1.1 Motivation

The development of information technology in recent years has led to the widespread use of information systems in various institutions and organizations, such as industries, banks, manufacture and defence systems. This has meant that more and larger amounts of data is processed, manipulated and managed by today's modern, large, complex information systems. In such an environment, security issues are becoming increasingly important. Security, like performance and accuracy, is now being considered as a key factor in the evaluation of the quality of an information system.

At a high level, security consists of maintaining three characteristics: confidentiality, integrity and availability [5, 67]. Confidentiality is concerned with only allowing authorized parties to access the resources and information in a computer system. Confidentiality is sometimes called secrecy or privacy. Integrity allows the detection of unauthorized modification of the resources of an information system. It is needed to ensure that the information is accurate and insulated against accidental and deliberate change. Availability indicates that the resources are accessible to authorized parties when needed. An authorized party should not be prevented from accessing objects to which the party has legitimate access. The mechanism associated with these three characteristics can sometimes overlap in some information systems.

Generally, there are two main security services to reach such a security goal: authentication and authorization. Authentication or identification verifies the identity of the party or user who is about to enter the system. This process ensures that only authorized party or user can enter the system. Many kinds of encryption and decryption
techniques have been used in the process of authentication. Authorization or access control enforces different levels of accesses to system resources for an authenticated party or user and provides for the protection of system resources against unauthorized use. It controls what kind of user or subject can have what kind of access right for what kind of resource or object. Access control ensures the availability of data to the right users. As a side benefit it also provides limited confidentiality: data not disclosed to unauthorized users and integrity: data not modified in an unauthorized way.

The authorization service is closely related to the authentication service. Usually, a user or a process acting on a user's behalf must be authenticated before being allowed to enter the system, then an access control service can effectively mediate access to the system resources [66].

This thesis deals with the authorization aspect of the security problem. We assume that the appropriate authentication service has been performed. Authorization specifies the security policy of an organization. A security policy is the set of decisions or rules which determine an organization's posture towards security. More precisely, a security policy determines the limits of acceptable access to the system and the acceptable manipulations against the resources of the system. Naturally, security policies differ from organization to organization and from application to application. Obviously, the security policy for a university's system is different from the policy for a military system.

For an organization, once its security policy requirements are determined, it needs a precise, flexible and expressive specification. Much research has been done in this area. So far there are many different authorization specifications. Some of the specifications are simple, precise but lack flexibility, or are expressive but lack precision. In this thesis, we address the issue of specification of authorization in a way that is precise, flexible and expressive. We use a formal method to specify and reason about authorization policies and to discuss their temporal property and transformations. We also consider the application of authorization specification in object oriented database systems.

THESIS
1.2 Security Policy and Access Control System

1.2.1 Security Policy

The security policy of a system is a high level guideline concerning design and management of an authorization system [35]. The policy expresses the basic choices taken by an organization for its own information security. The definition of a security policy leads to the explicit formulation of security strategies. Security strategies have been improving with the development of contemporary information systems. Separation of duty and other independent functional checks have been introduced, thereby helping to balance the relationship between the resources to be protected and the security surrounding the resources. Security strategies are highly organization dependent. They can vary from organization to organization and can also vary depending on the type of application. A precise and expressive specification of security strategies of an organization allows critical points and possible conflicts to be discussed and solved, and the objectives to be analyzed and concluded.

Security policies define the principles according to which an access request is granted or denied. Authorization or access control rules are the expression of security policies. They determine the system behaviour at run time. It is also necessary to state how the set of authorization rules is administered.

1.2.2 Access Control System

Access control in information systems is responsible for ensuring that all accesses to the system resources occur exclusively according to the access policies and rules specified by the security strategies. It detects and prevents any unauthorized attempts against the system. The access control system is the mechanism used to enforce the security as stated by the authorization policies. It uses the authenticated identities of users or processes, information about these users or processes, and/or capabilities to determine and enforce access. If a user or process attempts to use an unauthorized resource, or an authorized resource with an unauthorized type of access, the access control system will reject the attempt and may additionally report the incident for the
purposes of generating an alarm and recording it as part of a security audit trail. Figure 1.1 illustrates a typical access control system.

![Diagram of an access control system](image)

**Figure 1.1: An access control system.**

Generally, an access control system includes subjects (users, processes) who access objects (system resources, data) by certain type of operations (read, write, execute). It consists of two components:

1. A set of access policies and rules: it states the types of access rights the subjects authorized for the objects.

2. An evaluation mechanism: it evaluates the access requests against the stated access policies and rules to decide if to grant or deny these access requests.

When a subject requests a certain access to an object, the evaluation mechanism will check the request with respect to the access policies and rules. If such a request does not violate the access policies and rules, or it is evaluated true in the domain of the access policies and rules, then this request will be granted. If the request violates the access policies and rules, or it is evaluated false in the domain of the access policies
and rules, then this request will be denied. If it can not decide if the request violates the access police or not, or it is evaluated neither true nor false in the domain, the decision for such a request will be undecidable.

1.3 Authorization Specifications

Generally, an authorization or access control mechanism is based on three types of information.

1. Subject: A subject is an entity that is capable of accessing an object. A subject refers to an individual user, party, a host, a terminal or an application.

2. Object: An object is an entity to which access can be controlled. An object can refer to an individual data field, a program, a record, a file or entire database.

3. Access Right: Access right defines the way in which a subject can access an object. An access right may be specified for each subject-object pair. Examples of access rights can be read, write, execute or some other operations depending on different authorization policies and different applications.

Access control models are traditionally classified into two conventional categories: discretionary access control (DAC) and mandatory access control (MAC). A DAC model essentially enumerates all the subjects and objects in a system and regulates the access to the object by a subject based on the identity of the subject. It can be best represented by the HRU’s [42] access control matrix with a row for each object and a column for each subject, each entry of the matrix represents the authorization(s) the corresponding subject holds for the corresponding object. In this model, there is no attribute or sensitivity level associated with each subject and object. It is a loosely controlled model.

On the other hand, in a traditional MAC model, each subject and object has an attribute associated with it to indicate its sensitivity level. The information flow among these sensitivity levels forms a lattice. In MAC, the authorization to access an object by a subject is determined by the attributes of the subject and object. All authorizations are controlled by a reference monitor. The reference monitor enforces two rules.
for information flow in MAC: no-read-up and no-write-down which ensures that information can only flows from low sensitivity level to high sensitivity level and prevent information flows from high sensitivity level to low sensitivity level. It is a tightly controlled model.

Generally, there are access matrix and access control list and logic models for specifying authorization. The access matrix model as formalized by Harrison, Ruzzo and Ullman (HRU) [42] has broad expressive power. It is a commonly used form for representation of access control. In access matrix, the rows represent the subjects and the columns represent the objects. the intersections represent the access rights the subjects hold for the corresponding objects. Figure 1.2 shows a general access matrix.

<table>
<thead>
<tr>
<th></th>
<th>file1</th>
<th>file2</th>
<th>file3</th>
<th>file4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>R</td>
<td></td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Bob</td>
<td></td>
<td>W</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Carol</td>
<td>R</td>
<td>W</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>David</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 1.2: An access matrix.

From this matrix, we can get the following access information:

- Alice has read right for file1
- Alice has read right for file3
- Bob has write right for file 2
- Bob has read right for file3
- Carol has read right for file1
- Carol has write right for file2
- Carol has read right for file3
In an access matrix, when the number of subjects and objects are very large, it is quite complex to manipulate it. Also a matrix for a real system usually tends to be very sparse, and hence many systems do not store such a matrix for access rights. Instead, they tend to use an access control list or capability based approach.

In the access control list approach, for each object, there is an access control list which is used to define the access rights for each subject. For certain objects, the access control list could be empty. This means currently no subject has any access right for that object. Figure 1.3 shows the access control list for the four objects file1, file2, file3 and file4 corresponding to the example in Figure 1.2.

![Access control lists](image)

Figure 1.3: Access control lists.

In the capability based approach, every subject has a capability list. A capability list for a certain subject specifies the rights to access each object. For certain subject, its capability list could be an empty list. This means the subject does not have access right for any object corresponding to the example in Figure 1.2. Figure 1.4 shows the capability lists for four subjects Alice, Bob, Carol and David.

Traditionally, the access right information can be stored in an access matrix, or access control list, or capability list, or some kind of combination of these approaches. Practical implementation of an access matrix usually takes advantage of the sparse-
ness of the matrix. These approaches are usually specified using some low level system specific mechanism. The semantics of authorization is dependent on the semantics of the low level mechanisms, which is not formally defined and may vary from one implementation to another. This causes problems in large scale information systems with heterogeneous implementations. Also these approaches are not sufficiently flexible to deal with multi-domain authorization problems, and can not support different access control policies. To overcome these drawbacks, logic approaches for representing and evaluating authorization have been proposed [36, 48, 91].

With a logic approach, authorization is specified using a high level independent semantics and is separated from its implementation in system specific mechanisms. The problem of authorization can be divided into two related subproblems: authorization representation and authorization evaluation. Representation involves the specification of authorization policy while evaluation refers to the process of determining whether access is to be granted or not.

Normally, the logic approaches use propositional logic, first order logic, default logic or combinations of these logics. The alphabets of the logic languages generally have the following items:
• a finite set of variables
• a finite set of propositional variables
• two propositional constants $T$ and $F$
• a finite set of subject constants
• a finite set of subject variables
• a finite set of object constants
• a finite set of object variables
• a finite set of access right constants
• a finite set of access right variables
• a finite set of predicate symbols
• logical connectives

In a logic representation, the access facts can be represented by some predicates such as $holds(S,A,O)$ which refers to the fact that subject $S$ holds access right $A$ for object $O$. The access constraints or access rules can be specified by a logical formula. For instance, using default logic, a formula $f : f'/g$ where $f$, $f'$ and $g$ are formulas and are respectively called the prerequisite, assumption and consequent of the rule. This rule is interpreted as follows: if $f$ is true and there is no evidence to prove that $f'$ is true, then we can conclude that $g$ is true.

We can have other representations of authorization rules using a logic approach.

\[
holds(S_1, Modify, File) \land holds(S_2, Modify, File) \tag{1.1}
\]

\[
holds(S_1, Modify, File) \supset \neg holds(S_2, Modify, File) \tag{1.2}
\]

\[
\forall s g a o. s \in g \land holds(g, a, o) \supset holds(s, a, o) \tag{1.3}
\]

where (1.1) refers to the fact that both subjects $S_1$ and $S_2$ can modify objects $file$; (1.2) means that if subject $S_1$ has permission to modify the object $file$, then subject $S_2$ is
not allowed to do so; rule (1.3) can be interpreted as: for any subject $s$, access right $a$, object $o$ and subject group $g$, if $s$ is a member of group $g$ and $g$ holds access right $a$ for object $o$, then $s$ holds access right $a$ for object $o$ as well.

Using a logic approach, authorization policy can be specified by a knowledge base which comprises a finite set of access facts and a finite set of access rules. Unlike most other existing approaches, the semantics of authorization is defined independently and is separated from its implementation mechanism. The separation means that the authorization specification is independent from the system which enforces the authorization policy. This enables users to specify authorization policy according to which access control policy must be enforced. Different policies can be specified on different objects according to the needs of the users. The expressiveness of logic approaches is quite powerful. They can represent different access control policies, and the constraint rules which the system must obey. Actually, they can be used to specify nearly all of the existing authorization policies, and have a great potential for specifying more complex policies.

1.4 Authorization Transformations

An authorization policy states the access rights and rules to ensure system security. Such a policy has a temporal characteristic, that is, as the system and application requirements change, it needs to be updated as well to capture the changing needs. Formally, we refer to such a change as authorization transformation. Representation, evaluation and analysis of such changes form an important part of the design of authorization policies.

An authorization transformation changes the state of a policy base. Some subjects in the policy base may gain more access rights, while others may lose some access rights on some objects. The transformation is generally nonmonotonic. That is, in the process of transformation, the subjects do not always gain more access rights, they could also lose some of their existing access rights when performing the transformation. Therefore, an authorization transformation has a nonmonotonic characteristic.

In general, transformations can be classified as an internal transformation or an ex-
ternal transformation. Internal transformations allow a subject who possesses certain rights for an object to obtain additional rights. External transformations allow a subject to give some other right for an object to another subject.

Such a transformation can be represented using pseudo-code as follows:

\[
\text{if cond} \\
\text{then trans(user, file, r)} \Rightarrow (user, file, w)
\]

The above pseudo-code can be interpreted as: if permitted (condition satisfied), then perform transformation of the access right user holds for file from read to write.

A formal logic approach can also be used for specifying authorization transformations. In this approach, an authorization policy can be specified by a policy base, which is a knowledge base which comprises a finite set of facts and a finite set of access constraints. The facts represent explicitly the access rights the subjects hold for the objects. The access constraints are a set of rules which the authorization policy should satisfy. They are a set of closed first order formulas. Figure 1.5 illustrates the relationship between authorization policy transformations and knowledge base system updates.

![Diagram of authorization policy transformation and knowledge base update](image)

**Figure 1.5: Knowledge base update and authorization transformation.**

A structure of transformation can be defined on such a policy base which defines the precondition that needs to be satisfied before transformation, and postcondition
that becomes valid after the transformation. Several logic updating approaches can be employed in the procedure of policy base transformation (knowledge base update) such as model based updating and knowledge base updates. [51, 90].

Belief revision concepts [1, 44, 51] can also be applied to authorization transformation. Belief revision deals with the issue of updating one's state of beliefs with newly received information, in particular when the new information conflicts with the current state of belief [90]. This work addresses the issue of updating a knowledge base where the knowledge base is expressed as logical theory.

When using a logical theory to express the authorization policy as a knowledge base, these knowledge base updates and belief revision techniques can be employed with the authorization transformation. From the knowledge base update view point, authorization transformation can be partitioned into two classes: update only on a given set of facts and update on all of the formulas present in the knowledge base.

Considering the likelihood that access rights will change far more often than access control rules, most authorization transformations will deal with access rights update. That is, the access rights transformation. Access control rules are considered relatively stable. More comprehensive transformations deal with the updates of both the access rights and the access control rules.

1.5 Authorization Application in Database Systems

A database is a collection of certain relevant data. As more and larger database systems have been increasingly in widespread use, it has also posed serious problems of data security. With the ever increasing growth in the use of database, database security issues are increasingly important in the design and implementation of database systems.

Database security comprises a set of measures, policies and mechanisms to provide confidentiality, integrity and availability for the data objects in database. Ensuring database security means preventing the unauthorized disclosure, modification of data from the database or against attempts to deny service to authorized users. Achieving security of a database means the ability to identify the attack and choose a proper
policy.

Database security has been discussed by many researchers [35, 36, 40, 57, 59, 65, 95]. Some of the work addresses security in federated databases, some for commercial database systems, some for relational database systems and some for object oriented database systems.

In a relational database system, security issues include how to control the propagation and revocation of authorizations and how authorizations are related to authorizations on relations. Several relational database systems use grantflags to control the propagation of authorizations. In these systems, grantflags are specified for each user for a relation or view and access mode. The grantflag flag can have the value of "grant" or "rejection". The grant flag allows a user to grant and revoke the corresponding access mode. In addition, a user with a grant flag for a relation or view and mode can give and rescind that grantflag.

In many relational database systems, a user may be authorized for a view but not for the related relations. In such a system, granting authorization for a view but not for the related relation is a way to restrict authorization to a subset of the data contained in the relation.

Object oriented database systems have become very popular since they have the ability to support complex data objects and advanced applications such as computer-aided design, computer engineering and scientific applications. The security issues in object oriented database are increasingly being investigated in recent times and several models for authorization in object oriented database have been proposed. A fundamental concept in object oriented systems is inheritance. So a security model needs to capture such a property.

Questions such as the following arise: if a user is authorized to access a class, should this user be authorized to access all of its instances or all of its subclasses? If authorization for a class implies authorizations for all of its instances or all of its subclasses, then we can get into such a situation whereby some objects in the lower part of the object hierarchy can inherit a large number of authorizations and sometimes conflicts occur. We need to have some mechanism to restrict these undesired inherited authorizations as well as to resolve conflicts. One way to do this is to use

THESIS
negative authorizations to override these inherited positive authorizations. Another way is to explicitly define some authorizations to override these undesired inherited authorizations.

The general considerations for inheritance in an object oriented database can be summarized as follows:

1. A user who has access to a class is allowed to have similar types of access in the corresponding subclasses to the attributes inherited from that class.
2. Access to a complete class implies access to the attributes defined in that class as well as to attributes inherited from a higher class.
3. An attribute defined for a subclass is not accessible by accessing any of its superclass.

Some other models take into consideration not only the aspect of inheritance and composite objects but also the derivation of new authorizations on some objects from the existing authorizations. The authorizations specified by users are explicit, while the authorizations derived from the system are implicit. From the authorization rules specified by users, new authorizations are derived.

1.6 About This Thesis

In this thesis, we try to provide a formal method to specify authorization, its transformation and application in object oriented database systems.

Formal methods are required in the specification and development of secure systems which need high degree of assurance. A formal method typically introduces formal specification techniques which have a mathematical or logical foundation. A formal method provides a vocabulary, a syntax, a semantics and a calculus. The provision of a precise syntax and semantics separates these methods from other informal techniques, such as specification using pseudo-code, or development using a structured method.

A vocabulary is a way of recording something: a collection of signs and symbols. Sometimes, formal languages tend to use some exotic notations to reflect their basis
in mathematical logic such as $\Sigma, \Omega$ or $\Rightarrow$. A syntax, or grammar, provides the rules according to which signs and symbols are grouped together to form legal statements in the language. Semantics explains the meaning of the statements in the language and their claims about the world. Without semantics, a statement is just an arrangement of signs or symbols.

A calculus of a formal method facilitates reasoning about objects defined in the formal language, and to provide some guarantee with mathematical certainty that one specification is correctly derived from another specification. There is no such calculus in an informal method [47].

Any technique used to specify system security should provide a language with sufficient expressiveness to support the specification of complex concepts and procedures but it should also accompany this expressiveness with mathematical precision and conciseness. Most formal methods are based on relatively simple mathematics such as set theory, or some logic such as predicate calculus. Formal methods have the properties of expressiveness, precision and conciseness which are desirable of any specification technique. They also provide a means of communication which is unambiguous, concise and exact.

In this thesis, we will use a formal method which is based on propositional logic and first order logic to specify authorization policies, their transformations and applications.

In authorization specification, we propose a logic based formal approach to specify authorization policies. The authorization policy is specified using a policy base which comprises a finite set of facts and a finite set of access constraints. The facts represent explicitly the access rights. The access constraints are rules which the authorization policy should satisfy. The constraints together with these access rights can imply other access rights which can be deduced from the policy base.

We use a language to specify the policy base. The language is a sorted first order language with six disjoint sorts for $subject$, $group-subject$, $access-right$, $group-access-right$ and $object$, $group-object$ respectively. We also have some predicates to specify access rights. For instance, $s-holds$ takes arguments as $subject$, $access-right$ or $group-access-right$ and $object$ or $group-object$ respectively to represent that a certain subject has ac-
cess right(s) for certain object(s). *g-holds* takes arguments as *group-subject, access-right or group-access-right* and *object or group-object* respectively to represent that a certain group subject has access right(s) for certain object(s). We have logical connectives and punctuations and predicates ∈ and ⊆.

In authorization transformations, we employ a model-based semantics to perform single step policy base transformation. Under the model-based paradigm, the semantics of a transformation on a policy base is based not on the formulas presented in the policy base, but on the individual model of the policy base. A transformation is applied to each model individually. The performance of a transformation on a policy base is achieved based on the result of the transformation on every model of the policy base under the principle of minimal change. Informally, the minimal change principle says that during a state transformation, the difference between the initial state and the resulting state should be as minimal as possible under the restriction of policy constraints.

Furthermore, we use high level languages to specify the sequence of policy base transformations. The languages not only handle a problem such as: given a policy base and a sequence of transformations, what is the resulting policy base after performing such a sequence of transformations. We also provide mechanism for dealing with default authorizations which are needed to specify some desired authorizations.

Finally, we apply our formal method to object oriented database systems. We propose a logic language that has a clear and declarative semantics to specify the structural features of object oriented databases. Then we extend the language to include authorizations on data objects into database specification. The semantics of the resulting language is defined in such a way that both the inheritance property of authorizations in an object oriented database and authorization rules among different data objects can be formally justified. We discuss reasoning on authorizations of data objects in object oriented database and also discuss the properties of the reasoning.
A Formal Representation for Authorization Policy

Authorization policy plays an important role in a multi-user, information-sharing systems. It needs to reflect the user, system and application's requirements. Also it needs to be flexible so that the system can specify its required access rights and at the same time to be constrained to meet the overall objectives and policies of an organization. In this chapter, we first discuss the general background of authorization specification, review some existing authorization specification models, examine the advantages and drawbacks of these models, then we propose our model: a logic based approach for specifying authorization policy. This model provides an organized approach to specify and evaluate authorization policy and is able to accommodate constraints and inheritance rules as well as policy update into it. In our approach, authorization policy is specified by a policy base which consists of a finite set of facts and a finite set of access constraints. The facts represent explicitly the access rights the subjects hold for the objects. The access constraints, on the other hand, are rules which the system and authorization policy should satisfy. The constraints also imply other facts which can be deduced from the policy base and the inheritance property of the policy base. We use the concept of model to represent the state of the policy base. We give the formal definition of model and discuss the possible different states of a policy base. We also summarize the features and properties of our authorization model.
2.1 General Background

Authorization policy specification has long been an important issue in information system security. A variety of different kinds of authorization specifications have been proposed such as the access matrix model [42], typed access matrix model [78], capability list [39, 49] and formal approach, logic language model [48, 91]. Most of these models are typically specified in terms of the abstractions of subjects, objects and access rights.

2.1.1 Access Matrix Model

In an access matrix model, if an authorization policy is specified as "subject $S_1$ holds read and write rights for object $O_1$, and also holds execute and write rights for object $O_2$, subject $S_2$ holds execute and write rights for object $O_2$, and read right for object $O_3$", then using the access matrix model, this policy can be represented as follows, where $R, W, E$ represent the rights of Read, Write and Execute respectively.

<table>
<thead>
<tr>
<th></th>
<th>O1</th>
<th>O2</th>
<th>O3</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>R, W</td>
<td>E, W</td>
<td></td>
</tr>
<tr>
<td>S2</td>
<td></td>
<td>E, W</td>
<td>R</td>
</tr>
</tbody>
</table>

Figure 2.1: An access matrix.

This model has a simple form and clear presentation and also has broad expressive power. But it has a weak safety property. That is, the problem of determining whether or not a given subject can ever acquire access right to a given object. To overcome this weak property, the typed access matrix (TAM) [78] was proposed from the view point of safety determination.

The protection state of a system is defined by the privileges (access rights) possessed by the individual subjects. Once the initial state of a system has been established, the state evolves by the autonomous activity of subjects. A security model provides a framework for specifying the dynamics of the protection state. This is usually done
by stating rules which prescribe the authorization for making incremental changes in the state. Such a collection of rules is called an authorization scheme, or scheme for short. Now the safety determination issue is: for a given initial state and authorization scheme, how to characterize the protection states that are reachable? In other words, the safety problem asks: is there a reachable state in which a particular subject possesses a particular access right for a particular object?

Normally, there is an essential conflict between the expressive power of an access control model and tractability of safety analysis. This is based on the condition that subjects are authorized to create new subjects, hence the system is unbounded. TAM proposed an approach for achieving strong safety property while keeping the expressive power by introducing the notion of strong type into HRU model.

In the TAM system, there are a finite set of access rights $R$, a finite set of object types $T$, and a set of subject types $T_s$, $T_s \subseteq T$. The types and rights are defined when a system is initialized and thereafter $T$ and $R$ remain constant. Therefore, the protection state of a system is defined as a four tuple (SUB, OBJ, t, AM) where SUB, OBJ, t, AM represent the set of subjects, the set of objects, the type function which gives the type of every object and the access matrix respectively. So HRU model is a special case of the TAM model in which there are only two types: subject and object.

The protection state of the system is changed by means of commands. The commands are defined as follows:

```
command \alpha(X_1 : t_1, X_2 : t_2, \cdots, X_k : t_k)
    if r_1 \in [X_{s_1}, X_{a_1}] \land r_2 \in [X_{s_2}, X_{a_2}] \land \cdots \land r_m \in [X_{s_m}, X_{a_m}]
    then op_1; op_2; \cdots; op_n
end
```

where $\alpha$ is the name of the command; $X_1, X_2, \cdots, X_k$ are formal parameters whose types are $t_1, t_2, \cdots, t_k$ respectively; $r_1, r_2, \cdots, r_m$ are access rights; $s_1, s_2, \cdots, s_m$ and $a_1, a_2, \cdots, a_m$ are integers. Each $op_i$ is one of the primitive operations which include: enter or delete an access right, create or destroy a subject and create or destroy an object from the access matrix.

In the above command, "if" statement states the conditions needed to perform the
command; "then" statement states the operations that need to be performed once the "if" conditions are satisfied. If the set of conditions is empty, the command is called a unconditional command, otherwise, it is a conditional command.

Monotonic typed access matrix (MTAM) model is defined to be identical to TAM except that the delete, destroy subject and object primitive operations are omitted.

From expressive power point of view, TAM inherits the expressive power of HRU, which is one of the most general access control model to date. MTAM inherits the expressive power of monotonic HRU [43] which only allows revocation which is itself reversible in general. MTAM is not able to represent some desirable non-monotonic aspects of access control because of the limitations of the monotonic model. But many policies with non-monotonic components can be reduced to monotonic policies for the purpose of safety analysis.

It is shown in [78] that every MTAM scheme can be converted to an equivalent scheme in which all primitive create operations occur only in unconditional commands. The general undecidability of safety in MTAM follows the undecidability results for monotonic HRU [42, 43]. MTAM, however, has decidable safety cases identified on the basis of the types of subjects and objects involved in creation operations. These cases cannot be formulated in HRU model due to the lack of typing. The principal result of [78] is that it proved that safety is decidable for acyclic MTAM schemes. Acyclic MTAM scheme is a scheme in which the creation chain only goes in one direction and the chain is acyclic. Acyclic MTAM schemes are sufficient to express most of the practical monotonic security policies.

2.1.2 Capability-based Model

Capability based systems have been discussed by many researchers [22, 34, 39, 49, 50, 55] for specifying access control to improve the traditional access matrix model. In a matrix model, it is complex to manipulate the matrix directly because the number of objects can be very large. Also a matrix for a real system tends to be very sparse, so many systems do not store the access rights in a matrix form. Rather, they use either an access control list approach or a capability approach. In the access control
list approach, the matrix is viewed by column. Each object is associated with an access control list which stores the subjects and their access rights for the object. The list is checked to see whether to grant an access. In the capability approach, the matrix is viewed by row. Each subject is associated with a capability list which stores its access rights to all related objects. Possessing a capability is the evidence of possessing the corresponding access rights.

