and create operations. Instead we develop an abstract framework for specifying this selectivity. While we do not address implementation issues in this thesis we do believe that this framework can be efficiently implemented.

In chapter 3 we identify a basic property of a protection state called the flow function. We then formulate the **flow-analysis problem** which is concerned with determining bounds on the value of this function in every protection state. In a sense, this problem is a generalization of the safety problem. There are several approaches to the flow-analysis problem. We investigate these approaches in chapters 3, 4, and 5. In chapter 6 we investigate some analysis issues other than the flow-analysis problem. In chapter 7 we summarize our results and present further questions which must be pursued to better understand the protection mechanism studied in this thesis.

We caution the reader that we will be introducing a fair amount of notation. There is no escape from notation in a formal treatment, however, we are especially handicapped due to the recent emergence of the problems discussed in this thesis. For lack of historical precedent, most of this notation is of our own creation. For convenient reference, in appendix A we provide a list of the major terms and symbols defined through the course of this thesis along with the page number(s) where they are defined. Certain aspects of our notation have been specifically designed to clarify the role of various symbols. The reader will benefit by paying attention to these conventions, which are summarized in appendix B. Finally, we have isolated the presentation of algorithms and analysis of their cost in appendix C.

CHAPTER 2

SELECTIVE SEND-RECEIVE PROTECTION SCHEMES

In this chapter we define a restricted version of the send-receive transport mechanism. Stated informally, our restrictions amount to the requirement that properties of a subject, which occur in the qualifiers of send, receive, demand and create activators, are determined when the subject gets created and remain constant thereafter. These properties strongly influence the operations which a subject can successfully invoke. We find it useful to consider such properties as part of the domain of the subject. This is a reasonable extension of the notion of a domain, since these properties do confer privileges on a subject. Then, under our restriction that these properties be static, the domain of a subject comprises two components, as follows.

1. A static component consisting of the properties mentioned above and the activators possessed by the subject.

2. A dynamic component consisting of the set of tickets possessed by the subject.

We say that two subjects are of the same type if the static components of their domains are identical. We motivate this definition of a subject type in section 2.1. Once this connection between the activator formulation and types is introduced, it is convenient to conceive the design of a protected system directly in terms of types. The design of a protected system then involves two steps, as follows.

1. Specifying the set of types, as well as specifying, for each subject type, the static component of the domain of every instance of this subject type.

2. Specifying the initial set of subjects and objects, and the initial distribution of tickets for these subjects and objects.

In order to emphasize this fact, we call the result of the first step as a **selective send-receive protection scheme**, or simply a **protection scheme**.

The specification of a protection scheme implies that the designer must think in terms of types. For this reason, in section 2.2, we depart from the activator formulation and define **design parameters**, for specifying selectivity in the transport, demand and create operations, directly in terms of types.

We conclude this chapter by considering a number of miscellaneous issues. In section 2.3 we define the notions of a state transition and a transition sequence, in order to talk about the evolution of a system from its given initial state. In section 2.4 we show how entities which possess privileges but are passive, in the sense that they cannot autonomously invoke operations, can be accommodated in our design framework in a straightforward manner. Finally, in section 2.5 we outline our conventions for naming the sets, functions, and relations introduced throughout the thesis. The reader will benefit greatly by paying attention to these conventions.

2.1. TYPES OF OBJECTS, SUBJECTS AND TICKETS

The concept of type is familiar in the context of data structures. The type of a given data structure determines the set of operations which are meaningful. Thus push and pop are meaningful operations on a stack but not on a queue. The concept of an object type corresponds to this familiar notion.

A subject is an active entity in the system. There are operations which can be performed on a subject. For example, a process might be put to sleep or awakened. However, the more significant aspect of a subject is that a subject can autonomously invoke operations. For subjects we extend the notion of type to model both aspects, as follows.

1. The operations which are meaningful on a given subject are determined by the type of the subject.

2. The operations which a particular subject may invoke are also determined by the type of the subject.

Thus, for instance, a subject of type "ordinary user" cannot invoke certain sensitive operations which a subject of type "system manager" is allowed to invoke.

Let us see how this notion of a subject type relates to our earlier discussion of the send-receive transport mechanism. Now, the occurrence of types in the patterns of an activator is a matter of convenience[1] which can be easily eliminated by treating the type of an entity as another attribute, similar to attributes such as class and dept. For instance, the activator

can-send p:[*c*/x:c] **to** s:[*b*/s] **if** Q(p,s)

can be equivalently replaced by the activator

can-send p:[*c*/x:c] **to** s:[*sub*/s] **if** Q(p,s) ∧ type(s) = *b*

In this manner, we can eliminate any prior notion of subject types, so that there is no loss of generality in assuming that all subjects are instances of a single type *sub*. We will now reintroduce the notion of a subject type, so as to model the two aspects mentioned above. In order to do so, we define the following notion.

> **Definition 2.1:** Given any finite set of activators, the properties of a subject A which occur in the qualifiers of these activators are called the **facade** of the subject A. ∎

For instance, the facade of a subject A might be as follows.

facade(A) = {class(A) = 3, dept(A) = 'Marketing'}

Once the facade of a subject is known, it can be determined exactly which activators, and thereby which operations, can be successfully invoked with reference to this subject. Our first criterion for determining a subject type is that all subjects which are instances of the same type should have an identical facade. By this criterion, exactly the same set of operations can be invoked with respect to subjects of the same type.

To model the second aspect, we need to consider the operations which a given subject might invoke. By definition, the exact set of operations which a given subject is authorized to perform are determined by the activators and tickets in the domain of this subject. However, the activators themselves specify the general pattern of these operations. Our second criterion for determining a subject type is that all subjects of the same type should possess exactly the same set of activators. We then have the following notion of subject types.

> **Definition 2.2:** Two subjects are said to be instances of the same **subject type** if and only if the facade of both subjects is identical and both subjects possess the same set of activators. ∎

---

[1]Of course, it is also a recognition of the importance of this concept in modern methodologies for design of systems.

Now that we have indicated how the type of a subject is determined from its facade and the activators possessed by this subject, we will assume that a designer approaches the design of a system by thinking of subject types in this manner. The most basic design parameter is then the set of types. We require that this set be specified as the first step in designing a system, as follows.

> **Definition 2.3:** The designer of a system specifies a finite set of **subject types**, denoted by the symbol T_S, and a finite set of **object types**, denoted by the symbol T_O. These two sets must be mutually exclusive, so that $T_S \cap T_O = \phi$. We denote the union of these two sets by the boldface symbol **T**, i.e., $\mathbf{T} = T_S \cup T_O$. ■

We will frequently need to refer to the type of an entity and introduce the *t* function in order to do so conveniently. This function simply returns the type of its argument subject or object, i.e.,

$$t\colon \text{SUB} \cup \text{OBJ} \longrightarrow \mathbf{T}$$

Here we are assuming that every entity is an instance of exactly one type. There is no loss of generality in this assumption. For instance, consider a situation where entities of type *a* are also instances of type *b* but not vice versa. That is, *a* is a sub-type of the type *b*. This situation can be modeled under our assumption by ensuring that, for example, if the following activator

can-send p:[*b*/x:c] **to** s:[*sub*/s] **if** Q(p,s)

is in the domain of a subject A, then the activator

can-send p:[*a*/x:c] **to** s:[*sub*/s] **if** Q(p,s)

is also in the domain of A.

Due to the distinction between subject types and object types, we can further state that

$$t\colon \text{SUB} \longrightarrow T_S$$
$$t\colon \text{OBJ} \longrightarrow T_O$$

We assume the *t* function is computationally efficient, in that the type of a given subject or object can be quickly determined from its name. Since all addressing is done via tickets, this requires that the type be encoded into every ticket in some way. This assumption would be required anyway to facilitate type checking at run time.

We denote subject types by lower case italicized letters from the beginning of the alphabet. Thus the letters *a*, *b*, *c*, *d* denote specific subject types. We denote subjects by upper case letters from the beginning of the alphabet. Thus the letters A, B, C, D denote specific subjects. For the most part, we will ensure that the type of a subject is the corresponding lower case italic letter; so that *t*(A) = *a* and *t*(B) = *b*. We rarely introduce specific object types and specific objects in our discussion. When we do so, we adopt a similar naming convention.

Since the authorization for the transport operation is controlled by the send and receive rights, we call these rights the **transport rights**. In combination with the copy flag we then have four transport rights denoted as follows.

$$\mathbf{R}_T = \{s,\ r,\ sc,\ rc\}$$

All other rights are called inert rights. For our purpose, inert rights are treated as uninterpreted symbols, specified as follows.

> **Definition 2.4:** The designer of a system specifies a set $\mathbf{R}_I$ of **inert rights**. The set of rights **R** is then $\mathbf{R} = \mathbf{R}_I \cup \mathbf{R}_T$. ■

We extend the notion of type to apply to tickets as follows.

> **Definition 2.5:** Define the set of **ticket types** T X **R**, by saying that the ticket A/x:c is of type *t*(A)/x:c. ■

That is, the type of a ticket is an ordered pair whose first element is the type of the entity being addressed and whose second element is the right symbol carried by the ticket[2]. Modulo our assumption about the efficient computation of the *t* function, the type of a given ticket can be efficiently determined. The set $\mathbf{T}_S$ X $\mathbf{R}_T$ of **transport ticket types** is of particular significance.

Observe that, by our definition, the tickets A/x and A/xc are of different types; the former ticket being of type *t*(A)/x while the latter is of type *t*(A)/xc. This is a useful distinction, since the latter ticket can potentially be transported while the former ticket can never be transported.

In general, most of the rights will be type specific in the sense that the operation authorized by a particular right symbol x will be meaningful only if it is performed on an entity of a given type. For example, the authorization to perform a push is meaningful only if the object on which the push is performed is of type stack. Due to the type specific interpretation of rights, certain ticket types will correspond

---

[2]Recall that on page 14 we limited a ticket to carrying exactly one right symbol, and indicated how this assumption does not entail any loss of generality

to tickets which do not arise in practice. Hence, the size of the set T X R will generally be larger than the set of ticket types actually encountered. This fact should be kept in mind while interpreting the complexity of our algorithms.

2.2. A PROTECTION SCHEME AND ITS PARAMETERS

We are now ready to state the most important restrictions we impose on the version of the send-receive transport mechanism investigated in this thesis. These restrictions are as follows.

1. Every subject is an instance of exactly one type. The type is specified when the subject is created and thereafter cannot change. That is, the typing of subjects is static.

2. The set of types, $T = T_S \cup T_O$, cannot be changed at "run time". Thus types can only be introduced at "compile time" (i.e. when a system is defined).

3. Similarly, the set of inert rights R_I cannot be changed at "run time". Thus, inert rights can only be introduced at "compile time". It immediately follows that the set of ticket types, T X R, also cannot be changed at "run time".

As indicated in section 2.1, the type of a subject is determined by its facade and by the set of activators that the subject possesses. In the send-receive transport mechanism, the latter set is assumed to be static. Our assumption of static typing of subjects then amounts to additionally assuming that the facade of every subject is static.

Now consider the nature of the protection state under these assumptions. As defined earlier, the domain of a subject consists of the tickets and activators possessed by the subject. Let us additionally consider the facade of a subject to also be part of its domain. Since the facade of a subject influences the set of operations which a given subject can invoke, this is a reasonable extension to the notion of a domain. Then, the domain of a subject consists of two components, as follows.

1. A static component consisting of the facade of the subject and the activators possessed by the subject. By definition, this component is determined by the type of the subject.

2. A dynamic component consisting of the set of tickets possessed by the subject.

Correspondingly, the design of a system consists of two steps as follows.

1. Specifying the set of types and inert rights, as well as specifying, for all subject types, the facade and activators in the domain of every instance of this subject type. This specification determines the static component of every subject's domain.

2. Specifying the initial set of subjects and objects, and the initial distribution of tickets for these subjects and objects.

Since the selectivity in the transport, demand and create operations is determined by the facade and activators of the subjects, many policy decisions are made when the first step is completed. In order to emphasize this fact, we call the result of the first step as a **protection scheme** or simply a **scheme**.

The specification of a protection scheme implies that the designer must think in terms of types. For this reason, we depart from the activator formulation and define **design parameters**, for specifying selectivity in the transport, demand and create operations, directly in terms of types. The most basic design parameters are the sets T_S, T_O, and R_I. These sets serve as the foundation on which the remaining parameters are defined. The following four sub-sections, respectively, discuss the design parameters which control the transport, demand, and create operations, and constrain the initial state. The section concludes by summarizing our framework and providing examples of specific schemes.

There is an interesting analogy between our framework for specifying a protection scheme and the notion of a data model in the literature on data base systems, as follows.

- The design framework, to be shortly defined, corresponds to the notion of a data model. A data model determines what can and what must be specified while defining a particular schema. Similarly, our design framework determines what can and what must be specified while defining a particular scheme.

- Defining a protection scheme by specifying the parameters of the design framework corresponds to defining a particular schema within the scope of a given data model.

- Constructing an initial state for a given protection scheme corresponds to constructing a particular data base for a given schema.

Indeed, this analogy motivates our choice of the name protection scheme.

2.2.1. The Direct Flow Limit Function

We begin by reviewing the authorization required for a ticket to be transported from the domain of one subject to the domain of another. Specifically, consider subjects A and B and some ticket C/x:c. The most basic requirement for this ticket to be transported from A to B, is that A must possess the ticket C/xc. For example, if the domain of A contains the tickets C/xc, D/x and E/y, then it is not possible to transport the tickets D/x or E/y to the subject B. But it <u>might</u> be possible to transport the tickets C/x or C/xc to the subject B, provided that

1. There is a link from A to B.

2. A and B respectively possess a suitable send and a receive activator.

In our restricted version of the send-receive transport mechanism, the latter aspect is determined by the types of the subjects A and B, and by the type of the ticket C/x:c. We require that this aspect be specified in the following manner.

> **Definition 2.6:** The designer of a scheme specifies a function called the **direct flow limit function**
>
> $$dfl\colon \mathbf{T_S} \times \mathbf{T_S} \rightarrow \mathbf{power\text{-}set}(\mathbf{T} \times \mathbf{R})$$
>
> which limits the types of tickets which can be transported over a link in the manner discussed below. ■

The *dfl* function is best viewed as a filter which applies after the basic conditions for transport have been satisfied.

As before, let us indicate the tickets in the domain of a subject A by dom(A). We illustrate the role of the *dfl* function in the context of the following situation.

$$\mathrm{dom}(A) = \{C/xc,\ D/x,\ E/y,\ B/s\}$$
$$\mathrm{dom}(B) = \{A/r\}$$

There is a link from A to B and, since C/xc is in the domain of A, the basic conditions for transport of C/x:c to B are met. The transport of the ticket C/x:c from dom(A) to dom(B) is then permitted if and only if

$$t(C)/x{:}c \in dfl(t(A),t(B))$$

Observe, in particular, that the *dfl* function may allow the transport of the ticket C/x but not the ticket C/xc, as shown below.

$$t(C)/x \in dfl(t(A),t(B))$$
$$t(C)/xc \notin dfl(t(A),t(B))$$

In such a case, the ticket C/x can be moved from the domain of A to the domain of B but no further.

Lest there be any misunderstanding, we remind the reader at this point that the transport is effected by making a copy of the ticket possessed by subject A and A retains possession of the ticket C/xc.

Next we illustrate a specific example of a *dfl* function. We begin by defining the set of types as follows.

$$\mathbf{T}_S = \{a, b\}$$
$$\mathbf{T}_O = \{o\}$$

The set of transport rights is fixed as a basic component of our framework but we do need to specify the set of inert rights. Assume that there are actually some undetermined number of inert rights; but the only selectivity we wish to impose is in terms of two major categories called u (for update) and p (for preserve), in order to distinguish protected operations which may or may not alter the state of the object being addressed. Then the set of inert rights can be represented as follows.

$$\mathbf{R}_I = \{\mathrm{uc},\ \mathrm{u},\ \mathrm{pc},\ \mathrm{p}\}$$

We might define the *dfl* function to have the following values.

$$\begin{aligned} dfl(a,a) &= \{b/\mathrm{sc},\ b/\mathrm{s},\ o/\mathrm{pc},\ o/\mathrm{p},\ o/\mathrm{uc},\ o/\mathrm{c}\} \\ dfl(a,b) &= \{b/\mathrm{s},\ o/\mathrm{pc},\ o/\mathrm{p},\ o/\mathrm{u}\} \\ dfl(b,a) &= \{b/\mathrm{sc},\ b/\mathrm{s},\ o/\mathrm{pc},\ o/\mathrm{p},\ o/\mathrm{u}\} \\ dfl(b,b) &= \{o/\mathrm{p}\} \end{aligned}$$

Let us examine the value of *dfl*(*a*,*a*). This set describes the types of tickets which can be directly transported between two subjects of type *a*, provided the basic conditions are satisfied. Since *b*/sc is in this set, send tickets with the copy flag for subjects of type *b* can be directly transported between two subjects of type *a*. It is then reasonable to permit transport of *b*/s tickets also, and observe that *b*/s is in the set *dfl*(*a*,*a*). Indeed, by our earlier convention

$$\mathrm{B/sc} \in \mathrm{dom}(\mathrm{A}) \rightarrow \mathrm{B/s} \in \mathrm{dom}(\mathrm{A})$$

We can then assume, in general, that

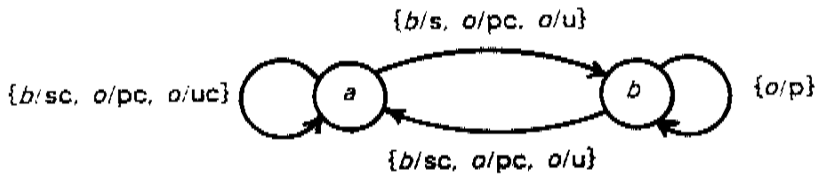
$$c/\mathrm{xc} \in dfl(a,b) \rightarrow c/\mathrm{x} \in dfl(a,b)$$

This helps reduce the verbosity in listing the values of a *dfl* function, since we can omit mention of *c*/x if *c*/xc appears in *dfl*(*a*,*b*). By this convention, the *dfl* function defined above can be rewritten as follows.

$$\begin{aligned} dfl(a,a) &= \{b/\text{sc},\ o/\text{pc},\ o/\text{uc}\} \\ dfl(a,b) &= \{b/\text{s},\ o/\text{pc},\ o/\text{u}\} \\ dfl(b,a) &= \{b/\text{sc},\ o/\text{pc},\ o/\text{u}\} \\ dfl(b,b) &= \{o/\text{p}\} \end{aligned}$$

The converse need not be true; observe that the ticket type *o*/u is in *dfl*(*a*,*b*) whereas *o*/uc is not in *dfl*(*a*,*b*). This allows for a subject of type *a* to transport an update ticket for an object to a subject of type *b* but the transported ticket cannot carry the copy flag.

A natural representation for a *dfl* function is a graph which we call a **dfl graph**. The nodes of this graph are subject types, and there is an edge from node *a* to node *b* if and only if $dfl(a,b) \neq \phi$. Every such edge is labeled by the value of *dfl*(*a*,*b*). The graph corresponding to our example *dfl* function is then as shown below.



Observe that a particular *dfl* function can be represented in terms of activators in a variety of ways. We leave it as an exercise for the reader to devise alternate representations of our example *dfl* function in terms of activators.

2.2.2. The Demand Function

The demand operation allows for acquisition of a ticket merely by requesting it. There are two aspects to this operation, viz.,

1. Obtaining tickets with inert rights.
2. Obtaining tickets with transport rights.

By definition, inert rights do not authorize operations which change the protection state. We do not explicitly model the authorization for obtaining tickets with inert rights via a demand operation, since the specific manner in which this is done does not influence our analysis.

Clearly transport tickets are crucial in determining the authorization for changing the protection state. For this aspect of the demand operation, we require the authorization to be in terms of types by providing the following design parameter.

Definition 2.7: The designer of a protection scheme specifies a **demand** function

$$\textit{demand}: T_S \rightarrow \textbf{power-set}(T_S \times R_T)$$

which authorizes acquisition of transport tickets on demand so that

$$b/\mathrm{x:c} \in \textit{demand}(a)$$

authorizes every subject A of type a to obtain the ticket B/x:c for every subject B of type b. The only requirement is that both A and B must exist. ■

A possible use of this feature is to allow certain types of subjects to establish links between themselves. For example, if we have

$$a/\mathrm{s} \in \textit{demand}(a)$$
$$a/\mathrm{r} \in \textit{demand}(a)$$

then two subjects of type a can establish links in both directions by demanding the send and receive tickets for each other. Whenever a new subject of type a is introduced, it can be linked in both directions to every existing subject of type a by use of the demand operation. Thus a policy that any two users of type a can directly transport tickets to each other, is enforced in a straightforward manner. Similarly, a policy that every user of type a can directly transport tickets to a user of type b, but not necessarily vice versa, is enforced as follows.

$$b/\mathrm{s} \in \textit{demand}(a)$$
$$a/\mathrm{r} \in \textit{demand}(b)$$

The ability to obtain transport tickets on demand also allows us to model degenerate cases of our send-receive framework in a straightforward way. Specifically, consider the following situations.

1. The *demand* function authorizes subjects of every type to obtain a send ticket for subjects of every type, i.e.,

   $$(\forall \langle a,b \rangle)\ [b/\mathrm{s} \in \textit{demand}(a)]$$

   For such schemes the distribution of send tickets is of no consequence, since these tickets can be obtained simply by demanding them. The transport operation is then effectively controlled by the distribution of receive tickets.

2. Similarly at the other extreme, the *demand* function authorizes subjects of every type to obtain a receive ticket for subjects of every type, i.e.,

$$(\forall\langle a,b\rangle)\ [b/r \in demand(a)]$$

For such schemes the distribution of receive tickets is of no consequence, since these tickets can be obtained simply by demanding them. The transport operation is then effectively controlled by the distribution of send tickets.

3. It is also possible to completely eliminate the demand operation, for transport tickets, by specifying that

$$(\forall a)\ [demand(a) = \phi]$$

2.2.3. The Can-Create Relation and Create-Rules

There are two aspects to the create operation, viz., creation of objects and creation of subjects. By definition, objects are passive entities incapable of any autonomous action. Moreover, objects are not permitted to possess tickets. Hence, creation of objects does not present any major problem in analysis. We do not explicitly model the authorization which controls creation of objects, since the specific manner in which this is done does not influence our analysis.

Creation of subjects introduces a major complication in analysis, since then there is potentially an infinite set of subjects all of which are capable, in principle, of transporting tickets to each other. There are two issues which have to be addressed.

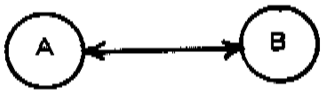
The first issue is the authorization required for a create operation. In line with our general theme, we require that the authorization be expressed in terms of types as follows.

Definition 2.8: The designer of a protection scheme specifies a **can-create** relation

$$can\text{-}create \subseteq T_S \times T_S$$

which authorizes creation of subjects so that a subject of type a can create a subject of type b if and only if *can-create*(a,b). ■

The second issue is the precise semantics of the create operation. Consider the creation of subject B by subject A. A reasonable semantics for this operation might be to require that immediately after the creation of B there is a link from A to B and vice versa as shown below.



This would authorize transport of tickets in both directions as permitted by the *dfl* function. Establishing these links requires placing transport tickets for A and B so that

B/s ∈ dom(A)	A/s ∈ dom(B)
B/r ∈ dom(A)	A/r ∈ dom(B)

Now there is an additional question as to whether any of these tickets should carry the copy flag. Since A is the creator of B, it might be useful to give A transport tickets for B with the copy flag so that the tickets B/sc and B/rc are placed in the domain of A. Then A can propagate these tickets to other subjects in the system.

On the other hand it may not be desirable to give A copiable transport tickets for B. For example, we might want to allow an ordinary user the ability to create a very powerful subject provided the created subject can be isolated from the rest of the system. The ordinary user might use this very powerful subject to create a subsystem for his own personal experiments. In such a case it might even be desirable to give the created subject copiable transport tickets for itself as this could aid in constructing a subsystem.

The point is that we must face several issues about the precise semantics of creation. In line with the theme of types we require the semantics for creation to be specified in terms of types. The designer of a scheme must specify the result of a subject of type *a* creating a subject of type *b*. For a given pair of subject types *<a,b>* we call this specification the *<a,b>* **create-rule**. As examples of create rules, we have the two cases mentioned earlier and repeated below.

1. If a subject A of type *a* creates a subject B of type *b* then immediately after the create operation has occurred

B/s ∈ dom(A)	A/s ∈ dom(B)
B/r ∈ dom(A)	A/r ∈ dom(B)

2. If a subject A of type *a* creates a subject B of type *b* then immediately after the create operation has occurred

B/sc ∈ dom(A)	A/s ∈ dom(B)
B/rc ∈ dom(A)	A/r ∈ dom(B)

Although we are willing to provide flexibility in the specification of create-rules, we do impose one major restriction as follows.

Definition 2.9: The operation of subject A creating subject B is **local** if the (immediate) impact is only on the domain of A and the domain of B in terms of transport tickets for A and B. ■

We intend to rule out a situation where some other subject C might be given a transport ticket for the created subject B as an immediate result of the create operation. We also rule out a situation where B is given some tickets other than the transport tickets for A and B. This later task can be accomplished after the create has occurred by a transport operation or by B demanding such tickets. The restriction that create-rules must be local embodies a fundamental principle of systems design, which we encountered earlier on page 6, viz., that any "small" or "frequent" event in the system state should result in a "small" change in the protection state.

The designer of a protection scheme then specifies the semantics of the create operation as follows.

Definition 2.10: For every *<a,b>* ∈ *can-create* the designer of a protection scheme specifies a **create-rule**. This rule must be local and determines the (immediate) consequence of a create operation, where a subject of type *a* creates a subject of type *b*. ■

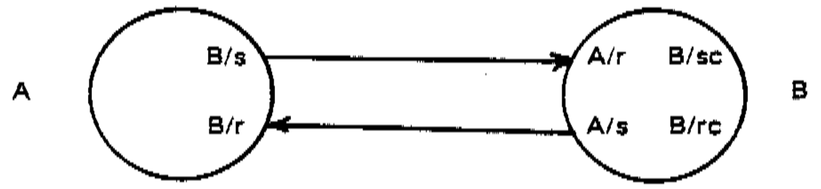
Observe that the designer is allowed to specify different create-rules for every pair in the *can-create* relation. Of course, the designer may choose to specify the same create-rule for every pair. We now single out a particular create-rule which we use extensively in our examples.

Definition 2.11: The **self-copy create-rule** specifies that immediately after a subject A creates a subject B

1. The tickets B/s, B/r are placed in dom(A).

2. The tickets A/s, A/r, B/sc, B/rc are placed in dom(B).

In terms of a diagram, the situation immediately after A creates B is as follows.



■

Unless otherwise mentioned, we will use this rule in our examples. This saves us the task of specifying create-rules every time we discuss an example involving creates.

2.2.4. Constraints on the Initial State

A particular system state is determined by the following aspects.

1. The set SUB of subjects.
2. The set OBJ of objects.
3. The distribution of tickets for subjects and objects.

By definition, objects play a limited role and we can ignore their existence in our analysis. We are interested in the following aspects of a system state.

1. The set SUB of subjects.
2. The distribution of transport tickets.

We will refer to this portion of a system state as the **protection state**.

In our framework, the designer of a particular system approaches this task in two stages. First he has to come up with a protection scheme by specifying the parameters defined in the previous three sections. Then he has to specify an initial state by defining the initial set of subjects, the initial set of objects, and the initial distribution of tickets.

In certain cases we will find it useful to constrain the nature of this initial state. It is convenient to consider such restrictions as part of the protection scheme itself. Adopting a specific scheme limits the range of policies. Restricting the initial state has a similar effect. Of course, these constraints apply only to the initial state and need not be true in subsequent states.

Abstractly, a **constraint** is simply some predicate which evaluates to true or false for a given protection state. Let us consider what kind of constraints[3] might be imposed on the initial state. There are two independent aspects to the initial distribution of tickets. One aspect is the distribution of send and receive tickets. We might require that if a subject A is given a send ticket for subject B then B must also be given a receive ticket for A, i.e.,

[3] Most of the constraints discussed here were earlier encountered in our discussion of the uniform send-receive mechanism in section 1.2.4.

$$(\forall \langle A,B\rangle \in SUB^{0} \times SUB^{0})\,[B/s \in dom^{0}(A) \Leftrightarrow A/r \in dom^{0}(B)]$$

This assumption forces the design of an initial state to be in terms of links without introduction of stray tickets. We refer to this constraint as the **balanced state** requirement.

A second aspect of the initial state, is the occurrence of tickets with the copy flag. We will find the following assumption useful.

$$(\forall A \in SUB^{0})\,[A/sc \in dom^{0}(A) \wedge A/rc \in dom^{0}(A)]$$

We refer to this as the **self-reference** assumption. We say that the tickets A/sc and A/rc are the **self-reference tickets** for the subject A. The self-reference assumption allows each subject a degree of autonomous control over tickets for itself. More importantly, it will enable us to ignore the create operation under certain conditions investigated in chapter 4. Thus it has a definite impact on the complexity of analyzing the behavior of a particular system. Moreover, this assumption can be made without substantial loss of generality. Indeed, we can effectively eliminate these tickets from the domain of a particular subject A by ensuring that

$$(\forall b \in T_S)\,[t(A)/x{:}c \notin dfl(t(A),b)]$$

Further, note that every initial state can be augmented to be a self-reference state simply by introducing the self-reference tickets in the domain of every subject. Hence, the self-reference assumption serves as a useful theoretical tool for an approximate analysis[4].

In chapters 5 and 6 we will go one step further and stipulate that the only tickets in the initial state with the copy flag are those required by the self-reference assumption, i.e.,

$$(\forall \langle A,B\rangle \in SUB^{0} \times SUB^{0})\,[A/sc \in dom^{0}(B) \Leftrightarrow A = B]$$
$$(\forall \langle A,B\rangle \in SUB^{0} \times SUB^{0})\,[A/rc \in dom^{0}(B) \Leftrightarrow A = B]$$

We refer to this as the **self-copy assumption**. This is a major restriction and to single out its importance we will name the class of schemes after this particular constraint. The idea here is that there should be a single source from which all

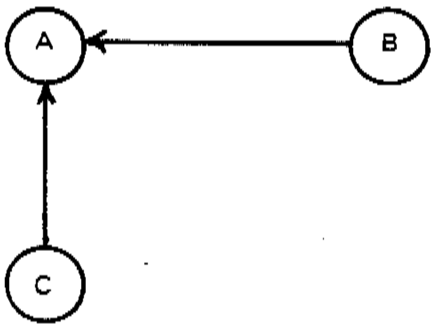
[4]In the following sense: if a particular configuration of the protection state cannot be realized starting from an initial state which has been augmented to satisfy the self-reference assumption, then the configuration certainly cannot be realized starting from the actual initial state.

copies of a ticket are obtained, so that mechanisms for revocation of these copies can be easily implemented.

For the sake of uniformity, in our examples we always begin with a balanced initial state which satisfies the self-copy assumption. This reduces our task in specifying an initial state, since it then suffices to show the initial set of links. The initial distribution of transport tickets can be easily deduced from the initial set of links, as follows.

- By the self-copy requirement, all links in the initial state are established by tickets without the copy flag.
- By the self-copy (and, hence, self-reference) requirement, every subject possesses the self-reference tickets for itself.
- By the balanced requirement, there are no tickets other than those required to establish the links and the self-reference tickets.

For example, consider the following initial state.



If this state satisfies the balanced self-copy requirement, it is implicit that transport tickets in this state are distributed as follows.

$$\begin{aligned}
\text{dom}^{0}(A) &= \{A/sc,\ A/rc\} \cup \{C/r,\ B/r\} \\
\text{dom}^{0}(B) &= \{B/sc,\ B/rc\} \cup \{A/s\} \\
\text{dom}^{0}(C) &= \{C/sc,\ C/rc\} \cup \{A/s\}
\end{aligned}$$

2.2.5. Summary and Examples

In the preceding sections we have defined a framework for the design of a protection scheme. Certain assumptions are basic to the framework and apply to any specific scheme. The most significant assumptions are as follows.

1. Every subject is created to be of a specific type and its type cannot change.

2. New types cannot be introduced at run time.

3. Selectivity in the transport, demand and create operations is determined by the types of subjects and tickets involved.

We defined a number of parameters which must be specified in order to define a particular scheme. These parameters are summarized below.

Definition 2.12: A **selective send-receive protection scheme** (or simply **scheme**) is defined by specifying

1. The set of types $\mathbf{T} = \mathbf{T}_S \cup \mathbf{T}_O$ where $\mathbf{T}_S \cap \mathbf{T}_O = \phi$.
2. The set of rights $\mathbf{R} = \mathbf{R}_I \cup \mathbf{R}_T$ where $\mathbf{R}_T = \{s, r, sc, rc\}$.
3. The function *dfl*: $\mathbf{T}_S \times \mathbf{T}_S \rightarrow$ **power-set**($\mathbf{T} \times \mathbf{R}$).
4. The function *demand*: $\mathbf{T}_S \rightarrow$ **power-set**($\mathbf{T}_S \times \mathbf{R}_T$).
5. The relation *can-create* $\subseteq \mathbf{T}_S \times \mathbf{T}_S$.
6. A local create-rule for every pair $<a,b> \in$ *can-create*.
7. Constraints on the initial state.

■

The first five parameters are each formalized in terms of a set, function, or relation. The precise definition of the last two parameters has been deliberately left open but we have indicated the scope we have in mind.

Let us consider some examples of schemes which can be defined using this framework. For simplicity, assume that, unless otherwise stated, the create-rule is the self-copy rule and there is no constraint on the initial state.

Example 1: For our first example we specify the design parameters as follows.

1. $\mathbf{T}_S = \{a\}$, $\mathbf{T}_O = \{o\}$
2. $\mathbf{R}_I = \{pc, p, uc, u\}$
3. *dfl* $(a,a) = \mathbf{T} \times \mathbf{R}$
4. *demand* $(a) = \phi$
5. *can-create* $= \{<a,a>\}$

For this scheme there is a single subject type *a* and a single object type *o*. The set of inert rights distinguishes two kinds of rights interpreted as follows.

p: authorizes operations which preserve the internal state of the entity
u: authorizes operations which update the internal state of the entity

The *dfl* function allows all types of tickets to be transported. The *demand* function does not allow transport tickets to be obtained by a demand operation. The *can-create* relation authorizes subjects to create other subjects.

We can also accommodate several variations of this scheme. For instance we can modify the *demand* function so that

$$demand(a) = \{a/s\}$$

Then any subject can obtain a send ticket for another subject merely by demanding it, and the transport operation is effectively controlled by receive tickets. Similarly, if we modify the *demand* function so that

$$demand(a) = \{a/r\}$$

any subject can obtain a receive ticket for another subject merely by demanding it, and the transport operation is effectively controlled by send tickets.

Example 2: For our next example we specify the design parameters as follows.

1. $T_S = \{a, b\}$, $T_O = \{o\}$

2. $R_I = \{pc, p, uc, u\}$

3. $dfl(a,a) = T \times \bar{R}$
   $dfl(a,b) = T \times R$
   $dfl(b,a) = T \times R$
   $dfl(b,b) = \phi$

4. $demand(a) = \phi$
   $demand(b) = \phi$

5. $can\text{-}create = \{<a,a>, <a,b>\}$

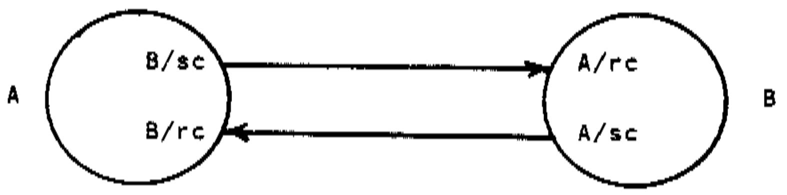
Now there are two subject types *a* and *b*. The *dfl* function allows for transport of all types of tickets between all pairs of subjects, unless both subjects are of type *b*. We can also consider variations of this example along the lines of the variations considered for example 1.

These two examples indicate the kinds of schemes which can be accommodated in our framework. In both cases there is a considerable amount of policy built into the scheme. We will consider more complex structures as we develop our analysis.

Finally, consider how the uniform send-receive mechanism, reviewed in section 1.2.4, can be modeled in our framework. We can eliminate the role of the copy flag as follows.

1. Require that every ticket in the initial state carries the copy flag.

2. Let the create-rule specify that immediately after a subject A_1 creates subject A_2 the situation is



The fact that transport tickets are universally transportable is easily expressed by the following requirement.

$$(\forall\langle a,b\rangle[T_S \times R_T \subseteq dfl\,(a,b)]$$

Any scheme which satisfies these requirements will, naturally, be called a **uniform scheme**.

2.3. TRANSITION SEQUENCES

There are several ways by which the protection state can change. Every **state transition** will be caused by one of the following operations.

1. A transport operation which moves a copy of a transport ticket from the domain of one subject to the domain of another.

2. A create operation which introduces a new subject in the system.

3. A demand operation which places a transport ticket in the domain of some subject.

It is the responsibility of the protection mechanism to ensure that there is proper authorization for every state transition. We say that a state transition is **legal** provided there is appropriate authorization for the operation which causes the state transition. A sequence of legal state transitions is called a **transition sequence**.

We will denote a transition sequence by upper case letters from the middle part of the alphabet, such as H and G. Unless otherwise mentioned, a transition sequence is applied to the initial state. The state resulting after all operations in the sequence have been executed will be denoted by the corresponding lower case letter, such as h and g respectively. Any state established by a transition sequence is called a **derived state**.

The reader may have observed that the authorization for create and demand operations is more or less independent of a particular protection state. If a subject A is allowed to create a subject B the requirements are

1. The subject A must exist
2. $<t(A), t(B)> \in$ *can-create*

The state dependent requirement that A must exist is relatively trivial. Similarly if A is authorized to demand a ticket B/x:c the requirements are

1. The subject A must exist
2. The subject B must exist
3. $t(B)/x{:}c \in$ *demand*$(t(A))$

Once again, the state dependent requirement that A and B must exist is relatively trivial.

It will occasionally be convenient for us to assume that the three kinds of state transitions occur in a particular order. Specifically, that all the create operations occur first, followed by all the demand operations, and finally followed by all the transport operations. We say that a transition sequence with this structure is in **canonical form**. Due to the (almost) state independent nature of authorization for create and demand operations, this assumption does not entail any loss of generality. Specifically, we have the following lemma.

Lemma 2.1: For every transition sequence H establishing state h, there is a transition sequence H' in canonical form which establishes state h.

Proof: Modify H to obtain H' by rearranging the operations in H to conform to the canonical form, while preserving the relative order of each kind of operation. Since the relative order of the create operations is preserved, the create operations in H' are legal. Since all subjects have been created prior to any demand operation, these operations are legal in H'. The transport operations in H' are in the same relative order as in H and are preceded by all the create and demand operations and, hence, are legal in H'. Since H' consists of exactly the same operations as H, the state h' resulting from H' is identical to the state h resulting from H. ■

We can then assume, without loss of generality, that any derived state is established by a transition sequence in canonical form.