In a distributed system environment, both approaches have their merits. An access control list approach implements some centralized control and supports administrative activities better. It can quickly answer questions such as: for a particular object, which subject holds what kind of access rights for it? But it is easier for the capability approach to answer questions such as: for a particular subject, what kind of access rights does it hold for which objects? These questions are commonly asked when something goes wrong with the system. Another important feature of a capability approach is that it supports better both the least privilege principle and protected subsystems. The least privilege principle is that to grant a subject as minimal access rights as possible to protect the system. These limit the damage caused when protection is partially broken.

A better protection system can be generated by combining the two approaches together. It can be better off by taking advantages of both approaches. In [50], an augmented capability architecture was designed and implemented to support lattice security and traceability of access rights. It uses the capability based approach at the lowest level for supporting access control lists which express security policies outside the security kernel. This architecture design is for a hardware supported centralized system.

By focusing on a distributed network environment and merging the two approaches in a different way, an identity based capability system was designed in [39] to improve the weakness of the traditional capability system. One of the weaknesses of the traditional capability system is that it cannot solve the capability propagation problem. It has to be modified to do some checking against the security policies to control capability propagations. In this identity based capability system, access control lists are used to support the capability protection mechanism. It incorporates identities and
access control lists together to solve the capability propagation and revocation problems. This design can enforce security policies, and it also can be used in a centralized environment.

Generally, a capability is a bit string which can propagate in many ways without being detected. That means the server normally cannot monitor and mediate capability propagations. In the identity based capability system, a capability is created in a way that it will fail the validity check when used by processes of users other than the owner. Only the server can propagate a valid capability. So the server can monitor and mediate any capability propagation. The idea of the identity based capability system is that when a capability is to be propagated, the access control server checks the security policy to see if such a propagation is allowed. It is not necessary to check against the security policy again when a capability is later in use. The motivation of this design is that the number of capability propagation is usually much less than the number of their use, so that it seems more reasonable to check the security policy at the propagation time than at the access time. In some situations, checking the security policy may be complex and expensive.

Several classes of capability system designs for managing access control have been discussed in [49]. A classic capability system cannot enforce the military security policies (mandatory) nor the Bell and LaPadula rules (discretionary). [49] discusses system design options that enforce DoD's mandatory and discretionary access control policies. Enforcing a security policy means controlling the access rights users can acquire. In a capability system, a user's access rights are defined by the capabilities he can obtain and the access rights conferred by those capabilities. Propagation of access rights can be limited based on the attributes of the capability, on the context in which the capability is used, or on both. Strategies for controlling access rights can be organized according to what is checked and when it is checked. For a system to enforce the DoD security policy, more checking is required. A set of attributes of capability system designs have been developed. By examining the system requirement against the capability system designs attributes, one can easily determine the limitations of the design with respect to a proposed security policy that the system is supposed to enforce.

THESIS
2.1.3 Logic Model

The above approaches specify authorization using some low level system specific mechanisms, such as protection bits, capability and access control lists. From the viewpoint that authorization is an independent concept which should be separated from implementation mechanism and given a precise semantics, Woo and Lam [91] proposed a formal approach which uses a logic language to specify authorization policy. They use a policy base to encode a set of authorization requirements and a precise semantics based on a formal notation of authorization policy is provided.

In this approach, the concept of authorization is divided into two related issues: representation and evaluation. Representation refers to the specification of authorization requirements, while evaluation refers to the actual determination of the access rights of subjects according to the authorization requirements. An authorization requirement is stated as a rule and a collection of such rules constitutes a policy base. There can be multiple policy bases in a system, each corresponding to a security administrator. A composition of all policy bases define the authorization in the entire system.

The language Woo and Lam used to specify authorization requirements in the policy base is a many sorted first order language with a rule construct. The rule construct is similar to the default construct in default logic [69] and is useful for specifying structural properties of authorization requirement.

Formally, there are three different sets named the set of subjects, set of objects and set of access rights respectively in the system. A rule is a form written as $f : f' / g$, where $f, f'$ and $g$ are formulas and $f$ is called prerequisite, $f'$ is called assumption and $g$ is called consequent of the rule respectively. A rule is ground if $f, f'$ and $g$ do not include ordinary variables. Usually, a rule including ordinary variables can be viewed as a set of all its ground instances. Therefore, in practice, only the ground rules need to be considered.

For convenience, if any component formula is missing from a rule, it is assumed to be true ($\top$). Furthermore, notation $f \Rightarrow g$ was used to represent a rule of the form $f : T / g$. $T \Rightarrow g$ was further abbreviated to $g$. The intuitive semantic meaning of a
rule $f : f'/g$ is as follows: If $f$ is believed and it is also consistent to believe $f'$, then $g$ is believed as well. The Woo-Lam policy base is a finite set of rules. The fact that a subject $S$ is explicitly granted access right $R$ to object $O$ is expressed as the form $(S,R,O) \in P^+$, while the fact that a subject $S$ is explicitly denied an access right $R$ to object $O$ is expressed as $(S,R,O) \in N^+$. Here $P^+$ records the access rights that are explicitly granted, while $N^+$ records the access rights that are explicitly denied.

The semantics of a policy base in Woo and Lam’s approach is based on a concept called extension, which is actually an analogy of the fixed point semantics of Reiter’s default logic [69]. Since the semantics is based on Reiter’s default logic, a policy base may have one or more than one or even no extension at all [91].

Authorization evaluation proceeds as follows: given a request from a subject $S$ for access $R$ to object $O$, $grant(R,S,O)$ is returned if the access request is granted, $deny(R,S,O)$ is returned if the access request is denied, $fail(R,S,O)$ is returned if it cannot determine either to grant or deny the access request.

Jajodia et al [48] proposed a similar logic language for specifying authorizations. Their aim is to provide a model which is able to support different access control policies. Such a model is based on the logic specification of authorizations. The language allows a user to specify the policy according to which access control decision is to be made. The policy is expressed by rules which can enforce derivation of authorizations, conflict resolution, access control and integrity constraint checking.

The basic elements on which this model is based are objects, subjects and actions. Actually, these correspond to the traditional terms of object, subject and access right. A set obj of objects defined in the system is a set on which some action can be executed. The choice of the specific objects and actions depends on the system to which the model is applied. In addition, the objects can be grouped into types. Type is used to specify a set of objects or a group of objects. There are three kinds of subjects: users U, groups G and roles R, where U represents individual user, G represents group of users and R is a named collection of privileges needed to perform a specific action in the system.

The authorization policy is defined as a mapping which maps 4 tuples (o,u,R,a) consisting of an object, a user, a role set and an action to the set {authorized, denied}. THESIS
The authorization language used to specify the policy consists of Constant Symbols, Variable Symbols and Predicate Symbols. The Constant Symbols refer to the sets of constant objects, types, users, groups and authorizations. The Variable Symbols refer to the sets of variable objects, types, users, groups and authorizations. The following is the Predicate Symbols:

1. A ternary predicate symbol \textit{cando}. Its arguments are object, subject and signed authorization term respectively. This predicate represents explicit authorizations.

2. A ternary predicate symbol \textit{dercando}, with same arguments as cando. It represents derived authorizations.

3. A ternary predicate symbol \textit{do}, with same arguments as cando. This predicate represents the authorizations that hold for each subject for each object.

4. A 4-term predicate symbol \textit{grant}. Its four arguments are object, user, role set and signed action respectively. It represents the accesses to be granted or denied to each subject on each object.

5. A 5-term predicate symbol \textit{done}. Its five arguments are object, user, role set, unsigned action and a natural number respectively. It represents the accesses executed by the subjects.

6. A binary predicate symbol \textit{active} whose arguments are user and role term respectively. It captures the concept of active roles for a user.

7. A binary predicate symbol \textit{typeof} which takes arguments as an object and an object type. It captures the group relationship between objects.

8. Two binary predicate symbols \textit{dirin} and \textit{in} that takes two subjects as arguments. They capture the direct and indirect membership relationship between subjects.

The authorization specification is defined as a set of authorization(cando), derivation(dercando), resolution(do), access control(grant) and integrity(error) rules. Every authorization specification in the system is a stratified datalog program. Upon each
access request by user \( u \) with active roles \( R \) to execute action \( a \) on object \( o \) the program is evaluated and access granted if and only if \( \text{grant}(o,u,R,a) \) is true according to the semantics of the specification.

Let us see an example of the static separation of duty specified by this model. For instance, consider the operations of submitting, evaluating and approving the budget. A static separation of duty requires that the same subject cannot be authorized for all the three operations above. This requirement can be expressed as follows:

\[
\text{error}() \leftarrow \text{do} (\text{budget}, s, \text{submitting}) \\
&\text{do} (\text{budget}, s, \text{evaluating}) \\
&\text{do} (\text{budget}, s, \text{approving}).
\]

The access matrix model has a simple form and broad expressive power. The typed access matrix model is established based on access matrix model. It retains all the properties of the access matrix model and the safety determination has been improved. But it is hard to specify system constraints in these models. And the incomplete, implicit information and inheritance cannot be specified.

Capability-based model improved the access matrix model in different ways. By combining access control lists and capabilities, a better system can be achieved to monitor and mediate capability propagations and hence to ensure the integrity of the system. In the same way as access matrix models, capability-based model cannot specify system constraints either. And these models are specified using low level specific mechanisms.

The logic models mentioned above specify authorizations in a higher level and the authorization specifications are separated from the implementation mechanisms. Woo and Lam's model uses default logic to represent and evaluate authorizations. Syntax and a precise semantics are provided. The model is powerful enough to specify constraint and inheritance rules. But default logic may have no extension, that means for a given access request, the system may neither authorize nor deny the request.

Jajodia et al's logic model improved Woo and Lam's model in the sense that their model is conclusive - it will either authorize or deny an access request. And it can well specify constraint, inheritance rules and conflict resolution rules. But this model does
not provide a precise semantics. We feel it is not necessary to specify rules in such a detail that original and derived rules are specified by different rules with different predicates.

Both logic models provided an elegant way to represent and evaluate authorizations. Apart from access facts, both models can specify system requirement rules, constraints and inheritance rules. However, the issue of authorization update is missed. Authorization policy has a temporal property. It needs to be updated as user, system and application requirements change.

In the following, we propose a logic model based on propositional logic and first order logic to specify authorization policies. The model is specified by a language $L$ and the constraints, inheritance rules as well as policy update can be accommodated into this model.

\section{A Formal Representation of Authorization Policy}

We now briefly review some basic concepts of propositional logic and first order logic on which our formal authorization policy specification language is based.

\subsection{Preliminary}

Despite its limited expressiveness, propositional logic serves to illustrate many of the concepts of logic just as first order logic [71]. Its syntax is simple. The symbols of propositional logic are the logical constants True and False, propositional symbols such as $P$ and $Q$, the logical connectives $\land$, $\lor$, $\neg$, $\supset$, $\equiv$ and parentheses. All sentences are formed by the following grammar:

- The logical constants True and False are sentences
- A propositional symbol such as $P$ and $Q$ is a sentence
- Wrapping parentheses around a sentence yields a sentence
- A sentence can be formed by combining simpler sentences with logical connectives

\section*{THESIS}
The semantics of propositional logic is also quite straightforward. It is defined by specifying the interpretation of the proposition symbols and constants, and specifying the meanings of the logical connectives.

The interpretation of a propositional symbol can be any arbitrary fact. A sentence containing just a propositional symbol is satisfiable but not valid: it is true just when the fact it refers to is the case. A complex sentence has a meaning derived from the meaning of its parts. Each connective can be thought of as a function. Truth tables define the semantics of sentences such as $True \lor True$. The semantics of a complex sentence is defined by a process of decomposition: first, determine the meaning of its parts, and then combine them using the definition of the connective functions.

A first order language with equality is specified by the following vocabulary of two disjoint sets of symbols:

1. Logical Symbols:
   
   (a) Parentheses of all shapes and sizes.
   
   (b) Logical connectives: $\neg$ and $\land$.
   
   (c) Variables: $x, y, z, \cdots$.
   
   (d) Equality: $=$.

2. Parameters:
   
   (a) Quantifier symbol: $\forall$.
   
   (b) Predicate symbols: For each $n \geq 0$, a set of symbols, called $n$-place or $n$-ary predicate symbols.
   
   (c) Function symbols: For each $n \geq 0$, a set of symbols, called $n$-place or $n$-ary function symbols. 0-ary function symbols are called constant symbols.

A first order term is either an individual constant or an individual variable. The well formed formulas (wff) are defined as follows:

1. If $s_1, \cdots, s_n$ are first order terms and $p$ is a $n$-ary predicate symbol, then $p(s_1, \cdots, s_n)$ is a wff.
2. If $\phi_1$ and $\phi_2$ are wff then $\neg \phi_1$ and $\phi_1 \land \phi_2$ are also wff;

3. If $\phi$ is a wff and $x$ is an individual variable, then $\forall x \phi$ is also a wff.

The symbols $\lor$ (or), $\supset$ (implies), $\equiv$ (equivalent) and $\exists$ (exists) can be defined in terms of $\neg$, $\land$ and $\forall$. Thus $\phi_1 \lor \phi_2$ is defined as $\neg (\neg \phi_1 \land \neg \phi_2)$, $\phi_1 \supset \phi_2$ is defined as $\neg \phi_1 \lor \phi_2$, $\phi_1 \equiv \phi_2$ is defined as $(\phi_1 \supset \phi_2) \land (\phi_2 \supset \phi_1)$, $\exists x \phi$ is defined as $\neg (\forall x) \neg \phi$. A sentence is a formula with no free variables. The language defined over the above vocabulary is defined to be the set of all well formed formulas constructible using this vocabulary.

We begin the semantics with the definition of a structure for a first order language. A structure sometimes is called an interpretation. It indicates the domain or universe of quantification [70].

Formally, a structure $S$ for a given first order language is a function whose domain is the set of parameters of the language and is defined by:

1. $\forall^s$ is a nonempty set, called the universe or the domain of the structure $S$. The universe is written as $|S|$.

2. For each n-ary predicate symbol $P$ of the first order language, $P^x \subseteq |S|^n$. These n-tuples of the universe are understood to be all, and only, those tuples on which $P$ is true in the structure. This is called the extension or model of $P$ in the structure $S$.

3. For each n-ary function symbol $f$ of the first order language, $f^x$ is a n-ary function on $|S|$. That is, $f^x : |S|^n \rightarrow |S|$. In particular, when n=0, $f$ is a constant symbol, $f^x$ is simply some element of the universe.

When a sentence $\sigma$ is true in a structure $S$, it is denoted by $\models \sigma$ The following are some useful definitions.

1. **Satisfiability:** A structure $S$ satisfies a sentence $\sigma$, or $\sigma$ is true in $S$, iff $\models \sigma$. a sentence is satisfiable iff there is a structure that satisfies it.

2. **Models:** A structure $S$ is a model of a sentence $\sigma$ iff $S$ satisfies $\sigma$. $S$ is a model of a set of sentences iff it is a model of each sentence in the set.
3. **Logical entailment**: Suppose $\Gamma$ is a set of sentences and $\sigma$ is a sentence. $\Gamma \models \sigma$ iff every model of $\Gamma$ is a model of $\sigma$.

In general, a sorted first order language has variables and terms of different sorts. Semantically, the universe is partitioned into disjoint sub-universes, one such sub-universe for each sort. A variable will range over its own sub-universe, and a term will denote an element in its corresponding sub-universe. A predicate symbol will be syntactically restricted to take arguments of certain prespecified sorts, as will functions. Moreover, functions will be required to deliver values of a prespecified sort.

### 2.2.2 Language $L$ for Authorization Policy

We use a language $L$ to specify authorization policy in our model. $L$ is a sorted first order language with six disjoint sorts for subject, group-subject, access-right, group-access-right and object, group-object respectively. Language $L$ has the following vocabulary:

1. Sort *subject*: with subject constants $S, S_1, S_2, \cdots$, and subject variables $s, s_1, s_2, \cdots$.

2. Sort *group-subject*: with group subject constants $G, G_1, G_2, \cdots$, and group subject variables $g, g_1, g_2, \cdots$.

3. Sort *access-right*: with access right constants $A, A_1, A_2, \cdots$, and access right variables $a, a_1, a_2, \cdots$.

4. Sort *group-access-right*: with group access right constants $GA, GA_1, GA_2, \cdots$, and group access right variables $ga, ga_1, ga_2, \cdots$.

5. Sort *object*: with object constants $O, O_1, O_2, \cdots$, and object variables $o, o_1, o_2, \cdots$.

6. Sort *group-object*: with group object constants $GO, GO_1, GO_2, \cdots$, and group object variables $go, go_1, go_2, \cdots$.

7. A ternary predicate symbol *s-holds* which takes arguments as subject, access-right or group-access-right and object or group-object respectively.

8. A ternary predicate symbol *g-holds* which takes arguments as group-subject, access-right or group-access-right and object or group-object respectively.

**THESIS**
9. A binary predicate symbol $\in$ which takes arguments as subject and group-subject or access-right and group-access-right or object and group-object respectively.

10. A binary predicate symbol $\subseteq$ whose both arguments are group-subjects, group-access-rights or group-objects.

11. Logical connectives and punctuations: as usual, including equality.

In language $L$, we define a term is one of the above six sorts constant or variable. An atom is a predicate as defined above. A literal is an atom or the negation of an atom. A formula is a literal, or literals connected by the logical connectives, it is defined the same way as a well formed formula. A ground formula is a formula without variable occurrence in it. For instance, $s$ and $O$ are terms, $s$-holds($s$, $a$, $o$) and $s \in g$ are atoms, $\neg s$-holds($s$, $a$, $o$) is a literal, $s$-holds($S$, $A$, $O$) $\lor$ $s$-holds($S_1$, $A$, $O$) is a formula and it is also a ground formula.

In our language, a subject $S$ has access right $R$ for object $O$ is represented using a ground formula $s$-holds($S$, $R$, $O$). Similarly, a group subject $G$ has access right $R$ for object $O$ is represented using a ground formula $g$-holds($G$, $R$, $O$). A group subject is a set of subjects, where these subjects are related in some way, or have some common characters. Similarly for group object and group access right. The following example shows how the traditional access matrix can be translated into our model by a set of ground formulas.

**Example 1** Consider the access matrix in Figure 2.1, This access matrix can be represented in our formalism as the following set of ground formulas.

$\{s$-holds($S_1$, $Read$, $O_1$), $s$-holds($S_1$, $Write$, $O_1$), $s$-holds($S_1$, $Write$, $O_2$), $s$-holds($S_1$, $Execute$, $O_2$), $s$-holds($S_2$, $Write$, $O_2$), $s$-holds($S_2$, $Execute$, $O_2$), $s$-holds($S_2$, $Read$, $O_3$)$\}$.

It is obvious that a capability based model can be easily translated into our model since the capability approach is just a matrix been viewed by row while the access control list approach is just a matrix been viewed by column.

**THESIS**
Group membership is represented as follows: for example, "a subject s is a member of group subject g" is represented using the formula $s \in G$. We can also represent inclusion relationships between subject groups such as $G_1 \subseteq G_2$ or between access right groups such as $GA_1 \subseteq GA_2$. Furthermore, we can represent constraints among subjects' authorizations. For example, the rule stating that for any subject and object, if the subject is the owner of this object, then the subject should have read and write rights for that object. This constraint can be represented as follows:

$$\forall o.s\text{-}holds(s, Own, o) \supset s\text{-}holds(s, Read, o) \land s\text{-}holds(s, Write, o).$$

If we want to state the rule that the object file can be updated by only one person(subject), the following constraint can be used:

$$\forall s_1.s\text{-}holds(s, Update, File) \supset \neg s\text{-}holds(s_1, Update, File).$$

On the other hand, suppose we have a constraint stating that for any subject s and group subject g, if s is a member of g, then s should have all the access rights that g has. This is the so-called inheritance property of authorizations. This constraint can be captured using the following formula:

$$\forall s g a o.s \in g \land g\text{-}holds(g, a, o) \supset s\text{-}holds(s, a, o).$$

2.2.3 Representation of Policy Base

If the occurrence of a variable in a formula is not bound by a quantifier, the variable is called a free variable. A formula is closed if no variable is free in it. A closed formula is also called a sentence. We will use the term of closed formula, sentence and formula interchangeably. Using the language $L$, we can now give a formal definition of the policy base.

**Definition 1** A policy base $PB$ is a pair $(F, C)$ where $F$ is a finite set of ground literals and $C$ is a finite set of closed first order formulas.

In a policy base $PB = (F, C)$, $F$ represents the agent's knowledge of access rights at the current stage, some access rights not related so far may not in $F$. $C$ represents the policy constraints about the domain of the system and it should be always satisfied.
Example 2 Consider the following information and rules of access requirements from a mental health hospital [92].

1. Al is a patient.

2. Ed is a primary physician and he may read patients’ records.

3. Jane is a primary physician.

4. Patients may not read their records.

5. The consulting physician may neither modify nor copy patients’ records.

6. The primary physician may modify or copy patients’ records.

Let $P, CP, PP$ be group constants that represent group patient, group consulting-physician and group primary-physician respectively. $P$-records represents the patients’ records object. The above information and rules can be represented as follows:

\begin{align*}
    Al & \in P \\
    Ed & \in PP \land s\text{-holds}(Ed, Read, P\text{-records}) \\
    Jane & \in PP \\
    \forall s. s \in P \supset \neg s\text{-holds}(s, Read, P\text{-records}). \\
    \forall s. s \in CP \supset \neg s\text{-holds}(s, Modify, P\text{-records}) \land \neg s\text{-holds}(s, Copy, P\text{-records}). \\
    \forall s. s \in PP \supset s\text{-holds}(s, Modify, P\text{-records}) \lor s\text{-holds}(s, Copy, P\text{-records}).
\end{align*}
In our system, this can be specified as $PB = \{ (2.1), (2.2), (2.3), (2.4), (2.5), (2.6) \}$, where $\{ (2.1), (2.2), (2.3) \}$ is the set of facts and $\{ (2.4), (2.5), (2.6) \}$ is the set of constraints.

A model of a policy base is the assignment of a truth value to every formula of the policy base in such a way that all formulas of the policy base are satisfied [31]. For example, if a policy base $PB = \{ A, B, A \supset C \}$, then the model of this policy base is $\{ A, B, C \}$. If another policy base $PB1 = \{ A, B \vee C \}$, then it has three models $\{ A, B \}, \{ A, C \}$ and $\{ A, B, C \}$. Again, if a policy base is $PB2 = \{ A, B, A \supset \neg B \}$, then this policy base has no model. So, a policy base can have none, one or more than one models.

A Herbrand model [31] is defined on its Herbrand base. The Herbrand base includes all of the possible ground literals which could appear in a policy base during its lifetime. If a ground literal is in the Herbrand base of a policy base but not in the policy base either explicitly or implicitly, then this means that the truth value of this ground literal is indeterminate, that is it could be either positive or negative. Therefore one Herbrand model specifies the ground literal to be positive and includes the literal, whereas the other Herbrand model takes the negative value for the literal and omits it. For convenience, in practice, we only specify those ground literals that are going to be used in the policy base rather than enumerating all the literals in the Herbrand base.

If a formula $\psi$ can be presented in a policy base $PB$ either explicitly or implicitly, we say that the formula $\psi$ is a logical consequence or consequence of policy base $PB$. $\psi$ presents in $PB$ explicitly means that it is explicitly specified in $PB$. On the other hand, $\psi$ presents in $PB$ implicitly means that $\psi$ can be deduced from the facts and constraints of the $PB$.

Formally, we give the following definition.

**Definition 2** A model of a policy base $PB = (F, C)$ is defined to be a Herbrand model of $F \cup C$. $PB$ is said to be consistent if there exists some model of $PB$. The set of all models of $PB$ is denoted as $Models(PB)$. A formula $\psi$ is a consequence of $PB$, denoted as $PB \models \psi$, if $F \cup C \models \psi$. In this case, we also say that $\psi$ is satisfied or is true in $PB$.

**Example 3** Consider a policy base $PB = (F, C)$, where

THESIS
§2.2. A Formal Representation of Authorization Policy

\[ F = \{S_1 \in G, S_2 \in G, g-holds(G, \text{Read}, O)\}, \text{ and} \]
\[ C = \{\forall s gao. s \in g \land g-holds(g, a, o) \supset s-holds(s, a, o)\}. \]

This policy base represents current information about access rights of the agent: subjects \( S_1 \) and \( S_2 \) belong to group subject \( G \); \( G \) can read object \( O \); and if a subject belongs to a group subject then this subject inherits all of the access rights the group subject holds. It is not difficult to see that facts \( s-holds(S_1, \text{Read}, O) \) and \( s-holds(S_2, \text{Read}, O) \) are consequences of \( PB \), and \( PB \) has a unique model \( m \) where:

\[ m = \{S_1 \in G, S_2 \in G, g-holds(G, \text{Read}, O), s-holds(S_1, \text{Read}, O), \]
\[ s-holds(S_2, \text{Read}, O)\}. \]

\[ \square \]

**Example 4** Consider a policy base \( PB = (F, C) \), where

\[ F = \{s-holds(S_1, \text{Read}, O), s-holds(S_2, \text{Read}, O)\}, \text{ and} \]
\[ C = \{s-holds(S_1, \text{Read}, O) \supset -s-holds(S_2, \text{Read}, O)\}. \]

Clearly this policy base is not consistent as there does not exist a model for \( PB \). That is, there does not exist such a state in which both the facts and constraints of the policy base are satisfied. Consistency of a policy base reflects the consistency of an authorization. If a policy base has no model, this means that the authorization policy is not properly stated. System error will be encountered if such an authorization is to be enforced. \( \square \)

**Example 5** Let the policy base be \( PB = (F, C) \), where

\[ F = \{s-holds(S, \text{Read}, O_1)\}, \]
\[ C = \{s-holds(S, \text{Read}, O_1) \supset s-holds(S, \text{Read}, O_2) \vee s-holds(S, \text{Read}, O_3)\}. \]

\( PB \) has three models \( m_1, m_2, m_3 \) as follows:

\[ m_1 = \{s-holds(S, \text{Read}, O_1), s-holds(S, \text{Read}, O_2)\}, \]
\[ m_2 = \{s-holds(S, \text{Read}, O_1), s-holds(S, \text{Read}, O_3)\}, \text{ and} \]
\[ m_3 = \{s-holds(S, \text{Read}, O_1), s-holds(S, \text{Read}, O_2), s-holds(S, \text{Read}, O_3)\}. \]

\( \text{THESIS} \)
Intuitively, a policy base represents the agent’s information about authorizations of the system and this information can be incomplete in the sense that some fact(s) may neither be explicitly represented in the policy base nor be consequence(s) of the policy base. On the other hand, a model represents a possible state of the system, and the set of all models of a policy base represents all possible current states of the policy base and these states are complete. Therefore, we refer to a model in Models(PB) as a possible state of PB. Here are some of the important features of our policy base:

1. Representing implicit information about access rights. In the above examples, the access rights represented by the facts are explicit. The access rights deduced from the facts and constraints are implicit. As in Example 3, facts $S_1 \in G, S_2 \in G$ and $g\text{-}holds(G, \text{Read}, O)$ are explicit and facts $s\text{-}holds(S_1, \text{Read}, O)$ and $s\text{-}holds(S_2, \text{Read}, O)$ are implicit.

2. Representing inheritance of access rights. In Example 3, the access rights $s\text{-}holds(S_1, \text{Read}, O)$ and $s\text{-}holds(S_2, \text{Read}, O)$ are inherited since both $S_1$ and $S_2$ are members of the group $G$, and from the constraint in $PB$, they inherit all the access rights that group $G$ has.

3. Representing incomplete information about access rights. The ground formulas which are neither explicitly nor implicitly specified in our policy base can be either true or false. That is, if $s\text{-}holds(S, R, O)$ is not present and it cannot be deduced from the policy base, $S$ may or may not hold the access right $R$ for $O$. It is important to be able to allow the incompleteness of a policy base because the agent may not have complete information about the system’s authorizations at the beginning; some access rights may arise later when the policy base is updated.

2.2.4 Evaluation of Authorizations

Evaluation is the other aspect of authorization apart from representation. Once an authorization is properly specified and applied to a system, its task is to make the
decision as which access is to be granted and which is to be denied according to authorization
specification. This is the so called evaluation of authorization. That is, given a policy base
specified in our model and an access request, how to decide either to grant or deny this access request?

Obviously, if the access request $\alpha$ is just one of the facts specified in the policy base,
then $\alpha$ is to be granted. If $\alpha$ is not one of the facts explicitly specified in the policy base,
does it mean that the request has to be denied?

Since the constraints are allowed in the policy base, it is possible that some access
facts are implicitly presented in the policy base. By applying inference rules on these
existing facts and constraints, more facts can be deduced from the policy base. If $\alpha$
is not one of the facts which are explicitly presented in the policy base but one of the
facts which are deduced. The access request $\alpha$ will still be granted.

We give the formal definition of the evaluation rule as follows:

**Definition 3** Let $PB$ be the policy base and $\alpha$ an access request. $\alpha$ is to be granted iff $PB \models \alpha$,
$\alpha$ is to be denied iff $PB \models -\alpha$, otherwise, it is uncertain.

Hence, the procedure to decide either to grant or deny an access request is: first,
compute every model of the policy base, then check the access request against each
model to see if it is true in these models. If the access request is true in every model, it
is granted. If it is false in every model, then it is denied. Otherwise, it is uncertain.

**Example 6** A policy base $PB = (F,C)$, where

$$F = \{ s-holds(S_1, \text{Read}, O), s-holds(S_2, \text{Read}, O) \},$$
$$C = \{ s-holds(S_1, \text{Read}, O) \supset s-holds(S_3, \text{Read}, O),$$
$$\quad s-holds(S_2, \text{Read}, O) \supset s-holds(S_4, \text{Read}, O) \}.$$ 

According to definition 2, $PB$ has a unique model $m$ as follows:

$$m = \{ s-holds(S_1, \text{Read}, O), s-holds(S_2, \text{Read}, O), s-holds(S_3, \text{Read}, O),$$
$$\quad s-holds(S_4, \text{Read}, O) \}.$$ 

$m$ represents the current state of the policy base, it contains all the authorized access
rights of the policy base. If there is an access request $s-holds(S_3, \text{Read}, O)$, it will be
granted since \( PB \models s\_holds(S_3, \text{Read}, O) \), so \( s\_holds(S_3, \text{Read}, O) \) is true in \( m \). If another access request is \( s\_holds(S_3, \text{Read}, O_1) \), it will be denied since \( s\_holds(S_3, \text{Read}, O_1) \) is false in \( m \).

\[ \]

**Example 7** Suppose there is another policy base \( PB = (F, C) \), where

\[
F = \{ s\_holds(S_1, \text{Read}, O) \}, \\
C = \{ s\_holds(S_1, \text{Read}, O) \lor s\_holds(S_2, \text{Read}, O) \}.
\]

According to definition 2, \( PB \) has two models:

\[
m_1 = \{ s\_holds(S_1, \text{Read}, O) \}. \\
m_2 = \{ s\_holds(S_1, \text{Read}, O), s\_holds(S_2, \text{Read}, O) \}.
\]

If an access request is \( s\_holds(S_1, \text{Read}, O) \), it will be granted since \( s\_holds(S_1, \text{Read}, O) \) is true in both \( m_1 \) and \( m_2 \). If an access request is \( s\_holds(S_2, \text{Read}, O) \), it will be uncertain since \( s\_holds(S_2, \text{Read}, O) \) is true in \( m_2 \), but false in \( m_1 \). If another access request is \( s\_holds(S_3, \text{Read}, O) \), it will be denied since it is false in both \( m_1 \) and \( m_2 \).  

In the evaluation of authorization, if the system only gives answers of grant or denial for any access request, it is complete. If the system gives not only grant or denial answers, but sometimes the uncertain answer as well, it is incomplete. In our specification and evaluation system, when \( PB \) is a complete logic theory, that is, for any access right \( \alpha \), either \( \alpha \) or \( \neg \alpha \) is in \( PB \), and there is no disjunction in \( PB \), then \( PB \) has a unique model, it will always give grant or denial answer for any access request. Otherwise, if there is disjunction in \( PB \), or \( PB \) is not a complete logic theory, that is, incomplete information is allowed in \( PB \), then the system is not complete, and it will answer grant, denial or uncertain for an access request.

### 2.3 Summary of The Authorization Model

In this section, we summarize our authorization model and conclude the properties of this model. Generally, our authorization model has the following features:

**THESIS**
1. Separation from authorization implementation:

In most systems, authorization policy is specified using some low level, system specific mechanisms such as protection bits, capabilities and access control lists. That means authorization policy must be specified in terms of the policy enforced by the system. Our authorization model has its own syntax and semantics and is independent from the implementation mechanism. The separation of authorization specification from implementation provides the flexibility to enforce the same policy via different implementation systems or different policies in the same implementation system. In addition, updating an authorization policy does not require any change of the implementation mechanism.

2. Powerful expressiveness:

Our model has powerful expressiveness. It can not only represent the traditional access matrix, capability approach and access control list, but also represent constraint rules. The constraints can be used to specify the system and user’s requirements, certain conditions which the system domain has to comply with. They also can be used as inference rules to infer more implicit access rights and inheritance properties.

3. Inconsistency detection:

In our model, authorization is specified by the policy base. A policy base can have one or more models, or have no model at all. A policy base is consistent if it has one or more models. On the other hand, a policy base is inconsistent if it does not have a model. Consistency of a policy base reflects the consistency of an authorization policy. An inconsistent authorization policy signals errors on policy statement, or errors on security administrators, or due to the composition of authorization requirements. Therefore, the semantics of a language for authorization must be able to detect inconsistencies. It is quite easy in our model to detect the inconsistency of an authorization policy just by checking if the corresponding policy base has a model or not.

4. Nonmonotonic authorization:

The authorization is nonmonotonic. If a set of authorization requirements is
augmented by a new requirement, an access right a subject previously holds for an object may no longer allowed.
In the real world, authorization policy has a temporal property. That is, once it is created, it is unusual for the policy to remain the same during its life span. Instead, it needs to be updated to capture the changing needs of applications, systems and users. This implies that situations can arise where some subjects (users or processes) can gain some access rights for some objects and at the same time can lose some access rights for the same or different objects. Representation, evaluation and analysis of such changes form an important part of the design of authorization policies. These changes are implemented via transformation of authorization policies. In general such transformations can be nonmonotonic in that some users or subjects may lose certain rights. Some work has been done in this area. In [73, 74, 75, 76] Sanhdu et al proposed and discussed the concepts of access rights transformations and the nonmonotonic properties of such transformations. Meadows [62] addressed the issue of dynamic upgrading of policies. In this chapter, we propose our authorization transformation model and address this issue from a logic point of view. The transformation model is based on our authorization specification model discussed in the previous chapter. In this approach, we define the structure of the policy transformation and employ a model-based semantics to perform the transformation. Since constraints are included in our policy base specification, conflicts may arise during transformation. Then we discuss conflict resolution and define the preference ordering to resolve the conflict of authorizations under the principle of minimal change. We also discuss the implementation issues of the transformation model and analyze the complexities of the
3.1 Background

In this section, we review some of the earlier works in the area of authorization transformations.

Sanhdu introduced the concept of access rights transformation in [73], then followed by a series of papers [74, 75, 76] on this topic. The initial concept of transformation is used to unify a variety of access control mechanisms which were proposed independently of each other to deal with various integrity issues. Their common foundation is abstracted in a model called transform. The transformation of access rights is considered as taking place in two different ways:

1. **Self transformation or internal transformation** allows a subject who possesses certain rights for an object to obtain additional rights.

2. **Grant transformation or external transformation** occurs in the granting of access rights by one subject to another. The general idea is that possessing of a right for an object by a subject allows that subject to give some other right for that object to another subject.

The transformation model is called transform. In this model, the sets $TS$ and $TO$ for subject types and object types respectively are defined. Each subject is an instance of some subject type and each object is an instance of some object type. The subjects are required to create objects. The access rights obtained as a result of creation need to be specified. In transform, the creation of objects is done by can-create function:

$$cc : TS \rightarrow 2^{TO}$$

The interpretation is that subjects of type $u$ are authorized to create objects of types in $cc(u)$.

The effect of creation is defined by create-rule of the following form, where $R$ is the set of rights:

$$cr : TS \times TO \rightarrow 2^{R}$$

THESIS
The interpretation is that when subject $U$ of type $u$ creates an object $O$ of type $o$, the creator $U$ obtains the rights $cr(u,o)$ for $O$.

The internal transformation function is defined as follows:

\[
itrans : TS \times TO \times R \rightarrow 2^R
\]

The interpretation of $itrans(u,o,x) = x_1, \cdots, x_n$ is that a subject of type $u$ who has the $x$ right for an object of type $o$ can obtain the $x_1, \cdots, x_n$ rights for that object by internal transformation.

Grant transformation function is defined as follows:

\[
grant : TS \times TS \times TO \times R \rightarrow 2^R
\]

The interpretation of $grant(u,v,o,x) = x_1, \cdots, x_n$ is that a subject of type $u$ who has the $x$ right for an object of type $o$ can grant one or more of the $x_1, \cdots, x_n$ rights for that object to a subject of type $v$.

Formally, $transform$ is defined as follows:

A policy for transformation of access rights is stated in $transform$ by specifying the following finite set of components.

1. Disjoint sets of subject types $TS$ and object type $TO$.
2. A set of access rights $R$. 
3. A can-create function $cc : TS \rightarrow 2^{TO}$. 
4. Create-rules $cr : TS \times TO \rightarrow 2^R$. 
5. An internal transformation function $itrans : TS \times TO \times R \rightarrow 2^R$. 
6. A grant transformation function $grant : TS \times TS \times TO \times R \rightarrow 2^R$. 

The relation of the model $transform$ to some existing access control models is also considered. It is shown that the model $transform$ can unify a variety of access control mechanisms to certain extent, and also showed that the model has powerful expressiveness.

Meadows [62] discussed security policy transformation from a different view point. Here transformation is regarded as upgrading. The paper addresses the question of

THESIS
dynamic upgrading security policy in database system. This paper examines some dynamic upgrading policies and the problems that have arisen accordingly.

The questions concerning dynamic upgrading of the security policy of a database system include:

1. When is dynamic upgrading necessary?
2. How to determine a dynamic upgrading policy is secure?
3. How to define restricted dynamic upgrading policies that are more secure than the unrestricted ones?
4. How does a dynamic upgrading policy affect the complexity of the safety problem?

These models have powerful expressiveness, and can be used to unify some existing access control models to a certain extent. But the model is specified in a low level language and an important question is overlooked, that is, conflict resolution. When an authorization policy is being transformed or upgraded, conflict is unavoidable. This is a problem that needs to be addressed when discussing authorization policy transformations.

In the following, we propose our authorization transformation model which is specified using a high level logic language [8, 12].

3.2 Policy Base Transformations

In our transformation model, we assume that there is an administrative officer administering the assignment and the transformation of access rights for each subject. We do not consider separately whether the access rights are assigned originally or granted later by another subject. All the access rights are treated equally. We do not use the concept of creation. All the possible subjects, objects and access rights are created with the creation of the policy base domain. If a certain access right is not specified in the policy base, that means such an access right is not possessed by any subject at this stage. Similarly, if certain subjects or objects are not specified in the
policy base, this simply means that such a subject does not hold any access rights for any objects or there is no subject currently hold any access rights for such an object. After performing some transformation, certain subjects, objects or access rights which previously were not specified may appear in the policy base.

3.2.1 Definition of Transformation

Transformations change the state of the policy base. Relatively, the individual access rights are far more likely to be updated than the constraints. Hence, in our transformation, we only consider the change of the individual access rights at this stage, that is, changes to the set of access facts. Therefore, for a policy base $PB = (F,C)$, we view $F$ as a set of changeable literals while $C$ is a set of non-changeable formulas. Therefore during transformation, $C$ always remains unchanged. We consider two basic types of transformations that can be performed on the policy base: addition of a new access right to the current policy base and deletion of a current access right from the policy base. Another type of transformation update or modification can be represented using the two basic types of transformations. For instance, the effect of updating $s$-holds$(S, Write, O)$ to $s$-holds$(S_1, Write, O)$ can be viewed as the equivalent to deleting $s$-holds$(S, Write, O)$ from and adding $s$-holds$(S_1, Write, O)$ to the policy base. Furthermore, in our system, deleting an access right from the current policy base is represented by the addition of the negation of such an access right to the policy base. Figure 3.1 shows the basic outline of a transformation on a policy base.

![Figure 3.1: A transformation tran on PB.](image)

**Definition 4** A transformation description $tran^1$ is a structure of the form

---

$^1$We will alternatively call $tran$ a transformation or transformation description whenever there is no confusion in the context.

THESIS
\[ \text{Pre}(\text{tran})|\text{Post}(\text{tran})], \text{ where} \\
\text{Pre}(\text{tran}) = \{h_1, \cdots, h_m\}, \\
\text{Post}(\text{tran}) = \{l_1, \cdots, l_n\}, \text{ and} \\
h_i, l_j (1 \leq i \leq m, 1 \leq j \leq n) \text{ are ground literals of } L.
\]

Intuitively, \text{Pre}(\text{tran}) represents the precondition of \text{tran} in which every ground literal must be satisfied in the current policy base before \text{tran} is performed, while \text{Post}(\text{tran}) represents the postcondition of \text{tran} in which every ground literal must be satisfied in the new policy base after \text{tran} is performed. For instance, let \( PB = (F, C) \) be a policy base and \( \text{tran} = [\{s\text{-holds}(S, \text{Read}, O)\}|\neg s\text{-holds}(S, \text{Read}, O)] \) be a transformation description. The intuitive meaning of performing \( \text{tran} \) on \( PB \) is that if \( PB \models s\text{-holds}(S, \text{Read}, O) \) then after performing \( \text{tran} \), in the resulting policy base \( PB' \), the condition \( PB' \models \neg s\text{-holds}(S, \text{Read}, O) \) should hold. If \( \text{Pre}(\text{tran}) \) is an empty set, then this means that there is no precondition for \( \text{tran} \) to be executed (i.e., transformation can always be executed). We say a transformation \( \text{tran} \) is executable on a policy base \( PB \) if for every ground literal \( h \) in \( \text{Pre}(\text{tran}) \), \( PB \models h \). We also denote \( PB \models \text{Pre}(\text{tran}) \) (or \( PB \models \text{Post}(\text{tran}) \)) if \( PB \models h \) for each \( h \) in \( \text{Pre}(\text{tran}) \) (or \( PB \models l \) for each \( l \) in \( \text{Post}(\text{tran}) \)).

Now we are ready to describe the transformation procedure formally. Under the model-based paradigm, the semantics of a transformation on a policy base is based not on the formulas presented in the policy base, but on the individual model of the policy base. A transformation is applied to each model individually. That is, the performance of a transformation \( \text{tran} \) on the policy base \( PB \) is achieved based on the result of the transformation \( \text{tran} \) on every possible state of \( PB \) under the principle of minimal change. Informally, the minimal change principle says that during a state transformation, the difference between the initial state and the resulting state should be as minimal as possible under the restriction of policy constraints.

Let \( PB = (F, C) \) be a policy base and \( m_1, m_2 \in \text{Models}(PB) \). \( \text{Diff}(m_1, m_2) \) denotes the set of ground atoms such that any ground atom only occurs in one of \( m_1 \) and \( m_2 \). For instance, consider example 5 presented in section 2.2,

\[
\text{Diff}(m_1, m_2) = \{s\text{-holds}(S, \text{Read}, O_2), s\text{-holds}(S, \text{Read}, O_3)\}, \\
\text{Diff}(m_1, m_3) = \{s\text{-holds}(S, \text{Read}, O_3)\}, \text{ and}
\]

\text{THESIS}


\[ \text{Diff}(m_2, m_3) = \{ s-holds(S, \text{Read}, O_2) \}. \]

**Definition 5** Let \( PB = (F, C) \) be a policy base, \( tran \) a transformation description that is executable on \( PB \) (i.e. \( PB \models \text{Pre}(\text{tran}) \)), and \( m \) a possible state of \( PB \). A Herbrand interpretation \( m' \) of \( L \) is called a possible resulting state after performing transformation \( tran \) on \( m \) if and only if \( m' \) satisfies the following conditions:

1. \( m' \models C \) and \( m' \models l \) for every ground literal \( l \) in \( \text{Post}(\text{tran}) \).

(In fact, \( C \) is a set of formulas. Notation \( m' \models C \) means that every formula in \( C \) is satisfied in \( m' \).)

2. There does not exist other Herbrand interpretation \( m'' \) of \( L \) such that \( m'' \) satisfies Condition 1 and \( \text{Diff}(m, m'') \subset \text{Diff}(m, m') \).

We denote all such possible resulting states as \( \text{Res}(m, \text{tran}) \). A policy base \( PB' = (F', C) \) is called the resulting policy base after performing transformation \( tran \) on \( PB \) if and only if

\[ \text{Models}(F' \cup C) = \bigcup_{m \in \text{Models}(PB)} \text{Res}(m, \text{tran}). \] (3.1)

Let us examine the above definition more closely. In order to perform the transformation on a policy base \( PB \), we need to compute the set of models (i.e. the set of possible states) of \( PB \) first, then compute transformation on every possible model of \( PB \). Condition 1 states that the resulting model \( m' \) should satisfy the constraint(s) and the postcondition of \( tran \), while Condition 2 says that there does not exist another model which satisfies Condition 1 and its difference with the original state is less than that of \( m' \) with the original state. This condition forces the change from \( m \) to \( m' \) to be as minimal as possible. (3.1) shows that the resulting policy base \( PB' \) after performing transformation \( tran \) on \( PB \) is based on \( \text{Res}(m, \text{tran}) \) for every \( m \) in the set of \( \text{Models}(PB) \).

### 3.2.2 Examples

We now give some examples to illustrate how transformations are performed using the model-based approach described above.

**Example 8** A policy base \( PB = (F, C) \), where

\[ \text{THESIS} \]
\[ F = \{ S_1 \in G, S_2 \in G, s-holds(S_1, \text{Write}, O), g-holds(G, \text{Read}, O) \}, \] and
\[ C = \{ \forall s \in G \wedge g-holds(G, \text{Read}, O) \supset s-holds(s, \text{Read}, O) \}, \]

where the constraint \( C \) says that if group \( G \) has the read right for object \( O \), then every member of \( G \) also has the read right for \( O \). Now consider a transformation "if \( S_1 \) is a member of group \( G \), then delete \( S_1 \)'s write right for object \( O \) from \( PB \)". In our system, this transformation can be formally described as follows:

\[ \text{tran} = [\text{Pre(tran)} | \text{Post(tran)}], \]
where
\[ \text{Pre(tran)} = \{ S_1 \in G, s-holds(S_1, \text{Write}, O) \}, \]
and
\[ \text{Post(tran)} = \{ \neg s-holds(S_1, \text{Write}, O) \}. \]

Clearly, \( PB \models \text{Pre(tran)} \). So \( \text{tran} \) is executable on \( PB \). The transformation is first performed on every model of \( PB \). \( PB \) has only one model:

\[ m = \{ S_1 \in G, S_2 \in G, s-holds(S_1, \text{Write}, O), g-holds(G, \text{Read}, O), s-holds(S_1, \text{Read}, O), s-holds(S_2, \text{Read}, O) \}. \]

According to Definition 5, we have \( \text{Res}(m, \text{tran}) = \{ m' \} \), where

\[ m' = \{ S_1 \in G, S_2 \in G, g-holds(G, \text{Read}, O), s-holds(S_1, \text{Read}, O), s-holds(S_2, \text{Read}, O) \}. \]

As a Herbrand model only describes positive literals, the newly added literal \( \neg s-holds(S_1, \text{Write}, O) \) does not appear in \( m' \) but appears in the resulting \( PB' \). From (3.1), we obtain the resulting policy base as follows:

\[ PB' = (F', C), \]
where
\[ F' = \{ S_1 \in G, S_2 \in G, \neg s-holds(S_1, \text{Write}, O), g-holds(G, \text{Read}, O) \}. \]

\[ \text{Example 9} \] Let \( PB \) be the same as the previous example. Now consider the transformation that "if \( S_1 \) is a member of group \( G \), then change \( S_1 \)'s write right for object \( O \) to the execute right". We can specify this transformation as follows:

\[ \text{tran} = [\text{Pre(tran)} | \text{Post(tran)}], \]
where
\[ \text{Pre(tran)} = \{ S_1 \in G, s-holds(S_1, \text{Write}, O) \}, \]
and
\[ \text{Post(tran)} = \{ \neg s-holds(S_1, \text{Write}, O), s-holds(S_1, \text{Execute}, O) \}. \]

\[ \text{THESIS} \]
Clearly, $PB \models \text{Pre(tran)}$. So transformation tran is executable on $PB$. It should be noted that since the term Execute occurs in the postcondition of tran (i.e. s-holds($S_1$, Execute, $O$)), it should also appear in the Herbrand base of our language $L$ used in this example. If a ground literal is in the Herbrand base of a policy base but not in the policy base either explicitly or implicitly, this means that the truth value of this ground literal could be either positive or negative. Therefore our $PB$ should have the following two Herbrand models:

$$m_1 = \{ S_1 \in G, S_2 \in G, s\text{-}holds(S_1, Write, O), g\text{-}holds(G, Read, O), s\text{-}holds(S_1, Read, O), s\text{-}holds(S_2, Read, O) \}, \text{ and}$$

$$m_2 = \{ S_1 \in G, S_2 \in G, s\text{-}holds(S_1, Write, O), g\text{-}holds(G, Read, O), s\text{-}holds(S_1, Read, O), s\text{-}holds(S_2, Read, O), s\text{-}holds(S_1, Execute, O) \}.$$

Using Definition 5, we have the following result:

$$Res(m_1, tran) = \{ m' \}, \text{ and}$$

$$Res(m_2, tran) = \{ m' \}, \text{ where}$$

$$m' = \{ S_1 \in G, S_2 \in G, s\text{-}holds(S_1, Execute, O), s\text{-}holds(S_1, Read, O), s\text{-}holds(S_2, Read, O), g\text{-}holds(G, Read, O) \}.$$

So, the final policy base is

$$PB' = (F', C), \text{ where}$$

$$F' = \{ S_1 \in G, S_2 \in G, \neg s\text{-}holds(S_1, Write, O), s\text{-}holds(S_1, Execute, O), g\text{-}holds(G, Read, O) \}.$$

---

### 3.3 Conflict Resolutions

#### 3.3.1 The Problem

As the policy constraints are explicitly taken into account in the policy base, the transformations can be nonmonotonic in the sense that the addition of new access right(s) in the current policy base may also lead to a loss of some other access right(s). For instance, consider the following example.
Example 10 Let $PB = (F, C)$ be a policy base, where

$$F = \{s\text{-}holds(S_1, \text{Read}, O_1), s\text{-}holds(S_2, \text{Read}, O_2)\},$$ and

$$C = \{\forall ao. s\text{-}holds(S_1, a, o) \supset \neg s\text{-}holds(S_2, a, o)\}.$$

Now suppose we want to add the fact $s\text{-}holds(S_1, \text{Read}, O_2)$ into $PB$, that is, subject $S_1$ obtains a read right for object $O_2$. As an indirect result, $\neg s\text{-}holds(S_2, \text{Read}, O_2)$ is introduced into the policy base by the constraint while adding $s\text{-}holds(S_1, \text{Read}, O_2)$. This conflicts with the fact $s\text{-}holds(S_2, \text{Read}, O_2)$. Hence, it needs to be decided whether $s\text{-}holds(S_2, \text{Read}, O_2)$ or $\neg s\text{-}holds(S_2, \text{Read}, O_2)$ takes precedence. Suppose we insist that the transformation is to be performed, so $\neg s\text{-}holds(S_2, \text{Read}, O_2)$ takes precedence since it is introduced by the transformation. Therefore, it is not difficult to conclude that the resulting policy base $PB'$ is $(F', C)$ where $F' = \{s\text{-}holds(S_1, \text{Read}, O_1), s\text{-}holds(S_1, \text{Read}, O_2), \neg s\text{-}holds(S_2, \text{Read}, O_2)\}$.

After this transformation, subject $S_1$ obtains a read right for object $O_2$ while subject $S_2$ loses its read right for object $O_2$. ■

From the above example we can see that by introducing policy constraints in our policy base, a transformation may cause some indirect changes to the policy base. However, in some situations, these indirect changes may lead to conflicts which need to be resolved. The approach described so far does not provide a resolution for conflicts. Let us consider the following example,

Example 11 Let $PB = (F, C)$ be a policy base, where

$$F = \{S \in G, \neg s\text{-}holds(S, \text{Read}, FILE), \neg g\text{-}holds(G, \text{Read}, FILE)\},$$ and

$$C = \{\forall sgo. s \in g \land g\text{-}holds(g, \text{Read}, o) \supset s\text{-}holds(s, \text{Read}, o)\}.$$

This policy base simply says that subject $S$ belongs to group subject $G$, neither subject $S$ nor group subject $G$ can read object $FILE$. A constraint of the policy base says that if a subject $s$ belongs to a group subject $g$, then $s$ inherits the read right of $g$ for all objects.