2.4. MODELING PASSIVE SUBJECTS

Let us review the subject object distinction[5]. There are two independent aspects to this distinction as follows.

1. Subjects may possess tickets while objects cannot possess tickets.

2. Objects are passive and cannot execute any operations in the system, whereas subjects are active and can autonomously execute various operations in the system.

There is a slight problem in coupling both these aspects. Consider a directory of files. In a ticket based approach, a directory is naturally viewed as a set of tickets with one or more tickets for each file in the directory. Since a directory contains tickets, it must be modeled as a subject. At the same time, a directory is a passive entity which does not match the active character of a subject.

This problem is not peculiar to our framework and will arise in any capability based scheme. Our approach is to relax the assumption that all subjects are active. We then have two categories of subjects, such that an **active subject** is capable of independently initiating actions in the system whereas a **passive subject** is not capable of doing so. In our framework, it is possible to model passive subjects by limiting the authorization of such subjects for invoking operations. First, consider the operation of transporting a ticket from subject A to subject B. There are four cases here.

1. Let A and B both be active subjects. The authorization for a direct transport of tickets from A to B is expressed by the requirement

   $$B/s \in \mathrm{dom}(A) \wedge A/r \in \mathrm{dom}(B)$$

   and is the same whether A or B initiates the operation. In such a case we assume that either A or B can initiate the transport operation. In either circumstance, both A and B must act to ensure that the transport operation is successful.

2. Let A be an active subject while B is a passive subject. By definition

[5]This section may be skipped on a first reading.

only A can initiate the transport operation. We simply assume that B is a willing receiver and will execute any action required to enable the transport operation. Such actions can be automatically executed on behalf of B. In particular, if

$$A/r \notin \mathrm{dom}(B) \ \wedge \ t(A)/r \in \mathit{demand}(t(B))$$

then the demand operation to obtain the A/r ticket should be automatically executed on behalf of B.

3. Let A be a passive subject while B is an active subject. This situation is similar to the previous situation. Here, we assume that A is a willing sender and will execute any action required to enable the transport operation. In particular, if

$$B/s \notin \mathrm{dom}(A) \ \wedge \ t(B)/s \in \mathit{demand}(t(A))$$

then the demand operation to obtain the B/s ticket should be automatically executed on behalf of A.

4. Let A and B both be passive subjects. By definition, neither A nor B can initiate a transport operation. We model this case by requiring that

$$\mathit{dfl}(t(A), t(B)) = \phi$$

Then there is no authorization for transport of tickets from A to B and, even if one of them was an active subject, the transport operation would not be successful.

In our discussion above, we assumed that a demand operation can be automatically executed, on behalf of a passive subject, when required for enabling a transport operation. The reader may wonder why we did not simply require that *demand*(a) be empty if subjects of type a are passive. Observe that our approach does not preclude the latter requirement, and the designer is free to ensure that indeed *demand*(a) is empty if subjects of type a are passive. At the same time, our approach allows us to accommodate passive subjects in the following degenerate cases of our send-receive framework.

1. A situation where the transport operation is controlled by a receive ticket alone and which is modeled in our framework by requiring that

$$(\forall\langle a,b\rangle)\,[b/s \in \mathit{demand}(a)]$$

2. A situation where the transport operation is controlled by a send ticket alone and which is modeled in our framework by requiring that

$$(\forall\langle a,b\rangle)\,[b/r \in \mathit{demand}(a)]$$

It remains to consider the create operation. In order to model the passive nature of subjects, it suffices to ensure that <*a*,*b*> is in the *can-create* relation only if subjects of type *a* are active.

Our approach to modeling passive subjects will work provided subjects of a given type are either all passive or all active. This is not a major restriction and is consistent with the notion of a type. Example 2 on page 48 illustrates a scheme for which subjects of type *b* can be interpreted as being passive.

2.5. NAMING CONVENTIONS

We conclude this chapter by apologizing to the reader for the variety of notation we have had to introduce, while cautioning that there is still more to come as we formulate analysis questions. There is no escape from notation in a formal treatment, but we are especially handicapped by the lack of standard terminology due to the recent emergence of the problems being addressed in this thesis.

Certain aspects of our notation have been deliberately designed to help in distinguishing the role of various symbols. In particular, we have followed the conventions stated below.

1. There are two basic sets which pervade the entire discussion, viz.,

   SUB: the current set of subjects
   T_S: the set of subject types

   Members of T_S are denoted by lower case italicized letters such as *a, b, c*. Members of SUB are denoted by upper case letters such as A, B, C. To the extent possible, we ensure that the type of a subject is the corresponding lower case italic letter.

2. Any relation which is a subset of $T_S \times T_S$ is named using italic script. Similarly, a function with domain T_S or $T_S \times T_S$ is named using italic script. As examples of this convention we have the *can-create* relation; and the *demand* and *dfl* functions.

3. Any relation which is a subset of SUB X SUB is named using normal script. Similarly, a function with domain SUB or SUB X SUB is named using normal script. As an example of this convention we have the link relation.

4. We have made one exception to these rules in naming the *t* function,

which tells us the type of a given subject. According to our stated convention, its name should be in normal script; since its domain is the set SUB. We have deviated from this convention to ensure that every member of T_S appears in italic script; either directly as *a*, *b*, *c* etc., or indirectly, via an application of *t*, as *t*(A), *t*(B), *t*(C) etc. The net result is that any function or relation with an italicized name must have italicized arguments.

Our methodology for naming is discussed in further detail in appendix B.

CHAPTER 3
THE FLOW-ANALYSIS PROBLEM

In this chapter we discuss a basic analysis question called the flow-analysis problem. We begin by defining the flow function in section 3.1. For every pair of subjects A and B, in a given state, this function expresses the authorization for transport of ticket types from the domain of A to the domain of B accounting for indirect as well as direct transport. The values of this function may change as the protection state evolves. In section 3.2 we discuss an example to demonstrate the nature of this change.

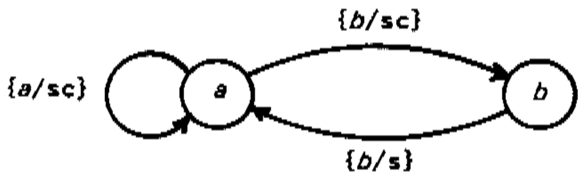
Stated broadly, the flow-analysis problem is to characterize properties of the flow function. In section 3.3 we formalize this problem and discuss a number of approaches to its solution. These approaches are then investigated in the remainder of this chapter and in the next two chapters.

3.1. THE FLOW FUNCTION

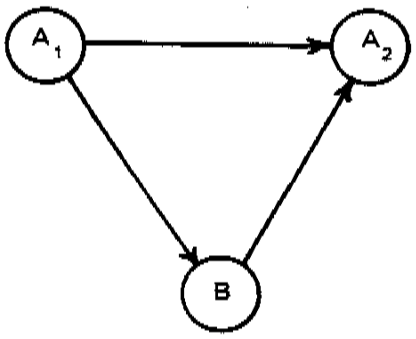
We motivate the definition of the flow function by an example. Consider a scheme with the *dfl* function shown below.

$$\begin{aligned} dfl(a,a) &= \{a/\mathbf{sc}\} \\ dfl(a,b) &= \{b/\mathbf{sc}\} \\ dfl(b,a) &= \{b/\mathbf{s}\} \\ dfl(b,b) &= \phi \end{aligned}$$

The *dfl* graph for this function is as follows.



Let A_1 and A_2 be subjects of type *a*, and B a subject of type *b*. Assume that we have the following situation.



The State k

Consider the nature of tickets which can be transported from A_1 to A_2. There is a link from A_1 to A_2. In conjunction with the *dfl* function, this link authorizes transport of tickets of type *a*/sc directly from A_1 to A_2. Similarly, it is possible to directly transport tickets of type *b*/sc from A_1 to B. While it is not possible to transport tickets of type *b*/sc directly from B to A_2, it is possible to transport tickets of type *b*/s. But then, tickets of type *b*/s can be indirectly transported from A_1 to A_2, via the subject B. The net authorization for transport of ticket types from A_1 to A_2 in state k can then be expressed as follows.

$$\mathrm{flow}^k(A_1, A_2) = \{a/sc,\ b/s\}$$

This set specifies the types of tickets which can be transported, either directly or indirectly, from A_1 to A_2 in the state k.

To formally define the flow function, it is useful to introduce the following notion of a path.

Definition 3.1: We say there is a **path** from subject A to subject B in state k provided

1. $\mathrm{link}^k(A,B)$

or 2. There exists a sequence of subjects $C_1, \ldots, C_n$ such that

(a) $\mathrm{link}^k(A, C_1)$

(b) $\mathrm{link}^k(C_i, C_{i+1}) \quad i=1,\ldots,n-1$

(c) $\mathrm{link}^k(C_n, B)$

In the latter case we say that the path **consists of the sequence** of subjects $C_1,\ldots,C_n$. ∎

Given a link from A to B, the types of tickets which can be directly transported from A to B is expressed by the set $dfl(t(A),t(B))$. Consider a path from A to B which consists of more than one link, for example,

$$\begin{tikzcd} A \arrow[r] & C_1 \arrow[r] & C_2 \arrow[r] & B \end{tikzcd}$$

Now a ticket of type $d/x{:}c$ can be transported from A to B over this path provided

1. A ticket of type d/xc can be directly transported from A to C_1 and from C_1 to C_2.

2. A ticket of type $d/x{:}c$ can be directly transported from C_2 to B.

In terms of the dfl function these two conditions are expressed as follows.

$$[d/xc \in dfl(t(A),t(C_1)) \cap dfl(t(C_1),t(C_2))]$$
$$\wedge$$
$$[d/x{:}c \in dfl(t(C_2),t(B))]$$

This leads to the following definition.

Definition 3.2: We say there is **authorization** to transport tickets of type $d/x{:}c$ over a specific path from subject A to subject B if

1. The path consists of a single link from A to B and

$$d/x{:}c \in dfl(t(A),t(B))$$

or 2. The path consists of a sequence of subjects $C_1,\ldots,C_n$ and

$$[d/xc \in dfl(t(A),t(C_1)) \cap \ldots \cap dfl(t(C_{n-1}),t(C_n))]$$
$$\wedge$$
$$[d/x{:}c \in dfl(t(C_n),t(B))]$$

■

The definition of the flow function is then as follows.

Definition 3.3: For every state k, define the associated **flow function**

$$flow^k\colon SUB^k \times SUB^k \longrightarrow \textbf{power-set}(T \times R)$$

by $c/x{:}c \in flow^k(A,B)$ if and only if

1. $A = B$

or 2. There exists a path from A to B in state k for which there is authorization to transport tickets of type $c/x{:}c$.

■

The first condition merely ensures that, for every subject A and every state k, we have $flow^k(A,A) = T \times R$. This is a matter of convenience for our exposition and is consistent with the intuition behind the flow function. The second condition is the interesting one. It ensures that, for two distinct subjects A and B, the value of $\text{flow}^k(A,B)$ summarizes the authorization for transport of ticket types over all paths, in state k, from A to B.

By defining the flow function in terms of ticket types, we avoid consideration of individual tickets in our analysis. In particular, tickets for objects do not influence the flow function in any way. By developing our analysis in terms of the flow function, we can simply ignore such tickets.

For a given state, the flow^k function can be computed from the link^k relation in a straightforward manner. As discussed in appendix C.3, this computation has a cost of $O(|T \times R| * |SUB^k|^3)$.

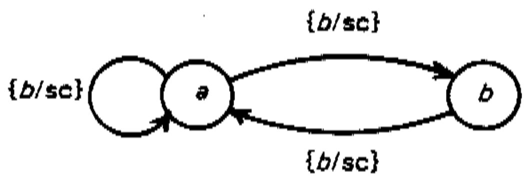
3.2. EVOLUTION OF THE FLOW FUNCTION

In this section we discuss an example which demonstrates how the flow function may evolve from a given initial state. The concepts, which are informally introduced here, will be formalized in section 3.3.

The scheme for our example has the following *dfl* function

$$
\begin{aligned}
dfl(a,a) &= \{b/\text{sc}\} \\
dfl(a,b) &= \{b/\text{sc}\} \\
dfl(b,a) &= \{b/\text{sc}\} \\
dfl(b,b) &= \phi
\end{aligned}
$$

with the associated *dfl* graph

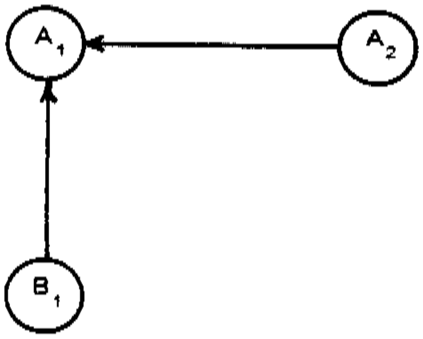


There are two subject types *a* and *b*. The only tickets which can be moved are of type *b*/sc and such tickets can potentially be transported between all pairs of subject types except for the pair <*b*,*b*>. We specify the *demand* function to be

$$
\begin{aligned}
demand(a) &= \{a/\text{r},\ b/\text{r}\} \\
demand(b) &= \{a/\text{r},\ b/\text{r}\}
\end{aligned}
$$

Any subject can then obtain a receive ticket for another subject simply by demanding it and, hence, the transport operation is effectively controlled by send tickets.

For the moment we do not allow any create operation. Now consider the following balanced self-copy initial state[1], where subjects A_1 and A_2 are of type *a* while subject B_1 is of type *b*.



The Initial State

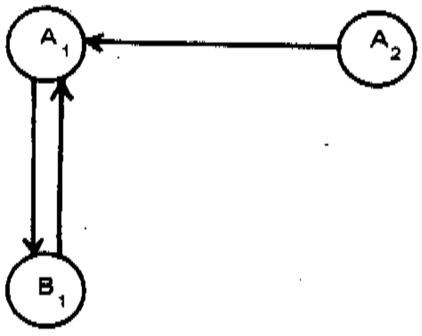
The only links in the initial state are from A_2 to A_1 and from B_1 to A_1. Both these links are directed towards A_1. The *dfl* function authorizes transport of the ticket type *b*/sc over each of these links. Hence, the initial flow of tickets from A_2 to A_1 and from B_1 to A_1 is {*b*/sc}. There are no links directed towards the subject A_2. Hence, the initial flow from both A_1 and B_1 to A_2 is the empty set. Similarly, there are no links directed towards the subject B_1 and the initial flow from A_1 and A_2 to B_1 is the empty set. By convention, the flow from every subject to itself is the entire set of ticket types T X R. Thus the flow^0 function, for this initial state, is as follows.

[1]See page 46.

$$\begin{array}{c|ccc|}
 & A_1 & A_2 & B_1 \\
\hline
A_1 & T \times R & \phi & \phi \\
A_2 & \{b/sc\} & T \times R & \phi \\
B_1 & \{b/sc\} & \phi & T \times R \\
\hline
\end{array}$$

The Flow$^{\circ}$ Function

Consider how this system may evolve from the initial state without any create operations. The only tickets which can be transported are tickets of type *b*/sc and there is only one subject B_1 of type *b*. By the self-reference assumption, B_1 possesses the ticket B_1/sc and, hence, this ticket can be transported to the domain of A_1. Since *a*/*r* ∈ *demand*(*b*), B_1 can obtain the ticket A_1/r by demanding it. These two operations will establish a link from A_1 to B_1 resulting in the state shown below.



The State k

The only ticket which can be moved is still the ticket B_1/sc. But there is no point in transporting it to the domain of B_1 or A_1, since both subjects already possess this ticket. Thus, in the absence of a create operation, no further links can be introduced and the maximum possible flow, between all pairs of subjects, has been realized in state k. We will call such a state a maximal state without creates, and denote the corresponding flow function as flow$^{\#}$.

It is a simple matter to compute the flow$^{\#}$ function for our example. The link from A_1 to B_1 enables transport of tickets of type *b*/sc tickets from A_1 to B_1. Since there is already authorization for transport of tickets of type *b*/sc tickets from A_2 to A_1, by transitivity tickets of type *b*/sc can be transported from A_2 to B_1. Hence, the flow from both A_1 and A_2 to B_1 is {*b*/sc}. However, there are still no links directed towards A_2 and the flow from both A_1 and B_1 to A_2 is still the empty set. Thus, we arrive at the following values for the flow* function.

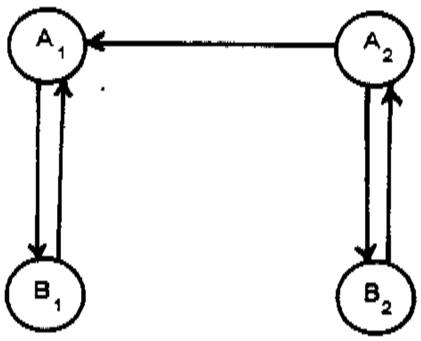
$$\begin{array}{c|ccc|}
 & A_1 & A_2 & B_1 \\
\hline
A_1 & T \times R & \phi & \{b/sc\} \\
A_2 & \{b/sc\} & T \times R & \{b/sc\} \\
B_1 & \{b/sc\} & \phi & T \times R \\
\hline
\end{array}$$

The Flow* Function

We now consider how the flow function may further evolve in the presence of create operations. We authorize subjects of type *a* to create subjects of type *b*, i.e.,

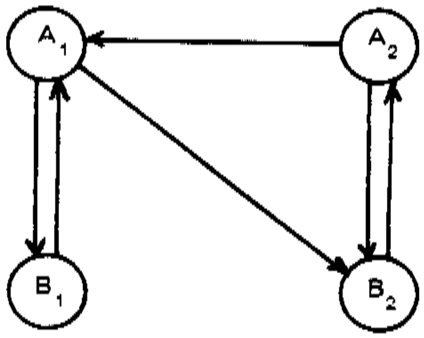
$$can\text{-}create = \{<a, b>\}$$

It is then permissible for the subject A_2 to create a subject B_2 of type *b*. This results in the following state.



A Create Operation

The ticket[2] B_2/sc can be moved to the domain of A_1, since there is a link from B_2 to A_2 and a link from A_2 to A_1, where both these links are authorized by the *dfl* function to transport tickets of type *b*/sc. The subject B_2 can obtain the ticket A_1/r by a demand operation. Hence, it is possible to establish a link from A_1 to B_2 as shown below.



The State n

In this state tickets of type *b*/sc can be transported from A_1 to A_2 via the subject B_2. Hence, the value of flow^n(A_1,A_2) is {*b*/sc}. Since tickets of type *b*/sc could already flow from B_1 to A_1, by transitivity the value of flow^n(B_1,A_2) is {*b*/sc}. The flow^n function is then as follows.

	A_1	A_2	B_1
A_1	T X R	{*b*/sc}	{*b*/sc}
A_2	{*b*/sc}	T X R	{*b*/sc}
B_1	{*b*/sc}	{*b*/sc}	T X R

The Flow^n Function

By examining the *dfl* graph on page 58, it is evident that only tickets of type

[2]By the self-copy create-rule, which is the used in all our examples, B_2 is created with B_2/sc in its domain.

b/sc can be transported. It follows that the maximum value of the flow function, between two distinct subjects, is $\{b/sc\}$. But this is exactly the value of the $flow^n$ function for every pair of distinct subjects. Hence, the maximum value of the flow function, between subjects which were present in the initial state, has been realized in state n. We will call such a state a maximal state, and denote the corresponding flow function as $flow^*$. We now formalize these concepts in the next section.

3.3. THE FLOW-ANALYSIS PROBLEM

There are two kinds of changes in the flow function, that occur as the protection state evolves. Firstly, the domain of the flow function changes as new subjects are created. Secondly, for a given pair of subjects, the value of the flow function changes as new links are established. In our analysis we will, for the most part, focus on the latter kind of change. Indeed, analysis questions with respect to subjects created subsequent to the initial state, can only be answered indirectly, as for example:

> For every pair of subjects A' and B', which are descendants of A and B respectively, there exists a derived state k such that for all derived states h we have
>
> $$flow^h(A',B') \subseteq flow^k(A,B)$$
>
> That is, if a flow can be realized between two descendants of A and B it can also be realized between A and B.

We will obtain such results as corollaries to our main theme of analysis, which focuses on the initial set of subjects.

Since there is no provision for deletion of subjects, the initial set of subjects continue to exist in all derived states. In order to focus on changes in the flow function with respect to this set, we introduce the following notion.

> **Definition 3.4:** For a given scheme and initial state, we say that a derived state m is a **maximal state** if it is the case that for all derived states h
>
> $$(\forall\langle A,B\rangle \in SUB^o \times SUB^o)[flow^h(A,B) \subseteq flow^m(A,B)]$$
>
> ■

We will shortly show that maximal states exist for every scheme and every initial state. For the moment, let us assume that this has been accomplished and consider why we should be interested in maximal states. By definition, in a maximal state the maximum possible value of the flow function between every pair of subjects, which were present in the initial state, has been realized. Indeed, this is the motivation for introducing this notion and leads us to the following definition.

Definition 3.5: For a given scheme and a given initial state, define the associated **maximal flow function**

$$\mathrm{flow}^{*}\colon \mathrm{SUB}^{\mathrm{O}} \times \mathrm{SUB}^{\mathrm{O}} \longrightarrow \textbf{power-set}\,(T \times R)$$

by restricting the domain of the flow function in a maximal state to the initial set of subjects, i.e.,

$$(\forall\langle A,B\rangle \in \mathrm{SUB}^{\mathrm{O}} \times \mathrm{SUB}^{\mathrm{O}})\,[\mathrm{flow}^{*}(A,B) = \mathrm{flow}^{m}(A,B)]$$

where m is a maximal state. ■

Clearly, a protection scheme constrains the evolution of an initial state only to the extent determined by the flow* function. Of course, the actual flow function in a derived state may be considerably less permissive, as determined by the actions undertaken by individual subjects. However, under a worst case scenario, the flow* function is useful in answering questions about tickets which can be acquired by a given subject. In particular, consider the **safety problem** of Harrison, Russo and Ullman [6]. This problem poses the following question: Is it possible for subject A to acquire the ticket B/x:c? Under the assumption[3] that the only copiable tickets B/xc are those introduced in the initial state, this question can be formulated in terms of the flow* function, as follows: Does there exist a subject C, who possesses the ticket B/xc in the initial state and t(B)/x:c is in flow*(C,A)?

Now let us return to the question of establishing the existence of maximal states. Due to the create operation, in general there are an infinite number of derived states. However, as far as the flow function between subjects present in the initial state is concerned, we can classify these states into a finite number of equivalence states as follows.

Definition 3.6: For a given scheme and initial state, two derived states g and h are said to be **equivalent** (with respect to the initial set of subjects) provided

$$(\forall\langle A,B\rangle \in \mathrm{SUB}^{\mathrm{O}} \times \mathrm{SUB}^{\mathrm{O}})\,[\mathrm{flow}^{g}(A,B) = \mathrm{flow}^{h}(A,B)]$$

■

[3] We will discuss what happens under the more general situation, where copiable tickets can be obtained by a demand operation, on page 85.

This notion of equivalence is clearly reflexive, symmetric and transitive, thereby partitioning the set of derived states into equivalence classes. Moreover, as argued below, there are a finite number of such classes.

> **Lemma 3.1:** For every scheme and every initial state, there are a finite number of equivalence classes of derived states.
>
> **Proof:** By examining the range of the flow function, it is evident there are at most $|SUB^{O}|^{2} * |\text{power-set}(T \times R)|$ distinct equivalence classes of derived states. This is clearly a finite number. ∎

Consider any pair, g and h, of derived states established by the transition sequences G and H respectively. By definition, H is a transition sequence which can be applied to the initial state. However, H can also be applied to the state g. The fact that the state transitions of G have occurred is of no consequence as far as the legality of the state transitions in H are concerned. This property is crucial for establishing the existence of maximal states and is formalized below.

> **Lemma 3.2:** Given a scheme and an initial state, then for every pair of derived states, g and h, there exists a derived state n such that
>
> $$(\forall \langle A,B\rangle \in SUB^{O} \times SUB^{O})\,[\mathrm{flow}^{g}(A,B) \cup \mathrm{flow}^{h}(A,B) \subseteq \mathrm{flow}^{n}(A,B)]$$
>
> **Proof:** Let the derived states g and h be established by the transition sequences G and H respectively. By lemma 2.1 on page 50, we can assume, without any loss of generality, that G and H are in canonical form[4]. Construct a sequence N of state transitions, in canonical form, as follows[5]:
>
> 1. The create operations of N are the create operations of G followed by the create operations of H, except that any create operation of H which duplicates a create operation of G is eliminated.
>
> 2. The demand operations of N are the demand operations of G followed by the demand operations of H.
>
> 3. The transport operations of N are the transport operations of G followed by the transport operations of H.

[4] A transition sequence is in canonical form if all create operations precede all demand operations which in turn precede all transport operations.

[5] There are several alternate ways by which N can be constructed from G and H. Somewhat trivially, we can switch the roles of G and H in the construction. More significantly, we can interleave the operations of G and H in any manner as long as the relative order of the operations of G and of the operations of H is preserved. Indeed, there is no need for any of G, H or N to be in canonical form, except that it does make the statement of the proof a little easier.

We need to show that N is a transition sequence, i.e., every state transition in N is legal. The authorization for the create and demand operations is determined by the types and is independent of the protection state except for an existence requirement[6]. The legality of the create and demand operations of N follows immediately from the legality of these operations in G and in H. The transport operations of N consist of two parts:

1. The transport operations of G.
2. The transport operations of H.

Once the create and demand operations of G have occurred, the transport operations of G are legal. The fact that there are some additional create and demand operations does not affect the legality of these operations. Similarly, once the create and demand operations of H have occurred, the transport operations of H are legal. The fact that there are some additional create, demand, and transport operations does not affect the legality of these operations. Hence, all operations in N are legal.

Now, consider the state n established by the transition sequence N. It is evident, from the construction, that any path in state g from a subject A to a subject B is duplicated in state n. Similarly, any path in state h from a subject A to a subject B is duplicated in state n. The lemma follows immediately. ∎

It is a simple matter to extend this union property to any finite collection of derived states.

Lemma 3.3: Given a scheme and an initial state, then for every finite collection $\underline{S}$ of derived states there exists a derived state m such that

$$(\forall s \in \underline{S})\ (\forall \langle A,B\rangle \in SUB^{O} \times SUB^{O})\ [\mathrm{flow}^{s}(A,B) \subseteq \mathrm{flow}^{m}(A,B)]$$

Proof: The proof is by induction on the size of $\underline{S}$. For the basis case let $\underline{S}$ be empty. The lemma is then trivially true, since the initial state meets the requirements for the state m. Assume, as an induction hypothesis, that the lemma is true for all $\underline{S}$ of size k. Consider the case where the size of $\underline{S}$ is k+1. Then, $\underline{S}$ can be expressed as

$$\underline{S} = \underline{S}' \cup \{h\}$$

where $\underline{S}'$ is of size k. By induction hypothesis, there exists a state n which satisfies the requirements of the lemma for the set $\underline{S}'$. But then, by applying lemma 3.2 to the states n and h, there exists a state m which satisfies the requirements of the lemma for the set $\underline{S}$. ∎

[6] That is, if A creates B then A must exist; and if A demands B/x:c then both A and B must exist.

We are now ready to prove the existence of maximal states.

Theorem 3.4: For every scheme and every initial state, there exists a maximal state; i.e., there exists a derived state m such that for every derived state h

$$(\forall \langle A,B \rangle \in SUB^{O} \times SUB^{O})\,[flow^{h}(A,B) \subseteq flow^{m}(A,B)]$$

Proof: By lemma 3.1, we know there are a finite number of equivalence classes of derived states. Let $\underline{S}$ be a collection of derived states such that $\underline{S}$ contains exactly one member of each equivalence class. Clearly $\underline{S}$ is finite. By lemma 3.3, there exists a derived state m such that

$$(\forall s \in \underline{S})\,(\forall \langle A,B \rangle \in SUB^{O} \times SUB^{O})\,[flow^{s}(A,B) \subseteq flow^{m}(A,B)]$$

By definition of $\underline{S}$, for every derived state h there is a member of $\underline{S}$ which is equivalent to h. The theorem follows from this fact and the above equation. ∎

We saw in section 3.2 that the flow* function is sensitive to the create operation. The example we presented was carefully designed so the that the maximum flow achievable under the constraints of the *dfl* function was quickly arrived at. Clearly this situation does not qualify as a generally applicable termination condition for computing the flow* function. The existence of a create operation allows the set of subjects to evolve in an unbounded manner. In our research we have not been able to formulate a suitable termination condition to determine whether creating more subjects might or might not affect the flow function. Our conjecture is that the flow* function is computable, but for now this is an open question.

Due to our inability to compute the flow* function, we introduce the following notion.

Definition 3.7: For a given scheme and a given initial state, we say that a function

$$flow^{\dagger}\colon SUB^{O} \times SUB^{O} \longrightarrow \textbf{power-set}\,(T \times R)$$

is an **(upper) bound on the flow function** if

$$(\forall \langle A,B \rangle \in SUB^{O} \times SUB^{O})\,[flow^{*}(A,B) \subseteq flow^{\dagger}(A,B)]$$

We say that a bound is **realizable** if it is identical to the flow* function. ∎

It is useful to introduce the following notion as a theoretical device in obtaining a bound on the flow function.

Definition 3.8: For a given scheme and a given initial state, define the associated **maximal flow function without creates**

$$\texttt{flow}^{\#}\texttt{:}\ \texttt{SUB}^{O} \times \texttt{SUB}^{O} \longrightarrow \textbf{power-set}\,(\mathbf{T} \times \mathbf{R})$$

to be the flow* function which results from the same initial state, but with the given scheme modified by setting *can-create* = ϕ while leaving all other parameters unchanged. ∎

Since the set of subjects cannot change, the flow$^{\#}$ function can be computed in a straightforward manner. This computation is described in appendix C.4, where it is shown that the cost is no worse than $O(|SUB^{O}|^{5})$.

It follows immediately from the definitions that for every pair of subjects, A and B, which were present in the initial state

$$\texttt{flow}^{O}\texttt{(A,B)} \subseteq \texttt{flow}^{\#}\texttt{(A,B)} \subseteq \texttt{flow}^{*}\texttt{(A,B)} \subseteq \texttt{flow}^{\dagger}\texttt{(A,B)}$$

This sequence of inclusions leads us to three different approaches to the flow-analysis problem, briefly sketched out below.

In this chapter we investigate two different techniques for obtaining a bound on the flow function. Due to the role of the *dfl* function, it is possible to obtain a bound on the flow function without considering the structure of a given initial state. We derive this bound in section 3.4. Such a bound is useful, since it can be computed quickly and applies to every initial state. However, since the structure of the initial state is not taken into account, this bound may be too permissive. In section 3.5, we develop a technique for computing and refining a bound on the flow function, which takes into consideration the given initial state . The basic idea is to reduce this problem to computing the flow$^{\#}$ function. In order to do so, we augment the initial state by introducing a finite number of new subjects. We show that the flow$^{\#}$ function which results from such an augmented initial state provides the desired bound. Our construction demonstrates that the impact of created subjects can be accounted for by introducing a finite number of subjects in an appropriate manner.

In chapter 4, we discuss the notion of **create-invariant schemes**. These schemes are defined by the requirement that for all initial states

$$(\forall\texttt{<A,B>} \in \texttt{SUB}^{O} \times \texttt{SUB}^{O})\,[\texttt{flow}^{*}\texttt{(A,B)} = \texttt{flow}^{\#}\texttt{(A,B)}]$$

For such schemes, computation of the flow* function reduces to a finite problem. Our analysis provides guidelines by which a designer can guarantee this property.

In chapter 5, we discuss the notion of **flow-invariant schemes**. These schemes are defined by the requirement that for all initial states

$$(\forall\langle A,B\rangle \in SUB^{O} \times SUB^{O})\,[\mathrm{flow}^{*}(A,B) = \mathrm{flow}^{O}(A,B)]$$

For such schemes, the initial authorization for transport of tickets can never change. Once again, computation of the flow* function reduces to a finite problem and our analysis provides guidelines by which a designer can guarantee this property.

3.4. THE INDIRECT FLOW LIMIT FUNCTION

The *dfl* function imposes a limit on the types of tickets which can be directly transported between two subjects. It also implicitly limits the types of tickets which can be indirectly transported. In this section we derive the indirect flow limit function, denoted as *ifl*, to express the limit on both indirect as well as direct transport of ticket types.

We first define the mediated flow limit function, denoted as *mfl*. This function takes into account the actual path which might be involved in an indirect transport of tickets. For our immediate purpose, it serves as a convenient notation which assists in expressing a formal definition of the *ifl* function. However, the *mfl* function will be a useful tool in the analysis of chapters 5 and 6.

Consider an example where we denote subjects of type *a* by A_i and subjects of type *b* as B_i. The situation below depicts a path from subject A_1 to subject B_1 via the subjects A_2, B_2 and A_3.

$$\begin{tikzcd} A_1 \arrow[r] & A_2 \arrow[r] & B_2 \arrow[r] & A_3 \arrow[r] & B_1 \end{tikzcd}$$

There is authorization to transport tickets of type *c*/x:c from the domain of A_1 to the domain of B_1 over this path if and only if

1. Tickets of type *c*/xc can be moved from dom(A_1) to dom(A_3).
2. Tickets of type *c*/x:c can be moved from dom(A_3) to dom(B_1).

The first condition can be written in terms of the *dfl* function as

$$c/\mathrm{xc} \in dfl(a,a) \cap dfl(a,b) \cap dfl(b,a)$$

The second condition is simply that

$$c/\mathrm{x{:}c} \in dfl(a,b)$$

Since the specific identity of the subjects involved in the path is not relevant, there is exactly the same authorization for transport of ticket types over the following path from A_1 to B_1

$$\begin{tikzcd} A_1 \arrow[r] & A_4 \arrow[r] & B_3 \arrow[r] & A_5 \arrow[r] & B_1 \end{tikzcd}$$

What is relevant is the type of the subjects in the path.

In order to avoid writing out a long sequence of intersections, we will introduce the *mfl* function so that, for the above example,

$$c/x{:}c \in mfl\,(a, aba, b)$$
$$\Longleftrightarrow$$
$$[c/xc \in dfl\,(a,a) \;\cap\; dfl\,(a,b) \;\cap\; dfl\,(b,a)\,] \;\wedge\; [c/x{:}c \in dfl\,(a,b)\,]$$

The *mfl* function takes three arguments. The first argument is the type of the subject at the source of the path. The third argument is the type of the subject at the destination of the path. The second argument is a string which represents the types of the subjects encountered while traversing the path from the source to the destination. The special case when the second argument is the empty string, quite naturally, accounts for the case where a path consists of a single link. By definition, the *dfl* function specifies the authorization for transport of ticket types over a single link. Hence,

$$mfl\,(a, \Lambda, b) = dfl\,(a,b)$$

where Λ denotes the empty string.

Let T_S^* denote the Kleene closure of T_S; and lower case Greek letters, such as α and β, denote strings from the set T_S^*. The *mfl* function is then defined recursively as follows.

Definition 3.9: For every *dfl* function, define the associated **mediated flow limit function**

$$mfl\colon \mathrm{T_S} \times \mathrm{T_S^*} \times \mathrm{T_S} \rightarrow \textbf{power-set}\,(\mathrm{T} \times \mathrm{R})$$

by

1. For the empty string Λ

$$mfl\,(a,\Lambda,b) = dfl\,(a,b)$$

2. For a string of the form αd

$$c/\mathrm{x{:}c} \in mfl\,(a,\alpha d,b)$$
$$\Leftrightarrow$$
$$[c/\mathrm{xc} \in mfl\,(a,\alpha,d)] \wedge [c/\mathrm{x{:}c} \in dfl\,(d,b)]$$

■

The *mfl* function expresses the authorization for transport of ticket types over a path consisting of a particular sequence of types of subjects. The limit on transport of tickets over any combination of paths is then captured by the following definition.

Definition 3.10: For every *dfl* function, define the associated **indirect flow limit function**

$$ifl\colon \mathrm{T_S} \times \mathrm{T_S} \rightarrow \textbf{power-set}\,(\mathrm{T} \times \mathrm{R})$$

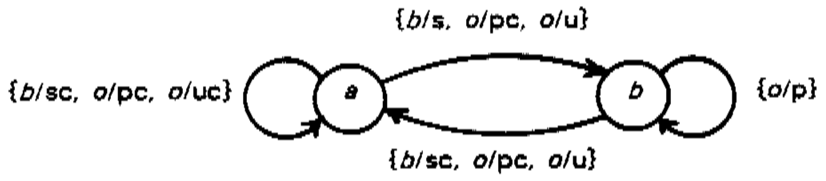
by

$$c/\mathrm{x{:}c} \in ifl\,(a,b) \Leftrightarrow (\exists\alpha)\,[c/\mathrm{x{:}c} \in mfl\,(a,\alpha,b)]$$

■

Thus $c/\mathrm{x{:}c} \in ifl(a,b)$ if and only if it is possible for a ticket of type $c/\mathrm{x{:}c}$ to be transported, either directly or indirectly, from a subject of type *a* to a subject of type *b*.

For the sample *dfl* function, whose graph is shown below



the associated *ifl* function is

$$\begin{aligned} ifl(a,a) &= \{b/\mathrm{sc},\ o/\mathrm{pc},\ o/\mathrm{uc}\} \\ ifl(a,b) &= \{b/\mathrm{s},\ o/\mathrm{pc},\ o/\mathrm{u}\} \\ ifl(b,a) &= \{b/\mathrm{sc},\ o/\mathrm{pc},\ o/\mathrm{u}\} \\ ifl(b,b) &= \{b/\mathrm{s},\ o/\mathrm{pc}\} \end{aligned}$$

Now consider two subjects B_1 and B_2 both of type *b*. The following facts can be deduced from the value of *ifl*(*b*,*b*).

1. It might be possible to transport a ticket of type *o*/pc from B_1 to B_2 even though this transport cannot be directly from B_1 to B_2.

2. A ticket of type *b*/sc can never be transported from B_1 to B_2.

3. It might be possible to transport a ticket of type *b*/s from B_1 to B_2 even though this transport cannot be directly from B_1 to B_2.

It follows immediately from the definition that the values of the flow function are constrained by the *ifl* function as stated below.

Theorem 3.5: For every scheme and every pair of distinct subjects A and B, in every state k

$$\mathrm{flow}^k(A,B) \subseteq ifl(t(A),t(B))$$

■

The *ifl* function then provides an upper bound on the flow function independent of the particular initial state. As discussed in appendix C.2, the *ifl* function can be computed from the *dfl* function in a straightforward manner with a cost of $O(|T \times R| * |T_S|^3)$.