Now consider the transformation of adding $g\text{-}holds(G, \text{Read}, FILE)$ to the policy base. This newly added fact directly conflicts with the existing fact $\neg g\text{-}holds(G, \text{Read}, FILE)$.
It needs to be decided either $\neg g\text{-}holds(G,\text{Read},\text{FILE})$ or $g\text{-}holds(G,\text{Read},\text{FILE})$ takes precedence. Suppose we allow the transformation to perform provided the precondition is satisfied, that is, in such a situation, we assume that the newly added fact takes precedence over the existing fact should conflict occurs.

Obviously $PB$ has a unique model $m = \{S \in G\}$. Therefore, according to Definition 5, there are two possible resulting states after performing such a transformation on $m$. These two resulting states conflict with each other. Again, we need some rules to decide the precedence.

$$m_1 = \{ S \in G, g\text{-}holds(G,\text{Read},\text{FILE}), s\text{-}holds(S,\text{Read},\text{FILE}) \}.$$  
$$m_2 = \{ g\text{-}holds(G,\text{Read},\text{FILE}) \}.$$  

Let us examine the two possible resulting states $m_1$ and $m_2$ more closely. In this situation, since our policy constraint is equivalent to $g\text{-}holds(G,\text{Read},\text{FILE}) \supset S \notin G \vee s\text{-}holds(S,\text{Read},\text{FILE})$, both $m_1$ and $m_2$ represent the minimal changes from $m$ with respect to this particular transformation. $m_1$ is obtained if the predicate $s\text{-}holds$ takes precedence over the predicate $\varepsilon$. This seems reasonable because after the addition of $g\text{-}holds(G,\text{Read},\text{FILE})$ into the policy base, $S$ obtains a read right for $\text{FILE}$ because of the inheritance property. On the other hand, $m_2$ is obtained if the predicate $\varepsilon$ takes precedence over the predicate $s\text{-}holds$. We have no reason to say that $m_2$ is not reasonable according to our approach described in section 3.2. To resolve such issues, we need to specify a proper preference ordering on these predicates which reflects the security policies of the organizations. ■

### 3.3.2 An Approach

Several approaches to conflict resolution have been proposed and used. Strong and weak authorizations have been proposed in the Orion authorization model [68]. In this model, Rabitti et al. used the concept of strong and weak authorizations where a strong authorization is the one which cannot be overridden by another authorization, a weak authorization is the one which can be overridden by a strong authorization.

---

$^2S \notin G$ is used for $\neg S \in G$.  

THESIS
In [27], Castano et al used the concept of strong and weak authorizations as well. The authorization base is grouped into two sets: strong authorization base (AB) and weak authorization base (WAB). AB consists of all strong authorizations while WAB consists of all weak authorizations. Authorizations derived from strong authorizations are also strong and Authorizations derived from weak authorizations are also weak. So AB also includes authorizations which are derived from these strong authorizations and WAB includes the authorizations which are derived from these weak authorizations. The basic idea behind this approach is that strong authorizations cannot be overridden, while weak authorizations can be overridden by strong or other weak authorizations, according to specified rules.

There is a function defined on the strong authorization base AB to determine whether an authorization is explicitly contained in AB or is derived from authorizations contained in AB. The following two properties are satisfied on the strong authorization base.

- Consistency of the AB property

  This property asserts that no two authorizations can exist simultaneously in AB such that one implies an authorization and the other implies the negation of that authorization.

- Nonredundancy of the AB property

  This property avoids the insertion of existing authorizations or of authorizations which are already implied in AB.

As for the strong authorization base, there is a function defined on the weak authorization base WAB to determine whether an authorization is explicitly contained in WAB or is derived from authorizations contained in WAB. For the WAB, the following properties hold.

- Completeness of the WAB property

  This means that for each authorization, either itself or its negation must either exist or be able to be derived.
• Consistency of the WAB property

This property states that an authorization and its negation cannot both exist or be derivable.

• Coexistence of the WAB and the AB property

This property should be always satisfied. In particular, if the insertion of a new weak authorization would not satisfy the consistency property, the authorization is not inserted. By contrast, if the insertion of a new strong authorization would not satisfy the consistency property, the strong authorization is inserted and all the weak authorization which cause non-satisfaction of the property are eliminated.

Lunt [56, 57] discussed various means of reconciling conflict authorizations. One of these is the most-specific rule approach where authorization policies applied in object oriented database systems. The most-specific rule requires that if an individual subject is specifically granted or denied authorization for an object, this takes precedence over any other authorizations for the object that are granted or denied to groups to which the subject belongs. With the most-specific rule, negative authorizations are simply a convenience in forming the access control lists. They can be used as follows: If user A wants to make object O available to everyone in group G except user B, then instead of enumerating everyone except B in the access control list, A could grant authorization to G and specifically deny authorization to B.

Another approach is denials take precedence which is also discussed in relation to object oriented database systems. With this approach, a subject or group’s denial of authorization for an object takes precedence over any authorizations that the subject or group may have been granted for the object. For example, if a user A explicitly denies user C authorization for object O, and a user B later grants a positive authorization for O to C, C will not become authorized. Thus, denial of authorization is a strong measure that can be taken to ensure that specific users and groups cannot obtain authorization to an object. In an object oriented database system, due to its hierarchical structure and natural inheritance property, the objects at the lower level may inherit more authorizations than desired. Negative(denial) authorizations pro
vide a way to restrict the inherited authorizations so that users can assign the precise authorizations desired for an object.

Both these conflict resolution approaches are two possible policies out of many possible ways of resolving conflicts among conflict authorizations. How such conflicts should be resolved is specific application dependent. Actually these two approaches can be subsumed to strong and weak authorizations by making individual authorizations strong and group authorizations weak in the most-specific rule approach and by making all negative authorizations strong and all positive authorizations weak in the denials take precedence approach.

We will use the concept of weak and strong authorization together with a preference ordering mechanism to resolve conflicts. We define existing fact(s) $\prec$ new fact(s), that is we assign the newly added authorization(s) to be strong and the previously existing authorization(s) to be weak. So when the newly added fact conflicts with an existing fact, the new fact will override the existing fact. This means if the precondition is satisfied, the transformation is always allowed to execute.

Now the problem that needs to be solved is: when conflicting resulting states are produced, which state is to be taken? As described in section 2.2, there are four predicates $\in$, $\subseteq$, $s$-holds and $g$-holds in our language $\mathcal{L}$. We assign $g$-holds a higher precedence than $\in$ and $\subseteq$, $\in$ and $\subseteq$ have a higher precedence than $s$-holds. This is achieved by introducing a preference ordering among the predicates in $\mathcal{L}$. By combining such an ordering mechanism in our model-based semantics, we provide a formal solution to the conflict problem.

Formally, a strict partial ordering $\prec$ (i.e. antireflexive, antisymmetric and transitive) among predicates $\in$, $\subseteq$, $s$-holds and $g$-holds is defined as $s$-holds $\prec \prec \prec g$-holds and $s$-holds $\prec \subseteq \prec g$-holds.

We now have two parallel streams of orderings: one is the ordering of the new and existing authorizations, the other one is the ordering of the predicates. We assign the first stream has higher preference ordering than the second stream. When conflict occurs between these two streams, that is, when a newly added fact conflicts with an existing fact, and the predicate of the new fact has lower preference ordering than the predicate of the existing fact, we define that the new fact overrides the existing fact.
We can now extend our model-based transformation given by Definition 5 to include a conflict resolution mechanism. First we need to introduce some additional notations. Let \( m, m' \) be two Herbrand interpretations of \( L \). \( m[g-holds], m[s-holds], m[\in] \) and \( m[\subseteq] \) denote the set of all interpretations of predicates \( g-holds, s-holds, \in \) and \( \subseteq \) in \( m \) respectively. For example, if a Herbrand interpretation is

\[
m = \{s-holds(S_1, A_1, O_1), s-holds(S_2, A_2, O_2), g-holds(G_1, A_3, O_3), \\
S_1 \in G_1, S_2 \in G_2, G_1 \subseteq G_2\},
\]

then we have

\[
m[g-holds] = \{g-holds(G_1, A_3, O_3)\},
\]
\[
m[s-holds] = \{s-holds(S_1, A_1, O_1), s-holds(S_2, A_2, O_2)\},
\]
\[
m[\in] = \{S_1 \in G_1, S_2 \in G_2\}, \text{ and}
\]
\[
m[\subseteq] = \{G_1 \subseteq G_2\}.
\]

On the other hand, \( Diff_{g-holds}(m, m'), Diff_{s-holds}(m, m'), Diff_{\in}(m, m') \) and \( Diff_{\subseteq}(m, m') \) denote the set of different interpretations on predicates \( g-holds, s-holds, \in \) and \( \subseteq \) in \( m \) and \( m' \) respectively. For instance, if

\[
m' = \{s-holds(S_2, A_2, O_2), S_2 \in G_2, G_1 \subseteq G_2\},
\]

then

\[
Diff_{g-holds}(m, m') = \{g-holds(G_1, A_3, O_3)\},
\]
\[
Diff_{s-holds}(m, m') = \{s-holds(S_1, A_1, O_1)\},
\]
\[
Diff_{\in}(m, m') = \{S_1 \in G_1\}, \text{ and}
\]
\[
Diff_{\subseteq}(m, m') = \{\}.
\]

The following is the formal definition of the extended model-based transformation based on the preference ordering \( \prec \).

**Definition 6** Let \( PB = (F, C) \) be a policy base, \( tran \) a transformation description that is executable on \( PB \) (i.e. \( PB \models Pre(tran) \)), and \( m \) a possible state of \( PB \). A Herbrand interpretation \( m' \) of \( L \) is called a possible resulting state after transformation \( tran \) on \( m \) based on the preference ordering \( \prec \) (called \( \prec \)-transformation \( tran \) on \( m \)), if and only if \( m' \) satisfies the following conditions:

**THESIS**
1. \( m' \models C \) and \( m' \models l \) for every ground literal \( l \) in \( \text{Post}(\text{tran}) \).

2. There does not exist other Herbrand interpretation \( m'' \) of \( L \) such that

   (a) \( m'' \) satisfies Condition 1;

   (b) \( \text{Diff}_{g-\text{holds}}(m,m'') \subseteq \text{Diff}_{g-\text{holds}}(m,m') \); or

   (c) \( m'[g-\text{holds}] = m''[g-\text{holds}] \) and

   \[ \text{Diff}_{\subseteq}(m,m'') \subseteq \text{Diff}_{\subseteq}(m,m') \]

   or

   \[ \text{Diff}_{\subseteq}(m,m'') \subseteq \text{Diff}_{\subseteq}(m,m') \]; or

   (d) \( m'[g-\text{holds}] = m''[g-\text{holds}] \) and

   \( m'[\in] = m''[\in] \) and

   \( m'[^{\subseteq}] = m''[^{\subseteq}] \) and

   \( \text{Diff}_{\subseteq-\text{holds}}(m,m'') \subseteq \text{Diff}_{\subseteq-\text{holds}}(m,m') \).

We denote all such possible resulting states based on the preference ordering \( \prec \) as \( \text{Res}^{\prec}(m,\text{tran}) \). A policy base \( PB' = (F',C) \) is called the resulting policy base by performing \( \prec \)-transformation \( \text{tran} \) on \( PB \) if and only if

\[
\text{Models}(F' \cup C) = \bigcup_{m \in \text{Models}(PB)} \text{Res}^{\prec}(m,\text{tran}).
\]  

(3.2)

Similarly to the previous definition for transformation without considering conflicts, in order to perform the transformation on a policy base \( PB \), we need to compute the set of models (i.e. the set of possible states) of \( PB \) first, then compute transformation on every possible state of \( PB \) based on the defined preference ordering. Condition 1 states that the resulting state \( m' \) should satisfy the constraint(s) and the postcondition of \( \text{tran} \), while overall, Condition 2 forces the change from \( m \) to \( m' \) to be as minimal as possible. The criteria used to measure the change are based on the preference ordering which are defined by the subconditions (a), (b), (c) and (d) of Condition 2.

The following example shows how a transformation is performed using the extended model-based transformation based on the preference ordering \( \prec \).

**Example 12** Let us consider the patient treatment policy in a hospital domain. Suppose \( S \) acts as both a consulting physician (CP) and a primary physician (PP), PP can
read and update patients' record (P-records), CP can read patients' record. The constraint is: "a member of a group has and only has those authorizations that the group has". This can be specified as: \( PB = (F, C) \), where

\[
F = \{ S \in CP, S \in PP, g-holds(CP, Read, P-records), \\
g-holds(PP, Update, P-records), g-holds(PP, Read, P-records) \}, \text{ and}
\]

\[
C = \{ \forall s gao.s \in g \land g-holds(g, a, o) \supset s-holds(s, a, o), \\
\forall s gao.s \in g \land \neg g-holds(g, a, o) \supset \neg s-holds(s, a, o) \}
\]

Now consider a transformation which causes the "group CP cannot update patients' record". That is, the addition of \( \neg g-holds(CP, Update, P-records) \) into the \( PB \) is required. Formally, it can be represented as follows:

\[
tran = [Pre(tran) \mid Post(tran)], \text{ where}
\]

\[
Pre(tran) = \{ \}, \text{ and}
\]

\[
Post(tran) = \{ \neg g-holds(CP, Update, P-records) \}.
\]

Here \( Pre(tran) \) is empty. It means that there is no precondition to perform the transformation. \( PB \) has two models:

\[
m_1 = \{ S \in CP, S \in PP, g-holds(CP, Read, P-records), \\
s-holds(S, Read, P-records), g-holds(PP, Read, P-records), \\
g-holds(PP, Update, P-records), s-holds(S, Update, P-records) \}
\]

\[
m_2 = \{ S \in CP, S \in PP, g-holds(CP, Read, P-records), \\
s-holds(S, Read, P-records), g-holds(PP, Read, P-records), \\
g-holds(PP, Update, P-records), s-holds(S, Update, P-records), \\
g-holds(CP, Update, P-records) \}
\]

In the second constraint, if we substitute \( s \) by \( S \), \( g \) by \( CP \), \( a \) by \( Update \) and \( o \) by \( P-records \) respectively, we get the following equivalent constraint instance:

\[
\neg g-holds(CP, Update, P-records) \supset S \notin CP \lor \neg s-holds(S, Update, P-records)
\]

That is, the addition of \( \neg g-holds(CP, Update, P-records) \) implies that we have to either remove \( S \in CP \) or \( s-holds(S, Update, P-records) \) but not both in accordance with

THESIS
the principle of minimal change. As we defined the ordering \( s \prec e \), we will remove \( \neg s\text{-}holds(S, Update, P\text{-}records) \) (i.e. adding \( \neg s\text{-}holds(S, Update, P\text{-}records) \)) and keep \( S \in CP \).

On the other hand, in the first constraint, if we substitute \( s \) by \( S \), \( g \) by \( PP \), \( a \) by \( Update \) and \( o \) by \( P\text{-}records \) respectively, we get the following equivalent constraint instance:

\[
\neg s\text{-}holds(S, Update, P\text{-}records) \supset S \notin PP \lor \neg g\text{-}holds(PP, Update, P\text{-}records)
\]

Similarly, as we defined \( e \prec g\text{-}holds \), we will keep \( g\text{-}holds(PP, Update, P\text{-}records) \) and remove \( S \in PP \).

Formally, using our conflict resolution and Definition 6, we have

\[
Res^{\prec}(m_1, tran) = \{m'\}, \text{ and}
\]

\[
Res^{\prec}(m_2, tran) = \{m'\}, \text{ where}
\]

\[
m' = \{ S \in CP, g\text{-}holds(CP, Read, P\text{-}records), s\text{-}holds(S, Read, P\text{-}records),
\]

\[
g\text{-}holds(PP, Read, P\text{-}records), g\text{-}holds(PP, Update, P\text{-}records) \}. 
\]

From (3.2), the resulting policy base is as follows:

\[
PB' = (F', C), \text{ where}
\]

\[
F' = \{ S \in CP, S \notin PP, g\text{-}holds(CP, Read, P\text{-}records),
\]

\[
g\text{-}holds(PP, Update, P\text{-}records), g\text{-}holds(PP, Read, P\text{-}records),
\]

\[
\neg g\text{-}holds(PP, Update, P\text{-}records) \}. 
\]

From the new policy base \( PB' \), we have \( PB' \models \neg s\text{-}holds(S, Update, P\text{-}records) \). Hence, the addition of \( \neg g\text{-}holds(CP, Update, P\text{-}records) \) will cause \( S \) to lose its membership of \( PP (S \notin PP) \) as well as its \( Update \) right for \( P\text{-}records \)(i.e. \( \neg s\text{-}holds(S, Update, P\text{-}records) \)).

From the above example, we can see that if a subject belongs to two different groups and it inherits the access rights from both groups, then a transformation of one of the two group's access right may affect the subject's access right, and such an effect may cause the subject to be removed from one of the two groups according to our conflict resolution. On the other hand, it is also possible that the changes of access

THESIS
rights within one group may affect a subject’s access right which is inherited from the other group, which again may cause a loss of its membership in one of these groups. This is illustrated in the next example.

Example 13 A policy base $PB = (F,C)$, where

$$F = \{ S_1 \in G, S \in G, S_2 \in G_1, S \in G_1, \}
\quad \text{g-holds}(G,\text{Read},O), \quad \text{g-holds}(G_1,\text{Execute},O) \},$$
and
$$C = \{ \forall s \in g \wedge \text{g-holds}(g,a,o) \supset \text{s-holds}(s,a,o) \},$$

Consider the transformation “the members of group $G$ cannot have execute right for $O$”. That is, the addition of $\neg \text{s-holds}(S,\text{Execute},O)$ and $\neg \text{s-holds}(S_1,\text{Execute},O)$ to $PB$.

Formally, the transformation can be represented as follows:

$$\text{tran} = [\text{Pre(tran)} \mid \text{Post(tran)}],$$
where
$$\text{Pre(tran)} = \{ \},$$
and
$$\text{Post(tran)} = \{ \neg \text{s-holds}(S,\text{Execute},O), \neg \text{s-holds}(S_1,\text{Execute},O) \}.$$

$PB$ has only one model:

$$m = \{ S_1 \in G, S \in G, S_2 \in G_1, S \in G_1, \text{g-holds}(G,\text{Read},O), \}
\quad \text{s-holds}(S,\text{Read},O), \quad \text{s-holds}(S_1,\text{Read},O), \quad \text{g-holds}(G_1,\text{Execute},O), \\
\quad \text{s-holds}(S,\text{Execute},O), \quad \text{s-holds}(S_2,\text{Execute},O) \}.$$

Since $S$ belongs to both groups $G$ and $G_1$ and using the constraint, it inherits the access rights from both groups. That is, $S$ holds $\text{Read}$ and $\text{Execute}$ rights for $O$. The transformation will change the access rights of $S$ and $S_1$. The result of the transformation is that $S$ and $S_1$ cannot have $\text{Execute}$ right for $O$, which conflicts with the access right $S$ inherited from group $G_1$. Since our conflict resolution policy includes that the newly added facts override the previously existing facts, this will force $S$ to lose the $\text{execute}$ right for $O$. But our constraint says that if $S$ is a member of $G_1$, $S$ holds the access right(s) that $G_1$ holds. That is, $S$ holds $\text{Execute}$ right for $O$. This again results in a conflict. As constraints need to be always true, this leads to $S$ being removed from $G_1$'s membership or $\text{g-holds}(G_1,\text{Execute},O)$ being removed from the policy base. Since we defined the preference ordering $\in \prec \text{g-holds}$, $S \in G_1$ will be removed from the policy base.

THESIS
Formally from Definition 6, we have $Res(m, tran) = \{m'\}$, where

$$m' = \{S_1 \in G, S \in G, S_2 \in G_1, g-holds(G, Read, O), g-holds(G_1, Execute, O),$$

$$s-holds(S, Read, O), s-holds(S_1, Read, O), s-holds(S_2, Execute, O)\}.$$ 

Our transformation is based on the model(s) of $PB$. For a subject, its access rights inherited from all of the groups which it belongs to are within the same model(s). When performing transformations, the consistency of the model(s) will guarantee the consistency of every related group. So we do not need to check the individual group for maintaining the consistency of $PB$.

From (3.2), the resulting policy base is as follows:

$$PB' = (F', C), \text{ where}$$

$$F' = \{S_1 \in G, S \in G, S_2 \in G_1, S \notin G_1, g-holds(G_1, Execute, O),$$

$$g-holds(G, Read, O), \neg s-holds(S, Execute, O), \neg s-holds(S_1, Execute, O)\}.$$ 

---

### 3.4 Implementation Issues

In this section, we consider the implementation issues of our extended model-based transformation approach. We discuss the basic idea of the implementation procedures and the algorithms employed in these procedures. Our method is similar to that of Winslett’s update theory [28, 90] in the sense of model-based semantics of change, but differs from it due to different ontologies [7].

#### 3.4.1 The Basic Idea

Let $PB = (F, C)$ be a policy base. A transformation $tran$ adds a literal or a set of literals $N = \{l_1, l_2, ..., l_n\}$ to $PB$. From Definition 6, the resulting policy base $PB' = (F', C)$ is based on the set of models of $PB'$:

$$Models(F' \cup C) = \bigcup_{m \in Models(PB)} Res^< (m, tran).$$
According to Definition 6 of $Res^< (m, tran)$, we can rewrite it as the following:

$$Res^< (m, tran) = Min(m, Models(N\cup C), \prec),$$

where $Min(m, Models(N\cup C), \prec)$ is the subset of $Models(N\cup C)$ such that each model $m'$ in it has minimal difference from $m$ with respect to the preference ordering $\prec$ defined in Condition 2 of Definition 6. So, our implementation comprises the following steps:

Step 1: Generate the set of models of $PB : Models(PB)$.

Step 2: Generate the set of models of $N \cup C : Models(N \cup C)$.

Step 3: For each model $m \in Models(PB)$,

- compute $Min(m, Models(N \cup C), \prec)$.
- Then $Models(PB') = \bigcup_{m \in Models(PB)} Min(m, Models(N \cup C), \prec)$.

Step 4: Generate resulting policy base $PB' = (F', C)$ from $Models(PB')$.

The above steps are summarized by Figure 3.2.

Since both step 1 and step 2 involve a procedure of model generation, we only need one model generator in our implementation. To obtain the resulting models, we need

THESIS
to compute $\text{Min}(m, \text{Models}(N \cup C), \prec)$ for each $m \in \text{Models}(PB)$. This will be achieved using an algorithm called resulting models finder algorithm which will be described later. Finally, to get the resulting policy base $PB' = (F', C)$ we need a procedure to generate $F'$ from $\text{Models}(PB')$.

Given the set of models of $PB'$, we may get different $PB'$ by different approach. But they are logically equivalent since they have the same set of models. In the following, we will give an approach of getting $PB'$ from its set of models.

### 3.4.2 The Algorithms

#### 3.4.2.1 The Model Generator

For a policy base $PB = (F, C)$, generally the constraints in $C$ may include universally quantified variables\(^3\). From the implementation consideration, we need to ground each constraint containing variables in $C$ to all of its propositional instances\(^4\). For example, in Example 3, $C$ contains one constraint:

$$\forall s, g, a, o \in g \land g\text{-holds}(g, a, o) \supset s\text{-holds}(s, a, o).$$

During implementation, this constraint needs to be replaced by its two ground instances:

$$S_1 \in G \land g\text{-holds}(G, Read, O) \supset s\text{-holds}(S_1, Read, O), \text{ and}$$

$$S_2 \in G \land g\text{-holds}(G, Read, O) \supset s\text{-holds}(S_2, Read, O).$$

Therefore, in the rest of this section, when we refer to a policy base $PB = (F, C)$, we assume that $C$ only contains constraints without variable occurrence.

From the implementation point of view, we need to define some additional concepts that will be used in our algorithms. For a policy base $PB = (F, C)$, an inconsistency is a set of literals whose conjunction is inconsistent with $C$. A minimal inconsistency is an inconsistency which has no subset that is also an inconsistency.

Let $L$ be the set of all ground literals of the language defined in section 2.3. To get the set of models of $PB = (F, C)$, first we need to find out the set $I$ of minimal

\[^3\]Technically, an existential quantifier in a formula can be eliminated by introducing Skolem function [31].

\[^4\]This technique is often used in the implementation of first order dynamic systems, eg.[90].

THESIS
inconsistencies between \( L \) and \( C \). This can be achieved using an inference engine. Given \( L \) and \( C \), we can use the resolution proof to find out all of the minimum-length proofs which lead to empty clauses, and the required inconsistencies can be directly read off from these proofs.

For instance, let us consider Example 3 and see how one can obtain the set \( I \) of minimal inconsistencies. To simplify the problem, let \( a \) stand for \( S_1 \in G \), \( b \) for \( S_2 \in G \), \( c \) for \( g\text{-}holds(G,Read,O) \), \( d \) for \( s\text{-}holds(S_1,Read,O) \) and \( e \) for \( s\text{-}holds(S_2,Read,O) \). Then this policy base can be viewed as \( PB = (F,C) \) where \( F = \{a,b,c\} \) and \( C = \{a \land c \supset d, b \land c \supset e\} \). Furthermore, \( a \land c \supset d \) is equivalent to \( \neg a \lor \neg c \lor d \) and \( b \land c \supset e \) is equivalent to \( \neg b \lor \neg c \lor e \). Here \( L = \{a, \neg a, b, \neg b, c, \neg c, d, \neg d, e, \neg e\} \). The resolution proof of Figure 3.3 shows the procedures to obtain the set \( I = \{\{a,c,\neg d\}, \{b,c,\neg e\}\} \) of minimal inconsistencies.

Once the set \( I \) of minimal inconsistencies between \( L \) and \( C \) is obtained, a model \( m \) of \( PB \) can be achieved by a maximal subset of \( L \) which contains \( F \) but does not contain any minimal inconsistency.

For the above example, considering the first inconsistency \( \{a,c,\neg d\} \), we get models \( \{a,b,c,d\} \) and \( \{a,b,c,d,e\} \). Taking the second inconsistency \( \{b,c,\neg e\} \) into account, leaves us the only model \( \{a,b,c,d,e\} \). That is, \( \{S_1 \in G, S_2 \in G, g\text{-}holds(G,Read,O_1), s\text{-}holds(S_1,Read,O_1), s\text{-}holds(S_2,Read,O_1)\} \).

---

\[ \neg a \lor \neg c \lor d \quad a \quad \neg b \lor \neg c \lor e \quad b \]

\[ \neg c \lor d \quad c \quad \neg c \lor e \quad c \]

\[ d \quad \neg d \quad e \quad \neg e \]

\[ \square \quad \square \]

**Figure 3.3: Resolution Proofs**

THESIS
The set of models of $N \cup C$ can be generated in a similar manner. The algorithm of model generator for $F \cup C$ is as follows:

Algorithm 1.

\textit{Input:} a finite set $F$ of ground literals, a finite set $C$ of ground formulas and a finite set $L$ of ground literals over $F \cup C$.

\textit{Output:} a finite set of models of $F \cup C$.