3.5. THE AUGMENTED FLOW FUNCTION

In this section we develop a technique for computing an upper bound on the flow function accounting for the structure of a particular initial state.

Let us refer to subjects not present in the initial state as created subjects. Every created subject has a unique creator. Thus, for every created subject there is some unique ancestor which was present in the initial state. For a created subject A we call this unique ancestor the originator of A, written as org(A). For those subjects A which were present in the initial state, we define org(A) to be A itself. The org function is formally defined below.

Definition 3.11: For every derived state k, define the associated **originator**[7] function

$$\text{org}: SUB^k \longrightarrow SUB^o$$

as follows

1. If $A \in SUB^o$ then $\text{org}(A) = A$.
2. If B was created by A then $\text{org}(B) = \text{org}(A)$.

A subject B such that $\text{org}(B) \neq B$ is called a **created subject**. A created subject B such that $\text{org}(B) = A$ is called a **descendant** of A. ■

Every subject can potentially spawn an unbounded number of descendants. We will simulate the impact of this unbounded set of subjects by augmenting the initial state. Specifically, we will introduce a finite number of new subjects in such a way that the actions of any descendant can be simulated by the actions of one of the subjects in the augmented initial state. We say that these subjects act as surrogates for all possible descendants.

We illustrate this construction by means of an example, before defining it formally. Consider a scheme with four subject types so that

$$\mathbf{T}_S = \{a,b,c,d\}$$

Let the *can-create* relation be

$$\textit{can-create} = \{\langle a,b\rangle,\ \langle b,a\rangle,\ \langle b,c\rangle,\ \langle c,c\rangle,\ \langle d,d\rangle\}$$

The transitive closure of this *can-create* relation is the set

$$\{\langle a,a\rangle,\ \langle a,b\rangle,\ \langle a,c\rangle,\ \langle b,a\rangle,\ \langle b,b\rangle,\ \langle b,c\rangle,\ \langle c,c\rangle,\ \langle d,d\rangle\}$$

Now consider any subject A of type *a* present in the initial state. The pairs in the transitive closure of the *can-create* relation with *a* as the first element are

$$\langle a,a\rangle,\ \langle a,b\rangle,\ \langle a,c\rangle$$

Hence, any descendant of A will be of type *a*, *b*, or *c*. We will augment the initial state by introducing two new subjects, say B and C, of type *b* and *c* respectively. The idea is that

- The presence of any type *a* descendant of A will be simulated by the presence of subject A itself.

[7] Since the org function is associated with a particular state, the astute reader may wonder why we have not indicated this by an explicit superscript. We have omitted doing so, because of the nature of this function. Specifically, once a subject A has been created, org(A) is defined and remains constant thereafter.

- The presence of any type *b* descendant of A will be simulated by the presence of subject B.
- The presence of any type *c* descendant of A will be simulated by the presence of subject C.

Due to their intended role we say that the subjects A, B, and C are, respectively, the *a*, *b*, and *c* surrogates of A. In order to achieve the desired effect, we distribute transport tickets for the subjects A, B, and C as determined by the following rule.

For every pair of subjects

$$\langle X, Y \rangle \in \{A, B, C\} \times \{A, B, C\}$$

such that

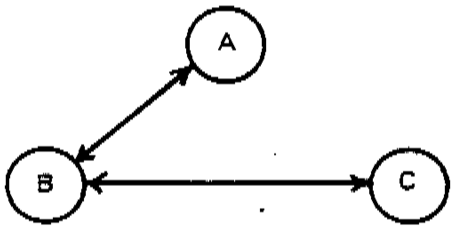
$$\langle t(X), t(Y) \rangle \in \textit{can-create}$$

introduce all tickets required by the create-rule by pretending that subject X creates subject Y. The pairs which satisfy this requirement are

$$\langle A,B\rangle,\ \langle B,A\rangle,\ \langle B,C\rangle,\ \langle C,C\rangle$$

Observe, in particular, occurrence of the pair <C,C> in the above rule. Now, of course, C cannot create itself. However, we do introduce all tickets required by the *<c,c>* create-rule in the domain of C itself.

Now, consider specifically the create-rule of self-copy creates. The tickets distributed would then result in the following links



These links would be established by tickets without the copy flag. Additionally the self-reference tickets would be given to each of these subjects so that

$$A/sc,\ A/rc \in \text{dom}(A)$$
$$B/sc,\ B/rc \in \text{dom}(B)$$
$$C/sc,\ C/rc \in \text{dom}(C)$$

We refer to this construction as appending surrogates to the subject A. The augmented initial state is constructed by appending surrogates to every subject present in the initial state. We now define these terms formally.

Definition 3.12: By the term **append surrogates** to subject A, we mean that a given state is augmented as follows.

1. Denote subject A as A_0 and its type *t*(A) as a_0.
2. For every subject type $a_i \neq a_0$, such that $<a_0,a_i>$ is in the transitive closure of *can-create*, introduce a new subject A_i of type a_i.
3. Refer to A_i as the a_i **surrogate** of A. Let the a_0 surrogate of A be the subject A itself.
4. For every pair $<A_i,A_j>$ of surrogates of A, such that $<a_i,a_j>$ is in *can-create*, place all tickets required by the $<a_i,a_j>$ create-rule by pretending that subject A_i creates subject A_j.

■

Definition 3.13: The **augmented initial state** is obtained by appending surrogates to every subject in the initial state. We denote the set of subjects in the augmented initial state by SUB^{aug}. ■

Next, we define a mapping from the set of subjects in any derived state to the set of subjects in the augmented initial state.

Definition 3.14: For every derived state k, define the **surrogate**[8] function

$$\text{surr: } SUB^k \longrightarrow SUB^{aug}$$

by

$$\text{surr}(A) = t(A) \text{ surrogate of org}(A)$$

■

In particular, observe that

1. Every subject present in the initial state is mapped to itself.
2. Two created subjects B_1 and B_2 both of type *b*, such that $org(B_1) = org(B_2)$, are mapped to the same surrogate subject in the augmented initial state.
3. The surrogate function preserves types, in the sense that

[8] See the footnote on page 73, in the context of the org function, for an explanation as to why the surr function is not defined with an explicit superscript to identify the state.

$$t(\mathrm{surr}(A)) = t(A)$$

We are now ready to prove that the augmented initial state serves as a basis for computing an upper bound on the flow function. We need one final definition.

Definition 3.15: For a given scheme and initial state, define the associated **augmented flow function**

$$\mathrm{flow}^{\dagger}: \mathrm{SUB}^{aug} \times \mathrm{SUB}^{aug} \rightarrow \textbf{power-set}(T \times R)$$

to be the flow$^{\#}$ function (i.e. the maximal flow function without creates) for the same scheme and the augmented initial state. ■

The following theorem establishes that the flow† function is an upper bound on the flow function. The proof demonstrates that the impact of any subject A can be simulated by the presence of surr(A).

Theorem 3.6: For every scheme and every initial state, it is the case that for every derived state h

$$(\forall\langle A,B\rangle \in \mathrm{SUB}^{h} \times \mathrm{SUB}^{h})[\mathrm{flow}^{h}(A,B) \subseteq \mathrm{flow}^{\dagger}(\mathrm{surr}(A),\mathrm{surr}(B))]$$

Proof: Consider any transition sequence H which applied to the initial state results in state h. We will modify[9] H to obtain a transition sequence G, with no creates, such that G applied to the augmented initial state establishes state g with the property that

$$\mathrm{flow}^{h}(A,B) \subseteq \mathrm{flow}^{g}(\mathrm{surr}(A),\mathrm{surr}(B))$$

Since our construction of G is for an arbitrary H, we will have established the desired result. It will be convenient for us to assume that H is in canonical form[10] and, by lemma 2.1 on page 50, we can do so without any loss of generality. G is obtained from H as follows.

1. Ignore all create operations in H.
2. Replace all demand operations in H as follows.

```
            A demands B/x:c in H
                  becomes
      surr(A) demands surr(B)/x:c in G
```

[9]The idea behind this construction was first used by Minsky [15] in the context of the uniform send-receive scheme.

[10]A transition sequence H is in canonical form if all create operations precede all demand operations which in turn precede all transport operations.

4. Replace all transport operations as follows.

move A/x:c from dom(B) to dom(C) in H
becomes
move surr(A)/x:c from dom(surr(B)) to dom(surr(C)) in G

In the first part of the proof, we will establish the following facts.

1. G is a transition sequence[11]
2. $A/x{:}c \in dom^h(B) \rightarrow surr(A)/x{:}c \in dom^g(surr(B))$

It follows immediately from assertion 2 that

$$link^h(A,B) \rightarrow link^g(surr(A), surr(B))$$

We will use this consequence of assertion 2 in the second part of the proof. The two assertions above are proved by induction on the number of transport operations in H.

Basis Case: For the basis case let the number of transport operations in H be 0. The sequence H then consists of a sequence of create operations followed by a sequence of demand operations, while G consists of a sequence of demand operations.

Assertion 1: In the construction of G the replacement for the demand operations is that

A demands B/x:c in H
becomes
surr(A) demands surr(B)/x:c in G

The authorization for demand operations is in terms of subject types and is independent of the particular state, except for the requirement that the subjects must exist[12]. By definition, the surr function preserves the subject type. By construction, surr(A) and surr(B) exist in the augmented initial state. But then, since the demand operation in H is legal the corresponding demand operation in G is also legal.

Assertion 2: Since there are no transport operations, there are only three ways by which the ticket A/x:c can appear in $dom^h(B)$.

Case (i): The ticket A/x:c is in the initial domain of B. This is possible only if A and B are in the initial set of subjects. But then, surr(A) = A and surr(B) = B, and assertion 2 becomes

[11]That is, every state transition in G is legal.

[12]That is, if A demands B/x:c in state k then both A and B must exist in state k for the operation to be meaningful.

$$A/x{:}c \in \mathrm{dom}^{o}(B) \rightarrow A/x{:}c \in \mathrm{dom}^{g}(B)$$

This is trivially true.

Case (ii): The ticket A/x:c appears in the domain of B due to a create operation. But then, by construction of the augmented initial state, the ticket surr(A)/x:c is placed in the domain of surr(B).

Case (iii): The only other way for the ticket A/x:c to appear in $\mathrm{dom}^{h}(B)$ is via a demand operation. But then, the corresponding demand operation in G ensures that assertion 2 is true.

Thus, for the basis case where H has no transport operations, both assertions are true.

Induction Step: Assume, as an induction hypothesis, that the assertions are true for every transition sequence H with k transport operations. Consider the case of a transition sequence H with k+1 transport operations. Then, H consists of an initial sequence H' with k transport operations followed by a single transport operation. Let the state established by the transition sequence H' be denoted as hk. Let G' be the required modification of H', and let the state established by G' be denoted as gk. By induction hypothesis G' is a transition sequence. Let the k+1st transport operation of H be

$$\text{move } A/x{:}c \text{ from } \mathrm{dom}(B) \text{ to } \mathrm{dom}(C)$$

Assertion 1: In order for the k+1st transport operation of H to be legal, we must have

1. $A/xc \in \mathrm{dom}^{hk}(B)$
2. $\mathrm{link}^{hk}(B,C)$
3. $t(A)/x{:}c \in dfl(t(B),t(C))$

By induction hypothesis 2, we then have

$$A/xc \in \mathrm{dom}^{hk}(B) \rightarrow \mathrm{surr}(A)/xc \in \mathrm{dom}^{gk}(\mathrm{surr}(B))$$
$$\mathrm{link}^{hk}(B,C) \rightarrow \mathrm{link}^{gk}(\mathrm{surr}(B),\mathrm{surr}(C))$$

Since the surr function preserves types, we have

$$t(A)/x{:}c \in dfl(t(B),t(C)) \rightarrow t(\mathrm{surr}(A))/x{:}c \in dfl(t(\mathrm{surr}(B)),t(\mathrm{surr}(C)))$$

But then, the three conditions required to authorize the corresponding transport operation in G are true.

Assertion 2: The state h differs from the state hk at most by

$$A/x{:}c \in dom^h(C)$$

But then, the corresponding transport operation in G ensures that

$$surr(A)/x{:}c \in dom^g(surr(C))$$

This completes the induction step and we have established both assertions.

It now remains to show that for every derived state h

$$(\forall \langle A,B\rangle \in SUB^h \times SUB^h)\,[flow^h(A,B) \subseteq flow^\dagger(surr(A),surr(B))]$$

Let $path^h$ and $path^g$, respectively, denote a path in state h and in state g. Given any $path^h$ from A to B, we will show there is a $path^g$ from surr(A) to surr(B) with at least the same authorization for transport of ticket types as the $path^h$ from A to B. The proof is by induction on the number of links in the $path^h$ from A to B. For the basis case, consider a $path^h$ from A to B of length 1, i.e, the path consists of a single link. By assertion 2, we have

$$link^h(A,B) \rightarrow link^g(surr(A),surr(B))$$

Since the surr function preserves types, the *dfl* function authorizes exactly the same transport of ticket types for both these links and the basis case is true. Assume that 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 subject C, such that there is $path^h$ from A to C of length k and $link^h(C,B)$. By induction hypothesis, there is a $path^g$ from surr(A) to surr(C) with at least the same authorization for transport of ticket types as the $path^h$ from A to C. By assertion 2, there is a $link^g$ from surr(C) to surr(B). Again, since the surr function preserves subject types, the authorization for transport of ticket types across this $link^g$ is the same as that for $link^h(C,B)$. But then, there is a $path^g$ from surr(A) to surr(B) with the same authorization for transport of ticket types as the $path^h$ from A to B. This establishes the induction step and concludes the proof of the theorem. ■

In appendix C.5 we show that the cost of computing the $flow^\dagger$ function is no worse than $O(|T_S|^5 * |SUB^0|^5)$. This computation may clearly be expensive in the worst case, but nevertheless it is tractable. We now consider an example to illustrate several aspects about this technique for computing a bound on the flow function. Specifically,

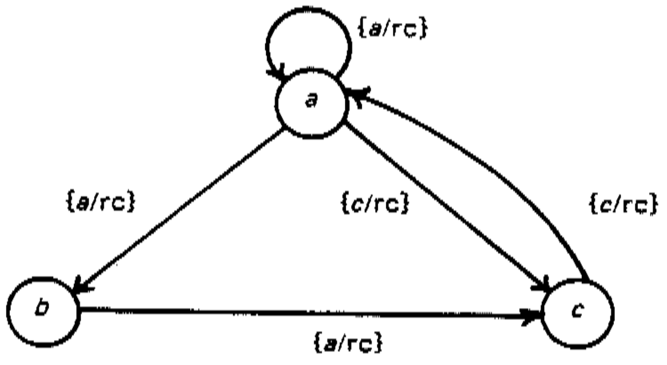
1. We show that the $flow^\dagger$ function provides a sharper bound than the *ifl* function.

2. We show that the flow† function may not be realizable and, hence, the bound may be too permissive.

3. Finally, we demonstrate that a sharper bound can be obtained by modifying the construction of the augmented initial state.

The scheme for our example has the *dfl* function shown below.

	a	*b*	*c*
a	{*a*/rc}	{*a*/rc}	{*c*/rc}
b	ϕ	ϕ	{*a*/rc}
c	{*c*/rc}	ϕ	ϕ



The Dfl Function

Let the authorization for the demand operation be

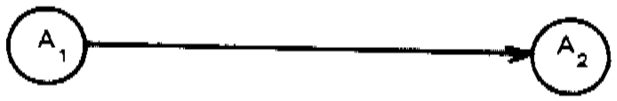
$$\begin{aligned} demand\,(a) &= \{a/\mathrm{s},\ b/\mathrm{s},\ c/\mathrm{s}\} \\ demand\,(b) &= \{a/\mathrm{s},\ b/\mathrm{s},\ c/\mathrm{s}\} \\ demand\,(c) &= \{a/\mathrm{s},\ b/\mathrm{s},\ c/\mathrm{s}\} \end{aligned}$$

Then, every subject can obtain a send ticket for any subject by demanding it and the transport operation is effectively controlled by the distribution of receive tickets. Let the authorization for the create operation be

$$can\text{-}create = \{a,b,c\} \times \{a,b,c\}$$

so that every subject is authorized to create subjects of every type. Finally, let the semantics of the create operation be as specified by the self-copy create rule (see definition 2.11 on page 43), and assume that the intial state is constrained to be a balanced self-copy state (see page 46).

Consider the initial state below with two subjects A_1 and A_2, both of type *a*, such that there is a link from A_1 to A_2 but not vice versa.

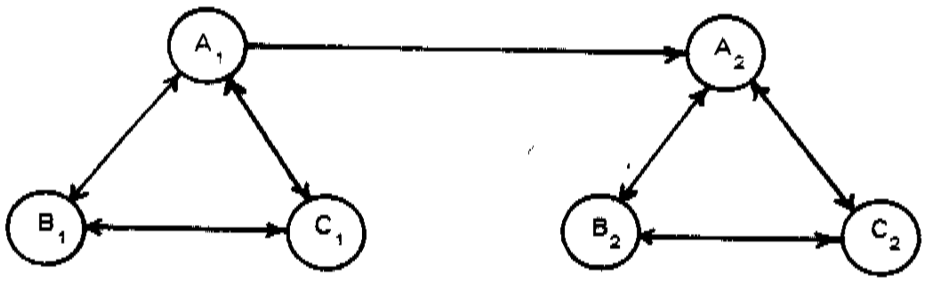


The Initial State

Since there is only one link, the $flow^0$ function is trivial to compute and we have the following values

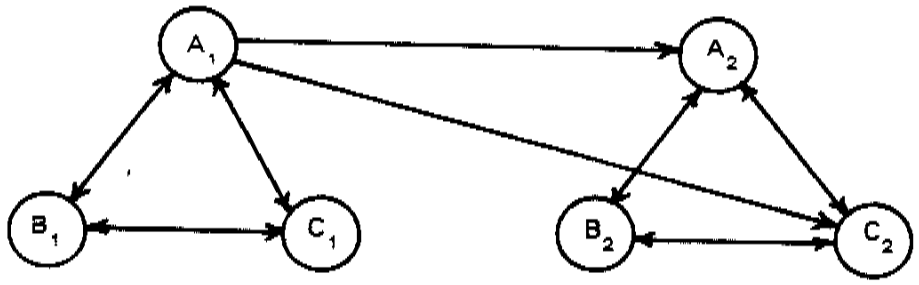
$$flow^0(A_1,A_2) = \{a/rc\}$$
$$flow^0(A_2,A_1) = \phi$$

The augmented initial state is obtained by appending surrogates to A_1 and A_2 resulting in



The Augmented Initial State

The $flow^+$ function can then be computed by the algorithm developed in appendix C.4 or simply by constructing all possible links by inspection. In particular, the self-reference ticket A_1/rc can be moved from the domain of A_1 to the domain of C_2 by using the path via A_2 and B_2. A_1 can acquire the ticket C_2/s by a demand operation, hence, establishing link(A_1,C_2). This results in the following state.



It turns out that the other links which can be established are irrelevant to computing the flow† function with respect to A_1 and A_2. These links are link(A_1,B_2), link(C_1,A_2), and link(C_1,C_2). None of these links establishes a path from A_2 to A_1. Hence, there is still no flow from A_2 to A_1. The path from A_1 to A_2 via C_2 can be used to transport tickets of type *c*/rc from A_1 to A_2. The flow† function is then as follows.

$$\text{flow}^\dagger(A_1,A_2) = \{a/rc,\ c/rc\}$$
$$\text{flow}^\dagger(A_2,A_1) = \phi$$

By inspection of the *dfl* graph on page 80, it is evident that

$$ifl(a,a) = \{a/rc,\ c/rc\}$$

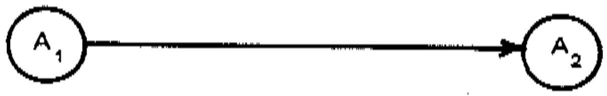
However, the flow† function specifies an empty flow from A_2 to A_1 and, hence, provides a better bound than the *ifl* function.

On the other hand, the flow† function allows for a flow of *a*/rc and *c*/rc tickets from A_1 to A_2. The flow of *a*/rc tickets was already authorized in the initial state, but it is not clear whether the flow of *c*/rc tickets from A_1 to A_2 can actually be attained. Indeed, we will shortly show that the latter flow cannot be achieved and, hence, the flow† function is too permissive. In doing so, we demonstrate a general technique which can be used to obtain a sharper bound on the flow function than provided by the flow† function.

Let us reconsider the motivation for appending surrogates to every subject in the initial state. By appending surrogates to a particular subject A, the idea was to be able to simulate all possible subjects that A might originate by mapping such subjects to one of the surrogates of A. Now the subjects originated by A, form a tree with A at the root and each created subject as a child of its creator. The mapping defined by the surrogate function treats subjects at all levels of this tree uniformly. Since there is no bound on the depth of such a tree, at some point we need to treat all levels uniformly. However, we might be able to improve our bound if we treat the first few levels of the tree differently.

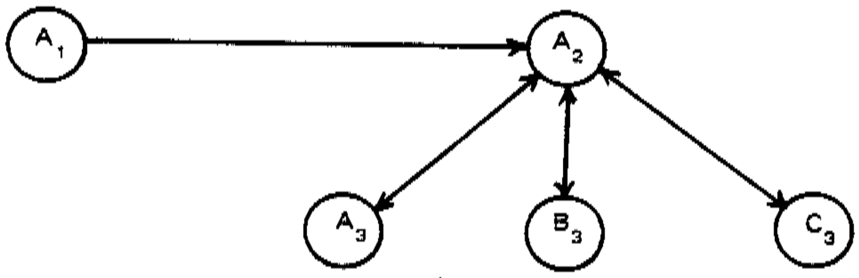
In particular, consider the first level of the tree. The subjects at this level are all directly created by A. Assume that the *can-create* relation authorizes A to create subjects of type *b*. Now, consider that A creates two subjects B_1 and B_2 both of type *b*. Since the parameters of any scheme are defined in terms of types, it is intuitively reasonable to expect that any impact that the presence of B_2 may have on the flow* function can as well be achieved by the presence of B_1. Indeed, the presence of B_2 can be simulated[13] simply by pretending that the subject B_1 is actually two subjects B_1 and B_2. By extension of this argument, B_1 can simulate the presence of any number of direct descendants of A which are of type *b*.

We use this idea to introduce the notion of an **unfolded state**. We say that the subject A is **unfolded one level** if A creates one subject of type *b* for every subject type *b* that A is permitted to create. Going back to our example initial state, reproduced below



The Initial State

we can unfold A_2 one level by assuming that A_2 creates subjects A_3, B_3, and C_3 of types *a*, *b*, and *c* respectively. This results in the following state



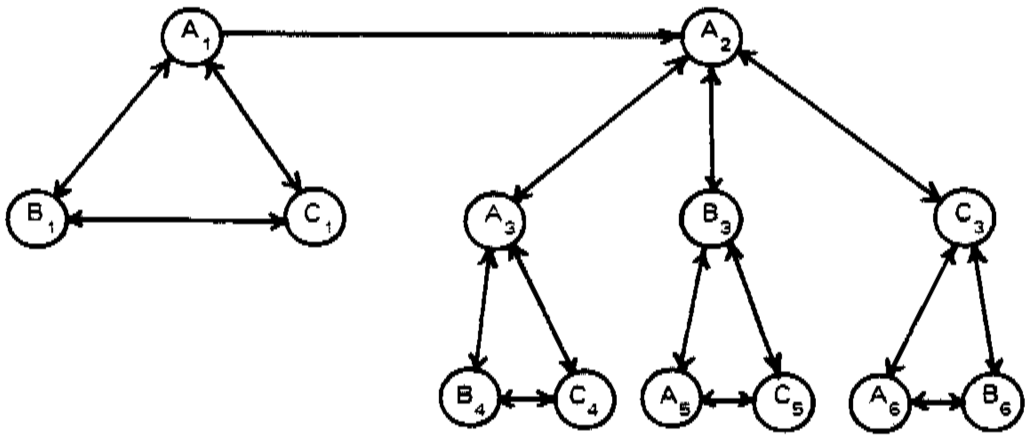
An Unfolded Initial State

We can now apply the technique of appending surrogates to this unfolded initial state. However, there is no need[14] to append surrogates to A_2, because the direct creates of A_2 are already accounted for by the unfolding of A_2 and the indirect creates will be accounted for by the surrogates of A_3, B_3, and C_3. This construction then results in the following state.

---

[13] A formal proof of this fact can be obtained by an induction similar to that used in the proof of theorem 3.6 on page 76.

[14] A formal proof of this fact can be obtained by an induction similar to that used in the proof of theorem 3.6 on page 76.



The Augmented Unfolded Initial State

As before, let us denote the flow$^{\#}$ function (i.e. the maximal flow function without creates) for this augmented initial state by flow†. We can compute this function by the technique discussed in appendix C.4 or, simply, by constructing all possible links by inspection. This set of links is somewhat messy to show and, perhaps, the reader will accept our claim[15] that

$$\text{flow}^{\dagger}(A_1, A_2) = \{a/rc\}$$
$$\text{flow}^{\dagger}(A_2, A_1) = \phi$$

By our earlier discussion, we know that the flow† function provides an upper bound on the flow function. In this particular case, the flow† function happens to be identical to the flow$^{\circ}$ function and, hence, provides an exact value for the flow* function.

For this example a single unfolding of one subject yielded an exact value for the flow* function. In general, this unfolding technique can be applied for several levels of the create tree and to several subjects in the initial state in an attempt to obtain a sharper bound.

Every unfolding increases the number of subjects in the unfolded initial state. This increase is reflected in the corresponding augmented unfolded initial state. At the same time, every unfolding may improve the bound on the flow function. There is

[15]The crucial point is that it is not possible to establish a link from A_1 to C_3.

an obvious tradeoff involved here. An important open question is to investigate the nature of this tradeoff, as well as to develop a systematic method for unfolding which progressively improves the bound.

We conclude this section by returning to the question of how the safety problem of Harrison, Russo, and Ullman [6] can be reduced to questions about the $\text{flow}^{\dagger}$ function. This problem poses the following question: Is it possible for subject A to acquire the ticket B/x:c? We observed, on page 64, that under the assumption that the only copiable tickets B/xc are those introduced in the initial state, this question can be formulated in terms of the flow^{*} function, as follows: Does there exist a subject C, who possesses the ticket B/xc in the initial state and t(B)/x:c is in flow^{*}(C,A)? In a more general situation, copiable tickets can also be obtained by a demand operation. At first thought, it seems that we can account for this simply by translating the safety problem as follows.

> Does there exist a subject C, who either possesses the ticket B/xc in the initial state or who can obtain it by a demand operation, and t(B)/x:c is in flow^{*}(C,A)?

However, this formulation does not account for the possibility that a created subject might obtain the ticket B/xc by a demand operation. In order to account for this possibility, we need some characterization of the flow function for such subjects. Fortunately, we have such a characterization in theorem 3.6 which states that for every derived state h

$$(\forall \langle A,B\rangle \in SUB^{h} \times SUB^{h})\,[\text{flow}^{h}(A,B) \subseteq \text{flow}^{\dagger}(\text{surr}(A),\text{surr}(B))]$$

In line with our basic theme of types, it is reasonable to assume that the ability to obtain tickets by means of a demand operation is determined by the type of a given subject[16]. Modulo this assumption, the safety problem can be formulated as follows.

> Does there exist a subject C in an augmented unfolded initial state, who either possesses the ticket B/xc or can obtain it by a demand operation, and t(B)/x:c is in $\text{flow}^{\dagger}$(C,A)?

Of course, since the $\text{flow}^{\dagger}$ function may not be realizable, an affirmative answer to this question does not necessarily imply an affirmative answer to the corresponding safety problem. At the same time, a negative answer to this question does imply a negative answer to the corresponding safety problem.

[16] In our design framework, we have made this assumption with respect to demanding transport tickets. Here, we are extending it to cover all tickets.

3.6. SUMMARY

In section 3.1 we defined the flow^k function to express the authorization for transport of tickets from dom(A) to dom(B), in state k, accounting for indirect as well as direct transport. In section 3.2 we discussed an example to demonstrate how the flow function may evolve from a given initial state.

In section 3.3 we introduced the notion of a maximal state, which is a derived state in which the maximum possible value of the flow function, between subjects present in the initial state, has been attained. We showed that a maximal state exists for every scheme and every initial state. We then defined the flow* function to be the flow function in a maximal state, but with the domain restricted to the initial set of subjects. A protection scheme constrains the evolution of an initial state only to the extent determined by the flow* function. The create operation presents a major complication in computing this function, since the protection state can evolve in an unbounded manner. We were not able to arrive at an algorithm for computing the flow* function or demonstrate that it is not computable. We introduced the notion of a bound flow† on the flow function, by the requirement that

$$(\forall \langle A,B\rangle \in SUB^O \times SUB^O)\,[\text{flow}^*(A,B) \subseteq \text{flow}^\dagger(A,B)]$$

We also defined the flow$^{\#}$ function to be the flow* function which would result if the create operation was eliminated. Since the set of subjects does not change the flow$^{\#}$ function is easily computed.

Stated broadly, the flow-analysis problem is to determine properties of the flow function in derived states. It immediately follows from our definitions, that for every pair <A,B> of subjects present in the initial state

$$\text{flow}^O(A,B) \subseteq \text{flow}^{\#}(A,B) \subseteq \text{flow}^*(A,B) \subseteq \text{flow}^\dagger(A,B)$$

This sequence of inclusions leads us to three different approaches to the flow-analysis problem as follows.

1. Develop techniques for computing a bound on the flow function. Two such techniques, viz., the *///* function and the flow† function, were discussed in this chapter.

2. Investigate conditions under which the flow* function is identical to the

flow$^{\#}$ function. Schemes with this property are said to be create-invariant and are studied in chapter 4.

3. Investigate conditions under which the flow* function is identical to the flow^{O} function. Schemes with this property are said to be flow-invariant and are studied in chapter 5.

In section 3.4 we derived the *ifl* function which provides a bound on the flow function without considering the structure of a particular initial state. This function is defined in terms of the *dfl* function and can be computed with a cost of $O(|T \times R| * |T_S|^3)$.

In section 3.5 we developed a technique for computing a bound on the flow function taking into account the particular initial state. This technique augmented the initial state, by introducing a finite number of new subjects for every subject present in the initial state. We proved in theorem 3.6 that the flow$^{\#}$ function which results from an augmented initial state provides a bound on the flow function. This bound can be computed with a cost no worse than $O(|T_S|^5 * |SUB^O|^5)$. We then demonstrated by means of an example that

1. This technique provides a sharper bound than the *ifl* function.
2. The bound may not be realizable.
3. The bound can be improved by unfolding the initial state.

It is an open question to determine the tradeoffs involved in improving the bound in this manner.

CHAPTER 4
CREATE-INVARIANT SCHEMES

The major complication in computing the flow* function arises from the create operation. In the previous chapter we showed that the impact of any number of created subjects could be accounted for by augmenting the initial state by introduction of a finite number of surrogate subjects. This enabled us to compute an upper bound on the flow function.

In this chapter we consider another method for simulating the impact of created subjects. The idea is that every subject will be able to simulate the actions of its descendants. To this end we define the following notion of a create-invariant scheme.

Definition 4.1: A scheme is **create-invariant** if for every initial state it is the case that

$$(\forall\langle A,B\rangle \in SUB^O \times SUB^O)\,[\mathrm{flow}^*(A,B) = \mathrm{flow}^{\#}(A,B)]$$

■

For a create-invariant scheme computation of the flow* function reduces to a finite problem which has a straightforward solution with a cost no worse than $O(|SUB^O|^5)$.

Minsky [15] demonstrated that the uniform send-receive scheme is create-invariant[1]. In this chapter we will generalize this result. The example discussed in section 3.2 demonstrates that our framework admits schemes which are not create-invariant. Our strategy here, is to devise constraints on the initial state and the *can-create* relation which enable us to ignore the effect of created subjects.

[1]For the uniform scheme the create-invariant property amounts to requiring that path*(A,B) if and only if path$^{\#}$(A,B). See theorem 1.1 on page 26.

Let us first consider the initial state. As shown in lemma 1.2 on page 26, for a uniform scheme there is no loss of generality in assuming that every initial state satisfies the self-reference assumption, i.e.,

$$(\forall A \in SUB^{O})\,[A/sc \in dom^{O}(A) \ \wedge \ A/rc \in dom^{O}(A)]$$

This aspect is crucial in establishing the create-invariant nature of this scheme. In particular the self-reference tickets for a subject A are used to simulate tickets for any descendant of A.

In this chapter we will impose the self-reference assumption as a constraint on the initial state. If this assumption is not true then the the initial state can be augmented by introducing the self-reference tickets in the domain of every subject. The resulting flow* function will then provide an upper bound on the flow function rather than an exact value of the flow* function. The cost of computing this bound is no worse than $O(|SUB^{O}|^{5})$ in contrast to the possibility of an $O(|T_{S}|^{5} * |SUB^{O}|^{5})$ cost for computing a bound via the technique of section 3.5. Hence, the results of this chapter are useful even if the self-reference assumption is not true. Moreover, this technique for obtaining a bound can be used after one or more subjects in the initial state have been unfolded one or more levels, as discussed on page 83 in the previous chapter. Then the self-reference tickets need not be introduced in the domains of subjects which have been unfolded. Thus, the self-reference assumption allows us a quicker technique for obtaining and refining a bound provided the scheme is create-invariant under this assumption.

In section 4.1 we propose a definition of weak creates and show that any scheme for which the *can-create* relation is constrained in this manner is create-invariant for self-reference initial states. This definition of weak creates, at the least, allows every subject A to create subjects of the same type as A and, hence, always admits a non-empty *can-create* relation.

Our definition of weak creates has four conditions. In section 4.2 we demonstrate that dropping any one of these conditions allows for schemes which are not create-invariant. In this sense, the four conditions proposed in section 4.1 are all required. In section 4.3 we discuss a particular class of schemes, called hierarchical schemes, to demonstrate the kind of situation for which weak creates are useful.

4.1. SCHEMES WITH WEAK CREATES

It is appealing to consider a situation where a subject A is allowed to create a subject A' only if A' is "no more powerful than" subject A. If the notion of "no more powerful than" has been correctly formulated, any scheme constrained in this manner should be create-invariant; since presumably the impact of a subject's actions cannot be amplified by creation of new subjects. This is the basic intuition underlying the developments of this chapter. For stylistic simplicity we shall use the term "weaker" to refer to the idea of "no more powerful than" even though the simpler term misses the connotation of possible equivalence. The problem then, is to formulate an appropriate notion of "weaker". Our approach is to formalize this notion in terms of properties of the *dfl* and *demand* function.

First consider the *dfl* function. For a given subject type *a*, this function limits the tickets which can be transported to and from any subject of type *a*. Additionally, it also limits the manner in which tickets of type *a*/x:c can themselves be transported. Both these aspects must be accounted for in determining whether a subject of type *b* is deemed to be weaker than a subject of type *a*.

Now clearly the ability of a subject of type *b* to acquire tickets from a subject of type *c* is weaker than the ability of a subject of type *a* to do the same provided

$$dfl(c,b) \subseteq dfl(c,a)$$

Similarly, the ability of a subject of type *b* to dispense tickets to a subject of type *c* is weaker than the ability of a subject of type *a* to do the same provided

$$dfl(b,c) \subseteq dfl(a,c)$$

Then our first criterion for saying that subjects of type *b* are weaker than subjects of type *a* is as follows.

$$(\forall c)\ [dfl(b,c) \subseteq dfl(a,c) \wedge dfl(c,b) \subseteq dfl(c,a)]$$

We next account for the mobility of tickets of a type *a*/x:c. Specifically consider two limiting cases.

$$1.\ (\forall \langle c,d \rangle)\ [a/x{:}c \in dfl(c,d)]$$
$$2.\ (\forall \langle c,d \rangle)\ [a/x{:}c \notin dfl(c,d)]$$

In the first case tickets of the type *a*/x:c are universally mobile whereas in the second case such tickets are completely immobile. Now in the first case there is ample scope for establishing links to and from a subject of type *a*. In the latter case this scope is considerably diminished. It is then reasonable to say that the scope for establishing links with a subject of type a is weaker in the second case. Based on this idea, our next criterion for saying that a subject of type b is weaker than a subject of type a is that a ticket of type a/x:c is at least as mobile as a ticket of type b/x:c, i.e.,

$$(\forall\langle c,d\rangle)\,[b/\mathrm{x{:}c} \in dfl(c,d) \rightarrow a/\mathrm{x{:}c} \in dfl(c,d)]$$

We must also consider the demand operation while assessing the ability of a subject. Clearly, the ability of a subject of type b to obtain tickets on demand is weaker than the ability of a subject of type a to do so provided

$$demand(b) \subseteq demand(a)$$

For the same reason that we took into account the mobility of a subject type in the context of the *dfl* function, we must additionally consider that a subject of type c can obtain transport tickets for subjects of type b and type a on demand. The ability of a subject of type c to demand tickets for subjects of type b is weaker than its ability to demand tickets for subjects of type a provided

$$b/\mathrm{x{:}c} \in demand(c) \rightarrow a/\mathrm{x{:}c} \in demand(c)$$

We are then led to formulate the following notion of weaker.

Definition 4.2: The subject type b is said to be **weaker** than the subject type a if and only if

1. $(\forall c)\,[dfl(c,b) \subseteq dfl(c,a) \wedge dfl(b,c) \subseteq dfl(a,c)]$
2. $(\forall\langle c,d\rangle)\,[b/\mathrm{x{:}c} \in dfl(c,d) \rightarrow a/\mathrm{x{:}c} \in dfl(c,d)]$
3. $demand(b) \subseteq demand(a)$
4. $(\forall c)\,[b/\mathrm{x{:}c} \in demand(c) \rightarrow a/\mathrm{x{:}c} \in demand(c)]$

Subject B is said to be weaker than subject A if and only if t(B) is weaker than t(A). ■

Definition 4.3: For a given pair of *dfl* and *demand* functions, define the **weak** *can-create* relation by

$$\langle a,b\rangle \in can\text{-}create \leftrightarrow b \text{ is weaker than } a$$

We say that a scheme with a weak *can-create* relation has **weak creates.** ■

By inspection of these definitions, it is evident that every weak *can-create* relation is reflexive and transitive. The reflexive property ensures that a weak *can-create* relation will be non-empty and, hence, non-trivial. Thus a weak *can-create* relation will at least authorize every subject to create subjects of its own type. Indeed, any notion of weaker for which this is not true would be suspect, since the parameters which define a scheme are specified in terms of subject types. In section 4.3 we will discuss a weak *can-create* relation which includes pairs other than those that are reflexive.

In the remainder of this section we establish that any scheme with weak creates is create-invariant for self-reference initial states. The proof is by an induction, similar to that used to prove theorem 3.6 on page 76, where we showed that the flow† function provides an upper bound on the flow function. In the proof of theorem 3.6, the impact of any subject A was simulated by presence of the subject surr(A). In the present context, we will simulate the impact of any subject A by presence of the subject[2] org(A). In order to do so, we need the following result.

Lemma 4.1: For every scheme with weak creates it is the case that for every triple of subjects A, B, and C

1. $dfl(t(A), t(B)) \subseteq dfl(t(org(A)), t(org(B)))$

2. $t(C)/x{:}c \in dfl(t(A), t(B)) \rightarrow$
   $t(org(C))/x{:}c \in dfl(t(org(A)), t(org(B)))$

3. $t(A)/x{:}c \in demand(t(B)) \rightarrow$
   $t(org(A))/x{:}c \in demand(t(org(B)))$

Proof: It follows from the reflexive and transitive nature of a weak *can-create* relation that every subject is weaker than its originator, i.e., D is weaker than org(D). This fact is useful in proving the three assertions of the lemma.

Assertion 1: Since A is weaker than org(A), we have

$$(\forall c)\ [dfl(t(A), c) \subseteq dfl(t(org(A)), c)]$$

Since B is weaker than org(B), we have

$$(\forall d)\ [dfl(d, t(B)) \subseteq dfl(d, t(org(B)))]$$

In particular, then for c = t(B) and d = t(org(A)), the two equations above become

$$dfl(t(A), t(B)) \subseteq dfl(t(org(A)), t(B))$$
$$dfl(t(org(A)), t(B)) \subseteq dfl(t(org(A)), t(org(B)))$$

Assertion 1 follows by composing these two equations.

[2] Recall that the org function maps (1) each subject present in the initial state to itself, and (2) each created subject to its (unique) ancestor which was present in the initial state. See definition 3.11 on page 73.

Assertion 2: Since C is weaker than org(C), we have for every pair of subject types $<a,b>$ that

$$t(\mathrm{C})/\mathrm{x{:}c} \in dfl(a,b) \twoheadrightarrow t(\mathrm{org}(\mathrm{C}))/\mathrm{x{:}c} \in dfl(a,b)$$

Assertion 2 follows from this fact and assertion 1.

Assertion 3: Since A is weaker than org(A), we have

$$(\forall b)\,[t(\mathrm{A})/\mathrm{x{:}c} \in demand(b) \twoheadrightarrow t(\mathrm{org}(\mathrm{A}))/\mathrm{x{:}c} \in demand(b)]$$

In particular, then for $b = t(\mathrm{B})$ we have

$$t(\mathrm{A})/\mathrm{x{:}c} \in demand(t(\mathrm{B})) \twoheadrightarrow t(\mathrm{org}(\mathrm{A}))/\mathrm{x{:}c} \in demand(t(\mathrm{B}))$$

Since B is weaker than org(B), we have

$$demand(t(\mathrm{B})) \subseteq demand(t(\mathrm{org}(\mathrm{B})))$$

Assertion 3 follows from the two equations above. ∎

We are now ready to establish the central result of this chapter.

Theorem 4.2: Any scheme with weak creates and self-reference initial states is create-invariant, i.e., for every initial state

$$(\forall<\mathrm{A},\mathrm{B}> \in \mathrm{SUB}^{\mathrm{O}} \times \mathrm{SUB}^{\mathrm{O}})\,[\mathrm{flow}^{*}(\mathrm{A},\mathrm{B}) = \mathrm{flow}^{\#}(\mathrm{A},\mathrm{B})]$$

Proof: We will establish the somewhat stronger result that for every derived state h

$$(\forall<\mathrm{A},\mathrm{B}> \in \mathrm{SUB}^{h} \times \mathrm{SUB}^{h})\,[\mathrm{flow}^{h}(\mathrm{A},\mathrm{B}) \subseteq \mathrm{flow}^{\#}(\mathrm{org}(\mathrm{A}),\mathrm{org}(\mathrm{B}))]$$

Then in particular, for A and B in the initial set of subjects, since org(A) = A and org(B) = B, we will have $\mathrm{flow}^{*}(\mathrm{A,B}) \subseteq \mathrm{flow}^{\#}(\mathrm{A,B})$. That $\mathrm{flow}^{\#}(\mathrm{A,B}) \subseteq \mathrm{flow}^{*}(\mathrm{A,B})$ is trivially true and we will have proved the theorem.

Now consider any transition sequence H, with weak creates, which results in state h. We will modify[3] H to obtain a transition sequence G, with no creates, resulting in state g such that

$$\mathrm{flow}^{h}(\mathrm{A},\mathrm{B}) \subseteq \mathrm{flow}^{g}(\mathrm{org}(\mathrm{A}),\mathrm{org}(\mathrm{B}))$$

Since our construction of G is for an arbitrary H, we will have established the desired result. It will be convenient for us to assume that

[3]The idea behind this construction was first used by Minsky [15] in the context of the uniform send-receive mechanism. The actual construction is quite similar to that used in the proof of theorem 3.6 on page 76, where we established that the $\mathrm{flow}^{\dagger}$ function provides a bound on the flow function.

H is in canonical form[4] and, by lemma 2.1 on page 50, we can do so without any loss of generality. Modify H to obtain G as follows.

1. Ignore all create operations in H.
2. Replace all demand operations in H as follows

$$\begin{gathered}\texttt{A demands B/x:c in H}\\ \textbf{becomes}\\ \texttt{org(A) demands org(B)/x:c in G}\end{gathered}$$

3. Ignore every transport operation in H of the form

$$\begin{gathered}\texttt{move A/x:c from dom(B) to dom(C) in H}\\ \texttt{where org(B) = org(C)}\end{gathered}$$

4. Replace every transport operation in H of the form

$$\begin{gathered}\texttt{move A/x:c from dom(B) to dom(C) in H}\\ \texttt{where org(B)} \neq \texttt{org(C)}\\ \text{by}\\ \texttt{move org(A)/x:c from dom(org(B)) to dom(org(C))}\end{gathered}$$

We will first establish the following facts

1. G is a transition sequence[5]
2. $A/x{:}c \in dom^{h}(B) \rightarrow org(A)/x{:}c \in dom^{g}(org(B))$

It follows immediately from assertion 2 that

$$link^{h}(A,B) \rightarrow link^{g}(org(A),org(B))$$

We will use this consequence of assertion 2 in the second part of the proof. The two assertions above are proved by induction on the number of transport operations in H.

Basis Case: For the basis case let the number of transport operations in H be 0. H then consists of a sequence of create operations followed by a sequence of demand operations, while G consists of a sequence of demand operations.

Assertion 1: The demand operations in G are legal, due to assertion 3 of lemma 4.1 on page 92, i.e.,

$$\begin{gathered}t(A)/x{:}c \in \textit{demand}(t(B)) \rightarrow\\ t(org(A))/x{:}c \in \textit{demand}(t(org(B)))\end{gathered}$$

Assertion 2: Since there are no transport operations, there are only three ways by which the ticket A/x:c can appear in dom^{h}(B).

[4]A transition sequence H is in canonical form if all create operations precede all demand operations which in turn precede all transport operations.

[5]That is, every state transition in G is legal.

Case (i): The ticket A/x:c is in the initial domain of B. This is possible only if A and B are in the initial set of subjects. But then, org(A) = A and org(B) = B, and assertion 2 becomes

$$A/x{:}c \in dom^{o}(B) \rightarrow A/x{:}c \in dom^{g}(B)$$

This is trivially true.

Case (ii): If the ticket A/x:c appears in the domain of B due to a create operation, then we have org(A) = org(B); and the right hand side of the implication in assertion 2 can be written as

$$org(B)/x{:}c \in dom^{g}(org(B))$$

Due to the self-reference requirement

$$org(B)/x{:}c \in dom^{o}(org(B))$$

But then, assertion 2 is trivially true.

Case (iii): The only other way for the ticket A/x:c to appear in $dom^{h}(B)$ is via a demand operation. But then, the replacement for the demand operation insures that assertion 2 is true.

Thus, for the basis case where H has no transport operations, both assertions are true.

Induction Step: Assume, as an induction hypothesis, that the assertions are true for every transition sequence H with k transport operations. Consider the case of a transition sequence H with k+1 transport operations. Then, H consists of an initial sequence H' with k transport operations followed by a single transport operation. Let the state established by the transition sequence H' be denoted as hk. Let G' be the required modification of H', and let the state established by G' be denoted as gk. By induction hypothesis G' is a transition sequence. Let the k+1st transport operation of H be

```
move A/x:c from dom(B) to dom(C)
```

Assertion 1: If org(B) = org(C), this transport operation is ignored in the construction of G. In this case, the induction step follows directly from the induction hypothesis. If org(B) ≠ org(A), the corresponding transport operation in G is

```
move org(A)/x:c from dom(org(B)) to dom(org(C))
```

Now, in order for the k+1st transport operation of H to be legal, we must have

$$\begin{aligned}&1.\ A/xc \in dom^{hk}(B)\\ &2.\ link^{hk}(B,C)\\ &3.\ t(A)/x{:}c \in dfl(t(B),t(C))\end{aligned}$$

By induction hypothesis 2, we have

$$A/xc \in dom^{hk}(B) \rightarrow org(A)/xc \in dom^{gk}(org(B))$$
$$link^{hk}(B,C) \rightarrow link^{gk}(org(B),org(C))$$

By assertion 2 of lemma 4.1 on page 92, we have

$$t(A)/x{:}c \in dfl(t(B),t(C)) \rightarrow t(org(A))/x{:}c \in dfl(t(org(B)),t(org(C)))$$

But then, the three conditions required to authorize the corresponding transport operation in G are true.

Assertion 2: The state h differs from the state hk at most by

$$A/x{:}c \in dom^{h}(C)$$

If org(B) ≠ org(C), the corresponding transport operation in G ensures that

$$org(A)/x{:}c \in dom^{g}(org(C))$$

If org(B) = org(C), then by condition 1 for the legality of the k+1st transport operation in H and by induction hypothesis 2, we have

$$org(A)/xc \in dom^{gk}(org(B))$$

from which it is immediately apparent that

$$org(A)/x{:}c \in dom^{g}(org(C))$$

This completes the induction step and we have established both assertions.

It now remains to show that for every derived state h

$$(\forall\langle A,B\rangle \in SUB^{h} \times SUB^{h})[flow^{h}(A,B) \subseteq flow^{*}(org(A),org(B))]$$

Let path^h and path^g, respectively, denote a path in state h and in state g. Given any path^h from A to B, we will show there is a path^g from org(A) to org(B) with at least the same authorization for transport of ticket types as the path^h from A to B. The proof is by induction on the number of links in the path^h from A to B. For the basis case, consider a path^h from A to B of length 1, i.e. the path consists of a single link. By assertion 2, we have that

$$link^{h}(A,B) \rightarrow link^{g}(org(A),org(B))$$

By assertion 1 of lemma 4.1 on page 92, we have

$$dfl(t(A), t(B)) \subseteq dfl(t(org(A)), t(org(B)))$$

Hence, the basis case is true. Assume that 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 subject C, such that there is $path^h$ from A to C of length k and $link^h(C,B)$. By induction hypothesis, there is a $path^g$ from org(A) to org(C) with at least the same authorization for transport of ticket types as the $path^h$ from A to C. By assertion 2, there is a $link^g$ from org(C) to org(B). Again, by lemma 4.1

$$dfl(t(C), t(B)) \subseteq dfl(t(org(C)), t(org(B)))$$

But then, there is a $path^g$ from org(A) to org(B) with at least the same authorization for transport of ticket types as the $path^h$ from A to B. This establishes the induction step and concludes the proof of the theorem. ∎

The proof of theorem 4.2 also provides an elegant characterization of the flow function, between subjects not present in the initial state, as follows.

Corollary 4.2.1: Given a scheme with weak creates and self-reference initial states then for every derived state h

$$(\forall \langle A,B\rangle \in SUB^h \times SUB^h)[flow^h(A,B) \subseteq flow^{\#}(org(A), org(B))]$$

∎

Thus, in every derived state, the flow function between two created subjects is a subset of the $flow^{\#}$ function between the originators of these subjects. In other words, if a flow can be realized between two created subjects then the same flow can also be realized between the originators of these subjects.

4.2. MODIFICATIONS OF THE WEAK CREATES DEFINITION.

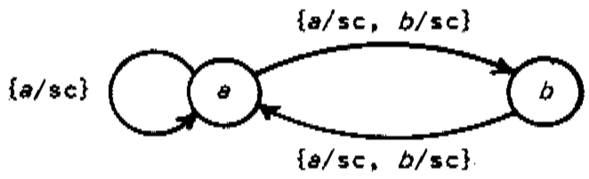
In this section we demonstrate that all aspects of our definition of weak creates on page 91 are required to ensure the create-invariant property. This definition requires the following conditions for the type *b* to be weaker than the type *a*.

1. $(\forall c)[dfl(c,b) \subseteq dfl(c,a) \wedge dfl(b,c) \subseteq dfl(a,c)]$
2. $(\forall \langle c,d\rangle)[b/x{:}c \in dfl(c,d) \rightarrow a/x{:}c \in dfl(c,d)]$
3. $demand(b) \subseteq demand(a)$
4. $(\forall c)[b/x{:}c \in demand(c) \rightarrow a/x{:}c \in demand(c)]$

We demonstrate that dropping any one of these conditions will allow for schemes which are not create-invariant. Of course, this does not mean that the conjunction of these four conditions is necessary for achieving the create-invariant property[6]. At the same time, this exercise does indicate there is little hope for relaxing these conditions.

We will drop each condition in turn and, hence, there are four cases to consider. The schemes which illustrate these four cases are similar to each other. In all cases there are two subject types *a* and *b*. The *dfl* and *demand* functions are constructed so that the type *b* satisfies all but one of the requirements for being weaker than the type *a*. In all cases we use the rule of self-copy creates[7]. For each case we exhibit a rather simple self-copy (and, hence, self-reference) initial state[8] with just two subjects A_1 and A_2, both of type *a*, to demonstrate that the scheme is not create-invariant. We have included all four cases for the sake of completeness. However, the general idea can be appreciated by studying any one of these cases.

<u>Case 1</u>: For this case we drop the first condition. Consider a scheme with the following *dfl* graph.



Let the *demand* function be

$$demand(a) = \{a/r,\ b/r\}$$
$$demand(b) = \{a/r,\ b/r\}$$

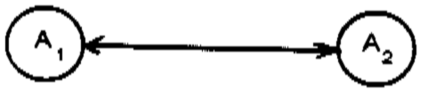
By inspection it is evident that conditions 2, 3, and 4 are satisfied for subjects of type *b* to be weaker than subjects of type *a*. Hence, if we drop condition 1 the pair <*a,b*> would be included in the weak *can-create* relation. Now consider the following initial state where A_1 and A_2 are subjects of type *a*.

---

[6]Indeed, our study of flow-invariant schemes in chapter 5, reveals that there are flow-invariant (and, hence, create-invariant schemes) which do not satisfy these conditions.

[7]See definition 2.11 on page 43.

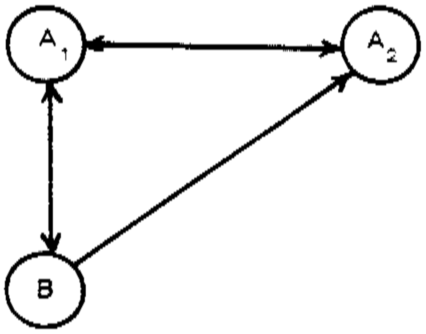
[8]See page 45.



In the absence of a create operation, these are the only possible links and the flow* function is easily computed to be as follows.

$$\text{flow}^*(A_1, A_2) = \{a/sc\}$$
$$\text{flow}^*(A_2, A_1) = \{a/sc\}$$

Let A_1 create a subject B of type *b*. By the self-copy create-rule, immediately after this create operation, there is a link from A_1 to B. By the self-reference assumption, A_2 possesses the ticket A_2/sc. This ticket can then be transported from dom(A_2) to dom(B) via A_1. A_2 can obtain the ticket B/r by a demand operation. Hence, it is possible to establish link(B,A_2), resulting in the following situation.

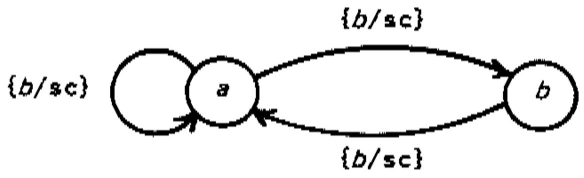


There is now authorization for transport of tickets of type *b*/sc from A_1 to A_2 indirectly via B. Hence,

$$[b/sc \in \text{flow}^*(A_1, A_2)] \wedge [b/sc \notin \text{flow}^*(A_1, A_2)]$$

But then, the scheme is not create-invariant.

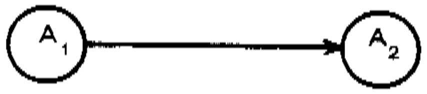
<u>**Case 2**</u>: For this case we drop the second condition. Consider a scheme with the following *dfl* graph.



Let the *demand* function be

$$demand(a) = \{a/r,\ b/r\}$$
$$demand(b) = \{a/r,\ b/r\}$$

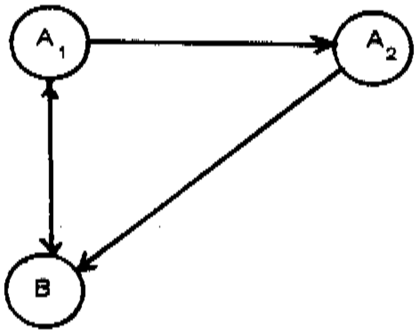
By inspection it is evident that conditions 1, 3, and 4 are satisfied for subjects of type *b* to be weaker than subjects of type *a*. Hence, if we drop condition 2 the pair <*a*,*b*> would be included in the weak *can-create* relation. Now consider the following initial state where A_1 and A_2 are subjects of type *a*.



Since transport tickets of the type *a*/x:c are not movable, in the absence of a create operation all possible links already exist, and the flow* function is as follows.

$$flow^*(A_1,A_2) = \{b/sc\}$$
$$flow^*(A_2,A_1) = \phi$$

Let A_1 create a subject B of type *b*. By the self-copy create-rule, immediately after this create operation, B possesses the ticket B/sc and there is a link from B to A_1. The ticket B/sc can then be transported from dom(B) to dom(A_2) via A_1. B can obtain the ticket A_2/r by a demand operation. Hence, it is possible to establish link(A_2,B), resulting in the following situation.

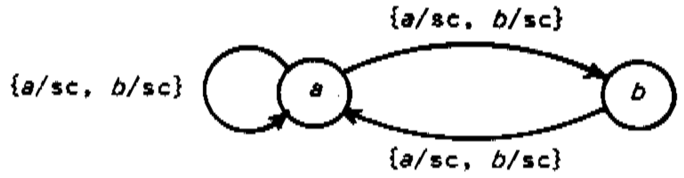


There is now authorization for transport of tickets of type *b*/sc from A_2 to A_1 indirectly via B. Hence,

$$[b/sc \in flow^*(A_2,A_1)] \wedge [b/sc \notin flow^*(A_2,A_1)]$$

But then, the scheme is not create-invariant.

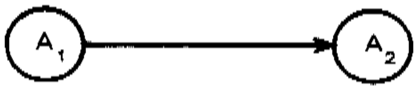
<u>**Case 3**</u>: For this case we drop the third condition. Consider a scheme with the following *dfl* graph.



Let the *demand* function be

$$demand(a) = \phi$$
$$demand(b) = \{a/r,\ b/r\}$$

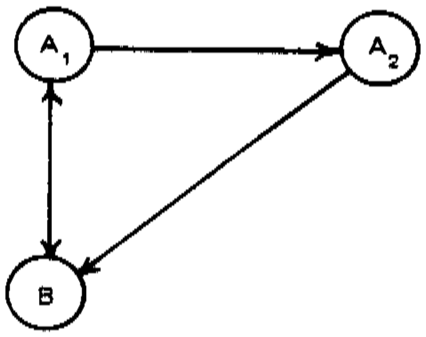
By inspection it is evident that conditions 1, 2, and 4 are satisfied for subjects of type *b* to be weaker than subjects of type *a*. Hence, if we drop condition 3 the pair *<a,b>* would be included in the weak *can-create* relation. Now consider the following initial state where A_1 and A_2 are subjects of type *a*.



By the self-reference assumption, A_1 possesses the ticket A_1/sc. This ticket can be moved to the domain of A_2. However, the ticket A_2/r cannot be obtained by A_1. Hence, in the absence of a create operation, all possible links already exist and the flow* function is as follows.

$$\text{flow}^*(A_1, A_2) = \{a/sc,\ b/sc\}$$
$$\text{flow}^*(A_2, A_1) = \phi$$

Let A_1 create a subject B of type *b*. By the self-copy create-rule, immediately after this create operation, B possesses the ticket B/sc and there is a link from B to A_1. The ticket B/sc can then be transported from dom(B) to dom(A_2) via A_1. B can obtain the ticket A_2/r by a demand operation. Hence, it is possible to establish link(A_2,B), resulting in the following situation.



There is now authorization for transport of tickets of type *a*/sc and *b*/sc from A_2 to A_1 indirectly via B. Hence,

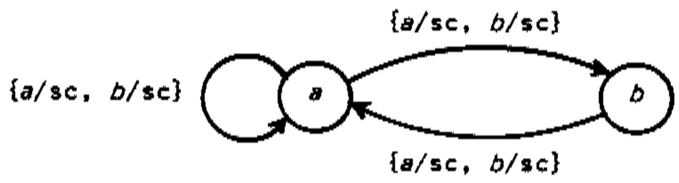
$$[b/\mathrm{sc} \in \mathrm{flow}^*(A_2,A_1)] \wedge [b/\mathrm{sc} \notin \mathrm{flow}^{\#}(A_2,A_1)]$$

and also

$$[a/\mathrm{sc} \in \mathrm{flow}^*(A_2,A_1)] \wedge [a/\mathrm{sc} \notin \mathrm{flow}^{\#}(A_2,A_1)]$$

But then, the scheme is not create-invariant.

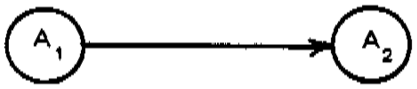
Case 4: For this case we drop the fourth condition. Consider a scheme with the following *dfl* graph.



Let the *demand* function be

$$demand(a) = \{b/\mathrm{r}\}$$
$$demand(b) = \{b/\mathrm{r}\}$$

By inspection it is evident that conditions 1, 2, and 3 are satisfied for subjects of type *b* to be weaker than subjects of type *a*. Hence, if we drop condition 4 the pair <*a,b*> would be included in the weak *can-create* relation. Now consider the following initial state where A_1 and A_2 are subjects of type *a*.



By the self-reference assumption, A_1 possesses the ticket A_1/sc. This ticket can be moved to the domain of A_2. However, the ticket A_2/r cannot be obtained by

A_1. Hence, in the absence of a create operation, all possible links already exist and the flow″ function is as follows.

$$\text{flow}''(A_1,A_2) = \{a/\text{sc},\ b/\text{sc}\}$$
$$\text{flow}''(A_2,A_1) = \phi$$

Let A_1 create a subject B of type *b*. By the self-copy create-rule, immediately after this create operation, B possesses the ticket B/sc and there is a link from B to A_1. The ticket B/sc can then be transported from dom(B) to dom(A_2) via A_1. B can obtain the ticket A_2/r by a demand operation. Hence, it is possible to establish link(A_2,B), resulting in the following situation.

$$\begin{tikzcd} A_1 \arrow[r] \arrow[d, leftrightarrow] & A_2 \arrow[dl] \\ B & \end{tikzcd}$$

There is now authorization for transport of tickets of type *a*/sc and *b*/sc from A_2 to A_1 indirectly via B. Hence,

$$[b/\text{sc} \in \text{flow}^*(A_2,A_1)] \wedge [b/\text{sc} \notin \text{flow}''(A_2,A_1)]$$

and also

$$[a/\text{sc} \in \text{flow}^*(A_2,A_1)] \wedge [a/\text{sc} \notin \text{flow}''(A_2,A_1)]$$

But then, the scheme is not create-invariant.

4.3. A CLASS OF HIERARCHICAL SCHEMES

In this section we consider a class of schemes for which the weak *can-create* relation arises as a natural consequence of the structure which motivates this class. Information systems are often structured as hierarchies. For example, the subjects might be organized in a hierarchy corresponding to a corporate organization chart. The class of schemes we study here models such situations[9] .

[9]The model studied here, is one particular model for hierarchical organizations. Our framework admits any number of such models. Indeed, no single model can cover the variety of hierarchical organizations encountered in practice.

In a hierarchical organization there are well defined channels for exchange of information. Thus, two clerical workers in different departments typically cannot exchange information directly, but must do so via some boss at an appropriate level in the hierarchy. The existence of such "proper channels" for communication is essential to preserving the sanity of a large organization. At the same time, it does lead to red-tape in that there will be delays in the communication. To counter this we need to provide a facility by which direct communication can be established, at the discretion of the bosses who would otherwise be involved in a "proper channel". Our model is motivated by this consideration.

Our starting point is a given hierarchy on the set of subject types, which presumably corresponds to some real-world structure. This hierarchy is specified by a 1-1 mapping between the nodes of a rooted tree and the set of subject types. We identify each node of the tree by the subject type associated with the node. The given hierarchy then induces the following structure on the subject types, defined in terms of standard properties of a rooted tree.

Definition 4.4: Define the **predecessor** relation

$$pred \subseteq T_S \times T_S$$

by *pred*(*a*,*b*) if and only if *a* is the root of some subtree, in the given hierarchy, which includes *b*. ∎

Definition 4.5: Define the **level** function

$$level: T_S \rightarrow \{1, \ldots, n\}$$

as follows.

1. If *a* is the root of the tree then *level*(*a*) = 1.

2. If *a* is not the root of the tree then

$$level(a) = \max\{level(b) \mid pred(b,a) \wedge b \neq a\} + 1$$

We say that the level of a subject A is *level*(*t*(A)). ∎

With respect to the *pred* relation, the idea is that a subject B is a subordinate or an equal of subject A if and only if *pred*(*t*(A),*t*(B)). For simplicity, we will use the term "subordinate" instead of the phrase "subordinate or equal", even though the simpler term misses the connotation of possible equivalence. The intention of the *level* function is that the lower the level of a subject type *a* the closer it is to the root of the hierarchy, and thereby the more trust and authority that may be assigned to subjects of type *a*.

Further, assume that object types are also classified into levels, so that there is at least one object type for every level of the tree. That is, the level function is extended to apply to object types as follows.

$$level: T_O \rightarrow \{1, \ldots, n\}$$

We then say that the level of an object O is *level*(*t*(O)). The intention is that the lower the level of an object, the more sensitive the information contained in it, and thereby the more restrictive the policy for transport of tickets for O.

Now consider the following policy for transport of tickets for objects.

> A subject B can obtain a ticket for an object O, only from a subject A whose level is less than or equal to the level of O.

That is, the more sensitive an object, the higher-up in the hierarchy must be a subject from whom B can obtain tickets for the object. This is exactly the kind of policy we encounter in real-world organizations. This policy is stated formally, as follows.

$$\begin{gathered}\text{O/x:c may be transported from dom(A) to dom(B)} \\ \rightarrow \\ level(t(A)) \leq level(t(O))\end{gathered}$$

Within our design framework, this policy is easily implemented by ensuring that for all pairs of subject types <*a*,*b*>

$$(\forall o/x{:}c \in T_O \times R_I)\,[o/x{:}c \in dfl(a,b) \rightarrow level(a) \leq level(o)]$$

Now there are many *dfl* functions which are consistent with this policy. As a trivial example, we can simply let *dfl*(*a*,*b*) be empty for all pairs of subject types. Here, we will interpret the stated policy in a permissive spirit, thereby not imposing any additional restrictions. This leads us to the following definition, for that portion of the *dfl* function which authorizes transport of tickets for object types.

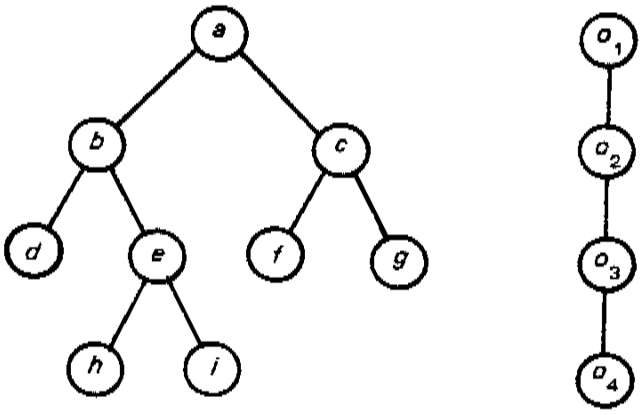
$$(\forall o/x{:}c \in T_O \times R_I)\,[o/x{:}c \in dfl(a,b) \leftrightarrow level(a) \leq level(o)]$$

Interestingly, the value of *dfl*(*a*,*b*) is determined by *level*(*a*) alone, and does not depend on *level*(*b*) in any way. In particular, if *a* is the root of the tree, then

$$(\forall b)\,[T_O \times R_I \subseteq dfl(a,b)]$$

Thus, there is no restriction on tickets for objects which can be obtained from a subject whose type is at the root of the hierarchy.

Let us consider this policy for transport of tickets for objects, in the context of the following hierarchy.



Here, the set of subject types is

$$T_S = \{a,b,c,d,e,f,g,h,i\}$$

with the hierarchical structure specified above. The set of object types is

$$T_O = \{o_1,o_2,o_3,o_4\}$$

with $level(o_i) = i$.

Consider a subject B of type *b*. Since *level*(*b*) is 2, B can transport tickets for objects of level 2, 3, and 4, to every subject X such that link(B,X). Similar remarks apply for a subject E of type *e*, except that E can only transport tickets for objects of level 3 and 4.

For the policy with respect to transport of transport tickets, we classify pairs of subject types into two categories, as follows.

1. The pairs <*a,b*> which are **comparable**, with respect to the *pred* relation, i.e., $pred(a,b) \vee pred(b,a)$.

2. The pairs <*a,b*> which are **incomparable**, with respect to the *pred* relation, i.e., $\sim pred(a,b) \wedge \sim pred(b,a)$.

We say that two subjects A and B are comparable or incomparable if their types are, respectively, comparable or incomparable. Consider the following policy for transport of transport tickets between two incomparable subjects.

1. Transport tickets cannot be directly transported between two incomparable subjects, A and B.

2. Transport tickets can be indirectly transported between two

> incomparable subjects A and B, only via a path which includes a subject whose type is a predecessor of both $t(A)$ and $t(B)$.

The first aspect amounts to requiring that, exchange of transport tickets between two incomparable subjects must take place through a "proper channel". This aspect is easily implemented, by ensuring that

$$\langle a,b\rangle \text{ are incomparable} \Rightarrow dfl(a,b) \cap T_S \times R_T = \phi$$

The second aspect defines a "proper channel", for this purpose, to be a path which includes at least one subject whose type is a predecessor of both the incomparable subjects in question.

This brings us to the question of the policy for transport of transport tickets between two comparable subjects. Now consider a subject A. We will allow A to transport tickets for every subject whose type is subordinate to $t(A)$ in the hierarchy. By virtue of its position in the hierarchy, A is a boss (or an equal) of all such subjects; and is thereby entitled to participate in the activity of establishing a link which bypasses a "proper channel". In addition, we will allow A to transport tickets for those subjects whose types are predecessors of $t(A)$. Then A can participate in the activity of establishing links between the subordinates of A and the bosses of A. These considerations lead us to the following policy for transport of transport tickets between two comparable subjects.

> Transport tickets of type $c/x{:}c$ can be directly transported between two comparable subjects, A and B, if and only if c is comparable to $t(A)$.

Within our design framework, this policy is easily implemented by ensuring that for all comparable pairs of subject types $\langle a,b\rangle$

$$(\forall c/x{:}c \in T_S \times R_T)\,[c/x{:}c \in dfl(a,b) \Leftrightarrow pred(a,c) \vee pred(c,a)]$$

We now illustrate these policies in the context of the hierarchy defined on page 106. First consider an incomparable pair of subjects, say B and C of type b and c respectively. Then, transport tickets can be moved from dom(B) to dom(C), or vice versa, only if at least one subject of type a is involved in this activity. Next consider a comparable pair of subjects, say B and E of type b and e respectively. Let there be a link from B to E and vice versa. Then transport tickets for subjects of type e, h, i, b and a can be moved from dom(E) to dom(B). These types of transport tickets can also be moved from dom(B) to dom(E). Additionally, transport tickets for subjects of type d can be moved from dom(B) to dom(E).

The following definition summarizes our derivation of the *dfl* function from the *pred* and *level* functions.

Definition 4.6: For a hierarchical scheme define the *dfl* function as follows.

1. If $pred(a,b) \vee pred(b,a)$ then

$$dfl(a,b) = \{c/x : c \in T_S \times R_T \mid pred(a,c) \vee pred(c,a)\} \cup \{o/x : c \in T_O \times R_I \mid level(a) \le level(o)\}$$

2. If $\sim pred(a,b) \wedge \sim pred(b,a)$ then

$$dfl(a,b) = \{o/x : c \in T_O \times R_I \mid level(a) \le level(o)\}$$

∎

So far, we have confined our discussion to the *dfl* function. Let us next consider the *can-create* relation. We allow every subject A to create subjects whose types are subordinate to $t(A)$ in the given hierarchy. This is easily achieved by defining the *can-create* relation to be identical to the *pred* relation.

Next consider the create-rules which specify the semantics of a create operation. Since our focus here is on the create-invariant property and the create-rules are not relevant to our characterization of this property, we will simply ignore this aspect. Then any set of create-rules is acceptable.

Finally, consider the demand operation for transport tickets. For the sake of simplicity, we require the *demand* function to be one of the following cases.

1. $(\forall a)\,[demand(a) = \phi]$
2. $(\forall a)\,[demand(a) = \{b/s \mid b \in T_S\}]$
3. $(\forall a)\,[demand(a) = \{b/r \mid b \in T_S\}]$

In the first case, transport tickets cannot be obtained by a demand operation. The second case, authorizes subjects of every type to obtain a send ticket for subjects of every type. Hence, the link relation is effectively determined by the distribution of receive tickets. This case then models a degenerate version of our send-receive framework. The third case is similar, except that the link relation is now effectively determined by the distribution of send tickets.

The following definition summarizes the above discussion.

Definition 4.7: A **hierarchical scheme** is defined by specifying the following parameters.

1. A rooted tree and a 1–1 mapping between T_S and the set of nodes of the tree. The *pred* relation and the *level* function for subject types are thereby implicitly defined (see definitions 4.4 and 4.5 on page 104.).

2. A function which assigns a level to each object type, i.e.,
$$\mathit{level}\colon T_O \rightarrow \{1, \ldots, n\}$$
where
$$n = \max\{\mathit{level}(a) \mid a \in T_S\}$$
such that there is at least one object type for every level, i.e.,
$$(\forall i \leq n)\ (\exists o \in T_O)\ [\mathit{level}(o) = i]$$

3. The *dfl* function as in definition 4.6.

4. The *can-create* relation as identical to the *pred* relation.

5. One of the three *demand* functions shown below.
$$\begin{aligned} &1.\ (\forall a)\ [\mathit{demand}(a) = \phi] \\ &2.\ (\forall a)\ [\mathit{demand}(a) = \{b/s \mid b \in T_S\}] \\ &3.\ (\forall a)\ [\mathit{demand}(a) = \{b/r \mid b \in T_S\}] \end{aligned}$$

■

We then have the following theorem.

Theorem 4.3: Every hierarchical scheme is create-invariant for self-reference initial states, i.e., the weak *can-create* relation for a hierarchical scheme is exactly the *pred* relation so that
$$\mathit{pred}(a, b) \leftrightarrow b \text{ is weaker than } a$$

Proof: $\rightarrow$: Observe that, for all three cases, the *demand* function trivially satisfies conditions 3 and 4 of definition 4.2 on page 91. Thus, it suffices to establish that *pred(a,b)* implies
$$\begin{aligned} &1.\ (\forall c)\ [\mathit{dfl}(c,b) \subseteq \mathit{dfl}(c,a) \wedge \mathit{dfl}(b,c) \subseteq \mathit{dfl}(a,c)] \\ &2.\ (\forall \langle c,d \rangle)\ [b/x{:}c \in \mathit{dfl}(c,d) \rightarrow a/x{:}c \in \mathit{dfl}(c,d)] \end{aligned}$$

In order to do so, we will use the following fact: Given *pred(a,b)*, if *b* and *c* are comparable then *a* and *c* are also comparable[10]. If *b* and *c* are comparable because *pred(b,c)*, then this statement follows from the

[10] See page 106 for the definition of comparable.

transitivity of the *pred* relation. If *b* and *c* are comparable because *pred*(*c*,*b*), then *b* has both *a* and *c* as predecessors. Since the subtrees with *a* and *c* are not disjoint, it follows that one of *a* or *c* must be a predecessor of the other. But this is exactly what it means to say that *a* and *c* are comparable. Now consider the two assertions which need to be established.

<u>Assertion 1</u>: There are parts to assertion 1, i.e., for every subject type *c*

$$[dfl(c,b) \subseteq dfl(c,a)] \wedge [dfl(b,c) \subseteq dfl(a,c)]$$

Consider the first part of this conjunction, i.e., $dfl(c,b) \subseteq dfl(c,a)$. To prove this statement, first consider tickets for objects. By definition of the *dfl* function, we know that for such tickets the value of *dfl*(*c*,*d*) is independent of *d*. Thus, for types of tickets for objects this statement is trivially true. Next consider tickets for subjects. By definition of the *dfl* function, we know that *d*/x:c is in *dfl*(*b*,*c*) if and only if both

1. *b* and *c* are comparable.
2. *b* and *d* are comparable.

Given *pred*(*a*,*b*), it follows from our earlier observation that

1. *a* and *c* are comparable.
2. *a* and *d* are comparable.

But then, *d*/x:c is in *dfl*(*a*,*c*). This completes the proof for the first part of assertion 1. The proof for the second part is similar.

<u>Assertion 2</u>: Consider any pair of subject types <*c*,*d*>. By definition, *b*/x:c is in *dfl*(*c*,*d*) if and only if both

1. *c* and *d* are comparable.
2. *c* and *b* are comparable.

Given *pred*(*a*,*b*) in conjunction with 2, it follows from our earlier observation that *a* and *c* are comparable. But then, by definition of the *dfl* function *a*/x:c is in *dfl*(*c*,*d*). This completes the proof of assertion 2 and establishes one direction of the theorem.

$\Leftarrow$: We need to establish that ~*pred*(*a*,*b*) implies *b* is not weaker than *a*. There are two cases to consider, depending on whether *pred*(*b*,*a*) or ~*pred*(*b*,*a*).

<u>Case 1</u>: Let *b* not be a predecessor of *a* so that

$$\sim pred(a,b) \wedge \sim pred(b,a)$$

Then, due to the structure of a rooted tree, there exists a subject type *c* which is a predecessor of <u>both</u> *a* and *b*. By definition of the *dfl* function, then

$$[b/x{:}c \in dfl(b,c)] \wedge [b/x{:}c \notin dfl(a,c)]$$

Hence, $dfl(b,c)$ is not a subset[11] of $dfl(a,c)$ and b is not weaker than a.