1. Use resolution proof to find the set $I$ of minimal inconsistencies between $L$ and $C$.

2. Find the maximal subset of $L$ which contains $F$ but does not contain any inconsistency of $I$.

3. From the set of all such maximal subsets of $L$ to form the set of models of $F \cup C$, i.e., $\text{Models}(F \cup C)$.

In the above algorithm, we achieve step 1 using a theorem prover OTTER [71]. In fact, step 1 can be pre-computed as a separate procedure for finding the set of minimal inconsistencies between $L$ and $C$. This can reduce the overall computation time significantly.

3.4.2.2 Resulting Models and New Policy Base Finders

Given the set of models of $PB$ and the set of models of $N \cup C$, the resulting models finder is to obtain a corresponding $m' \in \text{Models}(N \cup C)$ for every $m \in \text{Models}(PB)$ under the measure of minimal change from $m$ to $m'$. The set of models so obtained forms the set of resulting models. The algorithm is as follows:

Algorithm 2.

\textit{Input:} two finite sets of models: $\text{Models}(PB)$ and $\text{Models}(N \cup C)$.

\textit{Output:} a finite set of resulting models.

1. For every $m \in \text{Models}(PB)$ check every $m' \in \text{Models}(N \cup C)$ to find the $m'$ which is closest to $m$ under the condition 2
of Definition 6.

2. The set of models so obtained forms the set of resulting models.

The new policy base finder is to find $F'$ of $PB'$ from the set of the resulting models. Since the complementary literals are not included in these models, the new $F'$ is the union of the intersection of every resulting model and the complementary literals which are in $L$ but not in any resulting model. The following is the algorithm to find the new policy base.

**Algorithm 3.**

*Input: a finite set of resulting models and a finite set $L$ of literals.*

*Output: a finite set $F'$ of literals.*

1. Check every resulting model to find the set $T$ of ground atoms of $L$ which do not appear in any resulting model.

2. Obtain the set $T'$ of the intersection of every resulting model.

3. The union of $T'$ and the set of complements of every ground atom of $T$ gives $F'$.

For example, suppose the resulting models are $m_1 = \{a, b\}$ and $m_2 = \{b, c\}$. The set of ground literals over this policy base is $L = \{a, \neg a, b, \neg b, c, \neg c, d, \neg d\}$. According to the above algorithm, we get $T = \{d\}$ and $T' = m_1 \cap m_2 = \{b\}$, then $F'$ is formed by $\{b\} \cup \{\neg d\} = \{b, \neg d\}$.

### 3.5 Analysis of Implementation Algorithms

In this section, we summarize the implementation procedures and generally analyze the complexities of the algorithms introduced in these procedures.

Step 1 of Algorithm 1 is to compute the set of minimal inconsistencies between $L$ and $C$; this is achieved by employing a theorem prover OTTER. In fact, Algorithm 1 is implemented by two separate sub-procedures: computing minimal inconsistencies.
and finding models of \( F \cup C \), where the output of the first sub-procedure is the input of the second sub-procedure. In our implementation of the system, we can pre-compute minimal inconsistencies so that it does not affect the major activity of the system and can obviously reduce the overall computation time. Here we will only analyze the complexities of the second sub-procedure of Algorithm 1 (that is, step 2 of Algorithm 1), Algorithm 2 and Algorithm 3. Refer to [60] for a detailed analysis of the theorem prover OTTER.

Now let us consider the complexity of step 2 in Algorithm 1. We will use the notation \(|S|\) to indicate the cardinality of the set \( S \). Suppose that \( I \) is the set of minimal inconsistencies and \( L \) is the set of ground literals over \( F \cup C \). For the set \( L \), there are at most \( 2^{|L|} \) subsets. Since \( L \) includes both positive and negative ground literals, there are at most \( 2^{|L|}/2 \) (that is, \( 2^{|L|-1} \)) consistent subsets of \( L \). Therefore, we need only consider at most \( 2^{|L|-1} \) subsets of \( L \) in order to find out the maximal subsets of \( L \) containing \( F \) but not containing any element of \( I \). First, to find the subsets of \( L \) containing \( F \), we need to do at most \(|F| \cdot 2^{|L|-1}\) model checks. For each element \( i \) of \( I \), if \(|i| > |L|/2\), it must contain at least one pair of literals \( a \) and \( \neg a \), so it will contain a subset which is an inconsistency. Therefore, \(|i| \leq |L|/2\). Second, to get the maximal subsets of \( L \) that contain \( F \) and do not contain any \( I \) in \( I \), requires at most \( N \cdot |L|/2 \cdot |I| \) model checks, where \( N \) is the number of subsets of \( L \) containing \( F \). So \( N \leq |L|/2 \). Therefore, the total computational cost of step 2 is at most \( O(|F| \cdot 2^{|L|-1} + N \cdot |L|/2 \cdot |I|) \), that is, \( O(|F| \cdot 2^{|L|-1} + N^2/4 \cdot |I|) \).

Now consider Algorithm 2. Let \( N_1 = |\text{Models}(PB)| \) and \( N_2 = |\text{Models}(N \cup C)| \). Step 1 of Algorithm 2 is for each \( m \in \text{Models}(PB) \), compute \( \text{Min}(m, \text{Models}(N \cup C), <) \). From Definition 6, for each \( m' \) in \( \text{Models}(N \cup C) \), we need to compute \( \text{Diff}_{g\rightarrow \text{holds}}(m, m') \), \( \text{Diff}_{g\rightarrow \text{holds}}(m, m') \), \( \text{Diff}_{g\rightarrow \text{holds}}(m, m') \), and \( \text{Diff}_{g\rightarrow \text{holds}}(m, m') \) respectively. For each model \( m \), if \(|m| > |L|/2\), it must contain at least one pair of literals \( a \) and \( \neg a \), this contradicts with the model definition. Therefore, \(|m| \leq |L|/2\) and \(|m'| \leq |L|/2\), the complexity of each of the above computations is at most \(|L|^2/4\). The complexity of the above four computations is \(|L|^2\). Therefore, the total complexity of Algorithm 2 should be at most \( O(N_1 \cdot |L|^2 \cdot N_2) \), where \( N_1 \) and \( N_2 \) are cardinalities of \( \text{Models}(PB) \) and \( \text{Models}(N \cup C) \) respectively.
§3.6 Conclusions of The Transformation Model

Finally, let us consider Algorithm 3. From Algorithm 2, we get $Models(F' \cup C)$. Let $N = |Models(F' \cup C)|$. For each $m$ in $Models(F' \cup C)$, $|m| \leq |L|/2$. So the computational cost of both steps 1 and 2 is at most $N \cdot |L|^2/4 + N \cdot |L|^2/4 = N \cdot |L|^2/2$. So the maximum computational cost of Algorithm 3 is $O(N \cdot |L|^2/2)$.

3.6 Conclusions of The Transformation Model

In this chapter, we have developed an authorization transformation model based on our formal authorization specification proposed in the previous chapter. In this transformation model, we formally defined the policy transformation structure in which the pre and post conditions of the transformation are specified. The pre-condition is the condition which the current policy base must satisfy in order to perform the transformation, while the post-condition is the condition which the transformed policy base must satisfy. The transformation procedure employed a model based semantics in which the overall result is based on the transformation of the individual model of the policy base.

The transformation model has a nonmonotonic property. That is, the policy base does not always gain more access rights after performing transformation, instead, it may lose some of its access rights due to the transformation.

Furthermore, to resolve the possible conflicts introduced by the transformation, we provided a conflict resolution by defining weak and strong authorizations together with a preference ordering on the predicates. The transformation is performed under the principal of minimal change. We also provided the implementation algorithms and the analysis of these algorithms.
In Chapter 3, we proposed a model-based approach to specify the transformation of authorizations based on the *principle of minimal change* [28]. Nevertheless, there were some limitations in this approach. Firstly, we could not represent a sequence of transformations. For instance, this approach could not handle a problem like: given a policy base and a sequence of transformations, what is the resulting policy base after performing such a sequence of transformations? Secondly, default authorizations could not be expressed. In certain situations, default policies are needed to specify desired authorizations. In this chapter, we propose a high-level formal languages $L^t$ and then extend $L^t$ to $L^{td}$ to specify authorization transformations in secure computer systems. In particular, both $L^t$ and $L^{td}$ can specify sequence of authorization transformations. Though it has a simple syntax and semantics, we show that $L^t$ is expressive enough to specify some well-known examples of authorization transformations such as separation of duty and Chinese wall security policy [13]. Language $L^{td}$ is an augmentation of $L^t$ which includes *default propositions* into the domain description of authorization policies. However, the semantics of $L^{td}$ is not just a non-trivial extension of the semantics of $L^t$. We show that $L^{td}$ has more powerful expressiveness than $L^t$ in the sense that constraints, causal and inherited authorizations, and general default authorizations can be specified within $L^{td}$. For description purposes, we will refer to a policy base as a policy domain description in this chapter. The two terms are used interchangeably.

To simplify our presentation, we assume the existence of a system security officer administering the authorization transformations. This assumption enables us to concentrate on a single administering agent system and hence avoid the problem of
coordination among multiagents [11, 16].

4.1 Syntax and Semantics of Language $L^S$

4.1.1 Syntax of $L^S$

Language $L^S$ includes the following seven disjoint sorts for subject, group-subject, access-right, group-access-right, object, group-object and transformation together with predicate symbols holds, $\in$, $\subseteq$ and logic connectives $\land$ and $\neg$.

The first six disjoint sorts are the same as defined in Language $L$ of Chapter 3. The sort transformation is defined as follows:

Transformation: a set with a finite number of transformations $T, T_1, T_2, \ldots$.

In language $L^S$, the fact that a subject $S$ has access right $R$ for object $O$ is represented using a ground atom $\text{holds}(S, R, O)$. The fact that a subject $S$ is a member of $G$ is represented by $S \in G$. Similarly, we represent inclusion relationships between subject groups such as $G_1 \subseteq G_2$ or between access right groups such as $GA_1 \subseteq GA_2$.

In general, we define a fact $F$ to be an atomic formula of $L^S$ or its negation, while a ground fact is a fact without variable occurrence. We view $\neg\neg F$ as $F$. Fact expressions of $L^S$ are defined as follows: (i) each fact is a fact expression; (ii) if $\phi$ and $\psi$ are fact expressions, then $\phi \land \psi$ is also a fact expression. A ground fact expression is a fact expression without variable occurrence. A ground fact expression is called a ground instance of a fact expression if this ground fact expression is obtained from the fact expression by replacing each of its variable occurrence with the same sort constant. Now we are ready to formally define the propositions in $L^S$.

A policy proposition in $L^S$ is an expression of the form

$$\phi \text{ after } T_1, \cdots, T_m,$$

(4.1)

where $\phi$ is a ground fact expression and $T_1, \cdots, T_m$ ($m \geq 0$) are transformations. Intuitively, this proposition means that after performing transformations $T_1, \cdots, T_m$ se-
quently, the ground fact expression $\phi$ holds. If $m = 0$, we will rewrite (4.1) as

\[
\text{initially } \phi, \tag{4.2}
\]

which is called the initial policy proposition.

A transformation proposition is an expression of the form

\[
T \text{ causes } \phi \text{ if } \psi, \tag{4.3}
\]

where $T$ is a transformation, $\phi$ and $\psi$ are ground fact expressions. Intuitively, a transformation proposition expresses the following meaning: at a given state, if the precondition $\psi$ is true\(^1\), then after performing the transformation $T$ at this state, the ground fact expression $\phi$ will be true in the resulting state.

If the set of $\psi$ is empty, we will rewrite (4.3) as

\[
T \text{ causes } \phi, \tag{4.4}
\]

which means that there is no precondition to perform the transformation or the precondition is always true. That is, the transformation can always be performed.

A policy domain description $D$ in $L^i$ is a finite set of initial policy propositions and transformation propositions.

**Example 14** The following is a domain description:

\[
\text{initially } \text{holds}(S, \text{Read}, O) \land \text{holds}(S, \text{Write}, O),
\]

\[
\text{Delete-write}(S, \text{Write}, O) \text{ causes } \lnot \text{holds}(S, \text{Write}, O).
\]

This domain description expresses the following information: initially subject $S$ has Read and Write rights on object $O$. A transformation Delete-write($S, \text{Write}, O$) is available in this domain; if this transformation occurs, then the subject $S$ will no longer have the Write right for object $O$. ■

\(^1\)We will formally define a state and the semantics of a transformation proposition in the next subsection.
Example 15 Let us now consider the static separation of duty access policy. This policy refers to the fact that a certain set of access rights cannot be allowed for the same subject. For example, consider the operations Submit, Evaluate and Approve associated with a Budget object. A static separation of duty policy requires that the same subject cannot be authorized for all of the three operations.

Let $S$ be the subject, $B$ be the budget object. The domain description is specified as follows:

Initially $\neg (\text{holds}(S, \text{Submit}, B) \land \text{holds}(S, \text{Evaluate}, B) \land \text{holds}(S, \text{Approve}, B))$.

Example 16 Let us now consider the more general access policy on dynamic separation of duty. In this case, a subject can potentially execute any operation in a given set, though s/he cannot execute all of them. By executing some operation, s/he will automatically rule out the possibility of executing the others. The policy is referred to as dynamic in the sense that which actions a user can execute is determined by the user. For instance, consider the following simple example. Let a group officer be represented using a group-subject G-Officer. Let this group have access rights to submit, evaluate and approve a budget. Let the budget be represented using an object $B$. Now if a subject $S$ belongs to G-Officer, that is, $S \in \text{G-Officer}$, then $\text{holds}(S, \text{Submitable}, B)$, $\text{holds}(S, \text{Evaluateable}, B)$ and $\text{holds}(S, \text{Approveable}, B)$. Let the transformations be $Rqst(S, \text{Submit}, B)$, $Rqst(S, \text{Evaluate}, B)$ and $Rqst(S, \text{Approve}, B)$. The domain description $D$ can be represented as follows:

Initially $S \in \text{G-Officer}$,

Initially $\text{holds}(S, \text{Submitable}, B)$,

Initially $\text{holds}(S, \text{Evaluateable}, B)$,

Initially $\text{holds}(S, \text{Approveable}, B)$,

$Rqst(S, \text{Submit}, B)$ causes

$\text{holds}(S, \text{Submit}, B) \land$

$\neg \text{holds}(S, \text{Evaluateable}, B) \land$

THESIS
\(\neg \text{holds}(S, \text{Approveable}, B)\)

if \(S \in G\text{-Officer} \land \text{holds}(S, \text{Submitable}, B)\).

\(\text{Rqst}(S, \text{Evaluate}, B)\) causes

\(\text{holds}(S, \text{Evaluate}, B) \land\)

\(\neg \text{holds}(S, \text{Approveable}, B) \land\)

\(\neg \text{holds}(S, \text{Submitable}, B)\)

if \(S \in G\text{-Officer} \land \text{holds}(S, \text{Evaluateable}, B)\).

\(\text{Rqst}(S, \text{Approve}, B)\) causes

\(\text{holds}(S, \text{Approve}, B) \land\)

\(\neg \text{holds}(S, \text{Evaluateable}, B) \land\)

\(\neg \text{holds}(S, \text{Submitable}, B)\)

if \(S \in G\text{-Officer} \land \text{holds}(S, \text{Approveable}, B)\).

---

**Example 17** Now we consider the specification of the Chinese wall access policy [24] using our domain description. The Chinese wall access policy can be viewed as a special kind of dynamic separation of duty. In a Chinese wall policy, objects are grouped into *company datasets*, for instance Company-1 and Company-2. Company datasets whose organizations are in competition are then grouped together into *conflict of interest classes*. A subject can potentially access an object from either company dataset, but if the subject accesses an object in a company dataset 1, it cannot be allowed to access any object in a company dataset that is in a conflict of interest class with dataset 1. In our language, company datasets can be represented by a *group-object*. For instance, if *Company_1* and *Company_2* are in the same conflict of interest class, a subject who has accessed an object of *Company_1* will not be allowed to access any object in *Company_2* and vice versa.

Suppose that *Company_1* and *Company_2* are in the same conflict of interest class, *O_1* is an object of *Company_1* and *O_2* is an object of *Company_2* and *S* is a subject. We use \(\text{holds}(S, \text{Accessable}, O_1)\) and \(\text{holds}(S, \text{Accessable}, O_2)\) to represent that *S* can potentially...
access both \(O_1\) and \(O_2\). We have the following transformations: \(Rqst(S, Access, O_1)\) and \(Rqst(S, Access, O_2)\). The domain description \(D\) is specified as follows:

- **Initially** \(O_1 \in Company_1\),
- **Initially** \(O_2 \in Company_2\),
- **Initially** \(holds(S, Accessable, O_1)\),
- **Initially** \(holds(S, Accessable, O_2)\).

\(Rqst(S, Access, O_1)\) **causes**

\[
holds(S, Access, O_1) \land \\
\neg holds(S, Accessable, O_2) \\
\text{if } O_1 \in Company_1 \land O_2 \in Company_2 \land holds(S, Accessable, O_1),
\]

\(Rqst(S, Access, O_2)\) **causes**

\[
holds(S, Access, O_2) \land \\
\neg holds(S, Accessable, O_1) \\
\text{if } O_1 \in Company_1 \land O_2 \in Company_2 \land holds(S, Accessable, O_2).
\]

That is, if \(S\) accesses \(O_1\), then it will not be able to access \(O_2\) due to the transformation \(Rqst(S, Access, O_1)\). Similarly, if \(S\) accesses \(O_2\), it will not be able to access \(O_1\).

---

4.1.2 Semantics of \(L^f\)

Now we define the semantics of language \(L^f\). A **state** is a set of ground facts. Given a ground fact \(F\) (i.e. \(F\) is \(holds(S, R, O)\) or \(\neg holds(S, R, O)\)) and a state \(\sigma\), we say \(F\) is **true** in \(\sigma\) iff \(F \in \sigma\), and \(F\) is **false** in \(\sigma\) iff \(\neg F \in \sigma\). A ground fact expression \(\phi \equiv F_1 \land \cdots \land F_k\), where each \(F_i\) (\(1 \leq i \leq k\)) is a ground fact, is **true** in \(\sigma\) iff each \(F_i\) (\(1 \leq i \leq k\)) is in \(\sigma\). Furthermore, a fact expression with variables is true in \(\sigma\) iff each of its ground instances is true in \(\sigma\). A state \(\sigma\) is **complete** if for any ground fact \(F\) of \(L^f\), \(F\) or \(\neg F\) is in \(\sigma\). Otherwise \(\sigma\) is called a **partial state**. An **inconsistent state** \(\sigma\) is a state containing a pair of complementary ground facts \(F\) and \(\neg F\).
A transition function $\rho$ maps a set $(T, \sigma)$ into a state, where $T$ is a transformation and $\sigma$ is a state. Intuitively, $\rho(T, \sigma)$ denotes the resulting state caused by performing transformation $T$ in $\sigma$. A structure $M$ is a pair $(\sigma, \rho)$, where $\sigma$ is a state, and $\rho$ is a transition function. For any structure $M$ and any set of transformations $T_1, \ldots, T_m$, the notation $M^{T_1, \ldots, T_m}$ denotes the state

$$\rho(T_m, \rho(T_{m-1}, \ldots, \rho(T_1, \sigma) \cdots)),$$

where $\rho$ is the transition function of $M$, and $\sigma$ is the state of $M$.

We denote that a policy proposition $\phi$ is satisfied in a structure $M$ as $M \models_{\mathcal{U}} \phi$ after $T_1, \ldots, T_m$. This is true iff $\phi$ is true in the state $M^{T_1, \ldots, T_m}$. Given a domain description $D$, we say that a state $\sigma_0$ is the initial state of $D$ iff (i) for each initial policy proposition initially $\phi$ of $D$, $\phi$ is true in $\sigma_0$; (ii) if there is another state $\sigma$ satisfying condition (i), then $\sigma_0 \subseteq \sigma$ (i.e. $\sigma_0$ is the least state satisfying all initial policy propositions of $D$).

**Definition 7** A structure $(\sigma_0, \rho)$ is a model of a domain description $D$ iff $\sigma_0$ is a consistent initial state of $D$, and for any transformation $T$ and state $\sigma$, the following conditions hold:

1. if $D$ includes a transformation proposition $T$ causes $\phi$ if $\psi$, and $\psi$ is true in $\sigma$, then $\phi$ is true in $\rho(T, \sigma)$;

2. for each $F$ in $D$ which is not affected by $T$, $F \in \rho(T, \sigma)$ iff $F \in \sigma$;

3. for each $F$ in $D$ which occurs in $\phi$ in proposition $T$ causes $\phi$ if $\psi$, and $\psi$ is not true in $\sigma$, then $F \in \rho(T, \sigma)$ iff $F \in \sigma$.

Condition 1 says that if precondition $\psi$ for transformation $T$ is true in the current state $\sigma$, then after performing the transformation, the effect of the transformation $\phi$ should be true in the resulting state $\rho(T, \sigma)$. Condition 2 actually states a persistent rule which says for any fact that is not related to transformation $T$ should remain unchanged with respect to the performance of $T$. Condition 3 states that for some fact $F$ in $\phi$, if $\phi$ cannot be obtained by performing transformation $T$, then $F$ is true if it is originally true in the state before the transformation.

We say that a domain description $D$ is consistent if $D$ has a model. A policy proposition $\phi$ after $T_1, \ldots, T_m$ is entailed by $D$, denoted as $D \models_{\mathcal{U}} \phi$ after $T_1, \ldots, T_m$, iff it is true in each model of $D$. 

**THESIS**
Proposition 1 A consistent domain description $D$ has a unique model.

Example 18 Continuation of Example 16. For the dynamic separation of duty example, the initial state of $D$ is:

$$
\sigma_0 = \{ S \in \text{G-Officer}, \text{holds}(S,\text{Submitable},B), \text{holds}(S,\text{Evaluateable},B), \\
\text{holds}(S,\text{Approveable},B) \}.
$$

It can be easily shown that the following results hold:

$$
D \models_D S \in \text{G-Officer} \land \text{holds}(S,\text{Submit},B) \land \neg \text{holds}(S,\text{Evaluateable},B) \land \\
\text{after Rqst}(S,\text{Submit},B),
$$

$$
D \models_D S \in \text{G-Officer} \land \text{holds}(S,\text{Evaluate},B) \land \neg \text{holds}(S,\text{Submitable},B) \land \\
\text{after Rqst}(S,\text{Evaluate},B),
$$

$$
D \models_D S \in \text{G-Officer} \land \text{holds}(S,\text{Approve},B) \land \neg \text{holds}(S,\text{Submitable},B) \land \\
\text{after Rqst}(S,\text{Approve},B).
$$

Example 19 Continuation of Example 17. For the Chinese wall policy, the initial state of $D$ is:

$$
\sigma_0 = \{ O_1 \in \text{Company}_1, O_2 \in \text{Company}_2, \text{holds}(S,\text{Accessable},O_1), \\
\text{holds}(S,\text{Accessable},O_2) \}.
$$

From the above description, it is not difficult to show that

$$
D \models_D O_1 \in \text{Company}_1 \land O_2 \in \text{Company}_2 \land \text{holds}(S,\text{Access},O_1) \land \\
\neg \text{holds}(S,\text{Accessable},O_2) \land \\
\text{after Rqst}(S,\text{Access},O_1),
$$

$$
D \models_D O_1 \in \text{Company}_1 \land O_2 \in \text{Company}_2 \land \text{holds}(S,\text{Access},O_2) \land \\
\neg \text{holds}(S,\text{Accessable},O_1) \land \\
\text{after Rqst}(S,\text{Access},O_2),
$$

THESIS
4.1.3 Document Release Example

In this section, we consider a slightly modified version of the well-known document release example [75] and specify the authorization transformations using our language $L'$. 

Example 20 The following is cited from [75]:

"... a scientist creates a document and hence gets own, read and write access rights to it. After preparing the document for publication, the scientist asks for a review from a patent officer. In the process, the scientist loses the write right to the document, since it is clearly undesirable for a document to be edited during or after a (successful) review. After review of the document, the patent officer grants the scientist an approval. It is reasonable to disallow further attempts to review the document after an approval is granted. Thus the review right for the document is lost as approval is granted. After obtaining approval from the patent officer, the scientist can publish the document by getting a release right for the document. ..."

We use subject constant $Sci$ to denote the scientist, subject constant $PO$ to denote the patent officer, object constant $Doc$ to denote the document, access right constants $Own$, $Read$, $Write$, $Review$, $Pat-ok$, $Pat-reject$, $Release$ to denote the rights own, read, write, review, patent-ok, patent-reject and release respectively. We also have the following transformations $Rqst(Sci, Doc, PO)$, $Get-approval(Sci, Doc, PO)$, $Get-rejection(Sci, Doc, PO)$, $Release-doc(Sci, Doc)$ and $Revise-doc(Sci, Doc)$. The domain description $D$ expressing the access policy within our framework is given as follows:

\[ D \models L', O_1 \in Company_1 \land O_2 \in Company_2 \land holds(S, Access, O_1) \land \\
\neg holds(S, Accessable, O_2) \land \\
\text{after } Rqst(S, Access, O_1), Rqst(S, Access, O_2), \\
D \models L', O_1 \in Company_1 \land O_2 \in Company_2 \land holds(S, Access, O_2) \land \\
\neg holds(S, Accessable, O_1) \land \\
\text{after } Rqst(S, Access, O_2), Rqst(S, Access, O_1). \]
initially holds(Sci, Own, Doc),
initially holds(Sci, Read, Doc),
initially holds(Sci, Write, Doc),

\[ \text{Rqst}(Sci, Doc, PO) \text{ causes} \]
holds(PO, Review, Doc) \land \neg \text{holds}(Sci, Write, Doc)
if \text{holds}(Sci, Own, Doc) \land \text{holds}(Sci, Write, Doc),

\[ \text{Get-approval}(Sci, Doc, PO) \text{ causes} \]
holds(Sci, Pat-ok, Doc) \land \neg \text{holds}(PO, Review, Doc)
if \text{holds}(PO, Review, Doc) \land \text{holds}(Sci, Own, Doc),

\[ \text{Get-rejection}(Sci, Doc, PO) \text{ causes} \]
holds(Sci, Pat-reject, Doc) \land \neg \text{holds}(PO, Review, Doc),
if \text{holds}(PO, Review, Doc) \land \text{holds}(Sci, Own, Doc),

\[ \text{Release-doc}(Sci, Doc) \text{ causes} \]
holds(Sci, Release, Doc) \land \neg \text{holds}(Sci, Pat-ok, Doc)
if \text{holds}(Sci, Pat-ok, Doc),

\[ \text{Revise-doc}(Sci, Doc) \text{ causes holds}(Sci, Write, Doc) \]
if \text{holds}(Sci, Pat-reject, Doc).

The initial state of \( D \) is
\[ \sigma_0 = \{ \text{holds}(Sci, Own, Doc), \text{holds}(Sci, Read, Doc), \text{holds}(Sci, Write, Doc) \} \].

Let us now consider the policy propositions that are entailed from \( D \). From the semantics presented previously, we can prove that the following results hold.

\( D \models_{\Sigma} \text{holds}(PO, Review, Doc) \land \neg \text{holds}(Sci, Write, Doc) \)
after \text{Rqst}(Sci, Doc, PO),

THEESIS
§4.1 Syntax and Semantics of Language $\mathcal{L}^*$

\[
D \models_{\mathcal{L}^*} \text{holds}(\text{Sci}, \text{Pat-ok}, \text{Doc}) \land \neg\text{holds}(\text{PO}, \text{Review}, \text{Doc})
\]
\textbf{after} $\text{Rqst}(\text{Sci}, \text{Doc}, \text{PO}), \text{Get-approval}(\text{Sci}, \text{Doc}, \text{PO})$

\[
D \models_{\mathcal{L}^*} \text{holds}(\text{Sci}, \text{Pat-reject}, \text{Doc}) \land \neg\text{holds}(\text{PO}, \text{Review}, \text{Doc})
\]
\textbf{after} $\text{Rqst}(\text{Sci}, \text{Doc}, \text{PO}), \text{Get-rejection}(\text{Sci}, \text{Doc}, \text{PO})$

\[
D \models_{\mathcal{L}^*} \text{holds}(\text{Sci}, \text{Release}, \text{Doc}) \land \text{holds}(\text{Sci}, \text{Pat-ok}, \text{Doc})
\]
\textbf{after} $\text{Rqst}(\text{Sci}, \text{Doc}, \text{PO}), \text{Get-approval}(\text{Sci}, \text{Doc}, \text{PO})$

\[
\text{Release-doc}(\text{Sci}, \text{Doc})
\]

\[
D \models_{\mathcal{L}^*} \text{holds}(\text{Sci}, \text{Write}, \text{Doc})
\]
\textbf{after} $\text{Rqst}(\text{Sci}, \text{Doc}, \text{PO}), \text{Get-rejection}(\text{Sci}, \text{Doc}, \text{PO})$

\[
\text{Revise-doc}(\text{Sci}, \text{Doc}).
\]

The above results describe the expected solutions with respect to the execution of different sequences of transformations. Furthermore the following theorem shows that the performance of sequence of transformations will not affect the scientist’s own right for the document.

The following results hold.

\[
D \models_{\mathcal{L}^*} \text{holds}(\text{Sci}, \text{Own}, \text{Doc})
\]
\textbf{after} $\text{Rqst}(\text{Sci}, \text{Doc}, \text{PO})$

\[
D \models_{\mathcal{L}^*} \text{holds}(\text{Sci}, \text{Own}, \text{Doc})
\]
\textbf{after} $\text{Rqst}(\text{Sci}, \text{Doc}, \text{PO}), \text{Get-approval}(\text{Sci}, \text{Doc}, \text{PO})$

\[
D \models_{\mathcal{L}^*} \text{holds}(\text{Sci}, \text{Own}, \text{Doc})
\]
\textbf{after} $\text{Rqst}(\text{Sci}, \text{Doc}, \text{PO}), \text{Get-rejection}(\text{Sci}, \text{Doc}, \text{PO})$

\[
D \models_{\mathcal{L}^*} \text{holds}(\text{Sci}, \text{Own}, \text{Doc})
\]
\textbf{after} $\text{Rqst}(\text{Sci}, \text{Doc}, \text{PO}), \text{Get-approval}(\text{Sci}, \text{Doc}, \text{PO})$

\[
\text{Release-doc}(\text{Sci}, \text{Doc})
\]

\[
D \models_{\mathcal{L}^*} \text{holds}(\text{Sci}, \text{Own}, \text{Doc})
\]
\textbf{after} $\text{Rqst}(\text{Sci}, \text{Doc}, \text{PO}), \text{Get-rejection}(\text{Sci}, \text{Doc}, \text{PO})$

\[
\text{Revise-doc}(\text{Sci}, \text{Doc}).
\]

\begin{itemize}
  \item THESIS
\end{itemize}
4.1.4 Properties of Language $\mathcal{L}^3$

Language $\mathcal{L}^3$ has the following properties:

- Incomplete information is allowed. In $\mathcal{L}^3$, the state of the authorization policies can be specified to be incomplete in the sense that some authorizations may not be represented in the state. For instance, in Example 19, the initial state $\sigma_0$ is incomplete because it does not include the facts $\text{holds}(PO,\text{Review},\text{Doc})$ or $\neg\text{holds}(PO,\text{Review},\text{Doc})$.

- Denials are expressed explicitly. As incomplete information is allowed in the state of authorization policies, denials (negations of authorization policies) must be explicitly represented in the state.

- The entailment relation $\models_{\mathcal{L}}$ of $\mathcal{L}^3$ is nonmonotonic with respect to transformation propositions. Recall that a domain description $D$ is a finite set of policy propositions and transformation propositions. The nonmonotonicity of $\models_{\mathcal{L}}$ with respect to transformation propositions states that adding more transformation propositions into $D$ may result in a policy proposition being no longer entailed in the domain description. This is because a new transformation proposition may change an authorization policy to become negative. For example, consider a domain description $D$ consisting of the following policy and transformation propositions:

  \[ \text{initially } \text{holds}(S, \text{Read}, \text{File}), \]
  \[ \text{Assign-write}(S, \text{Write}, \text{File}) \text{ causes } \text{holds}(S, \text{Write}, \text{File}). \]

  Clearly, we have $D \models \text{holds}(S, \text{Write}, \text{File})$ after $\text{Assign-write}(S, \text{Write}, \text{File})$. However, if we add another transformation proposition into $D$:

  \[ \text{Delete-write}(S, \text{Write}, \text{File}) \text{ causes } \neg\text{holds}(S, \text{Write}, \text{File}) \]
  \[ \text{if } \text{holds}(S, \text{Write}, \text{File}), \]

  Then we have $D' \models \neg\text{holds}(S, \text{Write}, \text{File})$ after

  \[ \text{Assign-write}(S, \text{Write}, \text{File}), \]

THESIS
\( \text{Delete-write}(S, \text{Write, File}) \).

where \( D' \) is the new domain description with the above transformation proposition added into \( D \).

- \( \models_{D'} \) is also nonmonotonic with respect to policy propositions. This can be observed from the following example. Suppose a domain description \( D \) consists of the following policy and transformation propositions:

\[
\text{initially holds}(S, \text{Own, File}),
\]

\( \text{Delete-own}(S, \text{Own, File}) \) causes \( \neg \text{holds}(S, \text{Own, File}) \) if \( S \in G \).

As the pre-condition \( S \in G \) of \( \text{Delete-own}(S, \text{Own, File}) \) does not hold in the initial state, we have \( D \models_{D'} \text{holds}(S, \text{Own, File}) \) after \( \text{Delete-own}(S, \text{Own, File}) \). However, if we add a policy proposition \( S \in G \) into \( D \) to form a new domain description \( D' \), then we have

\( D' \models_{D'} \neg \text{holds}(S, \text{Own, File}) \) after \( \text{Delete-own}(S, \text{Own, File}) \).

Now let us discuss some limitations of \( L' \):

- \( L' \) cannot express inherited and causal authorization policies. For instance, in many situations, a subject can inherit an access right from its group’s access right. Unfortunately, this kind of fact cannot be expressed by \( L' \). Also an authorization policy may have a causal relationship with other policies. For example, we may need to express the fact that if subject \( S \) has \text{Write} right on object \( O \), then subject \( S' \) should also have \text{Read} right on object \( O \). \( L' \) is not able to express this kind of causal relation.

- \( L' \) cannot express default authorization policies. For instance, the closed world assumption on the state of authorization policies can be viewed as a general default authorization: any policy that is not explicitly represented in the state will be assumed to be its negation.

- \( L' \) cannot express constraints. Sometimes we need to specify some restrictions on authorization policies. These restrictions are represented by constraints which should be satisfied by any state.
4.2 Syntax and Semantics of Language $\mathcal{L}^{sd}$

To overcome these limitations mentioned above, we extend language $\mathcal{L}'$ to $\mathcal{L}^{sd}$ in this section.

4.2.1 Syntax of $\mathcal{L}^{sd}$

The language $\mathcal{L}^{sd}$ has the same sorts and types of propositions as the language $\mathcal{L}'$ but in addition has one more type of proposition that we refer to as default proposition of the form:

$$\phi \text{ implies } \psi \text{ with absence } \gamma,$$

(4.5)

where $\phi, \psi$ and $\gamma$ are fact expressions. Note that $\phi, \psi$ and $\gamma$ may contain variables. In this case, the default proposition (4.5) will be treated as a set of default propositions obtained from (4.5) by replacing $\phi, \psi$ and $\gamma$ with their ground instances respectively.

Intuitively, the default proposition says that if $\phi$ is true in a state $\sigma$, and it can not be derived that $\gamma$ is true in $\sigma$, then we will infer that $\psi$ is true in $\sigma$.

There is a special form of the default proposition when the set $\gamma$ is empty in (4.5). In this case, we rewrite (4.5) as

$$\phi \text{ provokes } \psi,$$

(4.6)

which is viewed as a causal or inheritance relation between $\phi$ and $\psi$. For example, in many situations, a system should satisfy the following relation:

$$\text{holds}(s, \text{Own}, o) \text{ provokes holds}(s, \text{Read}, o) \land \text{holds}(s, \text{Write}, o).$$

On the other hand, there is also a special form of (4.6) when the set $\phi$ is empty. In this case we rewrite (4.6) as

$$\text{always } \psi,$$

(4.7)

which represents a constraint that should be satisfied by any state in the domain. For instance, we may express a constraint stating that the Root has any right for any object as follows:

$$\text{always holds(Root, } r, o).$$

THESIS
§4.2 Syntax and Semantics of Language $L^{sd}$

We define a transformation-based policy domain description $D$ (or domain description for short) in language $L^{sd}$ as a finite set of initial policy propositions, transformation propositions and default propositions.

Example 21 The following is an example of domain description $D$ specified by $L^{sd}$:

- initially $\text{holds}(S, \text{Own}, O)$,
- $\text{holds}(S, \text{Own}, O)$ implies $\text{holds}(S, \text{Write}, O)$
- with absence $\neg\text{holds}(S, \text{Write}, O)$,
- $\text{Delete-write}(S, \text{Write}, O)$ causes $\neg\text{holds}(S, \text{Write}, O)$.

\[\]

4.2.2 Semantics of $L^{sd}$

The semantics of $L^{sd}$ is not just a non trivial extension of the semantics of $L'$ although the syntax of $L^{sd}$ is a simple augmentation of that of $L'$. The reason is that to define a proper semantics of the default proposition (4.5), we have to employ a fixed-point semantics that shares the spirit of fixed-point semantics used in logic programs [37].

Given a domain description $D$, we first define the initial state of $D$. Suppose that a domain description $D_p$ only contains initial policy propositions, default propositions of the special form (4.6) and transformation propositions.

Definition 8 A state $\sigma_0$ is an initial state of $D_p$ iff $\sigma_0$ is the smallest state that satisfies the following conditions:

1. for each initial policy proposition initially $\phi$, $\phi$ is true in $\sigma_0$;

2. for each default proposition with the form $\phi$ provokes $\psi$, if $\phi$ is true in $\sigma_0$, then $\psi$ is also true in $\sigma_0$.

This definition describes how to get the initial state (initial policy base) from a domain description which only consists of initial policy propositions, default propositions of the special form (4.6) and transformation propositions. Condition 1 states that some of the facts of the initial state are obtained from these initial policy propositions, while Condition 2 states that some of the facts of the initial state are obtained
from these default propositions \( \phi \) provokes \( \psi \). For these propositions, if \( \phi \) is already in the initial state \( \sigma_0 \), \( \psi \) will also be included in \( \sigma_0 \). Since we require that the state is the smallest one which satisfies both conditions, this makes the initial state unique.

For instance, if the domain description consists:

- initially \( a \),
- initially \( b \),
- initially \( c \), and
- \( c \) provokes \( f \),
- \( d \) provokes \( e \).

The initial state we get from this domain description is \( \{a,b,c,f\} \).

Now we consider a domain description \( D \) containing default propositions with the general form (4.5). To define the initial state of \( D \), we first translate \( D \) to domain description \( D_p \) described above.

**Definition 9** Let \( \sigma_0 \) be a state. Suppose domain description \( D_p \) is obtained from \( D \) as follows:

1. by deleting each default proposition \( \phi \) implies \( \psi \) with absence \( \gamma \) from \( D \) if for every \( F_i \) in \( \gamma \), \( F_i \) is true in \( \sigma_0 \);

2. by translating all other default propositions \( \phi \) implies \( \psi \) with absence \( \gamma \) to the form \( \phi \) provokes \( \psi \).

Now if this state \( \sigma_0 \) is an initial state of \( D_p \), then we also define it to be an initial state of \( D \).

In this definition, it further divides the general default propositions \( \phi \) implies \( \psi \) with absence \( \gamma \) into two categories, in which \( \gamma \) is either true or false in the initial state. (1) states that for those propositions in which \( \gamma \) is true, \( \phi \) implies \( \psi \) with absence \( \gamma \) will not hold, so they need to be deleted. (2) states that for those propositions in which \( \gamma \) is false, \( \phi \) implies \( \psi \) with absence \( \gamma \) holds and can be further simplified as \( \phi \) provokes \( \psi \).

\(^2\)Recall that \( \gamma \equiv F_1 \land \cdots \land F_i \land \cdots \land F_k \), each \( F_i \) (1 ≤ i ≤ k) is a ground fact.
Taking default propositions into account, it turns out that the initial state of a domain description may be not unique, or may not even exist. This is shown by the following example.

**Example 22** A domain description $D$ consists of the following propositions:

- initially $\text{holds}(S, \text{Own}, O)$,
- $\text{holds}(S, \text{Own}, O)$ implies $\text{holds}(S, \text{Write}, O)$ with absence $\lnot \text{holds}(S, \text{Write}, O)$,
- $\text{holds}(S, \text{Own}, O)$ implies $\lnot \text{holds}(S, \text{Write}, O)$ with absence $\text{holds}(S, \text{Write}, O)$,
- Delete-own($S, \text{Own}, O$) causes $\lnot \text{holds}(S, \text{Own}, O)$.

Clearly, $D$ has two initial states:

- $\sigma_0 = \{\text{holds}(S, \text{Own}, O), \text{holds}(S, \text{Write}, O)\}$, and
- $\sigma_0' = \{\text{holds}(S, \text{Own}, O), \lnot \text{holds}(S, \text{Write}, O)\}$.

Consider another domain description $D'$ consisting of the following propositions:

- initially $\text{holds}(S, \text{Own}, O)$,
- $\text{holds}(S, \text{Own}, O)$ implies $\text{holds}(S, \text{Write}, O)$ with absence $\text{holds}(S, \text{Write}, O)$,
- Delete-own($S, \text{Own}, O$) causes $\lnot \text{holds}(S, \text{Own}, O)$.

$D'$ has no initial state according to the definition described above. ■

Similarly to that defined for the language $L^+$, we define that a structure of $D$ to be a pair $(\sigma, \rho)$, where $\sigma$ is a state, and $\rho$ is a transition function introduced in section 4.1. A policy proposition (4.1) is satisfied in a structure $M$, denoted as $M \models_{L^+} \phi$ after $T_1, \ldots, T_m$, if $\phi$ is true in state $M^{T_1, \ldots, T_m}$.

Now, we define the model of a domain description.

**Definition 10** Given a domain description $D$, let $(\sigma_0, \rho)$ be a structure of $D$, where $\sigma_0$ is a consistent initial state of $D$, and $\rho$ is a transition function introduced in section 4.1. $(\sigma_0, \rho)$ is a model of $D$ iff for any transformation $T$ and state $\sigma$, the following conditions hold:

1. if $D$ includes a transformation proposition $T$ causes $\phi$ if $\psi$, and $\psi$ is true in $\sigma$, then $\phi$ is true in $\rho(T, \sigma)$;
2. for each $F$ in $D$ which is not affected by $T$, $F \in p(T, \sigma)$ iff $F \in \sigma$;

3. for each $F$ in $D$ which occurs in $\phi$ in proposition $T$ causes $\phi$ if $\psi$, and $\psi$ is not true in $\sigma$, then $F \in p(T, \sigma)$ iff $F \in \sigma$.

The above definition is similar to Definition 7, since after Definition 8 and 9, the initial state of a general domain description with default proposition is defined. So the model of a general domain description is defined the same way as the model of a domain description without default proposition does.

Clearly, a domain description $D$ may have one or more or no models. $D$ is consistent if $D$ has a model. A policy proposition $\phi$ after $T_1, \cdots, T_m$ is entailed by $D$, denoted as $D \models_{L^d} \phi$ after $T_1, \cdots, T_m$ iff it is true in each model of $D$. A model $M$ of $D$ is complete if for any policy proposition $\phi$ after $T_1, \cdots, T_m$, either $D \models_{L^d} \phi$ after $T_1, \cdots, T_m$ or $D \models_{L^d} \neg \phi$ after $T_1, \cdots, T_m$.

**Example 23** A credit union divides its customers into two classes $G_1$ and $G_2$. A member of $G_1$ has a credit limit of up to $5000$. A member of $G_2$ has a credit limit of up to $10000$. The credit union reviews the financial state of its customers and upgrades or downgrades their credit limits accordingly. $A$, $B$ and $C$ are three customers of this credit union. Suppose the information we know is that $B$ belongs to $G_2$, $A$ has a credit limit of up to $5000$ and $C$ belongs to $G_1$ and has a credit limit of $5000$. The domain description $D$ for this example is:

- initially holds($A$, Credit, $5000$),
- initially holds($C$, Credit, $5000$),
- initially $B \in G_2$,
- initially $C \in G_1$,
- holds($A$, Credit, $5000$) implies $A \in G_1$
  with absence $A \in G_2$,
- $B \in G_2$ provokes holds($B$, Credit, $10000$).

Suppose $x$ represents a general customer, the transformation propositions are:

- Upgrade ($x$) causes $x \in G_2$ if $x \in G_1$,
- Downgrade ($x$) causes $x \in G_1$ if $x \in G_2$.

**THESIS**
The initial state of $D$ is:

$$\sigma_0 = \{\text{holds}(A, \text{Credit}, 5000), \text{holds}(B, \text{Credit}, 10000), \text{holds}(C, \text{Credit}, 5000),$$
$$A \in G_1, B \in G_2, C \in G_1\}$$

Obviously, the following holds:

$$D \models \_L^d \text{holds}(A, \text{Credit}, 10000) \land \text{holds}(B, \text{Credit}, 10000) \land \text{holds}(C, \text{Credit}, 5000) \land$$
$$A \in G_2 \land B \in G_2 \land C \in G_1$$

after $Upgrade(A)$,

$$D \models \_L^d \text{holds}(A, \text{Credit}, 5000) \land \text{holds}(B, \text{Credit}, 5000) \land \text{holds}(C, \text{Credit}, 5000) \land$$
$$A \in G_1 \land B \in G_1 \land C \in G_1$$

after $Downgrade(B)$.

After sequentially executing $Upgrade(A)$ and $Downgrade(B)$, the following will hold:

$$D \models \_L^d \text{holds}(A, \text{Credit}, 10000) \land \text{holds}(B, \text{Credit}, 5000) \land \text{holds}(C, \text{Credit}, 5000) \land$$
$$A \in G_2 \land B \in G_1 \land C \in G_1$$

after $Upgrade(A), Downgrade(B)$.

\section*{4.3 Related Work}

We now study some related work done in this area. Jajodia et al [48] proposed a logic language for expressing authorizations. Similarly to ours, they use predicates and rules to specify the authorizations; their work mainly emphasizes the representation and evaluation of authorizations, not for authorization transformations. Woo and Lam proposed a formal approach using default logic to represent and evaluate authorizations [91]. Since their work is quite close to our work, we will study their approach in more details in the next section.

\subsection*{4.3.1 Woo and Lam's Approach: A Review}

As both Woo and Lam and our approaches are able to represent default authorizations, in this section, we investigate the relationship between our approach and Woo
and Lam's in detail. To keep our presentation and comparison clear and consistent, we now describe briefly a simplified version of Woo and Lam's formalism.

Basically, in Woo and Lam's system, a set of authorizations is specified by a policy base. The language used is a many sorted first order language with rule construct\(^3\). The rule construct is similar to the default construct in default logic [69].

Formally, there are three different sets named the set of subjects, set of objects and set of access rights respectively in the system. A rule is a form written as \( f : f' \rightarrow g \), where \( f, f' \) and \( g \) are formulas and called the prerequisite, assumption and consequent of the rule respectively. A rule is said to be ground if \( f, f' \) and \( g \) do not include ordinary variables. Usually, a rule including ordinary variables can be viewed as a set of all its ground instances. Therefore, in practice, we only need to consider ground rules.

For convenience, if any component formula is missing from a rule, it is assumed to be true (T\(^\). Furthermore, the notation \( f \Rightarrow g \) is used to represent a rule of the form \( f : T \rightarrow g \). T \( \Rightarrow g \) which is further abbreviated to \( g \). The intuitive semantic meaning of a rule \( f : f' \rightarrow g \) is as follows: If \( f \) is believed and if it is also consistent to believe \( f' \), then \( g \) is believed as well. The Woo-Lam policy base is a finite set of rules.

In Woo and Lam’s language, the fact that a subject \( S \) is explicitly granted an access right \( R \) to object \( O \) is expressed as the form \( (S, R, O) \in P^+ \), while the fact that a subject \( S \) is explicitly denied an access right \( R \) to object \( O \) is expressed as \( (S, R, O) \in N^+ \). These can be translated into \( \text{holds}(S, R, O) \) and \( \neg \text{holds}(S, R, O) \) respectively in our language\(^4\).

Therefore, when we express a policy base in Woo and Lam’s formalism, we will use a modified form as shown in the following example.

Example 24 A policy base \( B_1 \) is specified as:

\[
B_1 = \{ \neg \text{holds}(S, \text{Write}, O_1) \land \text{holds}(S, \text{Write}, O_2) / \text{holds}(S, \text{Write}, O_3), \text{holds}(S, \text{Read}, O_1) \Rightarrow \text{holds}(S, \text{Read}, O_2), \text{holds}(S, \text{Read}, O_1) \}\.
\]

\(^3\)Note that Woo and Lam's language is a restricted first order language as no quantifiers are considered in their language.

\(^4\)Note that Woo and Lam’s \( P^− \) and \( N^− \) which record those rights that cannot be explicitly granted and denied respectively cannot be expressed as fact expressions in our language. But it would not be difficult to extend our language to capture this capability. For instance, we can add one more predicate such as \( \text{holds}^−(s, r, o) \) in the language to achieve this purpose.
These three rules in this policy base represent the following knowledge respectively: (i) if it is consistent to assume that subject $S$ cannot write to object $O_1$ and can write to object $O_2$, then it is believed that $S$ can write to object $O_3$; (ii) if $S$ can read $O_1$, then $S$ can read $O_2$ as well; (iii) subject $S$ has Read right to object $O_1$.

The semantics of a policy base in Woo and Lam’s approach is based on a concept called extension, which is actually an analogy of the fixed point semantics of Reiter’s default logic [69]. Instead of describing Woo and Lam’s original definition of extension, here we give its alternative which is based on Reiter’s definition [69] but is semantically equivalent to Woo and Lam’s original definition\(^5\).

**Definition 11** Let $B$ be a policy base and $E$ be a set of formulas. We say that $E$ is a $r$-extension (i.e. Reiter’s extension) for $B$ if it is one of the smallest deductively closed sets of formulas $E'$ satisfying the condition: For any rule with the form $f : f' / g$ from $B$, if $f \in E'$ and $\neg f' \notin E$, then $g \in E'$. A set of ground literals $\Sigma$ is called a $wl$-extension (i.e. Woo-Lam’s extension) of $B$ if and only if there exists a $r$-extension $E$ of $B$ such that $\Sigma$ is the smallest set satisfying $F \in \Sigma$ for every formula $f \in E$. A ground literal $l$ is derivable from $B$, denoted as $B \vdash_{wl} l$, if and only if $l$ is in every $wl$-extension of $B$.

**Example 25** Example 24 continued. According to Definition 11, it is not difficult to see that the policy base $B$ in Example 24 has a unique $wl$-extension:

$$\Sigma = \{\text{holds}(S, \text{Read}, O_1), \text{holds}(S, \text{Read}, O_2), \text{holds}(S, \text{Write}, O_3)\}.$$  

Clearly, we also have $B \vdash_{wl} \text{holds}(S, \text{Read}, O_2)$ and $B \vdash_{wl} \text{holds}(S, \text{Write}, O_3)$.

Obviously, as Woo and Lam’s policy base is based on Reiter’s default logic, a policy base may have one or more than one or even no extension at all [91]\(^6\).

---

\(^5\)A detailed proof to show the equivalence between Definition 6 and Woo Lam’s original definition on extension of a policy base is given in appendix.

\(^6\)This is similar to the situation of the state for a domain description in our language $L^{ud}$.
4.3.2 Comparison

Now we can explore the connection between Woo and Lam's authorization specification and our domain description in language $L^{nd}$. Intuitively, as Woo and Lam's method only deals with the specification of authorizations, we have to restrict our domain description $D$ to the case of not including any transformation propositions $T$ causes $\phi$ if $\psi$.

A domain description $D$ without transformation proposition is called static domain description. That is, $D$ only includes the following kinds of propositions:

* initial policy proposition: initially $\phi$,

* default proposition: $\phi$ implies $\psi$ with absence $\gamma$, or its special forms:
  - $\phi$ provokes $\psi$, and
  - always $\psi$.

Note that here all occurrences of $\phi, \psi$ and $\gamma$ are ground fact expressions in our language because we assume that any fact expression including variables is viewed as a set of all its ground instances.

From our previous discussion, we can assume that Woo and Lam's policy base $B$ only includes ground rules$^7$. Furthermore, for each rule $f : f'/g$ in the policy base $B$, we restrict $f, f'$ and $g$ to be in the form of ground fact expressions in our language $L^{nd}$. This means that $f, f'$ and $g$ should only be of the form $[\neg \text{holds}(S_1, R_1, O_1) \land \cdots \land [\neg \text{holds}(S_k, R_k, O_k)]$, where notation $[\neg]$ means that the negation sign "$\neg$" may or may not occur.

Under these assumptions, we have the following results to illustrate the relationship between these two approaches. Proofs of the following Theorem 1 and Theorem 2 are straightforward but quite tedious and are referred to the Appendix.