Case 2: Let b be a predecessor of a. It follows that

$$level(b) < level(a)$$

By definition of a hierarchical scheme, there exists an object type o such that $level(o) = level(b)$. But then, by definition of the dfl function

$$[o/x{:}c \in dfl(b,a)] \wedge [o/x{:}c \notin dfl(a,a)]$$

Hence, $dfl(b,c)$ is not a subset[12] of $dfl(a,c)$ and b is not weaker than a. This completes the proof of the theorem. ∎

In conclusion, note that the weak *can-create* relation for hierarchical schemes, in general, will contain pairs of subject types which are not reflexive.

4.4. SUMMARY

In this chapter we discussed the notion of a create-invariant scheme. For such schemes the flow* function is identical to the flow$^\#$ function and, hence, can be computed in a straightforward manner.

In section 4.1 we developed a notion of weak creates defined in terms of the *dfl* and *demand* functions. The constraint of weak creates is non-trivial in the sense that it will always permit a subject to create another subject of the same type. We showed in theorem 4.2 that any scheme with weak creates and self-reference initial states is create-invariant. In section 4.2 we showed that relaxing the definition of weak creates will allow for schemes which are not create-invariant. In section 4.3 we discussed the class of hierarchical schemes to demonstrate the utility of create-invariant schemes.

[11]For this case, we have established this fact in the context of tickets types for subjects. We cannot reach a similar conclusion with respect to ticket types for objects. We leave it as an exercise for the reader to figure out the possibilities in the latter context.

[12]For this case, we have established this fact in the context of tickets types for objects. We cannot reach a similar conclusion with respect to ticket types for subjects. Again, we leave it as an exercise for the reader to figure out the possibilities in the latter context.

CHAPTER 5
FLOW-INVARIANT SCHEMES

In this chapter we study a class of schemes which maintain the flow function as an invariant property of a system. To this end we have the following definition.

Definition 5.1: A scheme is **flow-invariant** (in the strong sense) if for every initial state it is the case that

$$(\forall\langle A,B\rangle \in SUB^{o} \times SUB^{o})[flow^{o}(A,B) = flow^{*}(A,B)]$$

■

Flow-invariant schemes[1] prohibit any change in the authorization for transport of ticket types from subject A to subject B. Even such a severe constraint on evolution of the flow function is of interest. The system can evolve in two ways. Firstly, it is possible to establish new paths. For example, we might have an initial state where some component of the flow from A to B is necessarily via subject C. In a derived state it might be possible to bypass the subject C. Thus a flow-invariant scheme permits for alternate paths to be established for effecting the flow specified in the initial state. Secondly, new subjects can be introduced in the system and paths may be constructed involving these subjects.

A more interesting class of schemes is obtained by modifying the the definition of flow-invariant to apply only to the flow of transport tickets. In order to do so we introduce the following notation.

[1]The uniform send-receive mechanism of Minsky [15] is flow-invariant under certain constraints as discuussed in section 1.2.4. Here we generalize this property to apply to selective schemes.

Definition 5.2: For every $flow^k$ function define the associated **flow function for transport tickets**

$$flowt^k\text{: } SUB^k \times SUB^k \rightarrow \textbf{power-set}(T_S \times R_T)$$

by

$$flowt^k(A,B) = flow^k(A,B) \cap T_S \times R_T$$

Similarly, define the $flowt^{\#}$ and $flowt^{*}$ functions by, respectively, restricting the range of the $flow^{\#}$ and $flow^{*}$ functions to types of transport tickets. ∎

This leads us to the following modification of definition 5.1.

Definition 5.3: A scheme is **flow-invariant** (in the **weak** sense) if for every initial state it is the case that

$$(\forall\langle A,B\rangle \in SUB^{o} \times SUB^{o})[flowt^{o}(A,B) = flowt^{*}(A,B)]$$

∎

Henceforth, we will use the term flow-invariant to mean flow-invariant in the weak sense.

For flow-invariant schemes there is a considerable speed-up in computing a bound on the flow function via the $flow^{\dagger}$ function discussed in section 3.5. Specifically, as demonstrated in appendix C.5, the cost of computing the $flow^{\dagger}$ function for such schemes is no worse than $O(|T \times R|*|T_S|^3*|SUB^o|^3)$ in contrast to the possibility of an $O(|T_S|^5*|SUB^o|^5)$ cost in general.

In our discussion of flow-invariant schemes we need to focus on the transport of transport tickets. In order to do so conveniently, we introduce the following notation to isolate that portion of the *dfl* function which authorizes transport of such tickets.

Definition 5.4: For every *dfl* function define the associated **direct flow limit function for transport tickets**

$$dflt\text{: } T_S \times T_S \rightarrow \textbf{power-set}(T_S \times R_T)$$

by

$$dflt(a,b) = dfl(a,b) \cap T_S \times R_T$$

Similarly, define the *mflt* and *iflt* functions by, respectively, restricting the range of the *mfl* and *ifl* functions to types of transport tickets. ∎

Our discussion in this chapter is in terms of the weaker notion of flow-invariant

However, our results extend to the stronger definition of flow-invariant simply by replacing flowt everywhere by flow and any occurrence of *_flt* by *_fl*.

5.1. THE ROLE OF THE CREATE OPERATION

In this section we establish that, under certain conditions on the create-rules, the value of the *can-create* relation is not relevant in determining whether a scheme is flow-invariant. This fact simplifies our analysis, since we can then assume, without any loss of generality, that the *can-create* relation is empty; thereby effectively eliminating the create operation.

In lemma 2.1 on page 50, we observed that any derived state can be established by a transition sequence in canonical form so that the create operations precede the demand and transport operations. Let us call the state after all the create operations in a canonical transition sequence have taken place, a **create-augmented state**. Clearly, the nature of a create-augmented state is determined by the create-rules of the scheme and the nature of the initial state.

Now, one aspect of defining a scheme is to specify constraints on the initial state[2]. If every create-augmented state satisfies these constraints, then every such state is itself an acceptable initial state. This observation leads us to the following definition.

Definition 5.5: We say that the create-rules of a scheme **preserve the constraints on the initial state** provided every create-augmented state satisfies all constraints on the initial state required by the scheme[3]. ■

Observe, in particular, if there are no constraints on the initial state then any set of create-rules will preserve the constraints on the initial state.

We then have the following theorem.

---

[2]Abstractly, a constraint is simply some predicate which evaluates to true or false for any given protection state. The kinds of constraints we have in mind are discussed in section 2.2.4.

[3]Of course, by definition, the initial state must satisfy these constraints.

Theorem 5.1: Any scheme with create-rules which preserve the constraints on the initial state is flow-invariant if and only if for all initial states

$$(\forall \langle A,B\rangle \in SUB^{o} \times SUB^{o})[\text{flowt}^{o}(A,B) = \text{flowt}^{\#}(A,B)]$$

Proof: $\Rightarrow$: This direction follows trivially, since if

$$\text{flowt}^{*}(A,B) = \text{flowt}^{o}(A,B)$$

then certainly

$$\text{flowt}^{\#}(A,B) = \text{flowt}^{o}(A,B)$$

$\Leftarrow$: Consider any derived state h. By lemma 2.1 on page 50, we can assume, without loss of generality, that h is established by a transition sequence H in canonical form. Let g be the state immediately after all the create operations of H have occurred. It will suffice to show that

$$(\forall \langle A,B\rangle \in SUB^{g} \times SUB^{g})[\text{flowt}^{g}(A,B) \subseteq \text{flowt}^{o}(\text{org}(A),\text{org}(B))]$$

Since the create-rules preserve the constraints on the initial state, we can consider g to be an initial state. The non-create operations of the transition sequence H can then be applied to this initial state g, resulting in state h. Since this g to h transition does not include any creates, by assumption

$$(\forall \langle A,B\rangle \in SUB^{g} \times SUB^{g})[\text{flowt}^{g}(A,B) = \text{flowt}^{h}(A,B)]$$

By construction, we have that

$$SUB^{h} = SUB^{g}$$

By combining the three equations above, we have

$$(\forall \langle A,B\rangle \in SUB^{h} \times SUB^{h})[\text{flowt}^{h}(A,B) \subseteq \text{flowt}^{o}(\text{org}(A),\text{org}(B))]$$

In particular, for subjects A and B present in the initial state we have org(A) = A and org(B) = B, so that

$$(\forall \langle A,B\rangle \in SUB^{o} \times SUB^{o})[\text{flowt}^{h}(A,B) \subseteq \text{flowt}^{o}(A,B)]$$

This argument applies for any arbitrary state h and, hence,

$$(\forall \langle A,B\rangle \in SUB^{o} \times SUB^{o})[\text{flowt}^{*}(A,B) \subseteq \text{flowt}^{o}(A,B)]$$

It is trivially true that

$$(\forall \langle A,B\rangle \in SUB^{o} \times SUB^{o})[\text{flowt}^{o}(A,B) \subseteq \text{flowt}^{*}(A,B)]$$

The theorem will then follow from these two equations.

It remains to show that for every create-augmented state g (i.e. a state established by a transition sequence which consists entirely of create operations), we have

$$(\forall \langle A,B\rangle \in SUB^g \times SUB^g)[flowt^g(A,B) \subseteq flowt^0(org(A),org(B))]$$

We now prove this by induction on the number of create operations in the transition sequence which establishes state g. Let the state after the first n create operations be denoted as state n. We need to show that for all n

$$(\forall \langle A,B\rangle \in SUB^n \times SUB^n)[flowt^n(A,B) \subseteq flowt^0(org(A),org(B))]$$

For the basis case let n = 0. But then, state n is the initial state and org(A) = A and org(B) = B. Hence, the basis case is trivially true. For the induction step, let the k+1st create operation be the creation of subject D by subject C. The only links[4] in state k+1 which might be introduced by this operation are $link^{k+1}(C,D)$ and/or $link^{k+1}(D,C)$. There are several cases to consider.

<u>Case (i)</u>: Consider any pair of subjects <A,B> such that A ≠ D and B ≠ D. Every path in state k+1 from A to B which includes subject D must include both $link^{k+1}(D,C)$ and $link^{k+1}(C,D)$ contiguously in this order, since these are the only possible links to and from subject D in state k+1. This path can then be replaced by a shorter path, that does not include subject D, by eliminating these two links. This shorter path only involves subjects and links present in state k. But then,

$$flowt^{k+1}(A,B) \subseteq flowt^k(A,B)$$

The induction step, for such pairs of subjects, follows from the induction hypothesis.

<u>Case (ii)</u>: Consider any pair of subjects <A,D> such that A ≠ C and A ≠ D. Every path in state k+1 from A to D must include subject C, since the only possible link to subject D in state k+1 is $link^{k+1}(C,D)$. Hence,

$$flowt^{k+1}(A,D) \subseteq flowt^{k+1}(A,C)$$

By case (i) we have already established that

$$flowt^{k+1}(A,C) \subseteq flowt^0(org(A),org(C))$$

Since org(D) = org(C), by combining these two equations, it follows that

$$flowt^{k+1}(A,D) \subseteq flowt^0(org(A),org(D))$$

<u>Case (iii)</u>: Consider any pair of subjects <D,B> such that

[4]This follows from the constraint that the create-rules must be local as discussed on page 43.

$B \neq C$ and $B \neq D$. By a similar argument as in case (ii), we can show that

$$flowt^{k+1}(D,B) \subseteq flowt^{0}(org(D),org(B))$$

Case (iv): The only pairs left to consider are <C,C>, <C,D> and <D,C>. Since subject C creates subject D, org(D) = org(C). The induction step then follows trivially, since by definition

$$flowt^{0}(org(C),org(C)) = T_S \times R_T$$

This completes the proof of the theorem. ■

The proof of theorem 5.1 also provides the following characteristic of the flowt function for subjects created subsequent to the initial state.

Corollary 5.1.1: For every flow-invariant scheme and every initial state it is the case that for every derived state h

$$(\forall \langle A,B\rangle \in SUB^h \times SUB^h)[flowt^h(A,B) \subseteq flowt^0(org(A),org(B))]$$

■

This is an appealing property. Recall that we had a similar result, in the context of create-invariant schemes (see corollary 4.2.1 on page 97).

5.2. SELF-COPY SCHEMES

In the remainder of this chapter we characterize the flow-invariant property for a class of schemes called self-copy schemes. The constraints which define this class are presented in section 5.2.1. In section 5.2.2 we identify a straightforward but crucial consequence of these constraints, called the origination property.

5.2.1. The Self-Copy Constraints

Self-copy schemes are defined, by constraining the initial state and the *demand* function and adopting a specific rule for the create operation, as follows.

Definition 5.6: A scheme is said to be a **self-copy scheme** if

1. The initial state is constrained to be a balanced self-copy state, i.e., for all subjects A and B present in the initial state

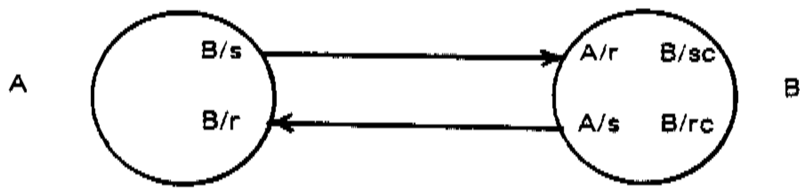
$$\text{(a)}\quad [B/s \in dom^0(A)] \leftrightarrow [A/r \in dom^0(B)]$$

$$\text{(b)}\quad [A/sc \in dom^0(B) \leftrightarrow A = B] \wedge [A/rc \in dom^0(B) \leftrightarrow A = B]$$

2. Copiable transport tickets cannot be obtained by a demand operation, i.e.,

   $$(\forall \langle a,b\rangle)\ [b/\mathrm{sc} \notin \mathit{demand}(a)\ \wedge\ b/\mathrm{rc} \notin \mathit{demand}(a)]$$

3. The create-rule is the self-copy rule which states that immediately after a subject A creates a subject B the situation is



■

Observe that there are no constraints on the *dfl* function and the *can-create* relation.

The restriction of a balanced initial state is mitigated by the existence of a *demand* function. Anyway, this restriction merely amounts to requiring that the initial state be specified without any stray tickets which do not establish links.

The self-copy restriction is considerably stronger. Of course, this constraint is on the initial state and in general will not be true in derived states. The restriction on the *demand* function ensures that the self-copy aspect of the initial state cannot be altered by a demand operation. The net effect is that there is a single source, the domain of a subject A, from which all copies of a transport ticket for subject A must be obtained. This single source property of a self-copy scheme facilitates implementation of mechanisms for revocation of these copies.

The self-copy create-rule ensures a uniformity with the constraints on the initial state. Specifically, consider any transition sequence in canonical form. The state after all the creates have occurred will then be a balanced self-copy state, since every individual create operation preserves the balanced self-copy property. In particular then, theorem 5.1 on page 115 applies to self-copy schemes, so that the value of the *can-create* relation is not relevant in determining the flow-invariant property for such schemes.

5.2.2. The Origination Property

The self-copy constraints have a simple but crucial consequence. Consider any subject A. These constraints ensure that transport tickets for A with the copy flag are initially present only in the domain of A. Any transport of the tickets A/s:c and A/r:c must then originate from A. Hence, if a subject B ever acquires one of these tickets by a transport operation it must be due to a series of transport operations which can be traced back to the domain of A. We call this the **origination property**. A formal statement of this property is as follows.

Lemma 5.2: Given a self-copy scheme and any pair of subjects A and B then for every transition sequence H whose final operation is a transport operation which places

$$A/x{:}c \text{ in } dom^h(B)$$

there is a proper (possibly empty) prefix G of H which establishes state g such that

$$t(A)/x{:}c \in flow^g(A,B)$$

Proof: Follows immediately from the above discussion. ■

In order to make use of this property in our analysis, we introduce the following notation to identify those tickets which can only be acquired by a transport operation.

Definition 5.7: For a given scheme and initial state we say that $B/x{:}c \in dom_T(A)$ if and only if

1. There is a transition sequence H whose final operation is a transport operation which places

$$B/x{:}c \text{ in } dom^h(A)$$

2. The ticket B/x:c is not in the initial domain of A, i.e.,

$$B/x{:}c \notin dom^0(A)$$

3. The ticket B/x:c cannot be obtained by A via a demand operation, i.e.,

$$t(B)/x{:}c \notin \mathit{demand}(t(A))$$

4. The ticket B/x:c is not placed in the domain of A by a create operation.

■

The first condition requires that the ticket can indeed be acquired by a transport operation. The other conditions exclude those tickets which are acquired or could be acquired in some other way. Lemma 5.2 can then be written as follows.

$$A/x{:}c \in \mathrm{dom}_T(B)$$
$$\Rightarrow$$
$$t(A)/x{:}c \in \mathrm{flow}(A,B) \text{ prior to } A/x{:}c \in \mathrm{dom}(B)$$

Here we omit explicit mention of the states g and h for simplicity.

5.3. FLOW-INVARIANT SELF-COPY SCHEMES

In this section we obtain necessary and sufficient conditions for a self-copy scheme to be flow-invariant. Due to theorem 5.1 on page 115, the value of the *can-create* relation is not relevant in determining the flow-invariant property. We will assume, without any loss of generality, that *can-create* = ϕ thereby effectively eliminating the create operation. It then suffices to define the *dfl* and *demand* functions, in order to define a particular self-copy scheme. Moreover, since our focus is on the flow-invariant property, we need only define that portion of the *dfl* function which authorizes transport of transport tickets (i.e., the *dflt* function).

The flow function, in general, and the flowt function, in particular, change due to establishment of links which were not present in the initial state. The evolution of the link relation is then a significant aspect for studying the flow-invariant property. We introduce the link$^+$ relation to refer conveniently to this evolution.

Definition 5.8: For a given scheme and initial state define the associated binary relation

$$\mathrm{link}^+ \subseteq SUB^O \times SUB^O$$

by

$$[\sim\mathrm{link}^O(A,B)] \wedge (\exists \text{ derived state } h)[\mathrm{link}^h(A,B)]$$

■

In order to understand the flow-invariant property it is essential to first understand the link$^+$ relation. The manner by which a link$^+$ must be established is strongly influenced by the *demand* function. Our strategy is to isolate four extreme cases of the *demand* function. This provides a convenient technique for developing our analysis by first considering these extreme cases and then consolidating the results. Moreover, these extreme cases are of interest by themselves.

5.3.1. Extreme Cases of the Demand Function

For a particular pair of subject types $<a,b>$, in a self-copy scheme, there are four possibilities with respect to the *demand* function as follows.

1. $[b/s \in demand(a) \wedge a/r \in demand(b)]$
2. $[b/s \in demand(a) \wedge a/r \notin demand(b)]$
3. $[b/s \notin demand(a) \wedge a/r \in demand(b)]$
4. $[b/s \notin demand(a) \wedge a/r \notin demand(b)]$

Now consider a subject A of type *a* and a subject B of type *b* such that $\sim link^{0}(A,B)$. In the first situation, it is a simple matter to establish $link^{+}(A,B)$, since both subjects can obtain the required ticket by a demand operation. In the next two situations, one of the subjects can obtain the required ticket by a demand operation. In the fourth situation, neither subject can obtain the required ticket by a demand operation. These four possibilities strongly influence the manner in which $link^{+}(A,B)$ must be established.

We define four extreme cases of the *demand* function in correspondence with the four possibilities above.

1. For the first case we require that

   $$(\forall <a,b>)\ [b/s \in demand(a) \wedge a/r \in demand(b)]$$

   In this case there is no control over the distribution of transport tickets, since these tickets can always be obtained by a demand operation. A scheme with such a *demand* function is called a **permissive scheme** to reflect this fact.

2. For the second case we require that

   $$(\forall <a,b>)\ [b/s \in demand(a) \wedge a/r \notin demand(b)]$$

   This case is a degenerate version of our send-receive framework, where the authorization for a transport operation is effectively achieved by the distribution of receive tickets. A scheme with such a *demand* function is called a **receive-controlled scheme** to reflect this fact.

3. For the third case we require that

   $$(\forall <a,b>)\ [b/s \notin demand(a) \wedge a/r \in demand(b)]$$

   This case is another degenerate version of our send-receive framework, where the authorization for a transport operation is effectively achieved by the distribution of send tickets. A scheme with such a *demand* function is called a **send-controlled scheme** to reflect this fact.

4. Finally, for the fourth case we require that

   $$(\forall\langle a,b\rangle)\ [b/\mathrm{s} \notin demand(a)\ \wedge\ a/\mathrm{r} \notin demand(b)]$$

   This case eliminates the demand operation for transport tickets. A scheme with such a *demand* function is called a **strict scheme** to reflect this fact.

Receive-controlled and send-controlled schemes are of particular interest. Analysis of such schemes reveals the limitations in controlling the transport operation by a single ticket, at the destination and source respectively. Inspite of the apparent duality of receive-controlled and send-controlled self-copy schemes, their properties are significantly different. This fact was first observed by Minsky [15] in his analysis of the uniform send-receive mechanism (see section 1.2.4).

Strict schemes are of interest, since they eliminate the demand operation for transport tickets. The analysis of such schemes then reveals the utility in providing a demand operation for transport tickets. Permissive schemes are of little intrinsic interest[5] by themselves and are introduced to cover all four cases of the *demand* function.

Our strategy in characterizing the nature of flow-invariant self-copy schemes will be, first of all to study three of these four extreme cases of the *demand* function in the following order

1. Receive-controlled self-copy schemes.
2. Send-controlled self-copy schemes.
3. Strict self-copy schemes.

Having done so we will consolidate the results to apply to an arbitrary self-copy scheme. We illustrate our results, for these three extreme cases of the *demand*

[5]This is not to say that such schemes are completely unusable. Indeed, a permissive scheme may well suit the particular policy needs of a designer. However, since links can be freely established, the flow analysis problem for such schemes is trivial. If we assume there is at least one subject of every type present in the initial state (or alternatively, that the create-rules allow this situation in a derived state), then computation of the $flow^*$ function reduces to computation of the indirect flow limit function discussed in section 3.4.

function, by determining the flow-invariant property for self-copy schemes with the following *dflt* functions.

1. A scheme similar to the uniform send-receive scheme, so that there is a single subject type *a* and

$$dflt(a,a) = \{a/sc,\ a/rc\}$$

2. A scheme with two subject types, *a* and *b*, such that so that

$$dflt(a,a) = \{a/sc,\ a/rc,\ b/sc,\ b/rc\}$$
$$dflt(a,b) = \{a/sc,\ a/rc,\ b/sc,\ b/rc\}$$
$$dflt(b,a) = \{a/sc,\ a/rc,\ b/sc,\ b/rc\}$$
$$dflt(b,b) = \phi$$

   This might correspond to a situation where subjects of type *a* are active whereas subjects of type *b* are passive, as discussed in section 2.4.

We will show that both schemes are flow-invariant for the receive-controlled and strict cases, while neither is flow-invariant for the send-controlled case.

5.3.2. The Receive-Controlled Case

For a receive-controlled self-copy scheme any subject can obtain a send ticket for any other subject by a demand operation whereas receive tickets cannot be obtained in this manner, i.e., for all pairs of subject types *<a,b>*

$$b/s \in demand(a)$$
$$b/r \notin demand(a)$$

Since the *demand* function is fully specified and the value of the *can-create* relation is irrelevant, for such schemes the only parameter of concern is the *dflt* function.

Now, consider how we might demonstrate that a receive-controlled self-copy scheme is <u>not</u> flow-invariant. It would suffice to exhibit an initial state with subjects A and B, of type *a* and *b* respectively, such that for some derived state h

$$\mathrm{flowt}^h(A,B) \neq \mathrm{flowt}^0(A,B)$$

The simplest manner in which this can be achieved is when

1. The link relation in these two states differs only by a single link from A to B, i.e.,

$$\mathrm{link}^h(A,B) \wedge \sim\mathrm{link}^0(A,B)$$

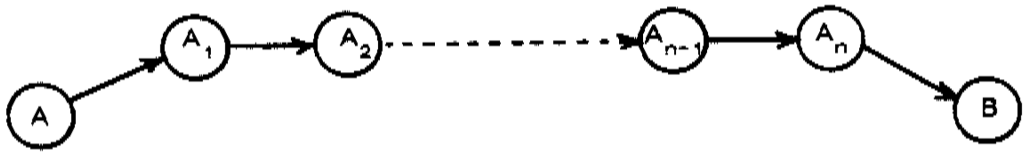
2. This additional link alters the value of the flowt function, i.e.,

$$dflt(a,b) \not\subseteq \text{flowt}^{o}(A,B)$$

To bring about the desired difference between the link^{h} and link^{o} relations it must be possible to establish link*(A,B). Subject A can obtain the B/s ticket by a demand operation. However, subject B must obtain the A/r ticket by a transport operation. By lemma 5.2 on page 119, transport of the A/r ticket must originate from the domain of A. If B gets this ticket directly from A then there is already a link from A to B prior to the transport operation. Hence, link*(A,B) must be established by moving the ticket A/r to the domain of B indirectly from the domain of A. Now this can be achieved only if the *dflt* function permits such an indirect transport. This requirement is formally stated as follows.

$$(\exists\alpha\neq\Lambda)\ [a/r \in mflt(a,\alpha,b)]$$

The simplest manner to establish link*(A,B) is then to construct an initial state with a path from A to B consisting of subjects whose types are represented by the string α. Such an initial state is shown below where we assume that $\alpha = a_1 ... a_n$ and subject A_i is of type a_i.



The derived state h differs from this initial state only by link^{h}(A,B). We would then have

$$\text{flowt}^{o}(A,B) = mflt(a,\alpha,b)$$
$$\text{flowt}^{h}(A,B) = mflt(a,\alpha,b) \cup dflt(a,b)$$

Our objective of demonstrating that the scheme is not flow-invariant will have been achieved if

$$dflt(a,b) \not\subseteq mflt(a,\alpha,b)$$

This approach to demonstrating that a receive-controlled self-copy scheme is not flow-invariant will work, provided there is a pair of subject types <*a,b*> such that

$$(\exists\alpha\neq\Lambda)\ [a/r \in mflt(a,\alpha,b) \ \wedge\ dflt(a,b) \not\subseteq mflt(a,\alpha,b)]$$

For the empty string Λ, the second term of the conjunction is trivially false. Hence, we can write the above requirement as

$$(\exists\alpha)\ [a/r \in mflt(a,\alpha,b) \ \wedge\ dflt(a,b) \not\subseteq mflt(a,\alpha,b)]$$

Conversely, our attempt will fail if we cannot find such a pair of subject types. Now, the negation of the above condition is easily seen to be

$$(\forall\alpha)\,[a/r \in mflt(a,\alpha,b) \rightarrow dflt(a,b) \subseteq mflt(a,\alpha,b)]$$

This requirement stipulates that any attempt at bypassing a path from a subject A of type *a* to a subject B of type *b*, by establishing a direct link from A to B, will not increase the value of flowt(A,B).

We are then led to the following theorem.

Theorem 5.3: A receive-controlled self-copy scheme is flow-invariant if and only if for all pairs of subject types <*a,b*>

$$(\forall\alpha)\,[a/r \in mflt(a,\alpha,b) \rightarrow dflt(a,b) \subseteq mflt(a,\alpha,b)]$$

Proof: $\rightarrow$: Consider the contrapositive statement that

For any receive-controlled self-copy scheme if there exists a pair of subject types <*a,b*> such that

$$(\exists\alpha)\,[a/r \in mflt(a,\alpha,b) \wedge dflt(a,b) \not\subseteq mflt(a,\alpha,b)]$$

then there is an initial state with subjects A and B such that for some derived state h

$$\mathrm{flowt}^h(A,B) \neq \mathrm{flowt}^0(A,B)$$

This statement follows immediately from the discussion above.

$\leftarrow$: This direction of the theorem is proved by induction on the number of link$^+$s established subsequent to the initial state. By theorem 5.1 on page 115, we can ignore the create operation and assume that all subjects are present in the initial state. Let us denote the state immediately after the first n link$^+$s are established as state n. We will show that for every n

$$(\forall\langle A,B\rangle \in SUB^0 \times SUB^0)\,[\mathrm{flowt}^n(A,B) = \mathrm{flowt}^0(A,B)]$$

For the basis case let n = 0. But then n is the initial state and the basis case is trivially true. Assume as an induction hypothesis that immediately after the first k link$^+$s are established

$$(\forall\langle A,B\rangle \in SUB^0 \times SUB^0)\,[\mathrm{flowt}^k(A,B) = \mathrm{flowt}^0(A,B)]$$

For the induction step consider the case of n = k+1. Let the nth link$^+$ established be from subject C to subject D. For a receive-controlled self-copy scheme this requires that

$$C/r \in \mathrm{dom}_T(D)$$

By the origination property, of lemma 5.2 on page 119, we must have

$$t(C)/r \in \mathrm{flowt}(C,D) \text{ prior to } \mathrm{link}(C,D)$$

By definition of the states k and k+1, the link relation in the state immediately prior to state k+1 is identical to the link relation in state k. It then follows from the above statement that

$$t(\mathrm{C})/\mathrm{r} \in \mathrm{flowt}^{k}(\mathrm{C},\mathrm{D})$$

By definition of the *mflt* and flowt functions, there must be some string α such that

$$t(\mathrm{C})/\mathrm{r} \in \mathit{mflt}(t(\mathrm{C}),\alpha,t(\mathrm{D})) \subseteq \mathrm{flowt}^{k}(\mathrm{C},\mathrm{D})$$

By assumption of the theorem it follows that

$$\mathit{dflt}(t(\mathrm{C}),t(\mathrm{D})) \subseteq \mathrm{flowt}^{k}(\mathrm{C},\mathrm{D})$$

But then, introduction of a link from C to D in state k+1 cannot affect the flowt function, and we have

$$(\forall\langle \mathrm{A},\mathrm{B}\rangle \in \mathrm{SUB}^{\mathrm{O}} \times \mathrm{SUB}^{\mathrm{O}})[\mathrm{flowt}^{k+1}(\mathrm{A},\mathrm{B}) = \mathrm{flowt}^{k}(\mathrm{A},\mathrm{B})]$$

The induction step then follows from the induction hypothesis. ■

Corollary 5.3.1: A receive-controlled self-copy scheme is flow-invariant, in the strong sense, if and only if for all pairs of subject types $\langle a,b\rangle$

$$(\forall\alpha)[a/\mathrm{r} \in \mathit{mfl}(a,\alpha,b) \Rightarrow \mathit{dfl}(a,b) \subseteq \mathit{mfl}(a,\alpha,b)]$$

Proof: Modify the proof of theorem 5.3 by replacing every occurrence of flowt by flow, and every occurrence of *_fl* by *_flt*. ■

Let us consider the *dflt* functions defined on page 123, in the context of a receive-controlled self-copy scheme.

Example 1: There is a single subject type *a* and

$$\mathit{dflt}(a,a) = \{a/\mathrm{sc},\ a/\mathrm{rc}\}$$

It follows that

$$(\forall\alpha)[\mathit{mflt}(a,\alpha,a) = \mathit{dflt}(a,a)]$$

Thus, the condition of theorem 5.3 is satisfied and this receive-controlled self-copy scheme is flow-invariant. ■

Example 2: There are two subject types *a* and *b* such that

$$\begin{aligned}\mathit{dflt}(a,a) &= \{a/\mathrm{sc},\ a/\mathrm{rc},\ b/\mathrm{sc},\ b/\mathrm{rc}\}\\ \mathit{dflt}(a,b) &= \{a/\mathrm{sc},\ a/\mathrm{rc},\ b/\mathrm{sc},\ b/\mathrm{rc}\}\\ \mathit{dflt}(b,a) &= \{a/\mathrm{sc},\ a/\mathrm{rc},\ b/\mathrm{sc},\ b/\mathrm{rc}\}\\ \mathit{dflt}(b,b) &= \phi\end{aligned}$$

It follows that for any string α and any pair of subject types

$$\langle c,d\rangle \in \{a,b\} \times \{a,b\}$$

either

$$1.\ mflt(c,\alpha,d) = \{a/sc,\ a/rc,\ b/sc,\ b/rc\}$$
$$\text{or}\ 2.\ mflt(c,\alpha,d) = \phi$$

Now in the first case it is evident that

$$dflt(c,d) \subseteq mflt(c,\alpha,d)$$

while, in the second case

$$c/r \notin mflt(c,\alpha,d)$$

In either case the condition of theorem 5.3 is satisfied and, hence, this receive-controlled self-copy scheme is flow-invariant. ■

Thus, both examples are flow-invariant for the receive-controlled case. There are a variety of receive-controlled flow-invariant self-copy schemes that can be constructed. We will study several examples in sections 5.3.6, 6.1 and 6.3.

5.3.3. The Send-Controlled Case

For a send-controlled scheme any subject can obtain a receive ticket for any other subject on demand whereas send tickets cannot be obtained in this manner, i.e., the *demand* function has the following form for all subject types a and b

$$b/\mathrm{r} \in demand(a)$$
$$b/\mathrm{s} \notin demand(a)$$

Since the *demand* function is fully specified and the value of the *can-create* relation is irrelevant, for such schemes the only parameter of concern is the *dflt* function.

Following the approach of the previous section, consider how we can demonstrate that a send-controlled self-copy scheme is <u>not</u> flow-invariant. We have seen that the simplest manner to do this is to exhibit an initial state with subjects A and B of type a and b respectively such that

1. For some derived state h we have

$$\mathrm{flowt}^{h}(A,B) \neq \mathrm{flowt}^{0}(A,B)$$

2. The link relation in these two states differs only by

$$\mathrm{link}^{h}(A,B) \wedge \sim\mathrm{link}^{0}(A,B)$$

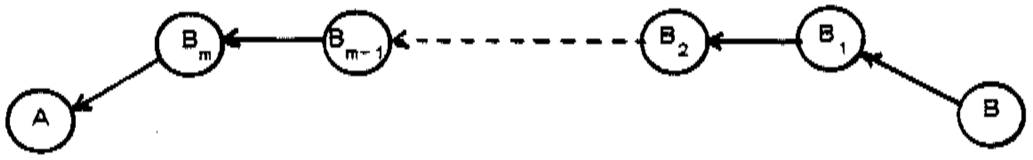
3. This additional link alters the value of the flowt function, i.e.,

$$dflt(a,b) \not\subseteq \mathrm{flowt}^{0}(A,B)$$

To bring about the desired difference between the link^{h} and link^{0} relations it must be possible to establish link$^+$(A,B). Subject B can obtain the A/r ticket by a demand operation. However, subject A must obtain the B/s ticket by a transport operation. By lemma 5.2 on page 119, transport of the B/s ticket must originate from the domain of B. Now this can be achieved only if the *dflt* function permits such a transport. This requirement is formally stated as follows.

$$(\exists\beta)\ [b/s \in mflt(b,\beta,a)]$$

The simplest manner to establish link$^+$(A,B) is then to construct an initial state with a path from B to A consisting of subjects whose types are represented by the string β. Such an initial state[6] is shown below where we assume that $\beta = b_1 \ldots b_m$ and subject B_i is of type b_i.



The derived state h would differ from this state only by link(A,B). We would then have

$$\text{flowt}^0(A,B) = \phi$$
$$\text{flowt}^h(A,B) = dflt(a,b)$$

Our objective of demonstrating that the scheme is not flow-invariant will have been achieved if $dflt(a,b) \neq \phi$.

This approach to demonstrating that a send-controlled self-copy scheme is not flow-invariant will work, provided there is a pair of subject types $<a,b>$ such that

$$(\exists\beta)\ [b/s \in mflt(b,\beta,a)\ \wedge\ dflt(a,b)\ \neq \phi]$$

Conversely, our attempt will fail if we cannot find such a pair of subject types. Now, the negation of the above condition is easily seen to be

$$(\forall\beta)\ [b/s \in mflt(b,\beta,a) \rightarrow dflt(a,b) = \phi]$$

This requirement stipulates that any attempt at using a path from a subject B of type *b* to a subject A of type *a*, to establish a link from A to B will result in a link which cannot be used to move transport tickets.

[6]If the string β happens to be empty then the sequence of links from B to A should be replaced by a single link from B to A.

We are then led to the following theorem.

Theorem 5.4: A send-controlled self-copy scheme is flow-invariant if and only if for all pairs of subject types $<a,b>$

$$(\forall\beta)\,[b/s \in \mathit{mflt}(b,\beta,a) \rightarrow \mathit{dflt}(a,b) = \phi]$$

Proof: $\rightarrow$: Consider the contrapositive statement that

For any send-controlled self-copy scheme if there exists a pair of subject types $<a,b>$ such that

$$(\exists\beta)\,[b/s \in \mathit{mflt}(b,\beta,a) \wedge \mathit{dflt}(a,b) \neq \phi]$$

then there is an initial state with subjects A and B such that for some derived state h

$$\mathrm{flowt}^h(A,B) \neq \mathrm{flowt}^0(A,B)$$

This statement follows immediately from the discussion above.

$\leftarrow$: In the discussion leading up to the theorem, we observed that for a send-controlled self-copy scheme

$$\mathrm{link}^+(A,B) \rightarrow (\exists\beta)\,[t(B)/s \in \mathit{mflt}(t(B),\beta,t(A))]$$

But then, by assumption of the theorem

$$\mathrm{link}^+(A,B) \rightarrow \mathit{dflt}(t(A),t(B)) = \phi$$

Thus, the only new links introduced cannot be used for the transport of transport tickets. Introduction of such links does not change the flowt function and, hence, the scheme is flow-invariant. ■

This proof also provides the following corollary.

Corollary 5.4.1: A send-controlled self-copy scheme is flow-invariant if and only if

$$(\forall<A,B>)\,[\mathrm{link}^+(A,B) \rightarrow \mathit{dflt}(t(A),t(B)) = \phi]$$

■

Thus, the dynamic aspect of such schemes is limited to the establishment of new links over which transport tickets cannot be moved. The reason why such schemes are flow-invariant is then self evident. Further, with the stronger definition of flow-invariant, we have the following result.

Corollary 5.4.2: A send-controlled self-copy scheme is flow-invariant, in the strong sense, if and only if for all pairs of subject types $<a,b>$

$$(\forall<A,B>)\,[\mathrm{link}^+(A,B) \rightarrow \mathit{dfl}(t(A),t(B)) = \phi]$$

■

Thus, with the stronger definition, the only links which can be established are completely useless, since no tickets can be transported over them.

Let us consider the *dflt* functions defined on page 123, in the context of a send-controlled self-copy scheme.