**Proposition 2** Let $D$ be a static domain description in language $L^{nd}$. We specify a Woo-Lam policy base $B$ in terms of $D$ as follows:

(i) for each initial policy proposition initially $\phi$ in $D$, we specify a rule $\phi$ in $B$;

---

$^7$Recall that a rule including ordinary variables can be viewed as a set of all instances of this rule.
(ii) for each default proposition \( \phi \) implies \( \psi \) with absence \( \gamma \) in \( D \), we specify a rule \( \phi : \neg \gamma / \psi \) in \( B \);

(iii) for each special default proposition \( \phi \) provokes \( \psi \), we specify a rule \( \phi \Rightarrow \psi \) in \( B \);

(iv) for each special default proposition always \( \psi \), we specify a rule \( \psi \) in \( B \).

Then for any ground fact expression \( \psi \), \( D \models_{L} \text{initially } \psi \) if and only if \( B \models_{wl} \psi \).

**Proposition 3** Let \( B \) be a Woo-Lam policy base where for each rule \( f : f' / g \) in \( B \), \( f \), \( f' \) and \( g \) are expressed in the form of ground fact expressions in language \( L^{ad} \). We specify a static domain description \( D \) in terms of \( B \) as follows:

(i) for each rule \( f : f' / g \) in \( B \), we specify a default proposition \( f \) implies \( g \) with absence
\[ \neg f_{1}' \land \cdots \land f_{k}' \text{, where } f' \equiv f_{1}' \land \cdots \land f_{k}' ; \]

(ii) for each rule \( g \) in \( B \), we specify an initial policy proposition initially \( g \);

(iii) for each rule \( f \Rightarrow g \) in \( B \), we specify a special default proposition \( f \) provokes \( g \) in \( D \).

Then for each ground fact expression \( \psi \), \( B \models_{wl} \psi \) if and only if \( D \models_{L} \text{initially } \psi \).

Propositions 2 and 3 reveal an important fact that under some conditions, given a domain description \( D \), there exists an equivalent Woo-Lam policy base \( B \), and vice versa. In this sense, our system has the same capability to specify authorizations as Woo and Lam's system. But the fundamental difference between our approach and Woo and Lam's and other related approaches is that our approach can specify the sequence of transformations of authorization, which is essential in real world applications.

### 4.4 Conclusions of The Sequence Transformation Model

In this chapter, we have proposed two higher level languages \( L^{ \prime } \) and \( L^{ad} \) to specify sequences of authorization transformations. We have shown that the language \( L^{ \prime } \) has a simple syntax and semantics, but is expressive enough to represent some well known access policy examples involving sequences of authorization transformations. Using
the definition of policy proposition, we are able to compute both the final state after performing a sequence of transformations as well as any intermediate state within the sequence of transformations. The language can represent incomplete information and allow denials to be represented explicitly. The entailment relation $\models_L$ of $L^i$ has a nonmonotonic property with respect to both policy propositions and transformation propositions. Language $L^{id}$ is an augmentation of $L^i$ which includes default propositions within the domain description of authorization policies. We have also shown that $L^{id}$ has more powerful expressiveness than $L^i$ in the sense that constraints, causal and inherited authorizations as well as general default authorizations can be specified. We have discussed the relationships between our approach and Woo and Lam's in detail.
Object Oriented Database with Authorization Policy

The increasing development of information technology in recent years has led to the widespread use of computer systems in a broad variety of organizations, institutions and applications. This has meant that more data than ever before is now stored and manipulated by computer systems, or database systems. This also has meant an increasing demand for the security of the data been stored and manipulated. Object oriented database system is one of the active trends of today’s database systems, hence data protection in object oriented databases has became an active research area recently. Authorization specification in object oriented databases is being increasingly investigated recently by many researchers [35, 36, 40, 57, 59]. Some of the work proposed a security model for object oriented databases, some presented a formal specification of an authorization model for object oriented database systems, also some design approaches has been proposed for securing object oriented database systems. However, most of the work to date suffers from a lack of formal logic semantics to characterize different types of inheritance properties of authorization policies among complex data objects. For instance, it is not clear how to provide a formal semantics of inheritance property of access control rights in the context of object oriented databases. Furthermore, it is also difficult to formally reason about authorizations associated with different objects in databases.

The purpose of this chapter is to address this issue from a formal logic point of view. In particular, we propose a logic language that has a clear and declarative se-
mantics to specify the structural features of object oriented databases and authorizations associated with complex data objects in databases. Our formalization characterizes the model-theoretic semantics of object oriented databases and authorizations associated with them. A direct advantage of this approach is that we can formally specify and reason about authorizations on data objects without losing inheritance and abstraction features of object oriented databases. We first propose a logic language for specifying object oriented databases. This language has a high level syntax and its semantics shares some features of Kifer et al's F-logic [53]. We then extend this language based on some features of our previous high level formal language $L^{rd}$ for authorization specification. The semantics of the resulting language is defined in such a way that both the inheritance property in an object oriented database (OODB) and authorization rules among different data objects can be formally justified.

5.1 Background

Here, we briefly review some related work and discuss the different approaches used in object oriented database security.

In [35], a security model for object oriented databases was proposed. This model consists of a set of policies, a structure for authorization rules and an algorithm to evaluate access requests. The database is composed of objects that include a collection of facts and a collection of relevant rules. An object binds knowledge rules to database facts. The database is specified by the OSAM* [85, 86] model, in which the generic properties are defined through a generalization association and the set of attributes of a class is defined by an aggregation association. Derived classes(subclasses) are viewed as generic. Class inheritance properties suggest that access to some attributes of a class also implies access to the corresponding values in its subclass. Generally, there are three types of access policies:

1. A user who has access to a class is allowed to have similar access in the corresponding subclasses to the attributes inherited from that class.

2. Access to a complete class implies access to the attributes defined in that class.
as well as to attributes inherited from a higher class.

3. An attribute defined for a subclass is not accessible by accessing any of its superclasses.

Additional policies are necessary to consider predicates and multiple inheritance. In general, an authorization rule is a tuple \((U, A, O, p, f)\) which defines that subject or user \(U\) has authorization of type \(A\) to those occurrences of object class \(O\) for which predicate \(p\) is true. User \(U\) can grant the access right \((O, A)\) if the copy flag \(f\) is true. This model has been often used to describe authorization systems for relational databases. There is a more specific version of these rules for authorization systems of object oriented databases as follows.

An authorization rule is a triple \((U, A, AO)\) where \(U\) is a user or user group, \(A\) is an access type or set of access types, and \(AO\) is the set of attributes of the object to be accessed. A rule can either refer to \(AO\) as a whole or to its individual components.

Access validation occurs by extracting a data request from a user query or from an existing program. This request has a structure \((U', A', O')\) where \(U'\) is the subject making the request, \(O'\) is the requested object, and \(A'\) is the requested access type. This request is validated against the authorization rules to decide if the request should be granted or not.

The placement of authorization rules affects considerably the way of validating access requests. Authorization rules can be placed at special classes, at the class to which they refer, or propagated throughout the structure hierarchy.

The access request evaluation algorithm is based on certain placement rules and that the security graph is a tree. For each node in the query graph the algorithm checks for authorization in the corresponding node in the query security graph. If it finds a rule authorizing all the requested access, it returns. Otherwise, it looks for more general rules in the ancestor of the current node. If still not fully authorized, it looks for more restrictive rules in its descendant subtrees using a certain search procedure, for example, depth-first search procedure.

In [36], a formal specification of authorization for object oriented databases has been presented. It is specified in language \(Z\). \(Z\) is a language as well as a style for
expressing formal specifications for computing systems [83]. In [36], it is assumed that the database and the authorization rules have already been defined, hence the work concentrates on the description of the structure and the properties the rules should satisfy to represent a secure object oriented database system.

Generally, an access rule has the structure:

\[(subject, access\_type, data\_object)\]

where \textit{subject} is the identifier of a user accessing the database; \textit{access\_type} is the intended mode of access, for example, read, write, scan, delete; and \textit{data\_object} is the data item whose access is controlled by this rule.

Access rules can be explicit or implicit. Explicit access rules are written by security administrators and are stored in some kind of data structure; implicit access rules are not stored, they are determined at user request validation time by the authorization policies, the data structure, and run time information. The introduction of implicit access rules saves the writing of a large number of rules and simplifies the authorization view for the security administrator. Access rules can also be negative, which means that the rule specifies a subject cannot access certain object. Negative access rules can be used to specify exceptions to granted access rights.

In Z specification, \textit{basic} types are used to build more complex types refereed to as \textit{derived} types. Using Z, the following basic types for a database are declared:

\[\text{[USER, CLASS\_NAME, ATTRIBUTE, METHOD]}\]

where \textit{USER} is a set of user identifiers; \textit{CLASS\_NAME} is a set of class names; \textit{ATTRIBUTE} is a set of attributes; and \textit{METHOD} is a set of methods.

A key point of this specification is the definition of a set \textit{pos\_rules} which contains all the authorized accesses of the database system. This set denotes a virtual existence because there may not be a corresponding data structure in an actual system and some of the rules may only be implicit. The derived type \textit{Auth\_rule} specifies the format of the elements in \textit{pos\_rules} while the security policies specify the relative properties of
each element in this set. When an access rule is issued by the security administrator, a corresponding explicit authorization rule and a set of implicit rules are added into the set pos.rules; when an explicit rule is deleted, the corresponding implicit rules are also deleted from the set pos.rules. The security policies specify the relationship between the defined access rules and the implied ones. It is necessary for each of its access rule to satisfy these specifications in order to secure the database system.

5.2 A Formal Specification for Object Oriented Databases

Now we propose a language \( L^o \) for specifying an object oriented database and its syntax and semantics [14, 15].

5.2.1 Syntax of \( L^o \)

We use a language \( L^o \) to specify object oriented databases. The vocabulary of \( L^o \) consists of:

1. A finite set of object variables \( OV = \{ o, o_1, o_2, \ldots \} \) and a finite set of object constants \( OC = \{ O, O_1, O_2, \ldots \} \). We will simply name \( O = OV \cup OC \) as object set.

2. A finite set \( F \) of function symbols as methods where each \( f \in F \) takes objects as arguments and maps to an object or a set of objects\(^1\). The set of object constructors are included in \( F \) since object constructors are a special kind of method.

3. Auxiliary symbols \( \Rightarrow \) and \( \rightarrow \).

An object proposition is an expression of the form

\[
\text{O has method } f_1(\cdots) \Rightarrow \Pi_1, \\
\ldots, \\
f_m(\cdots) \Rightarrow \Pi_m, \\
f_{m+1}(\cdots) \Rightarrow \Pi_{m+1},
\]

\(^1\)For the sake of simplicity, we also call an element in \( O \) as an object.

THESIS
\[ \ldots, \]
\[ f_n(\cdots) \mapsto \Pi_n. \]  
(5.1)

In (5.1) \( O \) is an object from the set \( O \cup O \mathcal{C} \) and \( f_1, \ldots, f_m, \ldots, f_n \) are function symbols (as object constructors). Each function symbol \( f \) takes objects as arguments and maps to some \( \Pi \) that is an object or a set of objects.

In an object proposition, a method of the form \( f(\cdots) \mapsto \Pi \) indicates that the arguments of \( f \) represent the types of actual objects that should be taken in any instance of this object proposition, and \( f \) returns a set of types of the resulting object/objects. It is important to note that in our context, the type of an object is also an object. We should also mention that as a special case, if a function symbol is 0-arity, then this function just presents a simple attribute of the associated object. On the other hand, a method of the form \( f(\cdots) \mapsto \Pi \) indicates that \( f \) takes actual objects as arguments and returns an actual object or a set of objects. For example, the following is an object description about a PhD student:

\[
\text{PhDStd has method name } \mapsto \text{String},
\]
\[
\text{area(Staff) } \mapsto \text{String},
\]
\[
\text{firstdegree } \mapsto \text{'Bachelor'},
\]

where \( \text{name } \mapsto \text{String} \) represents that the type of name is a string, and method \( \text{area} \) takes type \( \text{Staff} \) (i.e. another object, e.g. the student's supervisor) as a parameter and returns a type of string to indicate the research field, while \( \text{firstdegree } \mapsto \text{'Bachelor'} \) simply expresses that every PhD student should hold a Bachelor degree. An object proposition is called ground if there is no object variable occurrence in it.

An \textit{isa} proposition of \( L^O \) is an expression of one of the following two forms:

\[
O \text{ is a member of } C, \]
(5.2)

\[
O \text{ is a subclass of } C, \]
(5.3)

\text{THESIS}
where \( O \) and \( C \) are objects from \( O \), i.e., \( O \) and \( C \) may be object constants or variables. Clearly, *isa* propositions (5.2) and (5.3) explicitly represent the hierarchy relation between two objects. An isa proposition without containing any object variables is called a *ground isa* proposition.

We call an object or isa proposition a *data proposition*. A data proposition is *ground* if there is no object variable occurrence in it. If the detail of a data proposition is not interested in the context, we usually use notation \( \phi \) to denote the data proposition. We assume that any variable occurrence in a data proposition is universally quantified.

A *constraint proposition* is an expression of the form

\[
\phi \text{ if } \phi_1, \cdots, \phi_k, \tag{5.4}
\]

while \( \phi, \phi_1, \cdots, \phi_k \) are data propositions. A constraint proposition represents some relationship among different data objects. The intuitive meaning of (5.4) is that if \( \phi_1, \cdots, \phi_k \) are present in the current database state, then so is \( \phi \). With this kind of proposition, we can represent some useful deductive rules of the domain in our database. This will be illustrated later using some examples. A *database proposition* is an object proposition, isa proposition, or constraint proposition.

Now we can formally define our object oriented database as follows.

**Definition 12** An object oriented database (or simply called a database) \( \Sigma \) is a triplet \( (\Gamma, \Delta, \Omega) \), where \( \Gamma \) is a finite set of ground object propositions, \( \Delta \) is a finite set of ground isa propositions, and \( \Omega \) is a finite set of constraint propositions.

**Example 26** Consider a simplified domain about research people in a computer science department. The structure of such domain is illustrated in Figure 5.1. In Figure 5.1, line arrows indicate subclass relations while dotted line arrows indicate membership relations in the database.

Using language \( L^0 \), the database \( \Sigma = (\Gamma, \Delta, \Omega) \) is specified as follows:

1. the set of ground object propositions \( \Gamma \) consists of:

   \[
   \text{ResPeople has method name } \Rightarrow \text{String},
   \]

   THESIS
age \Rightarrow \text{Integer}, \hspace{2cm} \text{(5.5)}

\text{firstdegree} \Rightarrow \text{'Bachelor'}, \hspace{2cm} \text{(5.5)}

\text{Postgraduate has method id} \Rightarrow \text{Integer},

\text{degree} \Rightarrow \text{String}, \hspace{2cm} \text{(5.6)}

\text{area(Staff)} \Rightarrow \text{String}, \hspace{2cm} \text{(5.6)}

\text{Staff has method research} \Rightarrow \{\text{String, }, \ldots, \text{String}\},

\text{teaching} \Rightarrow \{\text{String, }, \ldots, \text{String}\}, \hspace{2cm} \text{(5.7)}

\text{salary} \Rightarrow \text{String}, \hspace{2cm} \text{(5.7)}

\text{Tutor has method topsalary} \Rightarrow \text{'$45,000'}, \hspace{2cm} \text{(5.8)}

\text{Lecturer has method topsalary} \Rightarrow \text{'$58,000'}, \hspace{2cm} \text{(5.9)}

\text{Professor has method topsalary} \Rightarrow \text{'$85,000'}, \hspace{2cm} \text{(5.10)}

\text{Tom has method name} \Rightarrow \text{'Tom'},

\text{age} \Rightarrow 21,

\text{id} \Rightarrow 96007,

\text{degree} \Rightarrow \text{'Master'},

\text{THESIS}
\begin{equation}
\text{area(Peter)} \mapsto \text{'Database'}, \tag{5.11}
\end{equation}

\textit{Sue} has method name \mapsto \text{'Sue'},

\begin{align*}
\text{age} & \mapsto 24, \\
\text{id} & \mapsto 95012, \\
\text{degree} & \mapsto \text{'PhD'},
\end{align*}

\text{area(James)} \mapsto \text{'Security'}, \\
\text{salary} & \mapsto \text{'$38,000'}, \tag{5.12}
\end{equation}

\textit{Faye} has method name \mapsto \text{'Faye'},

\begin{align*}
\text{age} & \mapsto 30, \\
\text{research} & \mapsto \{\text{'Networking'}\}, \\
\text{teaching} & \mapsto \{\text{'CS1', 'CS2'}\}, \\
\text{salary} & \mapsto \text{'$43,000'}, \tag{5.13}
\end{align*}

\textit{James} has method name \mapsto \text{'James'},

\begin{align*}
\text{age} & \mapsto 42, \\
\text{research} & \mapsto \{\text{'Security', 'Networking'}\}, \\
\text{Teaching} & \mapsto \{\text{'Networking'}\}, \\
\text{salary} & \mapsto \text{'$78,000'}, \tag{5.14}
\end{align*}

(2) the set of ground isa propositions $\Delta$ consists of:

\begin{equation}
\textit{Tom} \text{ isa member of Postgraduate}, \tag{5.15}
\end{equation}

\begin{equation}
\textit{Sue} \text{ isa member of Postgraduate}, \tag{5.16}
\end{equation}

\begin{equation}
\textit{Sue} \text{ isa member of Tutor}, \tag{5.17}
\end{equation}

\begin{equation}
\textit{Faye} \text{ isa member of Tutor}, \tag{5.18}
\end{equation}

\begin{equation}
\textit{Ann} \text{ isa member of Lecturer}, \tag{5.19}
\end{equation}

\begin{equation}
\textit{James} \text{ isa member of Professor}, \tag{5.20}
\end{equation}

\begin{equation}
\text{Postgraduate} \text{ isa subclass of ResPeople}, \tag{5.21}
\end{equation}

\begin{equation}
\text{Staff} \text{ isa subclass of ResPeople}, \tag{5.22}
\end{equation}

\textbf{THESIS}
Tutor isa subclass of Staff,  \hspace{1cm} (5.23)

Lecturer isa subclass of Staff,  \hspace{1cm} (5.24)

Professor isa subclass of Staff,  \hspace{1cm} (5.25)

and (3) \( \Omega \) consists of two constraint propositions:

\[
y \text{ isa member of } Staff \\
\text{if } x \text{ isa member of } Postgraduate, \\
x \text{ has method } area(y) \mapsto z,  \hspace{1cm} (5.26) \\
y \text{ has method } research \mapsto \{...,z,...\} \\
\text{if } x \text{ isa member of } Postgraduate, \\
x \text{ has method } area(y) \mapsto z,
\]

where \( x, y \) and \( z \) are object variables, and notation \( \{...,z,...\} \) means that set \( \{...,z,...\} \) includes element \( z \) which is of interest.

In database \( \Sigma \), we assume that objects Integer and String are primitive object constants and do not require explicit descriptions. That is, we omit isa propositions like \( 21 \text{ isa member of } Integer \) and 'Tom' \( \text{ isa member of } String \) from our database.

Our database also presents necessary inheritance properties among different objects. For instance, since object Tom is a member of Postgraduate, it should inherit method firstdegree \( \mapsto 'Bachelor' \). The semantics of inheritance will be described in next subsection.

Finally, in \( \Sigma, \Gamma \) and \( \Delta \) represent explicit data object descriptions and hierarchical relations among these objects, while \( \Omega \) describes constraints of the domain which characterize some implicit data objects and their properties. By using these rules in \( \Omega \) and facts in \( \Gamma \cup \Delta \), we actually can derive new data objects with some clear properties. For instance, in the above database, we do not give explicit description about object Peter. But from propositions (5.11) and (5.15) about object Tom, we can derive the facts that Peter is a member of Staff and has a research field 'Database'.

\[\blacksquare\]

**THESIS**
5.2.2 Semantics of $\mathcal{L}^o$

In this subsection, we define the semantics of our database language $\mathcal{L}^o$ using an approach similar to that employed in classical logic.

**Definition 13** Let $\mathcal{L}^o$ be the object oriented database language that we defined earlier. A structure of $\mathcal{L}^o$ is a tuple $I = (U, \mathcal{F}, \subseteq_U, \in_U, \Rightarrow_I, \mapsto_I)$, where

1. $U$ is a nonempty set called the universe of $I$ that represents the set of all actual objects in the domain.

2. For each $n$-ary function symbol $f$ in $\mathcal{F}$, there exists a $n$-ary function $f_i: U^n \rightarrow U$ in $\mathcal{F}_i$. For $n = 0$, $f_i$ is an element of $U$.

3. $\subseteq_U$ is a partial ordering on $U$ and $\in_U$ is a binary relation on $U$. We require that if $a \in_U b$ and $b \subseteq_U c$, then $a \in_U c$.

4. For symbol $\Rightarrow$ in $\mathcal{L}^o$, $\mapsto_I$ is a map: $\Rightarrow_I: U \rightarrow (H_0, \cdots, H_i, \cdots)$, where each $H_i$ is a set of $(i + 1)$-ary anti-monotonic functions with respect to ordering $\subseteq_U$\(^2\) such that for every $h_i \in H_i$,

   $$h_i : U^{i+1} \rightarrow \mathcal{P}(U),$$

   in which $\mathcal{P}(U)$ is the set of all upward-closed subsets of $U$ with respect to $\subseteq_U$.

5. For symbol $\mapsto$ in $\mathcal{L}^o$, $\Rightarrow_I$ is a map: $U \rightarrow (G_0, \cdots, G_k, \cdots)$, where $G_k$ is a set of $(k+1)$-ary functions such that for every $g_k \in G_k$,

   $$g_k : U^{k+1} \rightarrow U \cup \mathcal{P}(U),$$

   in which $\mathcal{P}(U)$ is the set of all subsets of $U$.

Now let us look at this definition more closely. In a structure $I$, $U$ represents all possible actual objects in the domain. That is, each element in $U$ is a real object in our real world. A function symbol (i.e. object constructor) $f$ in $\mathcal{F}$ corresponds to a

\(^2\)That is, if $u, v \in U^{i+1}$ and $v \subseteq_U u$, then $h_i(v) \supseteq h_i(u)$.
function $f_i$ in $\mathcal{F}_i$. Note that $f$ takes object constants or variables in $O$ as arguments while $f_i$ takes elements in $U$ as arguments and returns an element of $U$. The objective of ordering $\subseteq_U$ is to represent the semantics of isa subclass proposition in $\mathcal{L}^\rho$. For example, $a \subseteq_U b$ is the counterpart of proposition $A \text{ isa subclass of } B$, where $A$ and $B$ are elements in $O$ (i.e. object constants or variables) and are mapped to $a$ and $b$, which are the elements of $U$ respectively. We also write $a \subset_U b$ if $a \subseteq_U b$ but $a \neq b$. The semantics of isa membership proposition in $\mathcal{L}^\rho$ is provided by $\in_U$ in $I$ in a similar way.

The semantics of $\Rightarrow$, however, is not quite straightforward. As we mentioned earlier, a method of the form $f(\ldots) \Rightarrow \Pi$ actually defines the function type of $f$. That is, $f$ takes objects that represent types of actual objects and returns an object (or a set of objects) that indicates the type (or types) of resulting actual object (or objects). Suppose that $f$ is a $i$-ary function. Then the semantics of $\Rightarrow$ is provided by mapping $\Rightarrow^{(i)}_I$ which maps the resulting object represented by $f(\ldots)$ to a $(i+1)$-ary function $h_i : U^{i+1} \rightarrow \mathcal{P}(U)$, where the first $i$th arguments in $U^{i+1}$ are objects that corresponds to the $i$ arguments taken by $f$, and the $(i+1)$-th argument in $U^{i+1}$ is the object that corresponds to the object associated with function $f(\ldots)$ in the proposition (we also call it the host object of $f$). In $f(\ldots) \Rightarrow \Pi$, $\Pi$ denotes the type/types of resulting object/objects for which we use a subset of $U$ to represent all the possible actual objects that have type/types indicated by $\Pi$.

It is important to note that we require the subset of $U$ to be upward-closed with respect to ordering $\subseteq_U$. A subset $V$ of $U$ is upward-closed if for $v \in V$ and $v \subseteq_U v'$, then $v' \in V$. The purpose of this requirement is that if $V$ is viewed as a set of classes, upward closure ensures that for each class $v \in V$, $V$ also contains all the superclasses of $v$, which will guarantee the proper inheritance property of types. The same reason is used to define the anti-monotonicity of each $H_i$.

A similar explanation for $\Rightarrow_I$ can be given for the semantics of $\Rightarrow$. We now show that $\Rightarrow_I$ actually provides the type of the corresponding $\Rightarrow_I$.

To simplify our formalization, we will use Herbrand universe in any structures of $\mathcal{L}^\rho$. That is, the Herbrand universe $U_H$ is formed from the set of all object constants in

$^3$In expression $\Rightarrow_I : U \rightarrow (H_0, \ldots, H_i, \ldots)$, we use notation $\Rightarrow^{(i)}_I$ to denote the $i$-th component of $\Rightarrow_I$, i.e. $\Rightarrow^{(i)}_I : U \rightarrow H_i$. 

THESIS
OC and the objects built by function symbols on these object constants.

Definition 14 Let \( I = (U_H, \mathcal{F}_I, \subseteq_I, \in_I, \Rightarrow_I, \rightarrow_I) \) be a structure. We define entailment relation \( \models \) as follows:

1. For a ground isa membership proposition, \( I \models O \text{ isa member of } C \) if \( O \in_{U_H} C \), and for an isa subclass proposition, \( I \models O \text{ isa subclass of } C \) if \( O \subseteq_{U_H} C \).

2. For a ground object proposition,

\[
I \models O \text{ has method } f_1(\cdots) \Rightarrow \Pi_1, \\
\quad \vdots, \\
I \models f_m(\cdots) \Rightarrow \Pi_m, \\
I \models f_{m+1}(\cdots) \Rightarrow \Pi_{m+1}, \\
\quad \vdots, \\
I \models f_n(\cdots) \Rightarrow \Pi_n
\]

if the following conditions hold:

- For each \( f(O_1, \cdots, O_p) \) where \( f \) is in \( \{f_1, \cdots, f_m\} \),

\[
\Rightarrow^{(p)} (O')(O_1, \cdots, O_p, O) = \Pi, \text{ where } f_i(O_1, \cdots, O_p) = O', \text{ and}
\]

- For each \( f'(O'_1, \cdots, O'_q) \) where \( f \) is in \( \{f_{m+1}, \cdots, f_n\} \),

\[
\Rightarrow^{(q)} (O')(O'_1, \cdots, O'_q, O) = \Pi', \text{ where } f_j(O'_1, \cdots, O'_q) = O'.
\]

3. For a ground constraint proposition, \( I \models \phi \) if \( I \models \phi_1, \cdots, I \models \phi_k \) implies \( I \models \phi \).

4. For any proposition \( \psi \) including object variables, \( I \models \psi \) if for every instance \( \phi \) of \( \psi \) (i.e. \( \phi \) is obtained from \( \psi \) by substituting each variable in \( \psi \) with some element of \( U_H \)), \( I \models \phi \).