Example 1: There is a single subject type *a* and

$$dflt(a,a) = \{a/\mathrm{sc},\ a/\mathrm{rc}\}$$

It follows that

$$(\forall\alpha)\ [a/\mathrm{s} \in mflt(a,\alpha,a)\ \wedge\ dflt(a,a) \neq \phi]$$

But then the condition of theorem 5.4 is violated, and this send-controlled self-copy scheme is not flow-invariant. ■

Example 2: There are two subject types *a* and *b* such that

$$\begin{aligned} dflt(a,a) &= \{a/\mathrm{sc},\ a/\mathrm{rc},\ b/\mathrm{sc},\ b/\mathrm{rc}\} \\ dflt(a,b) &= \{a/\mathrm{sc},\ a/\mathrm{rc},\ b/\mathrm{sc},\ b/\mathrm{rc}\} \\ dflt(b,a) &= \{a/\mathrm{sc},\ a/\mathrm{rc},\ b/\mathrm{sc},\ b/\mathrm{rc}\} \\ dflt(b,b) &= \phi \end{aligned}$$

It follows that for any string α which consists of one or more occurrences of *a*

$$a/\mathrm{s} \in mflt(a,\alpha,a)\ \wedge\ dflt(a,a) \neq \phi$$

But then the condition of theorem 5.4 is violated, and this send-controlled self-copy scheme is not flow-invariant. ■

Thus, both examples are not flow-invariant for the send-controlled case.

5.3.4. The Strict Case

Next consider the case of strict self-copy schemes. For such schemes there is no authorization to obtain transport tickets by demand, i.e., for all subject types *a* and *b*

$$\begin{aligned} b/\mathrm{r} &\notin demand(a) \\ b/\mathrm{s} &\notin demand(a) \end{aligned}$$

Once again, the only parameter of concern is the *dflt* function.

Following the approach of the previous two sections, consider how we might demonstrate that a strict self-copy scheme is <u>not</u> flow-invariant. As before, the simplest manner to demonstrate this is to exhibit an initial state with subjects A and B, of type *a* and *b* respectively, such that

1. For some derived state h we have

$$\mathrm{flowt}^{h}(A,B) \neq \mathrm{flowt}^{0}(A,B)$$

2. The link relation in these two states differs only by

$$\mathrm{link}^h(A,B) \wedge \sim\mathrm{link}^o(A,B)$$

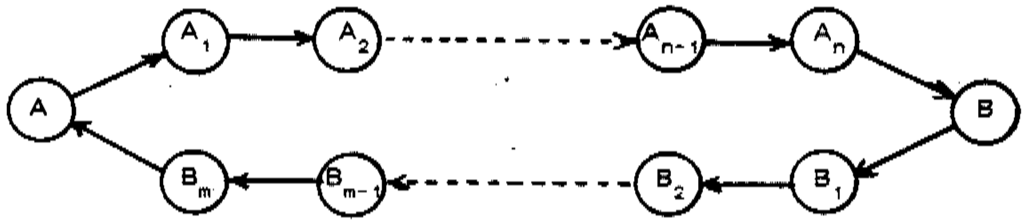
3. This additional link alters the value of the flowt function, i.e.,

$$dflt(a,b) \nsubseteq \mathrm{flowt}^o(A,B)$$

To bring about the desired difference between the link^h and link^o relations it must be possible to establish link*(A,B). To achieve this the ticket A/r must be placed in the domain of B by a transport operation and the ticket B/s must be placed in the domain of A by a transport operation. By the same reasoning used in the previous two sections, these two requirements respectively translate into the following conditions on the *mflt* function.

$$(\exists\alpha\neq\Lambda)\,[a/r \in mflt(a,\alpha,b)]$$
$$(\exists\beta)\,[b/s \in mflt(b,\beta,a)]$$

The simplest manner to establish link*(A,B) is then to construct an initial state with a path from A to B consisting of subjects whose types are represented by the string α and a path from B to A consisting of subjects whose types are represented by the string β. Such an initial state[7] is shown below, where we assume that $\alpha = a_1 ... a_n$, subject A_i is of type a_i, $\beta = b_1 ... b_m$, and subject B_i is of type b_i.



The derived state h would differ from this state only by link(A,B). We would then have

$$\mathrm{flowt}^o(A,B) = mflt(a,\alpha,b)$$
$$\mathrm{flowt}^h(A,B) = mflt(a,\alpha,b) \cup dflt(a,b)$$

Our objective of demonstrating that the scheme is not flow-invariant will have been achieved if

$$dflt(a,b) \nsubseteq mflt(a,\alpha,b)$$

[7] If the string β happens to be empty then the sequence of links from B to A should be replaced by a single link from B to A.

This approach to demonstrating that a strict self-copy scheme is not flow-invariant will work, provided there is a pair of subject types <*a,b*> such that

$$(\exists\alpha\neq\Lambda)\ (\exists\beta)\ [a/\mathrm{r} \in mflt(a,\alpha,b)\ \wedge\ b/\mathrm{s} \in mflt(b,\beta,a)\ \wedge\ dflt(a,b) \not\subseteq mflt(a,\alpha,b)]$$

For the empty string Λ, the third term of the conjunction is trivially false and, hence, we can write the above requirement as

$$(\exists\alpha)\ (\exists\beta)\ [a/\mathrm{r} \in mflt(a,\alpha,b)\ \wedge\ b/\mathrm{s} \in mflt(b,\beta,a)\ \wedge\ dflt(a,b) \not\subseteq mflt(a,\alpha,b)]$$

Conversely, our attempt will fail if we cannot find such a pair of subject types. Now, the negation of the above condition is easily seen to be

$$(\forall\alpha)\ (\forall\beta)\ [a/\mathrm{r} \in mflt(a,\alpha,b)\ \wedge\ b/\mathrm{s} \in mflt(b,\beta,a) \rightarrow dflt(a,b) \subseteq mflt(a,\alpha,b)]$$

This requirement stipulates that any attempt at bypassing a path from a subject A of type *a* to a subject B of type *b*, by establishing a direct link from A to B, will not increase the value of flowt(A,B).

We are then led to the following theorem.

Theorem 5.5: A strict self-copy scheme is flow-invariant if and only if for all pairs of subject types <*a,b*>

$$(\forall\alpha)\ (\forall\beta)\ [a/\mathrm{r} \in mflt(a,\alpha,b)\ \wedge\ b/\mathrm{s} \in mflt(b,\beta,a) \rightarrow dflt(a,b) \subseteq mflt(a,\alpha,b)]$$

Proof: $\rightarrow$: Consider the contrapositive statement that

For any strict self-copy scheme if there exists a pair of subject types <*a,b*> such that

$$(\exists\alpha)\ (\exists\beta)\ [a/\mathrm{r} \in mflt(a,\alpha,b)\ \wedge\ b/\mathrm{s} \in mflt(b,\beta,a)\ \wedge\ dflt(a,b) \not\subseteq mflt(a,\alpha,b)]$$

then there exists an initial state with subjects A and B such that for some derived state h

$$\mathrm{flowt}^{h}(A,B) \neq \mathrm{flowt}^{0}(A,B)$$

This statement follows immediately from the discussion above.

$\leftarrow$: This direction of the theorem is proved by induction on the number of link^{+}s established subsequent to the initial state. By theorem 5.1 on page 115, we can ignore the create operation and assume that all subjects are present in the initial state. Let us denote the state immediately after the first n link^{+}s are established as state n. We will show that for every n

$$(\forall\langle A,B\rangle \in SUB^{0} \times SUB^{0})\ [\mathrm{flowt}^{n}(A,B) = \mathrm{flowt}^{0}(A,B)]$$

For the basis case let n = 0. But then n is the initial state and the basis case is trivially true. Assume as an induction hypothesis that immediately after the first k link$^+$s are established

$$(\forall \langle A,B\rangle \in SUB^O \times SUB^O)[\mathrm{flowt}^k(A,B) = \mathrm{flowt}^O(A,B)]$$

For the induction step consider the case of n = k+1. Let the nth link$^+$ established be from subject C to subject D. For a strict self-copy scheme this requires that

$$[D/s \in \mathrm{dom}_T(C)] \wedge [C/r \in \mathrm{dom}_T(D)]$$

By the origination property, of lemma 5.2 on page 119, we then have

$$[t(D)/s \in \mathrm{flowt}(D,C)] \wedge [t(C)/r \in \mathrm{flowt}(C,D)]$$
$$\text{prior to link}(C,D)$$

By definition of the states k and k+1, the link relation in the state immediately prior to state k+1 is identical to the link relation in state k. It then follows from the above statement that

$$[t(D)/s \in \mathrm{flowt}^k(D,C)] \wedge [t(C)/r \in \mathrm{flowt}^k(C,D)]$$

By definition of the *mflt* function, there must be some string α such that

$$t(C)/r \in mflt(t(C),\alpha,t(D)) \subseteq \mathrm{flowt}^k(C,D)$$

and some string β such that

$$t(D)/s \in mflt(t(D),\beta,t(C))$$

By assumption of the theorem we then have

$$dflt(t(C),t(D)) \subseteq \mathrm{flowt}^k(C,D)$$

But then, introduction of a link from C to D in state k+1 cannot affect the flowt function, and we have

$$(\forall \langle A,B\rangle \in SUB^O \times SUB^O)[\mathrm{flowt}^{k+1}(A,B) = \mathrm{flowt}^k(A,B)]$$

The induction step then follows from the induction hypothesis. ■

We also have the following corollaries.

Corollary 5.5.1: For a given *dflt* function, if the receive-controlled self-copy scheme is flow-invariant then the strict self-copy scheme is flow-invariant.

Proof: If the receive-controlled scheme is flow-invariant then, by theorem 5.3 on page 125, for every pair of subject types *<a,b>*

$$(\forall\alpha)[a/r \in mflt(a,\alpha,b) \rightarrow dflt(a,b) \subseteq mflt(a,\alpha,b)]$$

By inspection, the condition of theorem 5.5 is then satisfied. ■

Corollary 5.5.2: For a given *dflt* function, if the send-controlled self-copy scheme is flow-invariant then the strict self-copy scheme is flow-invariant.

Proof: If the send-controlled scheme is flow-invariant then, by theorem 5.4 on page 129, for every pair of subject types $<a,b>$

$$(\forall\alpha)\,[a/s \in \mathit{mflt}(a,\alpha,b) \rightarrow \mathit{dflt}(b,a) = \phi]$$

By inspection, the condition of theorem 5.5 is then satisfied. ∎

Now consider the *dflt* functions defined on page 123, in the context of a strict self-copy scheme. On page 126, we saw that for the receive-controlled self-copy case both examples were flow-invariant. It follows, from corollary 5.5.1, that both examples are also flow-invariant for the strict self-copy case.

5.3.5. The General Case

In the three preceding sections, we characterized the nature of flow-invariant self-copy schemes with respect to three extreme cases of the *demand* function. By consolidating these results, we now obtain the following characterization of a flow invariant self-copy scheme.

Theorem 5.6: A self-copy scheme is flow-invariant if and only if for all pairs of subject types $<a,b>$

1. **If** $[b/s \in \mathit{demand}(a) \wedge a/r \notin \mathit{demand}(b)]$ **then**
   $(\forall\alpha)\,[a/r \in \mathit{mflt}(a,\alpha,b) \rightarrow \mathit{dflt}(a,b) \subseteq \mathit{mflt}(a,\alpha,b)]$

2. **If** $[b/s \notin \mathit{demand}(a) \wedge a/r \in \mathit{demand}(b)]$ **then**
   $(\forall\beta)\,[b/s \in \mathit{mflt}(b,\beta,a) \rightarrow \mathit{dflt}(a,b) = \phi]$

3. **If** $[b/s \notin \mathit{demand}(a) \wedge a/r \notin \mathit{demand}(b)]$ **then**
   $(\forall\alpha)\,(\forall\beta)\,[a/r \in \mathit{mflt}(a,\alpha,b) \wedge b/s \in \mathit{mflt}(b,\beta,a) \rightarrow \mathit{dflt}(a,b) \subseteq \mathit{mflt}(a,\alpha,b)]$

4. **If** $[b/s \in \mathit{demand}(a) \wedge a/r \in \mathit{demand}(b)]$ **then** $\mathit{dflt}(a,b) = \phi$

Proof: Note that exactly one of these four conditions will apply for a given pair of subject types $<a,b>$.

$\rightarrow$: This direction of the theorem is established by proving the contrapositive statement. Let us refer to the four conditions of the theorem by their designated numbers. The contrapositive statement is that

For any self-copy scheme if there exists a pair of subject types $<a,b>$ such that

$$\sim[1] \vee \sim[2] \vee \sim[3] \vee \sim[4]$$

then there exists an initial state with subjects A and B such that for some derived state h

$$flowt^h(A,B) \neq flowt^0(A,B)$$

Given ~[1], ~[2], or ~[3] we can construct an appropriate initial state by using the construction of sections 5.3.2, 5.3.3, or 5.3.4 respectively. Given ~[4] construction of such an initial state is trivial.

$\Leftarrow$: This direction of the theorem is proved by induction on the number of link$^+$s established subsequent to the initial state. The induction is similar to the induction used in the proofs of theorems 5.3 and 5.5 on pages 125 and 132 respectively. The difference is that the necessary condition for establishing link$^+$(C,D) is determined by the nature of the *demand* function with respect to the ability of C and D to respectively obtain the D/s and C/r tickets by a demand operation. There are four possibilities here and, hence, the induction step has four cases corresponding to the four cases of this theorem. ■

In order to obtain an algorithm for checking the conditions of theorem 5.6, we can eliminate all references to the *mflt* function by rewriting these conditions in terms of the *dflt* and *iflt* functions. This transformation is shown in appendix C.6 and results in an algorithm with a cost of $O(|T_S|^4)$.

We have seen that the conditions under which a send-controlled self-copy scheme is flow-invariant, are rather restrictive. On the other hand, the conditions under which a receive-controlled self-copy scheme is flow-invariant, are not too restrictive. For this reason, our examples of flow-invariant self-copy schemes are, for the most part, limited to the receive-controlled case. We do have the following result, by which our examples extend to a more general restriction on the *demand* function.

Corollary 5.6.1: For a given *dflt* function, if the receive-controlled scheme is flow-invariant than a self-copy scheme with the same *dflt* function and a *demand* function such that

$$(\forall\langle a,b\rangle)\ [b/r \notin demand(a)]$$

is also flow-invariant[8].

Proof: With the stated constraint on the *demand* function only the first

[8]Observe that corollary 5.5.1, on page 133 is a special case of this corollary.

and third conditions of theorem 5.6 are relevant. It follows that a self-copy scheme, with the stated constraint on the *demand* function, is flow-invariant if and only if for all $\langle a,b \rangle$

1. If $[b/\mathrm{s} \in demand(a)]$ **then**
$(\forall\alpha)\,[a/\mathrm{r} \in mflt(a,\alpha,b) \rightarrow dflt(a,b) \subseteq mflt(a,\alpha,b)]$

2. If $[b/\mathrm{s} \notin demand(a)]$ **then**
$(\forall\alpha)\,(\forall\beta)\,[a/\mathrm{r} \in mflt(a,\alpha,b) \wedge b/\mathrm{s} \in mflt(b,\beta,a) \rightarrow dflt(a,b) \subseteq mflt(a,\alpha,b)]$

Now for a given *dflt* function if the receive-controlled self-copy scheme is flow-invariant then for all pairs $\langle a,b \rangle$ we have

$$(\forall\alpha)\,[a/\mathrm{r} \in mflt(a,\alpha,b) \rightarrow dflt(a,b) \subseteq mflt(a,\alpha,b)]$$

But then, by inspection, it is evident that the **then** clauses of conditions 1 and 2 are true for every pair $\langle a,b \rangle$. ■

It is the ability to demand receive tickets which restricts us in constructing flow-invariant schemes. Once this ability is eliminated, there are a surprising variety of flow-invariant schemes which can be constructed, as discussed in sections 5.3.6, 6.1 and 6.3.

Moreover, note that the converse of corollary 5.6.1 is not true. Thus, in principle, restricting the ability to demand send tickets allows for a larger class of flow-invariant schemes than obtained for the receive-controlled case. However, the intuitive constraints that we have been able to devise actually result in self-copy schemes which are flow-invariant for the receive-controlled case. It is an interesting question, as to whether there are intuitively appealing *dflt* functions which lead to self-copy schemes which are flow-invariant for (say) the strict case but not for the receive-controlled case.

5.3.6. Nested Self-Copy Schemes

In this section we investigate a class of flow-invariant self-copy schemes. Due to theorem 5.1 on page 115, the value of the *can-create* relation is not relevant in determining the flow-invariant property for self-copy schemes. Since our focus here is on this property, we will ignore the existence of this design parameter. Similarly, we can ignore specification of the *dfl* function for all types of inert tickets. In order to specify a class of flow-invariant self-copy schemes it then suffices to describe the nature of the *dflt* and *demand* functions. As mentioned earlier, we will restrict our discussion to the receive-controlled case so that the *demand* function is specified as follows.

$$(\forall\langle a,b\rangle)\ [b/s \in demand\,(a)]$$

The only parameter of concern is then the *dflt* function.

We begin by considering a simple constraint on the *dflt* function, and lead up to a more complex structure. Specifically, we begin with an all-or-nothing situation by limiting the values of the *dflt* function to either allow transport of all types of transport tickets or of none. This condition is expressed as follows.

Definition 5.9: A scheme is said to be a **homogeneous scheme** provided the *dflt* function satisfies the following constraint.

$$dflt\colon \mathbf{T}_S \times \mathbf{T}_S \rightarrow \{\phi,\ \mathbf{T}_S \times \mathbf{R}_T\}$$

■

We will discuss these schemes in more detail in section 6.1. For the moment we have the following result.

Theorem 5.7: Every receive-controlled self-copy homogeneous scheme is flow-invariant.

Proof: For such schemes it is obvious that for all pairs of subject types *<a,b>* and for every string α

$$dflt\,(a,b) = \phi \text{ or } \mathbf{T}_S \times \mathbf{R}_T$$
$$mflt\,(a,\alpha,b) = \phi \text{ or } \mathbf{T}_S \times \mathbf{R}_T$$

But then,

$$a/r \in mflt\,(a,\alpha,b) \Rightarrow mflt\,(a,\alpha,b) = \mathbf{T}_S \times \mathbf{R}_T$$
$$\Rightarrow dflt\,(a,b) \subseteq mflt\,(a,\alpha,b)$$

Thus, the condition of theorem 5.3 on page 125 is satisfied. ■

It is instructive to consider what happens if we slightly modify the definition of a homogeneous scheme. In all cases we shall retain the two valued nature of the *dflt* function. However, we will modify the strict all-or-nothing nature of our definition.

In the first modification, we allow for a situation where transport tickets for certain subject types are immobile. Let the set of subject types whose tickets can be transported be denoted as $\mathbf{T}_S^{\prime}$, where $\mathbf{T}_S^{\prime}$ is some proper subset of $\mathbf{T}_S$. The maximum value of the *dflt* function is then $\mathbf{T}_S^{\prime} \times \mathbf{R}_T$, while the minimum value is ϕ, i.e.,

$$dflt\colon \mathbf{T}_S \times \mathbf{T}_S \rightarrow \{\phi,\ \mathbf{T}_S^{\prime} \times \mathbf{R}_T\}$$

The proof of theorem 5.7 works for this case also, by simply replacing $\mathbf{T}_S$

everywhere in the proof by T'_S. Thus, the above modification preserves the flow-invariant property.

For our next modification, we assume that tickets for certain types of subjects are universally mobile. Again, let T'_S be some proper subset of T_S. Consider a *dflt* function of the following form.

$$dflt\colon T_S \times T_S \rightarrow \{T'_S \times R_T,\ T_S \times R_T\}$$

Such a scheme has an all-or-some aspect, rather than the all-or-nothing aspect of homogeneous schemes. In this case, it is easy to construct an example scheme which is not flow-invariant. Specifically, let

$$\begin{aligned} T_S &= \{a,b,c\} \\ T'_S &= \{a\} \\ dflt(a,b) &= T'_S \times R_T \\ dflt(b,c) &= T'_S \times R_T \end{aligned}$$

where all other values of the *dflt* function are $T_S \times R_T$. These values violate the condition of theorem 5.3 on page 125, since

$$[a/r \in mflt(a,b,c) = T'_S \times R_T] \wedge [dflt(a,c) = T_S \times R_T]$$

Thus, this modification does not preserve the flow-invariant property.

However, if we impose an additional condition we will get a flow-invariant scheme. Specifically, consider as before that

$$dflt\colon T_S \times T_S \rightarrow \{T'_S \times R_T,\ T_S \times R_T\}$$

but, with the additional constraint that

$$a \in T'_S \rightarrow dflt(a,b) = T'_S \times R_T$$

The idea here is that, since transport tickets for subjects in the set T'_S are universally mobile, by curtailing the ability of such subjects to move transport tickets there will be a tighter control on the evolution of the flowt function. There are two cases to consider, in determining whether such a scheme is flow-invariant. First consider a subject type a from the set T'_S. Then, we have for all pairs $<a,b>$ and every string α that

$$\begin{aligned} a \in T'_S &\rightarrow [dflt(a,b) = T'_S \times R_T] \wedge [mflt(a,\alpha,b) = T'_S \times R_T] \\ &\rightarrow dflt(a,b) = mflt(a,\alpha,b) \end{aligned}$$

Next, consider a subject type a not in the set T'_S. Then, we have for all pairs $<a,b>$ and every string α that

$$a \notin T'_S \rightarrow [a/r \in mflt(a,\alpha,b) \rightarrow mflt(a,\alpha,b) = T_S \times R_T]$$
$$\rightarrow [a/r \in mflt(a,\alpha,b) \rightarrow dflt(a,b) \subseteq mflt(a,\alpha,b)]$$

In both cases, the condition of theorem 5.3 on page 125 is satisfied.

Based on the same idea, we can construct a variety of flow-invariant schemes. In the example above, the *dflt* function was restricted to being two valued. Let us first eliminate this constraint but in a structured manner.

Definition 5.10: A family of subsets of T_S

$$\{T_i : i=1,\ldots,n \mid T_i \subseteq T_S\}$$

is said to be **nested** provided

1. $T_S = T_1 \cup \ldots \cup T_n$
2. $(\forall\langle i,j\rangle) [[T_i \cap T_j = \phi] \vee [T_i \subseteq T_j] \vee [T_j \subseteq T_i]]$

■

We will restrict the range of the *dflt* function so that

$$dflt\colon T_S \times T_S \rightarrow \{T_i \times R_T \mid i=1,\ldots,n\}$$

for some nested family of subsets of T_S. If we do not impose any further restriction, we know from our earlier discussion that it is possible to construct schemes which are not flow-invariant. We will formulate an additional constraint, similar to what we did for the two valued case discussed above. In order to do so, we introduce the following notion.

Definition 5.11: Given a nested family of subsets of T_S we say that a subject type a is T_i**-mobile** if

$$[a \in T_i] \wedge (\forall j) [a \notin T_j - T_i]$$

■

Due to the nested nature of $\{T_1,\ldots,T_n\}$ there is a unique T_i for which any a is T_i-mobile. This leads to the following definition.

Definition 5.12: A scheme is said to be a **nested scheme** provided

1. $\{T_i : i=1,\ldots,n\}$ is a nested family of subsets of T_S
2. The *dflt* function satisfies the following constraint
   1. $dflt\colon T_S \times T_S \rightarrow \{T_i \times R_T \mid i=1,\ldots,n\}$
   2. a is T_i-mobile $\rightarrow (\forall b) [dflt(a,b) \subseteq T_i \times R_T]$

■

We then have the following result.

Theorem 5.8: Every receive-controlled nested self-copy scheme is flow-invariant.

Proof: Consider any subject type a which is T_i-mobile. It immediately follows that for every b and every α

$$a/r \in mflt(a,\alpha,b) \rightarrow mflt(a,\alpha,b) = T_i \times R_T$$

By definition of a nested scheme, for every b we have

$$dflt(a,b) \subseteq T_i \times R_T$$

The condition of theorem 5.3 on page 125 is then satisfied. ■

5.4. SUMMARY

We began this chapter by introducing two notions of a flow-invariant scheme. In the stronger sense, a flow-invariant scheme does not allow any change in the flow function, with respect to the initial set of subjects. In the weaker sense, the invariance is only with respect to the flow of transport tickets. In the former case, it is trivial to compute the flow* function, while in the latter case the cost of computing the flow† function is substantially reduced. Our discussion in the rest of the chapter was in the context of the weaker notion of flow-invariant, but our results are easily modified to apply to the stronger notion.

In section 5.1 we showed that, under some rather general conditions on the create rules, the value of the *can-create* relation is not relevant in determining whether a scheme is flow-invariant. This fact considerably simplifies our analysis, since we can then assume, without any loss of generality, that the *can-create* relation is empty; thereby effectively eliminating the create operation.

In section 5.2 we defined the class of self-copy schemes by constraining some of the design parameters. In particular, constraints were imposed on the *demand* function and the initial state and a specific create rule was adopted. The nature of these constraints led to the name self-copy, since the major constraint was on the occurrence of transport tickets with the copy flag.

In section 5.3 we obtained necessary and sufficient conditions for a self-copy scheme to be flow-invariant. The cost of testing these conditions for an arbitrary self-copy scheme is $O(|T_S|^4)$. Our strategy in developing these conditions was to first study extreme cases of the *demand* function. This study revealed that the properties are significantly different in these extreme cases, thus extending the observation first made by Minsky [15] in the context of the uniform send-receive mechanism. Finally, we demonstrated there a variety of flow-invariant self-copy schemes.

CHAPTER 6
FURTHER ASPECTS OF SELF-COPY SCHEMES

The preceding three chapters of this thesis have been concerned with the flow-analysis problem. In this chapter we discuss some other analysis issues in the context of self-copy schemes. The class of self-copy schemes is defined by imposing constraints on the initial state and the *demand* function and adopting a specific create-rule. We can define sub-classes of self-copy schemes by further constraints on the design parameters. Indeed, certain analysis issues can be formulated only in the context of such sub-classes.

In section 6.1 we study a class of self-copy schemes called **homogeneous schemes**. We introduced this class in the previous chapter. Here, we study it in more detail.

In section 6.2 we return to the general class of self-copy schemes. We now study the nature of the $link^+$ relation. Specifically, we characterize the types of subjects between which a $link^+$ may be established.

Finally in section 6.3 we study another class of self-copy schemes called **unique mediator schemes**. The analysis issues raised here are considerably different than those raised in the context of homogeneous schemes.

6.1. HOMOGENEOUS SCHEMES

The class of homogeneous schemes, introduced in section 5.3.6, is defined by constraining the *dflt* function to be two valued as follows.

$$dflt\colon T_S \times T_S \rightarrow \{\phi,\ T_S \times R_T\}$$

Thus, for a given pair of subject types *a* and *b*, the *dflt* function either permits transport of all transport tickets or of none.

6.1.1. Clusters of Subjects

It is evident that, for homogeneous schemes, not only the *dflt* function but also the flowt function is two valued. This fact leads us to introduce the following notion.

Definition 6.1: A **cluster** of subjects is a largest subset of SUB^O, such that for every pair of subjects A and B in the cluster

$$[flowt^*(A,B) = T_S \times R_T] \wedge [flowt^*(B,A) = T_S \times R_T]$$

■

By definition of the flowt function, it is evident that for homogeneous schemes

$$[flowt^k(A,B) = T_S \times R_T] \wedge [flowt^k(B,C) = T_S \times R_T] \Longrightarrow [flowt^k(A,C) = T_S \times R_T]$$

Moreover, for every state k and every subject A

$$flowt^k(A,A) = T_S \times R_T$$

Thus, the definition of a cluster amounts to the symmetric closure of a transitive and reflexive relation, whose characteristic function is defined by: <u>if</u> $flowt^*(A,B) = T_S \times R_T$ <u>then</u> 1 <u>else</u> 0. Clearly then, a cluster is an equivalence class of subjects and each subject belongs to a unique cluster.

Clusters play an important role in the analysis of homogeneous schemes. We will often need to refer to an individual subject, as well as the cluster to which this subject belongs. In order to do so conveniently, we introduce the following notation.

Definition 6.2: For every homogeneous scheme and every initial state define the associated **cluster** function

$$cls: SUB^O \rightarrow \textbf{power-set}(SUB^O)$$

by

$$cls(A) = \{B \in SUB^O \mid B \text{ is in the same cluster as } A\}$$

■

The $flowt^*$ function induces a natural partial ordering on clusters defined as follows.

Definition 6.3: Define the partial ordering < on clusters by

$$cls(A) < cls(B)$$
$$\Leftrightarrow$$
$$[flowt^*(A,B) = T_S \times R_T] \wedge [flowt^*(B,A) \neq T_S \times R_T]$$

If two distinct clusters are not comparable with respect to this partial ordering we say that cls(A) <> cls(B). ∎

6.1.2. The Link Relation

In this section we study properties of the link relation for self-copy homogeneous schemes. We need one final bit of notation before commencing our analysis.

Definition 6.4: For every initial state, define the associated binary relation

$$link^* \subseteq SUB^O \times SUB^O$$

by

$$link^*(A,B) \Leftrightarrow [link^O(A,B) \vee link^+(A,B)]$$

∎

The following theorem, then illustrates the crucial role of clusters.

Theorem 6.1: For every self-copy homogeneous scheme

$$[cls(A) = cls(B)] \Rightarrow [link^*(A,B) \wedge link^*(B,A)]$$

Proof: In conjunction with the self-reference assumption, the flowt* function enables transfer of the tickets required (if any) for establishing these links. ∎

Thus, within a cluster subjects can establish direct links with each other. Due to this constraint, a homogeneous scheme is suitable for a situation where the subjects can be partitioned into mutually exclusive components, corresponding to the clusters, such that the transport of tickets within a component is permissive.

Different classes of homogeneous schemes differ with respect to the policy for transport of tickets between subjects in distinct clusters. A comparative study of the policies, implemented by various self-copy homogeneous schemes, must necessarily concentrate on the patterns of links which can be established between subjects in different clusters. Let us refer to a link from a subject in one cluster to a subject in a different cluster as an **inter-cluster link**. In this section we characterize the nature of inter-cluster links in terms of the partial ordering on clusters.

The following theorem expresses the nature of links between subjects in incomparable clusters.

Theorem 6.2: For every homogeneous scheme and every initial state if cls(A) <> cls(B) then

$$\mathrm{link}^*(A,B) \Longleftrightarrow \mathrm{link}^{\circ}(A,B) \vee [t(B)/s \in \mathit{demand}(t(A)) \wedge t(A)/r \in \mathit{demand}(t(B))]$$

Proof: $\Leftarrow$: This direction is trivial, since it is a simple matter to establish a link from A to B if it does not already exist. Both subjects merely have to obtain the necessary tickets by a demand operation.

$\Rightarrow$: By definition, $\mathrm{link}^*(A,B)$ requires that either $\mathrm{link}^{\circ}(A,B)$ or $\mathrm{link}^+(A,B)$. Consider the latter possibility. Since the initial state is balanced, $\mathrm{link}^+(A,B)$ requires that A and B respectively obtain the tickets B/s and A/r in some derived state. There are two possibilities here, viz.,

1. Both A and B can obtain the required ticket by a demand operation.
2. At least one of A or B must obtain the required ticket by a transport operation.

Stated formally, we have argued that, for a self-copy scheme, $\mathrm{link}^+(A,B)$ implies

$$[t(B)/s \in \mathit{demand}(t(A)) \wedge t(A)/r \in \mathit{demand}(t(B))] \vee [A/r \in \mathrm{dom}_T(B) \vee B/s \in \mathrm{dom}_T(A)]$$

Consider the latter possibility. By the origination property, of lemma 5.2 on page 119, we have

$$A/r \in \mathrm{dom}_T(B) \rightarrow \mathrm{flowt}^*(A,B) \neq \phi$$

$$B/s \in \mathrm{dom}_T(A) \rightarrow \mathrm{flowt}^*(B,A) \neq \phi$$

In either case, we cannot have cls(A) <> cls(B) in violation of our assumption. The theorem follows immediately. ∎

Theorem 6.2 states that the only links between subjects in incomparable clusters are those present in the initial state or those which can be established purely by a demand operation.

We next characterize the nature of links between subjects which are in different but comparable clusters.

Theorem 6.3: For every homogeneous scheme if cls(A) < cls(B) then

1. $\text{link}^*(A,B) \leftrightarrow [\text{link}^O(A,B) \vee t(B)/s \in \textit{demand}(t(A))]$
2. $\text{link}^*(B,A) \leftrightarrow [\text{link}^O(B,A) \vee t(B)/r \in \textit{demand}(t(A))]$

Proof: We will prove assertion 1 of the theorem and the proof of assertion 2 is quite similar.

$\leftarrow$: This direction is trivial, since it is a simple matter to establish link(A,B) if it does not already exist. In particular, A can obtain the B/s ticket by a demand operation and the flowt* function authorizes transport of the A/r ticket from A to B.

$\rightarrow$: It suffices to show that

$$\text{link}^+(A,B) \rightarrow t(B)/s \in \textit{demand}(t(A))$$

Now in order to establish link$^+$(A,B), the ticket B/s must be placed in the domain of A. If A has this ticket in the initial state then by the balanced state assumption B has the A/r ticket. But then link^O(A,B), so that link$^+$(A,B) is not possible. Subject A cannot obtain the ticket B/s by transport, as this would require flowt*(B,A) $\neq \phi$ in contradiction to our assumption that cls(A) < cls(B). Hence, the only way for A to obtain this ticket is by a demand operation. ∎

Recall that in our analysis of the flow-invariant property for self-copy schemes we defined four extreme cases of the demand function as follows

1. For permissive schemes

$$(\forall\langle a,b\rangle)\,[b/s \in \textit{demand}(a) \wedge a/r \in \textit{demand}(b)]$$

2. For receive-controlled schemes

$$(\forall\langle a,b\rangle)\,[b/s \in \textit{demand}(a) \wedge a/r \notin \textit{demand}(b)]$$

3. For send-controlled schemes

$$(\forall\langle a,b\rangle)\,[b/s \notin \textit{demand}(a) \wedge a/r \in \textit{demand}(b)]$$

4. For strict self-copy schemes

$$(\forall\langle a,b\rangle)\,[b/s \notin \textit{demand}(a) \wedge a/r \notin \textit{demand}(b)]$$

We observed that permissive schemes are not very interesting and were introduced for the sake of completeness. With respect to the link relation it is clear that for any permissive scheme

$$(\forall\langle A,B\rangle)\,[\text{link}^*(A,B) \wedge \text{link}^*(B,A)]$$

Let us consider the other three extreme cases of the *demand* function in the context of homogeneous schemes. Specifically we will interpret the nature of inter–cluster links for these three cases.

Theorem 6.4: For receive–controlled homogeneous schemes

1. If cls(A) <> cls(B) then

$$\text{link}^*(A,B) \leftrightarrow \text{link}^O(A,B)$$

2. If cls(A) < cls(B) then

$$\text{link}^*(A,B) \wedge [\text{link}^*(B,A) \leftrightarrow \text{link}^O(B,A)]$$

Proof: Immediate from theorems 6.2 and 6.3. ∎

The policy, with respect to inter–cluster links, for any receive–controlled homogeneous scheme is then as follows.

1. The only possible links between subjects in incomparable clusters are those which were present in the initial state.

2. Links between subjects in distinct but comparable clusters can always be established in one direction but can exist in the other direction only if they were present in the initial state.

Next we see that the policy, with respect to inter–cluster links, for send–controlled schemes is in a sense the dual of the policy for receive–controlled homogeneous schemes. Specifically we have the following theorem.

Theorem 6.5: For send–controlled homogeneous schemes

1. If cls(A) <> cls(B) then

$$\text{link}^*(A,B) \leftrightarrow \text{link}^O(A,B)$$

2. If cls(A) < cls(B) then

$$\text{link}^*(B,A) \wedge [\text{link}^*(A,B) \leftrightarrow \text{link}^O(A,B)]$$

Proof: Immediate from theorems 6.2 and 6.3. ∎

Thus the policy, with respect to inter–cluster links, for any send–controlled scheme again has a one way permissiveness with respect to links between subjects in different, but comparable, clusters and a rigidity with respect to links in the other direction.

Finally consider the case of strict homogeneous schemes.

Theorem 6.6: For strict homogeneous schemes if cls(A) $\neq$ cls(B) then

$$\text{link}^{*}(A,B) \leftrightarrow \text{link}^{0}(A,B)$$

Proof: Immediate from theorems 6.2 and 6.3. ∎

Thus for strict homogeneous schemes the only possible links between subjects in different clusters are those which were present in the initial state.

We have demonstrated that the three extreme cases of the *demand* function lead to policies which are less general, in terms of links between subjects in different clusters, than the policy achievable by an arbitrary *demand* function. In all three cases the only possible links between subjects in incomparable clusters are those which were present in the initial state. With respect to links between subjects in different, but comparable, clusters the most restrictive policy is that of strict homogeneous schemes while receive–controlled and send–controlled homogeneous schemes are completely permissive in one direction and completely restrictive in the other.

The use of the term restrictive above is not intended to be pejorative, since this may well be a desirable result. The point is that the three extreme cases of the *demand* function do not cover every conceivable situation. Indeed the foregoing analysis demonstrates the utility of providing a non–trivial[1] demand operation.

6.2. THE MAY-LINK RELATION

In this section we return to the general class of self–copy schemes. We now study the nature of the link* relation. Recall that this relation specifies pairs of subjects between which there was no link in the initial state while a link could be established in some derived state, i.e.,

$$\text{link}^{*}(A,B) \leftrightarrow [\sim\text{link}^{0}(A,B)] \wedge (\exists \text{ derived state } h)[\text{link}^{h}(A,B)]$$

The possibility of establishing a link* from A to B, means that it is possible to realize a direct transport of tickets from A to B, where in the initial state such a direct flow was not authorized. Here, we characterize the types of subjects

[1] In the sense that the demand operation is not eliminated, as in the case of strict homogeneous schemes, or introduced to model degenerate versions of the send–receive principle, as in the case of receive–controlled and send–controlled homogeneous schemes.

between which a link$^+$ may be established, under a given self-copy scheme. In order to do so, we introduce the following notion.

Definition 6.5: For every protection scheme, define the associated binary relation

$$\textit{may-link} \subseteq \mathbf{T}_S \times \mathbf{T}_S$$

by *may-link*(*a*,*b*) if and only if there exists an initial state with subjects A and B, of type *a* and type *b* respectively, for which it is possible to establish link$^+$(A,B). ∎

The policy, with respect to direct transport of tickets from A to B where ~*may-link*(*t*(A),*t*(B)), is then decided when the initial state is established and cannot be changed thereafter.

In this section, we derive characteristics of the *may-link* relation for self-copy schemes. Given a pair of *dfl* and *demand* functions, the only parameter required to complete the specification of a self-copy scheme is the *can-create* relation. We first establish that the *may-link* relation is independent of the *can-create* relation in the following sense.

Theorem 6.7: For a given pair of *dfl* and *demand* functions

$$\textit{may-link}(a,b) \text{ for a self-copy scheme with } \textit{can-create} \neq \phi$$
$$\Leftrightarrow$$
$$\textit{may-link}(a,b) \text{ for the self-copy scheme with } \textit{can-create} = \phi$$

Proof: $\Leftarrow$: Trivial, since for given *dfl* and *demand* functions, any transition sequence for the self-copy scheme with *can-create* = ϕ is also a transition sequence for a self-copy scheme with *can-create* $\neq \phi$.

$\Rightarrow$: By definition, *may-link*(*a*,*b*) implies there is an initial state with subjects A and B, of types *a* and *b* respectively, such that

$$[\sim\text{link}^0(A,B)] \wedge (\exists \text{ derived state } h)[\text{link}^h(A,B)]$$

By lemma 2.1 on page 50, we can assume that the derived state h is established by a transition sequence H in canonical form so that the create operations precede all the demand and transport operations. Let g be the state immediately after all the creates have occurred. Due to the self-copy create-rule, this state preserves the balanced self-copy nature of the initial state. Thus the g state is itself a balanced self-copy state. The transition sequence H without the creates can then be applied to state g. Then

$$[\sim\text{link}^g(A,B)] \wedge (\exists \text{ derived state } h)[\text{link}^h(A,B)]$$

where state h is derived from state g without any create operations.

Hence, g is an initial state for which link$^+$(A,B) without any create operations. ∎

Thus for self-copy schemes[2], the *may-link* relation is determined by the *dfl* and *demand* functions. We then have the following theorem.

Theorem 6.8: For a self-copy scheme *may-link*(*a*,*b*) if and only if

1. $[b/s \in demand(a) \wedge a/r \notin demand(b)] \wedge$
   $(\exists \alpha \neq \Lambda)\,[a/r \in mflt(a,\alpha,b)]$

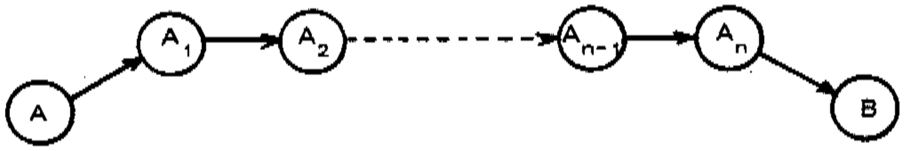
or 2. $[b/s \notin demand(a) \wedge a/r \in demand(b)] \wedge$
   $(\exists \beta)\,[b/s \in mflt(b,\beta,a)]$

or 3. $[b/s \notin demand(a) \wedge a/r \notin demand(b)] \wedge$
   $(\exists \alpha \neq \Lambda)\,[a/r \in mflt(a,\alpha,b)] \wedge (\exists \beta)\,[b/s \in mflt(b,\beta,a)]$

or 4. $[b/s \in demand(a) \wedge a/r \in demand(b)]$

Proof: Observe that exactly one of these conditions will apply for a given pair <*a*,*b*> of subject types.

$\Leftarrow$: We need to demonstrate in each case that there exists an initial state with subjects A and B of type *a* and *b* respectively for which it is possible to establish link$^+$(A,B). We demonstrate the construction for the first case and the construction for the other three cases is similar. In the first case, since $\alpha \neq \Lambda$, assume that $\alpha = a_1 ... a_n$. Let subject A_1 be of type a_1 and consider the following initial state.



It is then possible to transport the ticket A/r from dom(A) to dom(B) via the subjects A_1 to A_n. Subject A can obtain the B/s ticket by a demand operation. Then we have link$^+$(A,B).

$\Rightarrow$: By definition, link$^+$(A,B) requires that we must have one of the following mutually exclusive cases

[2]Actually theorem 6.7 applies to any scheme with create-rules that preserve the constraints on the initial state, in the sense of definition 5.5 on page 114.

1. $[b/s \in \mathit{demand}(a) \ \wedge\ a/r \notin \mathit{demand}(b)] \ \wedge$
   $A/r \in \mathrm{dom}_T(B)$

or 2. $[b/s \notin \mathit{demand}(a) \ \wedge\ a/r \in \mathit{demand}(b)] \ \wedge$
   $B/s \in \mathrm{dom}_T(A)$

or 3. $[b/s \notin \mathit{demand}(a) \ \wedge\ a/r \notin \mathit{demand}(b)] \ \wedge$
   $A/r \in \mathrm{dom}_T(B) \ \wedge\ B/s \in \mathrm{dom}_T(A)$

or 4. $[b/s \in \mathit{demand}(a) \ \wedge\ a/r \in \mathit{demand}(b)]$

In each of these four cases we can derive the corresponding condition of the theorem. We show this derivation for the first case and the derivation for the other three cases is similar. Now in the first case we have

$$\mathrm{link}^*(A,B) \rightarrow A/r \in \mathrm{dom}_T(B)$$

By the origination property, of lemma 5.2 on page 119, we have

$$A/r \in \mathrm{dom}_T(B) \rightarrow$$
$$A/r \in \mathrm{flow}(A,B) \text{ prior to } \mathrm{link}(A,B)$$

But then, by definition of the flow and *mflt* functions, we have

$$(\exists \alpha \neq A)\,[a/r \in \mathit{mflt}(a,\alpha,b)]$$

∎

In order to obtain an algorithm for checking the conditions of theorem 6.8, we can eliminate all references to the *mflt* function by rewriting these conditions in terms of the *iflt* function. This transformation is shown in appendix C.7 and results in an algorithm with a cost of $O(|T_S|^3)$.

6.3. UNIQUE-MEDIATOR SCHEMES

In this section we study a situation where there are two categories of subjects; these are called **mediators** and **users**. Transport tickets for mediators cannot be moved around, whereas transport tickets for users may be moved around. The latter transport is subject to certain constraints. Specifically, users are not allowed to transport such tickets directly between themselves but can do so only via one or more mediator subjects. The intention is that the activity of establishing a link from one user to another must involve the cooperation of at least one mediator subject. Presumably the mediator subjects are trusted to make such decisions in accordance with some broader policy considerations (which we do not specify here).

We allow for any (finite) number of user types but assume there is a single[3] mediator type. We call this class of schemes **unique-mediator schemes** to reflect this fact. Let us denote the single mediator type by the symbol *a*, and the user types by the symbol b_i for i ranging from 1 to some fixed value n. We now formalize and elaborate the constraints stated informally above.

1. Our first constraint is that transport tickets for mediator subjects cannot be moved around. This is easily enforced by requiring that the ticket types *a*/s and *a*/r not appear in the range of the *dflt* function.

2. Our second constraint is that two user subjects should not be able to move transport tickets directly between themselves. This constraint is expressed by the following requirement.
$$(\forall \langle b_i, b_j \rangle)\ [dflt(b_i, b_j) = \phi]$$

3. We assume that mediator subjects are very powerful with respect to their ability for moving transport tickets between themselves. Specifically, we require that
$$dflt(a,a) = \{b_i/\mathrm{sc},\ b_i/\mathrm{rc} \mid i=1,\ldots,n\}$$
Thus, two mediator subjects can potentially move transport tickets for all types of users directly between themselves. We could be even more permissive in this respect and allow such subjects to move transport tickets for mediators also, so that
$$dflt(a,a) = \{a/\mathrm{sc},\ a/\mathrm{rc}\} \cup \{b_i/\mathrm{sc},\ b_i/\mathrm{rc} \mid i=1,\ldots,n\}$$
This well might be the situation encountered in a practical context. However, the issues we wish to illustrate by this case study can be brought out, and indeed are easier to highlight, in the simpler context where transport tickets of the type *a*/x:c are not movable.

4. Recall that our context is that of self-copy schemes. We allow users to transport their self-reference tickets to a mediator subject so that their connection to the system may evolve. This translates to the following requirement on the *dflt* function.
$$(\forall b_i)\ [dflt(b_i, a) = \{b_i/\mathrm{sc},\ b_i/\mathrm{rc}\}]$$

5. In order for the link relation to evolve, we must allow users to obtain transport tickets for other users. At the same time, we do not want users to serve as indirect channels for transport of tickets between

[3]Based on the same idea, it is possible to build up hierarchies of users and mediators in several ways. Considerations of time and space, do not permit us to study such hierarchies here.

mediator subjects. The latter objective is enforced by prohibiting users from obtaining transport tickets with the copy flag, so that

$$(\forall \langle b_i, b_j \rangle)\ [b_j/sc \notin dflt(a,b_i) \wedge b_j/rc \notin dflt(a,b_i)]$$

The objective of allowing the link relation to evolve, is met by permitting mediator type subjects to move transport tickets for user types to every b_i, i.e.,

$$(\forall b_i)\ [dflt(a,b_i) \subseteq \{b_j/s,\ b_j/r \mid j=1,\dots,n\}]$$

The exact values of this portion of the *dflt* function are left open, as a parameter to be specified when a particular unique-mediator scheme is defined.

6. Finally, we wish to control the evolution of the link relation by insisting that new links cannot be established by a demand operation alone so that, for every $\langle c,d \rangle \in T_S \times T_S$

$$c/s \in demand(d) \rightarrow d/r \notin demand(c)$$
$$c/r \in demand(d) \rightarrow d/s \notin demand(c)$$

The following definition summarizes the various restrictions we have imposed.

Definition 6.6: A self-copy scheme is a **unique-mediator scheme** if

1. The set of subject types is

$$T_S = \{a, b_1, \dots, b_n\}$$

2. The *dflt* function is such that

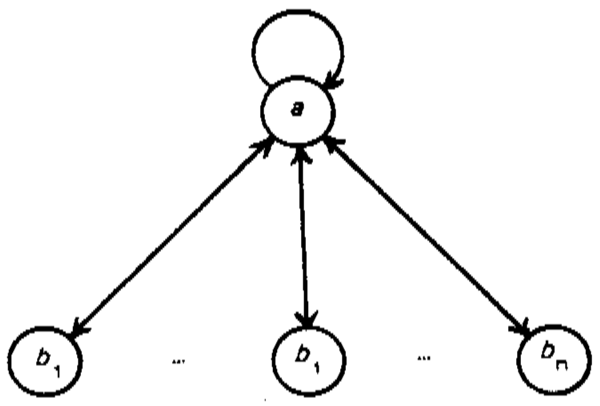
$$dflt(b_i,\ a) = \{b_i/sc,\ b_i/rc\}$$
$$dflt(b_i, b_j) = \phi$$
$$dflt(a\ ,\ a) = \{b_j/sc,\ b_j/rc \mid j=1,\dots,n\}$$
$$dflt(a\ , b_i) \subseteq \{b_j/s,\ \ b_j/r \mid j=1,\dots,n\}$$

3. The *demand* function is such that for all pairs of subject types $\langle c,d \rangle$

$$c/s \in demand(d) \rightarrow d/r \notin demand(c)$$
$$c/r \in demand(d) \rightarrow d/s \notin demand(c)$$

■

Observe that we do not constrain the *can-create* relation in any way. The *dflt* graph for a unique-mediator scheme has the structure indicated below.



The Dflt Graph for Unique-Mediator Schemes

It is apparent that the only way to move transport tickets from a subject of type b_i to a subject of type b_j is via one or more subjects of type *a*.

The definition of unique-mediator schemes immediately leads to the following property.

Theorem 6.9: For every unique-mediator scheme ~*may-link*(*a*,*a*).

Proof: Theorem 6.8 on page 149, characterizes the *may-link* relation for self-copy schemes. In the context of the <*a*,*a*> pair, for unique-mediator schemes the first three conditions are false, due to the constraint on the *dflt* function; and the fourth condition is false, due to the constraint on the *demand* function. ■

Thus, for such schemes it is not possible to introduce any links between mediator subjects. This is a notable property. In particular, this property guarantees that mediator subjects which are created (or, more strongly, originated) by a user subject will remain isolated from all other subjects which were present in the initial state (as well as from the descendants of such subjects).

6.3.1. The Flow-Invariant Property

Following our now familiar approach[4], we investigate the flow-invariant property for unique-mediator schemes in the context of extreme cases of the *demand* function. Due to the constraint on the *demand* function, there are three extreme cases to consider as follows.

[4]See sections 5.3 and 6.1.2.

1. For receive-controlled self-copy schemes

$$(\forall \langle a,b\rangle)\ [b/\mathrm{s} \in demand(a) \ \wedge\ a/\mathrm{r} \notin demand(b)]$$

2. For send-controlled self-copy schemes

$$(\forall \langle a,b\rangle)\ [b/\mathrm{s} \notin demand(a) \ \wedge\ a/\mathrm{r} \in demand(b)]$$

3. For strict self-copy schemes

$$(\forall \langle a,b\rangle)\ [b/\mathrm{s} \notin demand(a) \ \wedge\ a/\mathrm{r} \notin demand(b)]$$

We will establish that receive-controlled and strict unique-mediator schemes are flow-invariant, while send-controlled unique-mediator schemes are not. Consider first the case of receive-controlled schemes.

Theorem 6.10: Receive-controlled unique-mediator schemes are flow-invariant.

Proof: From theorem 5.3 on page 125, we know that receive-controlled self-copy schemes are flow-invariant if and only if for all pairs of subject types *<c,d>*

$$(\forall\alpha)\ [c/\mathrm{r} \in mflt(c,\alpha,d) \rightarrow dflt(c,d) \subseteq mflt(c,\alpha,d)]$$

There are three cases to consider.

Case (i): Consider pairs of subject types where the first member of the pair is a, i.e., $c = a$ in the above condition. For a unique-mediator scheme, tickets of type a/r are not movable and, hence, the condition is trivially true for such pairs.

Case (ii): Consider pairs of user types, i.e., $c = b_i$ and $d = b_j$ in the above condition. For every such pair, we have

$$dflt(b_i,b_j) = \phi$$

The condition is then trivially satisfied.

Case (iii): The only pairs left to consider are of the form $c = b_i$ and $d = a$ in the above condition. For every type b_i, we have

$$dflt(b_i,a) = \{b_i/\mathrm{sc},\ b_i/\mathrm{rc}\}$$

By inspection of the *dflt* function, for unique-mediator schemes it is evident that for every string α

$$mflt(b_i,\alpha,a) = \phi \text{ or } \{b_i/\mathrm{sc},\ b_i/\mathrm{rc}\}$$

But then,

$$\begin{aligned} & b_i/r \in mflt(b_1,\alpha,a) \\ \rightarrow\ & mflt(b_1,\alpha,a) = \{b_1/sc,\ b_1/rc\} \\ \rightarrow\ & dflt(b_1,a) \subseteq mflt(b_1,\alpha,a) \end{aligned}$$

Thus in all cases the condition of theorem 5.3 on page 125 is satisfied. ∎

Next, consider strict unique-mediator schemes.

Corollary 6.10.1: Strict unique-mediator schemes are flow-invariant.

Proof: From theorem 6.10 and corollary 5.5.1 on page 133. ∎

Finally, consider send-controlled unique-mediator schemes.

Theorem 6.11: Send-controlled unique-mediator schemes need not be flow-invariant.

Proof: From theorem 5.4 on page 129, we know that send-controlled self-copy schemes are flow-invariant if and only if for all pairs of subject types $<c,d>$

$$(\forall\alpha)\ [c/s \in mflt(c,\alpha,d) \rightarrow dflt(d,c) = \phi]$$

It suffices to exhibit a pair which violates this condition, to establish that send-controlled unique-mediator schemes need not be flow-invariant. Consider any pair $<b_i,a>$. By definition, we have

$$dflt(b_i,a) = \{b_1/sc,\ b_1/rc\}$$

while in general,

$$dflt(a,b_1) \neq \phi$$

This violates the stated condition. ∎

6.3.2. The May-Link Relation

Next we investigate properties of the *may-link* relation for the extreme cases of the *demand* function. Recall that this relation specifies those types of subjects between which links, which were not present in the initial state, may be established. We observed, in theorem 6.9 on page 153, that for unique-mediator schemes $\sim$*may-link*(a,a). Hence, it is not possible to establish new links between mediator subjects.

Now consider the types of user subjects between which a link* can be established. This is a policy decision to be made when a specific unique-mediator scheme is defined. Indeed, this decision determines the extent to which mediators can change the initial protection state. For convenience in our discussion, let the symbol B denote a specific subset of $\{b_1,...,b_n\}$ X $\{b_1,...,b_n\}$. It should be possible to accommodate any B in the *may-link* relation, in order to provide the designer of a specific unique–mediator scheme complete freedom in this respect.

The definition of unique–mediator schemes leaves the exact values of $dflt(a,b_i)$ unspecified, as a parameter to be defined for a particular unique–mediator scheme. We now show that, by appropriate design of these values, an arbitrary B can be included in the *may-link* relation. Indeed, we establish that this is feasible even if the *demand* function is fully specified.

Theorem 6.12: Given any subset B of $\{b_1,...,b_n\}$ X $\{b_1,...,b_n\}$, and any *demand* function[5] we can design the *dflt* function for a unique–mediator scheme so that

$$may\text{-}link(b_i,b_j) \Leftrightarrow \langle b_i,b_j\rangle \in B$$

Proof: Construct the *dflt* function as follows.

1. Initialize $dflt(a,b_i)$ to be empty for all b_i.

2. For every $\langle b_i,b_j\rangle$ in the specified set B do the following:

```
if $b_j/s \notin demand(b_i)$ then $dflt(a,b_i) \leftarrow dflt(a,b_i) \cup \{b_j/s\}$
if $b_i/r \notin demand(b_j)$ then $dflt(a,b_j) \leftarrow dflt(a,b_j) \cup \{b_i/r\}$
```

The intuition here is rather straightforward. If subjects of type b_i and b_j cannot obtain the required tickets by a demand operation[6] then they must be allowed to do so via a transport operation. In conjunction with theorem 6.8 on page 149, it is then a simple exercise to show that

$$may\text{-}link(b_i,b_j) \Leftrightarrow \langle b_i,b_j\rangle \in B$$

■

Thus as far as user types are concerned, there is complete flexibility in determining the *may-link* relation. In particular, for all three extreme cases[7] of the *demand* function, an arbitrary B can be accommodated in the *may-link* relation. However, there are significant differences when we consider the possibility of establishing new links involving a mediator subject. The following theorem establishes this difference.

---

[5]Subject, of course, to the constraint which rules out establishing links purely by demand operations.

[6]Actually we need not even bother to test this condition, since it does not matter whether there are alternate ways of obtaining the required tickets.

[7]That is, the receive–controlled, send–controlled, and strict cases.

Theorem 6.13: For a unique-mediator scheme in the three extreme cases of the *demand* function we have

1. For receive-controlled unique-mediator schemes
$$(\forall b_1)\ [\textit{may-link}(b_1,a)\ \wedge\ \sim\textit{may-link}(a,b_1)]$$

2. For send-controlled unique-mediator schemes
$$(\forall b_1)\ [\sim\textit{may-link}(b_1,a)\ \wedge\ \textit{may-link}(a,b_1)]$$

3. For strict unique-mediator schemes
$$(\forall b_1)\ [\sim\textit{may-link}(b_1,a)\ \wedge\ \sim\textit{may-link}(a,b_1)]$$

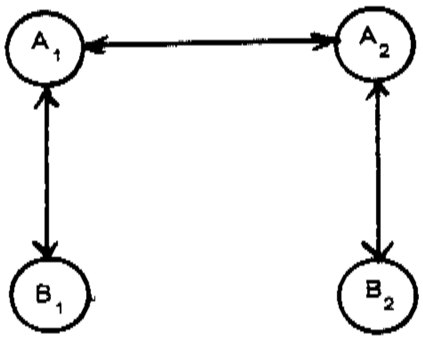
Proof: We will prove the first statement of the theorem. The proofs of the other two statements are similar. By theorem 6.8, we know that for a receive-controlled unique-mediator scheme
$$\textit{may-link}(b_1,a)\ \leftrightarrow\ (\exists\alpha\neq\Lambda)\ [b_1/r \in \textit{mflt}(b_1,\alpha,a)]$$
By definition of unique-mediator schemes, this condition is true for every string α which consists of one or more occurrences of a. Again by theorem 6.8 on page 149, we know that for a receive-controlled unique-mediator scheme
$$\textit{may-link}(a,b_1)\ \leftrightarrow\ (\exists\alpha\neq\Lambda)\ [a/r \in \textit{mflt}(a,\alpha,b_1)]$$
By definition of unique-mediator schemes, tickets of type a/r are not movable and this condition is trivially false. ∎

We will discuss the consequences of this theorem in the next section.

6.3.3. The Control-Invariant Property

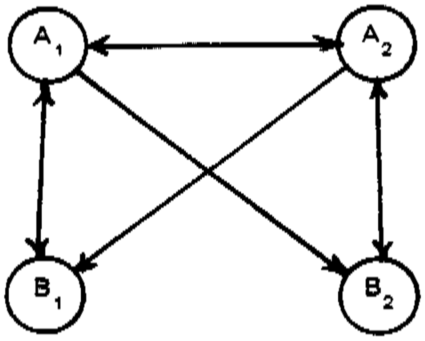
In this section we conclude our study of unique-mediator schemes by discussing the notion of a control-invariant scheme. Just as the notion of clusters is peculiar to homogeneous schemes the property of control-invariance is peculiar to unique-mediator schemes.

We motivate the introduction of this notion by considering send-controlled unique-mediator schemes. In theorem 6.13, we observed that for such schemes
$$(\forall b_1)\ [\textit{may-link}(a,b_1)]$$
It is then possible to establish a link$^+$ from a mediator subject to a user subject. In particular, consider the following initial state where A_1 and A_2 are subjects of type a and B_1 and B_2 are subjects of type b_1 and b_2.



The Initial State

In this state the subject A_1 has control over the subject B_1, since B_1 is connected to A_1 in both directions by links, but is not connected to any other subject of type *a*. It is quite reasonable to require that this control be maintained in all subsequent states. Then we know that whatever links may be established to and from B_1 must involve the mediation of A_1. Similar remarks apply to A_2 and B_2. For a send-controlled unique-mediator scheme this control cannot be maintained. In particular, in the initial state shown above, the self-reference ticket B_1/sc can be transported from dom(B_1) to dom(A_2). The ticket A_2/r can be obtained by B_1 on demand. This results in a link from A_2 to B_1. Similarly, there can be a link from A_1 to B_2. The resulting state is shown below.



A Derived State

The situation with respect to receive-controlled unique-mediator schemes is similar, since for these schemes, from theorem 6.13 we have

$$(\forall b_1)\ [\textit{may-link}\,(b_1, a)\,]$$

Indeed, the only way to ensure that links are not established between users and mediators, is to make sure that

$$(\forall b_i)\ [\sim\textit{may-link}(a,b_i)\ \wedge\ \sim\textit{may-link}(b_i,a)]$$

This leads us to introduce the following notion.

Definition 6.7: We say that a unique-mediator scheme is **control-invariant** if and only if

$$(\forall b_i)\ [\sim\textit{may-link}(a,b_i)\ \wedge\ \sim\textit{may-link}(b_i,a)]$$

■

The corollary below follows from this definition and theorem 6.13.

Corollary 6.13.1: Strict unique-mediator schemes are control-invariant whereas receive-controlled and send-controlled unique-mediator schemes are not control-invariant. ■

It is possible to slightly modify the *demand* function for receive-controlled and send-controlled unique-mediator schemes, so that the resulting schemes become control-invariant. All we need do, is eliminate the ability of user subjects to acquire tickets for mediator subjects by a demand operation. This is easily achieved by modifying the *demand* function as follows.

Definition 6.8: A unique-mediator scheme is **receive-oriented** provided

$$(\forall b_i)\ [\textit{demand}(b_i)\ =\ \{b_j/\mathrm{s}\,|\,j=1,\ldots,n\}]$$

Similarly, a unique-mediator scheme is **send-oriented** provided

$$(\forall b_i)\ [\textit{demand}(b_i)\ =\ \{b_j/\mathrm{r}\,|\,j=1,\ldots,n\}]$$

■

We then have the following corollary.

Corollary 6.13.2: Receive-oriented and send-oriented unique-mediator schemes are control-invariant.

Proof: By inspection of the conditions of theorem 6.8 on page 149. ■

An interesting aspect is that control-invariant schemes are also flow-invariant as shown below.

Theorem 6.14: Every unique-mediator scheme which is control-invariant is flow-invariant.

Proof: We know, from theorem 6.9 on page 153, that for every unique-mediator scheme $\sim$*may-link*(a,a). For a control-invariant unique-mediator scheme, by definition

$$(\forall b_i)\ [\sim\textit{may-link}(a,b_i)\ \wedge\ \sim\textit{may-link}(b_i,a)]$$

It follows that a link$^+$ can be established only between user subjects. Introduction of such links cannot change the flowt function. ■

Hence, the analysis of control-invariant schemes can be carried out efficiently. Observe that the converse of this theorem is not true, since receive-controlled unique-mediator schemes are flow-invariant but not control-invariant.

6.4. SUMMARY

In this chapter we investigated some policy issues other than the flow analysis problem.

In section 6.1 we studied the nature of the link relation in the context of homogeneous schemes. We defined the notion of a cluster of subjects and demonstrated that the nature of links between subjects in different clusters is strongly influenced by the *demand* function. In particular we showed that the three extreme cases of the *demand* function are limited in this respect.

In section 6.2 we defined the *may-link* relation. This relation specifies the types of subjects between which it is possible to establish a link which did not exist in the initial state. We characterized the nature of this relation, for self-copy schemes, and showed that it can be computed with a cost of $O(|T_S|^3)$.

In section 6.3 we defined the class of unique mediator schemes and studied the nature of the *may-link* relation for such schemes. We also introduced the notion of control-invariance and showed that, while the degenerate send-controlled and receive-controlled cases are not control-invariant, a slight modification of the *demand* function leads to unique-mediator schemes which are control-invariant.

CHAPTER 7
SUMMARY AND CONCLUSION

We have raised a number of questions in the course of our exposition. While most of these questions have only been partially answered we do believe that our results provide useful guidelines to the designer of a specific system. We review the major results of our investigation in section 7.1. In section 7.2 we discuss issues which have not been resolved in the analysis presented in this thesis. Resolution of these issues will provide a sharper insight than we have been able to provide. Finally in section 7.3 we briefly discuss some crucial aspects which were not addressed at all in the thesis.

7.1. MAJOR RESULTS AND INSIGHTS

In chapter 2 we proposed a framework for specifying the dynamics of a protection scheme. Certain assumptions are basic to the framework and apply to any specific scheme. Specifically,

1. The transport operation is controlled by the send–receive protocol.
2. There is a copy flag which distinguishes tickets whose copies might be transported from tickets whose copies simply cannot be transported.
3. Every subject is created to be of a specific type and its type cannot change thereafter.
4. New types cannot be introduced at run time.

Our framework provides the following parameters which must be specified in order to define a particular scheme.

1. The set of types $\mathbf{T} = \mathbf{T}_S \cup \mathbf{T}_O$ where $\mathbf{T}_S \cap \mathbf{T}_O = \phi$.
2. The set of rights $\mathbf{R} = \mathbf{R}_I \cup \mathbf{R}_T$ where $\mathbf{R}_T = \{s, r, sc, rc\}$.
3. The function *dfl*: $\mathbf{T}_S \times \mathbf{T}_S \rightarrow$ **power-set**$(\mathbf{T} \times \mathbf{R})$.
4. The function *demand*: $\mathbf{T}_S \rightarrow$ **power-set**$(\mathbf{T}_S \times \mathbf{R}_T)$.
5. The relation *can-create* $\subseteq \mathbf{T}_S \times \mathbf{T}_S$.
6. A local create-rule for every pair *<a,b>* $\in$ *can-create*.
7. Constraints on the initial state.

The first five parameters are each formalized in terms of a set, function, or relation. The precise definition of the last two parameters has been deliberately left open but we indicated the scope we have in mind.

In chapter 3 we defined the flow^k function to express the authorization for transport of tickets from dom(A) to dom(B), in state k, accounting for indirect as well as direct transport. The values of this function will change as the protection state evolves and the flow-analysis problem is to determine properties of this function. We defined the flow* function to express the maximum value of the flow function in any derived state. A protection scheme constrains the evolution of an initial state only to the extent determined by the flow* function. The create operation presents a major complication in computing this function, since the protection state can evolve in an unbounded manner. We were not able to arrive at an algorithm for computing the flow* function or demonstrate that it is not computable. We introduced the notion of a bound flow† on the flow function by the requirement that

$$(\forall \langle A,B\rangle \in \mathrm{SUB}^O \times \mathrm{SUB}^O)\,[\mathrm{flow}^*(A,B) \subseteq \mathrm{flow}^\dagger(A,B)]$$

We defined the *ifl* function which provides a bound on the flow function without considering the structure of a particular initial state. This function is defined in terms of the *dfl* function and can be computed with a cost of $O(|\mathbf{T} \times \mathbf{R}| * |\mathbf{T}_S|^3)$.

We also defined the flow$^\#$ function to express the maximum value of the flow function in any state derived without use of the create operation. The flow$^\#$ function is easily computed with a cost no worse than $O(|\mathrm{SUB}^O|^5)$. We developed a technique for computing a bound on the flow function taking into account the particular initial state. This technique augmented the initial state by introducing a finite number of new subjects for every subject present in the initial state. We defined the flow† function to be the flow$^\#$ function which results from an augmented initial state. We proved in theorem 3.6 that the flow† provides a bound on the flow function. This bound can be computed with a cost no worse than $O(|T_S|^5 * |SUB^O|^5)$. We demonstrated, by means of an example, that this technique provides a sharper bound than the *ifl* function. We also indicated how the bound can be improved by unfolding the initial state.

In chapter 4 we discussed the notion of a create-invariant scheme. For such schemes the flow* function is identical to the flow$^{\#}$ function and, hence, can be computed in a straightforward manner. We developed a notion of weak creates defined in terms of the *dfl* and *demand* functions. The constraint of weak creates is non-trivial in the sense that it will always permit a subject to create another subject of the same type. We showed in theorem 4.2 that any scheme with weak creates and self-reference initial states is create-invariant. We demonstrated that relaxing the definition of weak creates will allow for schemes which are not create-invariant. We also discussed the class of hierarchical schemes to demonstrate the utility of schemes with weak creates.

In chapter 5 we defined two notions of a flow-invariant scheme. In the stronger sense, a flow-invariant scheme does not allow any change in the flow function, with respect to the initial set of subjects. In the weaker sense, the invariance is only with respect to the flow of transport tickets. In the former case, it is a trivial matter to compute the flow* function. In the latter case, a bound on the flow function, using the technique of section 3.5, can be computed with a cost no worse than $O(|T \times R| * |T_S|^3 * |SUB^O|^3)$ which is considerably faster than the general case, which might require a cost of $O(|T_S|^5 * |SUB^O|^5)$. Our discussion in the rest of the chapter was in the context of the weaker notion of flow-invariant, but our results are easily modified to apply to the stronger notion. We showed that, under some rather general conditions on the create-rules, the value of the *can-create* relation is not relevant in determining the flow-invariant nature of a scheme. We defined the class of self-copy schemes by constraining some of the design parameters. The nature of these constraints led to the name self-copy, since the major constraint was in terms of transport ticket types with the copy flag. We obtained necessary and sufficient conditions for a self-copy scheme to be flow-invariant. The cost of testing these conditions for an arbitrary self-copy schemes is $O(|T_S|^4)$. Our strategy in developing these conditions was to first study extreme cases of the *demand* function. This study revealed that the properties are significantly different in these extreme cases, thus extending the observation first made by Minsky [15], in the context of the uniform send-receive mechanism. Finally, we demonstrated that there a variety of self-copy schemes which are flow-invariant.

In chapter 6 we investigated some policy issues other than the flow-analysis problem. Specifically

1. We studied the nature of the link* relation in the context of homogeneous schemes. We defined the notion of a cluster of subjects and demonstrated that the nature of links between subjects in different clusters is strongly influenced by the *demand* function. In particular we showed that the three extreme cases of the *demand* function are limited in this respect.

2. We defined the *may-link* relation as a measure of the scope for evolution of the link relation allowed by a particular self-copy scheme. We characterized the nature of this relation and showed that it can be computed with a cost of $O(|T_S|^3)$.

3. We defined the class of unique mediator schemes and studied properties of the *may-link* relation for such schemes.

7.2. UNRESOLVED ISSUES

The major unresolved issue is that of an exact computation of the flow* function. Recall that the flow* function expresses the maximum flow of ticket types which can be achieved in the presence of a create operation. We strongly believe that this function is indeed computable by some closure technique. However, we have not been able to formulate a suitable termination condition. One open problem is then to demonstrate the computability of the flow* function and determine the complexity of such a computation.

A related issue is to obtain some measure on how good the bound obtained by the flow† function actually is. We demonstrated by an example that this bound is considerably better than the bound imposed by the *ifl* function. Our example also showed that the flow† function may be too permissive. It would be interesting to know under what conditions the flow† function is identical to the flow* function. We could also look for some systematic technique to improve the bound along the lines of our example where we obtained an exact value of the flow* function by unfolding one level of creates prior to construction of the augmented initial state.

It is also possible to further investigate the concept of a create–invariant scheme. We have arrived at a sufficient condition which guarantees this property. That this sufficient condition is not necessary is easy to see. For example, the receive–controlled mediator schemes of section 6.3 were shown to flow–invariant. By extending the constraints on the *dflt* function to the *dfl* function these schemes become create–invariant in the presence of an arbitrary *can-create* relation. Such a create relation does not satisfy our criterion of weak creates. It would be of interest to develop necessary and sufficient conditions for a scheme to be create–invariant.

We developed the concept of a flow–invariant scheme in the context of self–copy schemes. This concept can be generalized to apply to a larger class of schemes and it would interesting to now how far the constraints can be relaxed.

7.3. AVENUES FOR FURTHER RESEARCH

There are several pragmatic issues which have not been addressed in this thesis. Perhaps, the most significant is that of implementing the design framework. Conceptually the framework can be implemented by a kernel, consisting of a combination of hardware and software, which provides the machinery for interpreting a specific protection scheme. The tradeoffs involved in such an implementation can be appreciated only after investigating the possible architectures for such a kernel. In particular, if there is significant economy in implementing restricted versions of the framework then it is worth investigating these versions further.

Another issue is the need to demonstrate the utility of our schemes by a large scale case study. Such an exercise would help refine those aspects of the framework which have been left unspecified or only partially specified. More significantly this exercise would (hopefully) demonstrate that our analysis techniques provide usable results in such a context.

APPENDIX A
INDEX OF DEFINITIONS

Appendix B
NAMING CONVENTIONS

It has been necessary to introduce a fair amount of notation, to describe formally the class of protection schemes considered in this thesis and to analyze their properties. Due to the lack of historical precedent, most of this notation is of our own creation. While we have attempted to introduce the minimal amount of notation necessary for pursuing our investigation, we are aware there is ample potential for a reader to get bewildered. In this appendix we summarize those features of the notation which were designed for clarifying the role of various symbols. The specific aspects covered are as follows.

1. Names of Relations
2. Names of Functions
3. Names of Subjects and Subject Types
4. The Set T_S^*
5. The Type Function
6. Transition Sequences and Resulting States
7. The Flow Limit Functions
8. The Flow Functions

Names of Relations

There are two basic sets which pervade the entire discussion. These are

SUB: the current set of subjects
T_S: the set of subject types

We have introduced several relations which are subsets of T_S X T_S. As examples we have the *can-create* and *may-link* relations. Any relation which is a subset of T_S X T_S is named using italic script.

Similarly, we have introduced several relations which are subsets of SUB X SUB. As examples we have the link and link$^+$ relations. Any relation which is a subset of SUB X SUB is named using normal script.

Names of Functions

The convention for names of relations extends to names of functions, by applying the same principle to function domains. If the domain of a function is T_S or $T_S \times T_S$ or $T_S \times T_S^* \times T_S$, then the function is named using italic script. As examples we have the *dfl*, *ifl*, and *mfl* functions, respectively.

If the domain of a function is SUB or SUB X SUB, then the function is named using normal script. As examples we have the org and flow functions, respectively.

We have made one exception to these rules in naming the *t* function, which tells us the type of a given subject. We explain the reason for this deviation after discussing the convention for naming subjects and subject types.

Names of Subjects and Subject Types

We denote subject types by lower case italicized letters from the beginning of the alphabet. Thus the letters *a, b, c, d* denote specific subject types.

We denote specific subjects by upper case letters from the beginning of the alphabet. Thus the letters A, B, C, D denote specific subjects. To the extent possible, we ensure that the type of a subject is the corresponding lower case italic letter.

On occasion, we need to introduce several subjects of the same type. We distinguish them by an integer subscript. Thus, A_1 and A_2 are two distinct subjects both of type *a*. Similarly, B_1 and B_2 are subjects of type *b*. We also refer to such subjects by a generic subscript as, for example, subject A_i of type *a*.

On occasion, we place integer subscripts on lower case italicized letters to denote subject types, so that a_1 and a_2 are distinct subject types. We can then refer to these types by a generic subscript as, for example, a_i. When we have subscripted types, we designate a subject of one of these types by the corresponding upper case letter in normal script and the same subscript. Thus, A_1 is a subject of type a_1 and A_2 is a subject of type a_2. We can also use a generic subscript so that A_i is a subject of type a_i.

There is an obvious ambiguity[1] in the above convention. Thus, it is not clear whether A_1 and A_2 are two distinct subjects both of type *a*; or whether they are, respectively, of type a_1 and a_2. This conflict is resolved by the particular context.

The Set T_S^*

There is a need to denote members of the set T_S^*, which is the Kleene closure of T_S. We denote strings from this set by lower case Greek letters, such as α and β. For the case of a non-empty string, we write out the string explicitly as follows.

$$\alpha = a_1 a_2 \ldots a_{n-1} a_n$$
$$\beta = b_1 b_2 \ldots b_{m-1} b_m$$

The empty string is denoted by the symbol Λ.

The Type Function

The type function

$$t\colon \mathrm{SUB} \longrightarrow T_S$$

returns the type of the argument subject. According to our stated convention, its name should be in normal script; since its domain is the set SUB. We have deviated from this convention to ensure that every member of T_S appears in italic script; either directly as *a*, *b*, a_1 etc. or indirectly, via an application of *t*, as *t*(A), *t*(B), $t(A_1)$ etc. The net result is that any function or relation with an italicized name must have italicized arguments.

Transition Sequences and the Resulting States

We denote a transition sequence by upper case letters from the middle of the alphabet typically, H and G. The resulting state is denoted by the corresponding lower case letter, h and g, respectively.

The Flow Limit Functions

The mnemonic used for flow limit is *fl*. If followed by a *t*, as in *flt*, then the flow limit is on tickets with transport rights.

If the letter *d* precedes these strings, then the flow limit is on a direct transport from one subject to another. If the letter *m* precedes these strings, then the flow limit is on a mediated transport from one subject to another. Finally, if the letter *i* precedes these strings, then the flow limit is on a direct or indirect transport from one subject to another.

[1] In addition to following these conventions, we explicitly indicate the type of every subject introduced in the discussion so there is never really any doubt about our intention.

By taking all combinations of the two independent aspects mentioned above, we get the following names.

dfl	*mfl*	*ifl*
dflt	*mflt*	*iflt*

The Flow Functions

The same convention used to qualify the *fl* symbol is also used to qualify the flow function. Thus,

flow: refers to the flow of any ticket type
flowt: refers to the flow of transport tickets

Appendix C
ALGORITHMS AND THEIR COMPLEXITY

In this appendix we develop algorithms for computing various functions and relations introduced in this thesis. We also derive the complexity of these algorithms.

Each algorithm is written as a procedure which returns a value. We assume that the design parameters of the scheme are available as global variables, so that they need not be passed as parameters to our algorithms. For the same reason, we also assume that the current set of subjects is globally available. Hence, the following variables are globally accessible.

1. The *dfl* function.
2. The *demand* function.
3. The *can-create* relation.
4. The current set of subjects, SUB.

Several algorithms require computation of the transitive closure of some relation. We assume there is a procedure called **trclosure** available for this.

The keywords and syntax used for writing these algorithms conform to standard practice in modern programming languages. However, there are certain aspects which need to be explained. These aspects are discussed in section C.1. In particular, we present the assumptions underlying our complexity analysis. The remaining sections develop and analyze the cost of specific algorithms.

C.1. Conventions for Presenting Algorithms

The particular aspects we explain here are, respectively:

1. The **forall** loop.
2. The ∃ predicate.
3. The use of set operations.
4. The representation of relations.
5. The representation of functions.
6. The transitive closure operation.

The Forall Loop

The **forall** loop involves iteration over all members of some set. The general form of this loop is

forall *member* ∊ *SET* **do** *statement*;

The *statement*, called the body of the loop, may be a simple statement or a compound statement enclosed within a **begin end** pair. The semantics is that the body of the loop is executed once for every member of the indicated set. At each iteration, the selection of a member from the indicated set is done non-deterministically; and each member is chosen exactly once. The *SET* being iterated over cannot be changed by the body of the loop.

If the cost of executing the body of the loop is independent of the set member chosen for a particular iteration, then the cost of executing the entire loop is

$$O(|SET| * \textit{cost-body})$$

where, $|SET|$ is the size of the *SET* and *cost-body* is the cost of a single iteration.

On occasion, the cost of executing the body of the loop depends on the particular set element selected for the iteration. In such cases, we compute the cost of the loop by assigning the maximum cost for every iteration. The cost of the entire loop is then

$$O(|SET| * \textit{cost-body-max})$$

where, *cost-body-max* is the maximum cost of a single iteration.

The Existential Predicate

The existential predicate involves an iteration over all members of some set, searching for an element which satisfies a designated predicate. The general form of this construct is

$$(\exists\ \textit{member} \in \textit{SET})\ \ \textit{predicate}(\textit{member})$$

This construct is used within an **if** statement. The semantics is that the *predicate* is tested for every member of the the indicated set. The moment a member which satisfies the predicate is encountered, the existential predicate evaluates to **true**. If the entire set is exhausted without finding a member for which the predicate is true, then the existential predicate evaluates to **false**. The order in which members are selected from the indicated set is non-deterministic, but each member is chosen exactly once. The *SET* being iterated over cannot be changed by evaluation of the predicate.

If the cost of evaluating the designated predicate is independent of the set member chosen for a particular iteration, then we assign the cost of evaluating the existential predicate as

$$O(|SET| * \textit{cost-predicate})$$

where, $|SET|$ is the size of the *SET* and *cost-predicate* is the cost of evaluating the designated predicate for a specific set member. Note that, if the existential predicate is false this is the actual cost; whereas if the existential predicate is true then it is a worst case cost.

On occasion, the cost of evaluating the predicate depends on the particular set element selected for the iteration. In such cases, we compute the cost of evaluating the existential predicate by assigning the maximum cost for every iteration. The cost of evaluating the existential predicate is then

$$O(|SET| * \textit{cost-predicate-max})$$

where, *cost-predicate-max* is the maximum cost of evaluating the designated predicate.

Set Operations

We assume that the union and intersection of sets can be performed in constant time. Also that, membership in a set can be tested in constant time. These costs are realizable for a bit-string representation of a set with known maximum size. The sets involved in our algorithms satisfy this requirement.

Relations

The declaration for a relation defined on a given set has the following form.

declare *relation-name* (1 . . . |*SET*| , 1 . . . |*SET*|)

We assume that membership in a relation can be tested in constant time. Also that, a particular pair can be inserted or deleted in constant time. The concrete implementation we have in mind is that of a bit-matrix. We insert a pair in the relation by setting the appropriate bit to **true**, and similarly, remove a pair by setting the appropriate bit to **false**.

Functions

Our functions are all set valued and have the following structure

function-name: *DSET* X *DSET* ⟶ **power-set** (*RSET*)

mapping the cross product of *DSET* to a subset of *RSET*. We declare a function as follows.

declare *function-name* (1... |*DSET*|, 1... |*DSET*|)

The *RSET* is implicitly determined by the *function-name*. The concrete implementation we have in mind is that of a matrix of bit-vectors. Each bit-vector has a size equal to |*RSET*|.

The Transitive Closure Operation

The transitive closure operation is used in a number of our algorithms. We assume there is a procedure called **trclosure** available for effecting this computation. The input to this procedure is a relation shown as a set of ordered pairs. For example,

trclosure ({<A,B> ∊ SUB X SUB| link(A,B)})

computes the transitive closure of the link relation.

The standard transitive closure algorithms compute the closure in $O(n^3)$ time, and we assume this cost in our complexity calculations. We are aware that the fast multiplication algorithms can be used to obtain an $O(n^{2.78} * \log(n))$ algorithm, which is asymptotically better. However, for practical situations this is not very helpful. We are also aware that there are algorithms which have an expected[1] complexity of $O(n^2)$. These algorithms are discussed in several books, including the one by Reingold, Neivergelt, and Deo [16].

[1]Since we adopt a worst case viewpoint in our analysis, this fact is not immediately relevant. However, this would be an important consideration when implementing the algorithms.

C.2. The Ifl Function

In order to compute the *ifl* function, it is convenient to first compute a closely related function defined as follows.

Definition C.1: For every *ifl* function define the associated function

$$iflc\colon T_S \times T_S \rightarrow \textbf{power-set}(T \times R)$$

by

$$c/xc \in iflc(a,b) \leftrightarrow c/xc \in ifl(a,b)$$

■

The *iflc* function is easily computed, by evaluating a transitive closure for every ticket type with the copy flag, as shown below.

```
procedure get-iflc
  declare $iflc(1...|T_S|, 1...|T_S|)$, $may\text{-}flow(1...|T_S|, 1...|T_S|)$;
  forall $<a,b> \in T_S \times T_S$ do $iflc(a,b) \leftarrow \phi$;
  forall $c/xc \in T \times R$ do
    begin
      $may\text{-}flow \leftarrow$ trclosure$(\{<a,b> \in T_S \times T_S \mid c/xc \in dfl(a,b)\})$;
      forall $<a,b> \in T_S \times T_S$ do
        if $may\text{-}flow(a,b)$ then $iflc(a,b) \leftarrow iflc(a,b) \cup \{c/xc\}$;
    end;
  return $iflc$;
end get-iflc;
```

There is a cost of $O(|T_S|^3)$ for the transitive closure operation and, since this operation must be performed for every copiable member of T X R, the complexity of the **get-iflc** algorithm is $O(|T \times R| * |T_S|^3)$.

The *ifl* function can be expressed in terms of the *dfl* and *iflc* functions, by the following transformation. By definition

$$c/x{:}c \in ifl(a,b) \leftrightarrow (\exists\alpha)\,[c/x{:}c \in mfl(a,\alpha,b)]$$

There are two cases to consider. For the first case, let α be empty. By definition of the *mfl* function, we then have

$$c/x{:}c \in mfl(a,\Lambda,b) \leftrightarrow c/x{:}c \in dfl(a,b)$$

For the second case, α is non-empty. Then, α can be written as βd where β may or may not be empty. But then,

$$\begin{aligned} & (\exists\beta d)\,[c/x{:}c \in mfl(a,\beta d,b)] \\ \leftrightarrow\ & (\exists\beta d)\,[c/xc \in mfl(a,\beta,d) \wedge c/x{:}c \in dfl(d,b)] \\ \leftrightarrow\ & (\exists d)\,[c/xc \in iflc(a,d) \wedge c/x{:}c \in dfl(d,b)] \end{aligned}$$

By combining both cases, the *ifl* function can be expressed as follows.

$$c/x{:}c \in ifl(a,b)$$
$$\Leftrightarrow$$
$$[c/x{:}c \in dfl(a,b)] \vee (\exists d)\,[c/xc \in iflc(a,d) \wedge c/x{:}c \in dfl(d,b)]$$

The *ifl* function can then be easily computed from the *iflc* function as shown below.

```
procedure get-ifl
  declare $ifl(1...|T_S|,1...|T_S|)$, $iflc(1...|T_S|,1...|T_S|)$;
  $iflc \leftarrow$ get-iflc;
  forall $<a,b>$ $T_S \times T_S$ do
    begin
      $ifl(a,b) \leftarrow dfl(a,b)$;
      forall $d \in T_S$ do $ifl(a,b) \leftarrow ifl(a,b) \cup [iflc(a,d) \cap dfl(d,b)]$;
    end;
  return $ifl$;
end get-ifl;
```

The cost of the nested **forall** loops is $O(|T_S|^3)$. This is dominated by the cost of computing the *iflc* function. Hence, the cost of computing the *ifl* function is $O(|T \times R|*|T_S|^3)$.

C.3. The Flow Function

In order to compute the flow function, it is convenient to first compute the flow of tickets with the copy flag. The latter flow is expressed by the flowc function, defined as follows.

Definition C.2: For every flow^k function define the associated function

$$\text{flowc}^k\text{: SUB} \times \text{SUB} \rightarrow \textbf{power-set}(T \times R)$$

by

$$c/xc \in \text{flowc}^k(A,B) \Leftrightarrow c/xc \in \text{flow}^k(A,B)$$

∎

The flowc function can be computed from the link relation, by evaluating a transitive closure for every ticket type with the copy flag. This computation is shown below.

```
procedure get-flowc(link)
  declare flowc(1...|SUB|,1...|SUB|),
          can-flowc(1...|SUB|,1...|SUB|);
  forall <A,B> ∈ SUB X SUB do flowc(A,B) ← ϕ;
  forall c/xc ∈ T X R do
    begin
      can-flowc ← trclosure({<A,B> ∈ SUB X SUB|
                              link(A,B) ∧ c/xc ∈ dfl(t(A),t(B))});
      forall <A,B> ∈ SUB X SUB do
        if can-flowc(A,B) ∨ A = B
        then flowc(A,B) ← flowc(A,B) ∪ {c/xc};
    end;
  return flowc;
end get-flowc;
```

The cost of each transitive closure operation is O(|SUB|^3). Hence, the cost of computing the flowc function is O(|T X R|*|SUB|^3).

It is then a simple matter to compute the flow function. We need only account for ticket types which do not carry the copy flag. The latter step is straightforward, since such tickets can be transported at most over a single link. This computation is shown below.

```
procedure get-flow(link)
  declare flow(1...|SUB|,1...|SUB|), flowc(1...|SUB|,1...|SUB|);
  flowc ← get-flowc(link);
  forall <A,B> ∈ SUB X SUB do
    begin
      if A = B then flow(A,B) ← T X R
               else flow(A,B) ← flowc(A,B);
      forall C ∈ SUB do if link(C,B) then
          flow(A,B) ← flow(A,B) ∪ [flowc(A,C) ∩ dfl(t(C),t(B))];
    end;
  return flow;
end get-flow;
```

The cost of executing the nested **forall** loops is O(|SUB|^3). This cost is dominated by the cost of computing the flowc function. Hence, the overall cost of computing the flow function is O(|T X R|*|SUB|^3).

C.4. The Maximal Flow Function without Creates

In order to compute the flow# function, we need to focus on the transport of transport tickets. To do so conveniently, we introduce the following notation[2] to isolate that portion of the *dfl* and flow functions which authorize transport of such tickets.

Definition C.3: For every *dfl* function define the associated **direct flow limit function for transport tickets**

$$dflt\colon T_S \times T_S \rightarrow \textbf{power-set}(T_S \times R_T)$$

by

$$dflt(a,b) = dfl(a,b) \cap T_S \times R_T$$

■

Definition C.4: For every flow^k function define the associated **flow function for transport tickets**

$$\texttt{flowt}^k\colon \texttt{SUB}^k \times \texttt{SUB}^k \longrightarrow \textbf{power-set}(T_S \times R_T)$$

by

$$\texttt{flowt}^k(\texttt{A},\texttt{B}) = \texttt{flow}^k(\texttt{A},\texttt{B}) \cap T_S \times R_T$$

■

In the previous section, we developed the **get-flow** algorithm to compute the flow function from the link relation. The flowt function can be computed in a similar manner, by replacing every occurrence of T X R by $T_S \times R_T$ and every occurrence of *dfl* by *dflt* in the **get-flow** algorithm. Call the resulting algorithm **get-flowt**. The cost of computing the flowt function is then $O(|T_S \times R_T| * |SUB|^3)$. Since, $|R_T|$ is known to be four, the cost of computing the flowt function is $O(|T_S| * |SUB|^3)$.

The **get-flowt** algorithm computes the flowt function from the link relation. In order to compute the evolution of the link relation, we next develop an algorithm called **get-link**, which given the initial domain and the flowt function returns the set of links which can then be established.

For a given pair of subjects A and B, there are two aspects which have to be

[2]This notation is introduced, in the main body of the text, in chapter 5. We define it at this point, since the interested reader may wish to read this appendix prior to reading chapter 5.

checked in order to determine whether a link can be established from A to B. These are that the ticket B/s must appear in dom(A) and that the ticket A/r must appear in dom(B). Let us refer to these two conditions as follows.

$$\text{slink} \equiv B/s \in \text{dom}(A)$$
$$\text{rlink} \equiv A/r \in \text{dom}(B)$$

Now, there are three ways by which the slink condition can be satisfied.

1. The ticket B/s is in the initial domain of A, i.e.,

$$B/s \in \text{dom}^{O}(A)$$

2. The subject A obtains the ticket B/s by demanding it, i.e.,

$$t(B)/s \in \textit{demand}(t(A))$$

3. There is some subject C which possesses the ticket B/sc and can transport the ticket B/s to the subject A. There are two ways[3] by which C might obtain the ticket B/sc. The subject C might have this ticket in the initial state or C might obtain the ticket by a demand operation. This case then requires that

$$[B/sc \in \text{dom}^{O}(C) \ \vee \ t(B)/sc \in \textit{demand}(t(C))]$$
$$\wedge$$
$$t(B)/s \in \text{flowt}(C,A)$$

A similar set of conditions can be developed for checking whether the rlink condition is satisfied.

The algorithm shown below checks these conditions for every pair of subjects A and B. If both conditions are satisfied, the pair <A,B> is inserted in the link relation otherwise the pair is omitted from the relation.

[3]We need not consider the case where C obtains the ticket B/sc from some subject D and then transports it to subject A, since this amounts to subject D transporting the ticket to A and will be accounted for.

```
function get-link (dom$^O$, flowt)
  declare link (1...|SUB|, 1...|SUB|), slink, rlink;
  forall <A, B> ∈ SUB X SUB do
    begin
      if B/s ∈ dom$^O$(A) ∨ $t$(B)/s ∈ demand($t$(A))
      then slink ← true else slink ← false;
      if A/r ∈ dom$^O$(B) ∨ $t$(A)/r ∈ demand($t$(B))
      then rlink ← true else rlink ← false;
      forall C ∈ SUB do
          begin
            if [B/sc ∈ dom$^O$(C) ∨ $t$(B)/sc ∈ demand($t$(C))] ∧
              $t$(B)/s ∈ flowt(C, A)
            then slink ← true;
            if [A/rc ∈ dom$^O$(C) ∨ $t$(A)/rc ∈ demand($t$(C))] ∧
              $t$(A)/r ∈ flowt(C, B)
            then rlink ← true;
          end;
      if slink ∧ rlink then link(A, B) ← true
                        else link(A, B) ← false;
    end;
  return link;
end get-link;
```

The inner **forall** loop checks for the existence of a suitable subject C who can assist in setting up link(A,B). The cost of this inner loop is $O(|SUB|)$. Since this loop is executed for every pair of subjects A and B, the net cost of the **get-link** algorithm is $O(|SUB|^3)$.

Establishing the new set of links may change the flowt function. In turn enhancement of the flowt function may allow more links to be established. We can alternately execute the **get-flowt** and the **get-link** algorithms until the link relation stabilizes. This computation is shown below.

```
procedure get-flowt# (dom°)
  declare flowt(1...|SUB|,1...|SUB|),
          link^k(1...|SUB|,1...|SUB|), link^{k+1}(1...|SUB|,1...|SUB|);
  forall <A,B> ∈ SUB X SUB do flowt(A,B) ← ∅;
  link^k ← get-link(dom°,flowt);
  repeat forever
    begin
      flowt ← get-flowt(link^k);
      link^{k+1} ← get-link(dom°,flowt)
      if link^{k+1} - link^k = ∅ then return flowt
                                 else link^k ← link^{k+1};
    end;
end get-flowt#;
```

The initial set of links is computed by setting all values of the flowt function to be empty and invoking the **get-link** algorithm. The flowt function and the link relation are then computed alternately till the link relation stabilizes. Both the **get-flowt** and **get-link** computations have a cost of O($|SUB|^3$), which is then also the cost of each iteration of the repeat loop. Since at most $|SUB|^2$ links can be established, there are at most $|SUB|^2$ iterations of this loop. Hence, the time complexity of the **get-flowt#** algorithm is no worse than O($|SUB|^5$).

Once the flowt# function has been computed, computation of the flow# function is straightforward, as shown below.

```
procedure get-flow# (dom°)
  declare flow(1...|SUB|,1...|SUB|), flowt(1...|SUB|,1...|SUB|),
          link(1...|SUB|,1...|SUB|);
  flowt ← get-flowt#(dom°);
  link ← get-link(dom°,flowt);
  flow ← get-flow(link);
  return flow;
end get-flow#;
```

The complexity of the **get-flow** algorithm is O($|T \times R| * |SUB|^3$). This is dominated by the O($|SUB|^5$) cost of computing the flowt# function. Hence, the complexity of the **get-flow#** algorithm is O($|SUB|^5$).

C.5. The Augmented Flow Function

Computation of the flow† function amounts to computing the flow$''$ function for the augmented initial state. Construction of the augmented initial state is a straightforward matter and we can ignore the cost of doing so.

For each subject present in the initial state construction of the augmented initial state introduces at most $|T_S|-1$ new subjects. Hence,

$$|SUB^{aug}| \leq |T_S| * |SUB^{O}|$$

The flow$''$ function can be computed in $O(|SUB|^5)$ time, as discussed in appendix C.4. Then, the cost of computing the flow† function is no worse than $O(|T_S|^5 * |SUB^{O}|^5)$. This computation clearly is expensive in the worst case, but nevertheless it is tractable.

Flow-Invariant Schemes:[4] For the special case of schemes, which are flow-invariant (in the weak sense[5]), there is a considerable speed-up in computation of the flow† function. The flowt† function can now be computed, from the link relation in the augmented initial state, by using the **get-flowt** algorithm on page 182. The cost of this computation is $O(|T_S| * |SUB|^3)$. Since there are at most $|T_S| * |SUB^{O}|$ subjects in an augmented initial state, the cost of computing the flowt† function is no worse than $O(|T_S|^4 * |SUB^{O}|^3)$. The flow† function can then be computed from the flowt† function, by using the **get-flow$''$** algorithm on page 185. The complexity of this computation is $O(|T \times \mathbf{R}| * |T_S|^3 * |SUB^{O}|^3)$. This is considerably better than the possibility of an $O(|T_S|^5 * |SUB^{O}|^5)$ cost for computing the flow† function in general.

[4]The remainder of this section is relevant to the discussion of chapter 5 and can be ignored till that point.

[5]See definition 5.3 on page 113.

C.6. The Flow-Invariant Property

Theorem 5.6 on page 134 characterizes the flow-invariant property for self-copy schemes as follows.

A self-copy scheme is flow-invariant if and only if for all pairs of subject types $<a,b>$

1. **If** $[b/s \in demand(a) \wedge a/r \notin demand(b)]$ **then** $(\forall\alpha)\,[a/r \in mflt(a,\alpha,b) \rightarrow dflt(a,b) \subseteq mflt(a,\alpha,b)]$

2. **If** $[b/s \notin demand(a) \wedge a/r \in demand(b)]$ **then** $(\forall\beta)\,[b/s \in mflt(b,\beta,a) \rightarrow dflt(a,b) = \phi]$

3. **If** $[b/s \notin demand(a) \wedge a/r \notin demand(b)]$ **then** $(\forall\alpha)\,(\forall\beta)\,[a/r \in mflt(a,\alpha,b) \wedge b/s \in mflt(b,\beta,a) \rightarrow dflt(a,b) \subseteq mflt(a,\alpha,b)]$

4. **If** $[b/s \in demand(a) \wedge a/r \in demand(b)]$ **then** $dflt(a,b) = \phi$

By negating both sides of this theorem and applying some straightforward logical transformations[6], we obtain the equivalent statement that

A self-copy scheme is <u>not</u> flow-invariant if and only if there exists a pair of subject types $<a,b>$ such that

1. $[b/s \in demand(a) \wedge a/r \notin demand(b)] \wedge (\exists\alpha)\,[a/r \in mflt(a,\alpha,b) \wedge dflt(a,b) \not\subseteq mflt(a,\alpha,b)]$

or 2. $[b/s \notin demand(a) \wedge a/r \in demand(b)] \wedge (\exists\alpha)\,[a/s \in mflt(a,\alpha,b) \wedge dflt(b,a) \neq \phi]$

or 3. $[b/s \notin demand(a) \wedge a/r \notin demand(b)] \wedge (\exists\alpha)\,(\exists\beta)\,[a/r \in mflt(a,\alpha,b) \wedge b/s \in mflt(b,\beta,a) \wedge dflt(a,b) \not\subseteq mflt(a,\alpha,b)]$

or 4. $[b/s \in demand(a) \wedge a/r \in demand(b)] \wedge dflt(a,b) \neq \phi$

In order to obtain an algorithm for checking these conditions, we will eliminate all references to the *mflt* function. In each case, testing that part of the condition which refers to the *demand* function is straightforward and we will focus on the remaining part. In particular, testing condition 4 is trivial. We consider the first three conditions in turn.

[6] It is important to recognize that the condition of theorem 5.6 can be treated as a conjunction of implications. This is because exactly one of the conditions of the four if statements will be satisfied for a given pair of subject types.

<u>**Condition 1**</u>: We begin by defining the following notation, for denoting that part of the *dflt* function which authorizes transport of copiable transport tickets.

Definition C.5: For every *dflt* function define the associated function

$$dfltc\colon T_S \times T_S \rightarrow \textbf{power-set}\,(T_S \times R_T)$$

by

$$c/xc \in dfltc\,(a,b) \leftrightarrow c/xc \in dflt\,(a,b)$$

■

Consider the requirement that $dflt(a,b)$ is not a subset of $mflt(a,\alpha,b)$. If α is the empty string then, by definition, $mflt(a,\Lambda,b)$ is the same as $dflt(a,b)$. Hence, α must be non-empty. Let α be of the form $a_1 ... a_n$, so that

$$mflt\,(a,a_1 \ldots a_n,b) =$$
$$dfltc\,(a,a_1) \cap \ldots \cap dfltc\,(a_i,a_{i+1}) \cap \ldots \cap dflt\,(a_n,b)$$

Then, $dflt(a,b)$ is not a subset of $mflt(a,a_1 ... a_n,b)$ if and only if at least one of the following conditions is satisfied.

$$\text{(i)}\quad dflt\,(a,b) \not\subseteq dfltc\,(a,a_1)$$
$$\text{(ii)}\quad (\exists i)\,[dflt\,(a,b) \not\subseteq dfltc\,(a_i,a_{i+1})]$$
$$\text{(iii)}\quad dflt\,(a,b) \not\subseteq dflt\,(a_n,b)$$

We will consider each one of these possibilities.

<u>Case (i)</u>: Let α have the form $c\beta$ where β may possibly be the empty string. Then,

$$a/r \in mflt\,(a,c\beta,b)$$
$$\leftrightarrow$$
$$a/rc \in dflt\,(a,c) \wedge a/r \in mflt\,(c,\beta,b)$$

The condition that the scheme is not flow-invariant, because of the pair $<a,b>$, can then be expressed as

$$(\exists c\beta)\,[a/rc \in dflt\,(a,c) \wedge a/r \in mflt\,(c,\beta,b) \wedge$$
$$dflt\,(a,b) \not\subseteq dfltc\,(a,c)\,]$$

By definition of the *iflt* function, this is identical to

$$(\exists c)\,[a/rc \in dflt\,(a,c) \wedge a/r \in iflt\,(c,b) \wedge$$
$$dflt\,(a,b) \not\subseteq dfltc\,(a,c)\,]$$

<u>Case (ii)</u>: For this case to apply the length of α must be at least two. Let α be of the form $\beta_1 cd\beta_2$, where β_1 and/or β_2 may possibly be empty. Then,

$$a/r \in mflt\,(a,\beta_1 cd\beta_2,b)$$
$$\leftrightarrow$$
$$[a/rc \in mflt\,(a,\beta_1,c)\,] \wedge [a/rc \in dflt\,(c,d)\,] \wedge [a/r \in mflt\,(d,\beta_2,b)\,]$$

The condition that the scheme is not flow–invariant, because of the pair $<a,b>$, can then be expressed as

$$(\exists\beta_1 cd\beta_2)\,[a/rc \in mflt(a,\beta_1,c) \wedge a/rc \in dflt(c,d) \wedge a/r \in mflt(d,\beta_2,b) \wedge dflt(a,b) \not\subseteq dfltc(c,d)]$$

By definition of the *iflt* function, this is identical to

$$(\exists cd)\,[a/rc \in iflt(a,c) \wedge a/rc \in dflt(c,d) \wedge a/r \in iflt(d,b) \wedge dflt(a,b) \not\subseteq dfltc(c,d)]$$

Case (iii): Let α have the form βd where β may possibly be the empty string. Then,

$$a/r \in mflt(a,\beta d,b)$$
$$\Leftrightarrow$$
$$[a/rc \in mflt(a,\beta,d)] \wedge [a/r \in dflt(d,b)]$$

The condition that the scheme is not flow–invariant, because of the pair $<a,b>$, can then be expressed as

$$(\exists\beta d)\,[a/rc \in mflt(a,\beta,d) \wedge a/r \in dflt(d,b) \wedge dflt(a,b) \not\subseteq dflt(a,d)]$$

By definition of the *iflt* function, this is identical to

$$(\exists d)\,[a/rc \in iflt(a,d) \wedge a/r \in dflt(d,b) \wedge dflt(a,b) \not\subseteq dflt(a,d)]$$

Thus, in each of the three cases we can eliminate reference to the *mflt* function. The resulting conditions are in terms of the *dflt, dfltc* and *iflt* functions. These conditions can be easily tested. This computation is coded as a boolean procedure, **flow-invariant-r** shown below, which takes a pair of subject types as input and determines whether this pair satisfies the condition for the scheme to be flow-invariant. The three **if** statements correspond to the three cases discussed above.

```
procedure flow-invariant-r $(a,b)$
  if $(\exists c \in T_S)$
     $[a/rc \in dflt(a,c) \wedge a/r \in iflt(c,b) \wedge dflt(a,b) \not\subseteq dfltc(a,c)]$
  then return false;
  if $(\exists <c,d> \in T_S \times T_S)$
     $[a/rc \in iflt(a,c) \wedge a/rc \in dflt(c,d) \wedge a/r \in iflt(d,b) \wedge$
                                                  $dflt(a,b) \not\subseteq dfltc(c,d)]$
  then return false;
  if $(\exists d \in T_S)$
     $[a/rc \in iflt(a,d) \wedge a/r \in dflt(d,b) \wedge dflt(a,b) \not\subseteq dflt(a,d)]$
  then return false;
  return true;
end flow-invariant-r;
```

The cost of evaluating this procedure is dominated by the cost of evaluating the existential predicate in the second **if** statement. Since this involves an iteration through the set $\mathbf{T}_S \times \mathbf{T}_S$, the cost is $O(|\mathbf{T}_S|^2)$.

<u>**Condition 2**</u>: This condition is easy to check, since by definition

$$(\exists\alpha)\,[a/s \in \mathit{mflt}(a,\alpha,b) \wedge \mathit{dflt}(b,a) \neq \phi]$$
$$\Leftrightarrow$$
$$[a/s \in \mathit{iflt}(a,b)] \wedge [\mathit{dflt}(b,a) \neq \phi]$$

<u>**Condition 3**</u>: This condition can be written as a conjunction of two separate conditions, one involving the string α and the other involving the string β, as follows.

$$(\exists\alpha)\,[a/r \in \mathit{mflt}(a,\alpha,b) \wedge \mathit{dflt}(a,b) \not\subseteq \mathit{mflt}(a,\alpha,b)]$$
$$\wedge$$
$$(\exists\beta)\,[b/s \in \mathit{mflt}(b,\beta,a)]$$

The first term of this conjunction is exactly the same as condition 1. Hence, the **flow-invariant-r** procedure can be used to evaluate the first term. By definition of the *iflt* function, the second term of the conjunction is exactly that $b/s \in \mathit{iflt}(b,a)$.

<u>**The Resulting Algorithm**</u>: It is now a simple matter to obtain a algorithm for testing whether the requirements of theorem 5.6 are satisfied. The algorithm shown below performs this computation by testing the appropriate condition for every pair of subject types.

```
procedure flow-invariant
  forall $\langle a,b \rangle \in T_S \times T_S$ do
    begin
      if $[b/s \in demand(a) \wedge a/r \notin demand(b)] \wedge$
        $\sim$flow-invariant-r $(a,b)$
      then return false;
      if $[b/s \notin demand(a) \wedge a/r \in demand(b)] \wedge$
        $b/s \in iflt(a,b) \wedge dflt(a,b) \neq \phi$
      then return false;
      if $[b/s \notin demand(a) \wedge a/r \notin demand(b)] \wedge$
        $b/s \in iflt(b,a) \wedge \sim$flow-invariant-r $(a,b)$
      then return false;
      if $[b/s \in demand(a) \wedge a/r \in demand(b)] \wedge$
        $dflt(a,b) \neq \phi$
      then return false;
    end;
  return true;
end flow-invariant;
```

The first three **if** statements correspond to the three conditions discussed above, while the final **if** statement corresponds to the fourth (trivial) condition. Since the **flow-invariant-r** predicate requires $O(|T_S|^2)$ time for its evaluation, the cost of this algorithm is no worse than $O(|T_S|^4)$; which is of the same order as the cost of computing the *iflt* function.

C.7. The May-Link Relation

In theorem 6.8 on page 149, we characterized the *may-link* relation for self-copy schemes as follows.

$$\textit{may-link}\,(a,b)$$
$$\Leftrightarrow$$

1. $[b/s \in \textit{demand}\,(a) \;\wedge\; a/r \notin \textit{demand}\,(b)] \;\wedge$
   $(\exists\alpha\neq\Lambda)\,[a/r \in \textit{mflt}\,(a,\alpha,b)]$

or 2. $[b/s \notin \textit{demand}\,(a) \;\wedge\; a/r \in \textit{demand}\,(b)] \;\wedge$
   $(\exists\beta)\,[b/s \in \textit{mflt}\,(b,\beta,a)]$

or 3. $[b/s \notin \textit{demand}\,(a) \;\wedge\; a/r \notin \textit{demand}\,(b)] \;\wedge$
   $(\exists\alpha\neq\Lambda)\,[a/r \in \textit{mflt}\,(a,\alpha,b)] \;\wedge\; (\exists\beta)\,[b/s \in \textit{mflt}\,(b,\beta,a)]$

or 4. $[b/s \in \textit{demand}\,(a) \;\wedge\; a/r \in \textit{demand}\,(b)]$

For any pair $<a,b>$ of subject types exactly one of these conditions needs to be tested. In order to obtain an algorithm for checking these conditions, we will eliminate all references to the *mflt* function. In each case testing that part of the condition which refers to the *demand* function is straightforward and we will focus on the remaining part. In particular, testing condition 4 is trivial. We consider the first three conditions in turn.

<u>Condition 1</u>: Since the string α is non-empty, we can assume that it contains at least one subject type, say c; so that α is of the form $\beta_1 c\beta_2$ where β_1 and/or β_2 may possibly be empty. But then, the condition can be transformed as follows.

$$\begin{aligned}
& (\exists\alpha\neq\Lambda)\,[a/r \in \textit{mflt}\,(a,\alpha,b)] \\
\Leftrightarrow\; & (\exists\beta_1 c\beta_2)\,[a/r \in \textit{mflt}\,(a,\beta_1 c\beta_2,b)] \\
\Leftrightarrow\; & (\exists\beta_1 c\beta_2)\,[a/rc \in \textit{mflt}\,(a,\beta_1,c) \;\wedge\; a/r \in \textit{mflt}\,(c,\beta_2,b)] \\
\Leftrightarrow\; & \quad (\exists c)\,[a/rc \in \textit{iflt}\,(a,c) \;\wedge\; a/r \in \textit{iflt}\,(c,b)]
\end{aligned}$$

<u>Condition 2</u>: For this case the existential condition on the *mflt* function is exactly that

$$b/s \in iflt(b,a)$$

<u>**Condition 3**</u>: Here, there are two parts to the condition on the *mflt* function; equivalent to conditions 1 and 2 respectively. Condition 3 can then be written as follows.

$$(\exists c)\,[a/rc \in iflt(a,c) \;\wedge\; a/r \in iflt(c,b)]$$

Condition 3 is then equivalent to

$$[b/s \in iflt(b,a)] \;\wedge\; (\exists c)\,[a/rc \in iflt(a,c) \;\wedge\; a/r \in iflt(c,b)]$$

<u>**The Resulting Algorithm**</u>: The *may-link* relation for self-copy schemes can now be computed by testing the appropriate condition for every pair of subject types. This computation is shown below, where the four **if** statements respectively correspond to the four conditions on page 191.

```
procedure get-may-link
  declare may-link (1...|$T_S$|,1...|$T_S$|);
  forall <a,b> ∈ $T_S \times T_S$ do
    begin
      if $[b/s \in demand(a) \wedge a/r \notin demand(b)] \wedge$
         $(\exists c \in T_S)[a/rc \in iflt(a,c) \wedge a/r \in iflt(c,b)]$
      then may-link(a,b) ← true else may-link(a,b) ← false;
      if $[b/s \notin demand(a) \wedge a/r \in demand(b)] \wedge$
         $b/s \in iflt(b,a)$
      then may-link(a,b) ← true else may-link(a,b) ← false;
      if $[b/s \notin demand(a) \wedge a/r \notin demand(b)] \wedge$
         $b/s \in iflt(b,a) \wedge$
         $(\exists c \in T_S)[a/rc \in iflt(a,c) \wedge a/r \in iflt(c,b)]$
      then may-link(a,b) ← true else may-link(a,b) ← false;
      if $[b/s \in demand(a) \wedge a/r \in demand(b)]$
      then may-link(a,b) ← true else may-link(a,b) ← false;
    end;
  return may-link;
end get-may-link;
```

Since at most one condition with complexity $O(|T_S|)$ must be tested for every pair of subject types and there are $|T_S|^2$ such pairs, the cost of this computation is $O(|T_S|^3)$.

REFERENCES

[1] Cohen, E., and Jefferson, D.
Protection in the Hydra Operating System.
Fifth Symposium on Operating Systems Principles :141–160, 1975.

[2] Denning, D.E., and Denning, P.J.
Data Security.
Computing Surveys 11(3):227–249, 1979.

[3] Denning, E. R.
Cryptography and Data Security.
Addison–Wesley, 1982.

[4] Dennis, J.B., and Van Horn, F.C.
Programming Semantics for Multiprogrammed Computations.
CACM 9(3):143–155, 1966.

[5] Graham, G.S., and Denning, P.J.
Protection – Principles and Practice.
AFIPS 40:417–429, 1972.

[6] Harrison, M.H., Russo, W.L., and Ullman, J.D.
Protection in Operating Systems.
CACM 19(8):461–471, 1976.

[7] Jones, A.K., Lipton, R.J., and Snyder, L.
A Linear Time Algorithm for Deciding Security.
Proc. 17th Symp. on the Foundations of Comp. Sci. , 1976.

[8] Kieburtz, R.B., and Silberschatz, A.
Capability Managers.
IEEE Trans. on Software Engg. SE–4(4):467–477, 1978.

[9] Lampson, B. W.
Protection.
Proc. 5th Princeton Symp. of Info. Sci. and Syst. :437–443, March, 1971.

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

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

[12] Minsky, N.
An Operation-Control Scheme for Authorization in Computer Systems.
Int. J. Comp. & Info. Sc. 7(2):157-191, 1978.

[13] Minsky, N.
The Principle of Attenuation of Privileges and its Ramifications.
In DeMillo, R.A. et al (editors), *Foundations of Secure Computations*, pages 255-276. Academic Press, 1978.

[14] Minsky, N.
Synergistic Authorization in Database Systems.
Proc. 7th Int Conf on Very Large Data Bases , 1981.

[15] Minsky, N.
Selective and Locally Controlled Transport of Privileges.
ACM Transactions on Programming Languages and Systems , To appear, 1983.

[16] Reingold, E.M., Nievergelt, J., and Deo, N.
Combinatorial Algorithms: Theory and Practise.
Prentice Hall, 1977.

[17] Snyder, L.
Formal Models of Capability-Based Protection Systems.
IEEE Trans. on Computers C-30(3):172-181, March, 1981.

[18] Wilkes, M.V., and Needham, R.M.
The Cambridge CAP Computer and its Operating System.
Elsevier North Holland, 1979.

[19] Wulf, W., Cohen, E., Corwin, W., Jones, A., Levin, R., Pierson, C., and Pollack, F.
Hydra: The Kernel of a Multiprocessor Operating System.
CACM 17(6):337-345, 1974.

VITA

RAVINDERPAL SINGH SANDHU

1953	Born August 9 in Pune, Maharashtra, India.
1968	Graduated from the Doon School, Dehra Dun, India.
1974	B.Tech. in Electrical Engineering, Indian Institute of Technology, Bombay, India.
1976	M.Tech. in Electrical Engineering (Computer Technology), Indian Institute of Technology, Delhi, India.
1975–77	Research Assistant, Department of Electrical Engineering, Indian Institute of Technology, Delhi, India.
1977	Lecturer, School of Computer and Systems Sciences, Jawaharlal Nehru University, Delhi, India.
1977–78	Systems Executive, Head Office Systems Group, Hindustan Computers Limited, Delhi, India.
1978–82	Teaching and Research Assistantships, Department of Computer Science, Rutgers University, New Brunswick, New Jersey.
1979–82	University Fellowship, Rutgers University.
1980	M.S. in Computer Science, Rutgers University.
1983	Ph.D. in Computer Science, Rutgers University.
1983	Assistant Professor, Department of Computer Science, Ohio State University, Columbus, Ohio.