Now we can formally define the model of a database \( \Sigma \) as follows:

Definition 15 A structure \( M \) of \( \mathcal{L}^\Sigma \) is a model of a database \( \Sigma = (\Gamma, \Delta, \Omega) \) if

\(^4\text{Note that under Herbrand universe } U_H, \text{ an object constant is mapped to itself in } U_H.\)
1. For each proposition $\psi$ in $\Gamma \cup \Delta \cup \Omega$, $M \models \psi$.

2. For each object proposition $\phi$, if $M \models \phi$, then $M \models \phi'$ where $\phi'$ is obtained from $\phi$ by omitting some methods of $\phi$.

3. For any isa proposition $O$ isa member of $C$ and object propositions $C$ has method $f(\cdots) \Rightarrow \Pi$ and $C$ has method $f(\cdots) \Rightarrow \Pi$, $M \models O$ isa member of $C$ and $M \models C$ has method $f(\cdots) \Rightarrow \Pi$ imply $M \models O$ has method $f(\cdots) \Rightarrow \Pi$.

4. for any isa proposition $O$ isa subclass of $C$ and object proposition $C$ has method $f(\cdots) \Rightarrow \Pi$, $M \models O$ isa subclass of $C$ and $M \models C$ has method $f(\cdots) \Rightarrow \Pi$ imply $M \models O$ has method $f(\cdots) \Rightarrow \Pi$.

Condition 1 in the above definition is the basic requirement for a model. Condition 2 allows us to partially represent an object with only those methods that are of interest in a given context. Condition 3 is a restriction to guarantee necessary inheritance of membership, while Condition 4 is needed for the purpose of subclass value inheritance.

Let $\Sigma$ be a database and $\phi$ be a database proposition. If for every model $M$ of $\Sigma$, $M \models \phi$, we also call that $\phi$ is entailed by $\Sigma$, denoted as $\Sigma \models \phi$.

### 5.2.3 Inheritance Properties

In this subsection, we explore inheritance properties of object oriented databases under our framework. First, from Definitions 13 and 14, it is easy to prove the method projection and method aggregation with respect to an object.

**Proposition 4 (Method Projection)** Let $I$ be a structure of $L^0$ and $\phi$ a ground object propo
sition with the form:

\[ O \text{ has method } f_1(\cdots) \rightarrow \Pi_1, \]
\[ \cdots, \]
\[ f_n(\cdots) \rightarrow \Pi_n, \]

where \( \rightarrow \) is a symbol for \( \Rightarrow \) or \( \rightarrow \). Then \( I \models \phi \) implies that
\( I \models O \text{ has method } f_1(\cdots) \rightarrow \Pi_1, \cdots, I \models O \text{ has method } f_n(\cdots) \rightarrow \Pi_n. \)

Proposition 5 (Method Aggregation) Let \( I \) be a structure of \( \mathcal{L}^o \) and \( \phi_1 \) and \( \phi_2 \) be ground object propositions with forms:

\[ O \text{ has method } f_1(\cdots) \rightarrow \Pi_1, \]
\[ \cdots, \]
\[ f_n(\cdots) \rightarrow \Pi_n, \]

and

\[ O \text{ has method } f'_1(\cdots) \rightarrow \Pi'_1, \]
\[ \cdots, \]
\[ f'_m(\cdots) \rightarrow \Pi'_m, \]

respectively, where \( \rightarrow \) is a symbol for \( \Rightarrow \) or \( \rightarrow \). Then \( I \models \phi_1 \) and \( I \models \phi_2 \) imply that \( I \models \phi \), where \( \phi \) is a ground object proposition of the form:

\[ O \text{ has method } f_1(\cdots) \rightarrow \Pi_1, \]
\[ \cdots, \]
\[ f_n(\cdots) \rightarrow \Pi_n, \]
\[ f'_1(\cdots) \rightarrow \Pi'_1, \]
\[ \cdots, \]
\[ f'_m(\cdots) \rightarrow \Pi'_m. \]

THESIS
Although the above two properties are quite obvious, they are actually useful for our reasoning about object methods. Method projection allows us to focus on some particular method of an object when needed, while method aggregation allows us to get an object proposition associated with all of its methods.

**Theorem 1 (Subclass Inclusion)** If $I \models O$ isa member of $C_1$, and $I \models C_1$ isa subclass of $C_2$, then $I \models O$ isa member of $C_2$.

**Theorem 2 (Subclass Type Inheritance)** The following results hold.

(i) If $I \models O$ has method $f(Q_1, \cdots, Q_k) \Rightarrow S$ and $I \models R$ isa subclass of $O$, then $I \models R$ has method $f(Q_1, \cdots, Q_k) \Rightarrow S$.

(ii) If $I \models O$ has method $f(Q_1, \cdots, Q_i, \cdots, Q_k) \Rightarrow S$ and $I \models Q'_i$ isa subclass of $Q_i$, then $I \models O$ has method $f(Q_1, \cdots, Q'_i, \cdots, Q_k) \Rightarrow S$.

(iii) If $I \models O$ has method $f(\cdots) \Rightarrow S$ and $I \models S$ isa subclass of $R$, then $I \models O$ has method $f(\cdots) \Rightarrow R$.

In Theorem 2, (i) and (ii) follow directly from the anti-monotonicity constraint on mapping $\Rightarrow_I$ and (iii) follows from the upward-closure of $\Rightarrow_I$ as defined in Definition 12. The following two propositions directly follow from Definition 15.

**Proposition 6 (Subclass Value Inheritance)** If $I \models O$ isa subclass of $C$ and $I \models C$ has method $f(\cdots) \Rightarrow \Pi$, then $I \models O$ has method $f(\cdots) \Rightarrow \Pi$.

**Proposition 7 (Membership Type and Value Inheritance)** The following results hold.

(i) If $I \models O$ isa member of $C$ and $I \models C$ has method $f(\cdots) \Rightarrow \Pi$, then $I \models O$ has method $f(\cdots) \Rightarrow \Pi$.

(ii) If $I \models O$ isa member of $C$ and $I \models C$ has method $f(\cdots) \Rightarrow \Pi$, then $I \models O$ has method $f(\cdots) \Rightarrow \Pi$.

**Example 27** Let us return to the research people domain example described earlier. From previous propositions and theorems, it is easy to verify that every class and object inherits all of the type and value methods from all of its superclasses. For instance, we have the following results:

**THESIS**
\[ \Sigma \models \text{Tom has method } \text{firstdegree} \mapsto '\text{Bachelor}', \]
\[ \Sigma \models \text{Sue has method } \text{tospalary} \mapsto '$45,000', \text{ and} \]
\[ \Sigma \models \text{Tutor has method } \text{name} \Rightarrow \text{String}. \]

Also note that object Peter is only implicitly described in \( \Sigma \). But from constraint propositions (5.26) and (5.27), we can further derive
\[ \Sigma \models \text{Peter is a member of Staff}, \text{ and} \]
\[ \Sigma \models \text{Peter has method } \text{research} \mapsto \{\cdots, 'Database', \cdots\}. \]

\[ \blacksquare \]

5.3 Authorization Specification in Object Oriented Database

In this section, we will extend our formal language \( L^o \) to \( L^{oa} \) which is used in the specification of authorization in object-oriented databases. First let us consider the following requirements in the specification of access policies in object oriented databases.

1. If a subject (user) has an access right to a complete object (class), then this should imply that this subject has the same access right to every method of the object (class). That means, we should allow default authorization policy in our system. But there are some exceptions. For example, an administration officer usually can access every method (attribute) of a postgraduate. But if a postgraduate student is also a tutor, this officer should not be able to access the postgraduate’s salary record.

2. If a subject has an access right to a class, there may be a need that this subject should be generally allowed to access all of its subclasses. Again, some exceptions should be taken into account. For example, a general research officer can access all the research records of the class Staff except that of the class Professor. A similar requirement is also needed for memberships.

3. We also need to represent causal or conditional authorization policies. For instance, an access policy such as "if subject A can access object O then subject B can also access O" is very useful in certain applications. For instance, "if the
member of a team can access the Budget object, then the team leader can also access the Budget object”.

4. Sometimes it is also necessary to represent negative authorizations in our system. For instance, a subject $S$ can access all methods (attributes) of class Postgraduate except salary.

5.3.1 Syntax of $L^{oa}$

The vocabulary of $L^{oa}$ includes the vocabulary of $L^o$ together with the following additions:

1. A finite set of subject variables $SV = \{s, s_1, s_2, \ldots\}$ and a finite set of subject constants $SC = \{S, S_1, S_2, \ldots\}$. We denote $S = SV \cup SC$.

2. A finite set of access-rights variables $AV = \{r, r_1, r_2, \ldots\}$ and a finite set of access-right constants $AC = \{R, R_1, R_2, \ldots\}$. We denote $A = AV \cup AC$.

3. A ternary predicate symbol $\text{holds}$ taking arguments subject, access-right, and object/method respectively.

4. Logic connectives $\land$ and $\neg$.

In language $L^{oa}$, a fact that a subject $S$ has access right $R$ for object $O$ is represented using a ground atom $\text{holds}(S, R, O)$. A fact that $S$ has access right $R$ for object $O$’s method $f(\ldots) \leftrightarrow \Pi^6$ is represented by ground atom $\text{holds}(S, R, O|f)$.

In general, we define an access fact to be an atomic formula $\text{holds}(s, r, o)$ (or $\text{holds}(s, r, o|f)$, where $o|f$ indicates a method associated with object $o$) or its negation. A ground access fact is an access fact without any variable occurrence. We view $\neg \neg F$ as $F$. An access fact expression in $L^{oa}$ is defined as follows: (i) each access fact is an access fact expression; (ii) if $\psi$ is an access fact expression and $\phi$ is an isa or object proposition, then $\psi \land \phi$ is an access fact expression; (iii) if $\psi$ and $\phi$ are access fact expressions, then $\psi \land \phi$ is an access fact expression. A ground fact expression is a fact expression with

\[ \text{Recall that symbol } \leftrightarrow \text{ indicates symbol } \Rightarrow \text{ or } \Rightarrow. \]

THESIS
no variable occurrence in it. An access fact expression is pure if it does not have an isa proposition occurrence in it.

Based on the above definition, the following are access fact expressions:

\( \text{holds}(S, R, O) \land O \text{ isa subclass of } C, \neg \text{holds}(S, R, o) \land o \text{ isa member of } C \), where \( o \) is an object variable.

Now we are ready to define propositions in language \( \mathcal{L}^{oa} \). Firstly, \( \mathcal{L}^{oa} \) has the same types of database propositions as \( \mathcal{L}^o \), i.e. object proposition, isa proposition and constraint proposition. It also includes the following additional type of access proposition:

\[
\psi \text{ implies } \phi \text{ with absence } \gamma, \tag{5.30}
\]

where \( \psi \) is an access fact expression, and \( \phi \) and \( \gamma \) are pure access fact expressions. Note that \( \psi, \phi \) and \( \gamma \) may contain variables. In this case, as before, (5.30) will be treated as a set of access propositions obtained by replacing \( \psi, \phi \) and \( \gamma \) with their ground instances respectively.

As an example, consider the following access proposition

\[
\text{holds}(S, \text{Read}, \text{Tutor|salary}) \land \text{Sue is a member of Tutor}
\]
\[
\text{implies} \text{holds}(S, \text{Read}, \text{Sue|salary})
\]
\[
\text{with absence} \neg \text{holds}(S, \text{Read}, \text{Sue|salary}),
\]

Intuitively, this expression says that if subject \( S \) can read tutor’s salary record and Sue is a member of tutor, then \( S \) can also read Sue’s salary record if the fact that \( S \) cannot read Sue’s salary does not currently hold.

A special form of access proposition (5.30) occurs when \( \gamma \) is empty. In this case, we can rewrite (5.30) as

\[
\psi \text{ provokes } \phi, \tag{5.31}
\]

THESIS
which is viewed as a causal or conditional relation between \( \psi \) and \( \phi \). For instance, we may have an access proposition like:

\[
\text{holds}(s, r, c) \land o \text{ is subclass of } c \\
\text{provokes } \text{holds}(s, r, o).
\]

This access proposition expresses that for any subject \( s \), access right \( r \) and objects \( o \) and \( c \), if \( s \) has access right \( r \) on \( c \) and \( o \) is a subclass of \( c \), then \( s \) also has access right \( r \) on \( o \). This is also an example of access inheritance.

On the other hand, there is also another special form of (5.30) when \( \psi \) is also empty. In this case, we can rewrite (5.30) simply as

\[
\text{always } \phi. \tag{5.32}
\]

For example, we can express a fact that the database administrator (DBA) should have any access right on any object as follows:

\[
\text{always } \text{holds}(\text{DBA}, r, o).
\]

5.3.2 Databases with Associated Authorizations

In the previous subsection, we introduced language \( L^a \) which was developed from language \( L^o \). It is clear that our access propositions (5.30), (5.31) and (5.32) provide flexibility to express different types of authorization policies on objects. However, to ensure the proper inheritance of access policies on different objects, some specific types of access policies are particularly important for database systems. The set of these kinds of authorization policies is referred to as the generic authorization scheme for databases.

Consider

\[
\text{holds}(s, r, o) \quad \text{implies} \quad \text{holds}(s, r, o|f) \\
\text{with absence} \quad \neg \text{holds}(s, r, o|f), \tag{5.33}
\]

THESIS
Intuitively, (5.33) says that if $s$ has access right $r$ on object $o$, then $s$ also has access right $r$ on each of its methods under the assumption that $\neg holds(s, r, o|f)$ is not present.

We also have the following two generic access propositions:

\[
holds(s, r, c) \land o \text{ isa subclass of } c \\
\quad \text{implies } holds(s, r, o) \\
\quad \text{with absence } \neg holds(s, r, o), \quad (5.34)
\]

and

\[
holds(s, r, c|f) \land o \text{ isa subclass of } c \\
\quad \text{implies } holds(s, r, o|f) \\
\quad \text{with absence } \neg holds(s, r, o|f). \quad (5.35)
\]

(5.34) and (5.35) guarantee the proper inheritance of access policies on subclasses.

Finally, the following two propositions ensure the membership inheritance of access policies.

\[
holds(s, r, c) \land o \text{ isa member of } c \\
\quad \text{implies } holds(s, r, o) \\
\quad \text{with absence } \neg holds(s, r, o), \quad (5.36)
\]

and

\[
holds(s, r, c|f) \land o \text{ isa member of } c \\
\quad \text{implies } holds(s, r, o|f) \\
\quad \text{with absence } \neg holds(s, r, o|f). \quad (5.37)
\]

Now we can formally define our database with associated authorizations as follows. We will refer to this kind of database as extended object oriented database.

THESIS
Definition 16 An extended object oriented database in $L^{oa}$ is a pair $\Lambda = (\Sigma, \Xi)$, where
$\Sigma = (\Gamma, \Delta, \Omega)$ is the database as defined in Definition 12, and $\Xi = GA \cup A$ is an authorization
description on $\Sigma$ where $GA$ is a collection of generic authorization propositions (5.33) - (5.37),
and $A$ is a finite set of user-defined access propositions.

5.3.3 Semantics of $L^{oa}$

Now we consider the semantics of language $L^{oa}$. We will see that the semantics of
$L^{oa}$ is not just a trivial extension of the semantics of $L^o$. In fact, to define a proper
semantics of our access proposition (5.30), we need to employ a fixed-point semantics
that shares the spirit of fixed-point semantics used for logic programs [11, 37].

Formally, a structure $I^\Lambda$ of $L^{oa}$ is a pair $(I^\Sigma, I^\Xi)$, where $I^\Sigma$ is a structure of $L^o$
as defined in Definition 12 and $I^\Xi$ is a finite set of ground literals with forms
$\text{holds}(S, R, O)$, $\text{holds}(S, R, O|f)$, $\neg\text{holds}(S, R, O)$ or $\neg\text{holds}(S, R, O|f)$. Now we can define the entailment
relation $\models_\lambda$ of $L^{oa}$.

Definition 17 Let $I^\Lambda = (I^\Sigma, I^\Xi)$ be a structure of $L^{oa}$. We define the entailment relation $\models_\lambda$
of $L^{oa}$ as follows.

1. For a database proposition $\psi$, $I^\Lambda \models_\lambda \psi$ iff $I^\Sigma \models \psi$.

2. For a pure ground access fact expression $\psi \equiv F_1 \land \cdots \land F_k$, where each $F_i$ is a ground
access fact, $I^\Lambda \models_\lambda \psi$ iff for each $i$, $F_i \in I^\Xi$.

3. For a ground access fact expression $\psi$, $I^\Lambda \models_\lambda \psi$ iff for each isa or object proposition $\phi$
occuring in $\psi$, $I^\Xi \models \phi$, and for each ground access fact $\psi'$ occurring in $\psi$, $\psi' \in I^\Xi$.

4. For an access fact expression $\psi$, $I^\Lambda \models_\lambda \psi$ iff for each ground instance $\psi'$ of $\psi$, $I^\Lambda \models_\lambda \psi'$.

Now we are in the position to formally define a model of $\Lambda = (\Sigma, \Xi)$.

Definition 18 Consider an extended database $\Lambda = (\Sigma, \Xi)$ and a structure $I^\Lambda = (I^\Sigma, I^\Xi)$. Let
$\Xi'$ be an authorization description obtained from $\Xi$ in the following way:

(i) by deleting each access proposition $\psi$ implies $\phi$ with absence $\gamma$ from $\Xi$ if for some $F_i$
in $\gamma$, $F_i \in I^{\Xi'}$;

\footnote{Recall that $\gamma \equiv F_1 \land \cdots \land F_k$ is a pure access fact expression, i.e. each $F_i$ ($1 \leq i \leq k$) is a ground access fact.}

THESIS
(ii) by translating all other access propositions \( \psi \) implies \( \phi \) with absence \( \gamma \) to the form \( \psi \) provokes \( \phi \), or to the form always \( \phi \) if \( \psi \) is empty.

**Definition 19** Consider an extended database \( \Lambda = (\Sigma, \Xi) \) and a structure \( I^\Lambda = (I^E, I^\Xi) \). Let \( \Xi' \) be an authorization description obtained from \( \Xi \) as described in Definition 18. \( I^\Lambda = (I^E, I^\Xi) \) is a model of \( \Lambda = (\Sigma, \Xi) \) if and only if

(i) \( I^\Xi \) is a model of \( \Sigma \);

(ii) \( I^\Xi \) is a smallest set satisfying the following conditions:

(a) for each access proposition always \( \phi \) in \( \Xi' \), \( I^\Lambda \models \phi \);

(b) for each access proposition of the form \( \psi \) provokes \( \phi \) in \( \Xi' \), if \( I^\Lambda \models \psi \), then \( I^\Lambda \models \phi \).

Taking default access proposition into account, it turns out that the models of an extended object oriented database may not be unique. This is shown by the following example.

**Example 28** Given an extended database \( \Lambda = (\Sigma, \Xi) \), where \( \Sigma \) is the research people database defined in Example 26, and \( \Xi = GA \cup A \), where \( A \) is a collection of the following access propositions:

\[
\text{holds}(S, \text{Own}, \text{Tom}),
\]  
\[
\text{holds}(S, \text{Own}, o) \text{ implies } \text{holds}(S, \text{Update}, o)
\]  
\[
\text{with absence } \neg\text{holds}(S, \text{Update}, o),
\]  
\[
\text{holds}(S, \text{Own}, o) \text{ implies } \neg\text{holds}(S, \text{Update}, o)
\]  
\[
\text{with absence } \text{holds}(S, \text{Update}, o).
\]  

(5.38) simply says that the user \( S \) owns object \( \text{Tom} \) in the database. (5.39) expresses that if \( S \) owns an object \( o \), then \( S \) will be able to update this object under the absence of the fact that \( S \) cannot update \( o \), whereas (5.40) states that if \( S \) owns an object \( o \), then \( S \) will not be able to update this object under the absence of the fact that \( S \) can update \( o \).
Clearly, (5.39) and (5.40) override each other. By using Definitions 17 and 18, it follows that \( \Lambda \) has two different models \( I_1^\Lambda \) and \( I_2^\Lambda \) such that

\[
I_1^\Lambda \models_{\lambda} holds(S, Update, Tom) \text{ and } \\
I_2^\Lambda \models_{\lambda} \neg holds(S, Update, Tom).
\]

Given an extended database \( \Lambda \), let \( \psi \) be a database proposition or an access fact expression. We say that \( \psi \) is entailed by \( \Lambda \), denoted as \( \Lambda \models_{\lambda} \psi \), if \( \psi \) is entailed by every model of \( \Lambda \).

5.4 Reasoning about Authorizations in Object Oriented Database

In this section, we first investigate some properties on the inheritance of authorizations on objects in database. Then we describe an application of our formalism presented in section 5.3 to an extended research people database domain in which a complex set of authorizations is associated with the database.

5.4.1 Authorization Inheritance

An extended database may have more than one model. In this case, every model actually represents one possible interpretation for the database with associated authorizations. However, the class of extended databases having unique models presents some interesting inheritance properties of authorizations with respect to subclass and membership relationships among objects in databases. An extended database is well-specified if it has a unique model.

Theorem 3 (Subclass Authorization Inheritance) Let \( \Lambda \) be a well-specified extended database and \( S, R, C \) and \( O \) be arbitrary subject constant, access right constant and object constants respectively. Then the following results hold.

(i) If \( \Lambda \models_{\lambda} holds(S, R, C) \land O \text{ is a subclass of } C \) and \( \Lambda \not\models_{\lambda} \neg holds(S, R, O) \), then \( \Lambda \models_{\lambda} holds(S, R, O) \).

THESIS
(ii) If $\Lambda \models_{\lambda} \text{holds}(S, R, C|f) \land O \text{ is subclass of } C \text{ and } \Lambda \not\models_{\lambda} \neg \text{holds}(S, R, O|f)$, then $\Lambda \models_{\lambda} \text{holds}(S, R, O|f)$.

**Theorem 4 (Membership Authorization Inheritance)** Let $\Lambda$ be a well-specified complex database, and $S$, $R$, $C$ and $O$ be arbitrary subject constant, access right constant and object constants respectively. Then the following results hold.

(i) If $\Lambda \models_{\lambda} \text{holds}(S, R, C) \land O \text{ is member of } C \text{ and } \Lambda \not\models_{\lambda} \neg \text{holds}(S, R, O)$, then $\Lambda \models_{\lambda} \text{holds}(S, R, O)$.

(ii) If $\Lambda \models_{\lambda} \text{holds}(S, R, C|f) \land O \text{ is member of } C \text{ and } \Lambda \not\models_{\lambda} \neg \text{holds}(S, R, O|f)$, then $\Lambda \models_{\lambda} \text{holds}(S, R, O|f)$.

The above two theorems directly follow from generic authorization schemes (5.33) - (5.37) and Definitions 17 and 18. The following two theorems, on the other hand, represent that these subclass and membership authorization inheritance can be overridden such that the consistency of authorizations can be maintained.

**Theorem 5 (Overriding of Subclass Authorization Inheritance)** Let $\Lambda$ be a well-specified complex database, and $S$, $R$, $C$ and $O$ be arbitrary subject constant, access right constant and object constants respectively. Then the following results hold.

(i) If $\Lambda \models_{\lambda} \text{holds}(S, R, C) \land O \text{ is subclass of } C \text{ and } \Lambda \not\models_{\lambda} \text{holds}(S, R, O)$, then $\Lambda \models_{\lambda} \neg \text{holds}(S, R, O)$.

(ii) If $\Lambda \models_{\lambda} \text{holds}(S, R, C|f) \land O \text{ is subclass of } C \text{ and } \Lambda \not\models_{\lambda} \text{holds}(S, R, O|f)$, then $\Lambda \models_{\lambda} \neg \text{holds}(S, R, O|f)$.

**Theorem 6 (Overriding of Membership Authorization Inheritance)** Let $\Lambda$ be a well-specified complex database, and $S$, $R$, $C$ and $O$ be arbitrary subject constant, access right constant and object constants respectively. Then the following results hold.

(i) If $\Lambda \models_{\lambda} \text{holds}(S, R, C) \land O \text{ is member of } C \text{ and } \Lambda \not\models_{\lambda} \text{holds}(S, R, O)$, then $\Lambda \models_{\lambda} \neg \text{holds}(S, R, O)$.

(ii) If $\Lambda \models_{\lambda} \text{holds}(S, R, C|f) \land O \text{ is member of } C \text{ and } \Lambda \not\models_{\lambda} \text{holds}(S, R, O|f)$, then $\Lambda \models_{\lambda} \neg \text{holds}(S, R, O|f)$.

**THESIS**
For example, in a well-specified complex database $\Lambda, \Lambda \models \text{holds}(\text{Anne, Update, Staff | record})$ and $\text{GenStaff}$ and $\text{AcadStaff}$ are subclasses of $\text{Staff}$. If neither $\neg \text{holds}(\text{Anne, Update, GenStaff | record})$ nor $\neg \text{holds}(\text{Anne, Update, AcadStaff | record})$ are entailed by $\Lambda$, from Theorem 3, we can get that $\text{holds}(\text{Anne, Update, GenStaff | record})$ and $\text{holds}(\text{Anne, Update, AcadStaff | record})$.

5.4.2 Research People Database Revisited

In this subsection, we revisit the research people database domain discussed in Example 26. We will consider a set of authorizations on objects in this database and show how our approach can be used to reason about authorizations in this object oriented database environment.

Example 29 Research people database domain revisited. Consider an extended research people database $\Lambda = (\Sigma, \Xi)$, where $\Sigma$ is specified to be the database described in Example, and $\Xi$ is a set of authorization policies on $\Sigma$.

We assume that in this domain, there are three layers of subjects:

(i) A super user called a database administrator $\text{DBA}$, who can read and update class $\text{ResPeople}$ and all its subclasses and members through inheritance;

(ii) A student administrator $\text{StdAdm}$ who can read and update class $\text{Postgraduate}$ and all its members through the inheritance. An academic administrator $\text{AcadAdm}$ who can read and update class $\text{Staff}$ and all its subclasses and members through inheritance.

(iii) Five individual users named $T, S, F, P$ and $J$ indicating people Tom, Sue, Faye, Peter and James respectively, who can read their own objects $\text{Tom, Sue, Faye, Peter}$ and $\text{James}$.

The structure of this extended database is shown in Figure 5.2.

Now we can specify $\Xi$ as follows. Let $\Xi = GA \cup A$, where $GA$ is a collection of (5.33) - (5.37) (i.e. generic authorization scheme). Based on the above descriptions (i), (ii)
and (iii), clearly $A$ should include the following access propositions:

\[
\begin{align*}
\text{always} & \, holds(DBA, \text{Read}, \text{ResPeople}), & (5.41) \\
\text{always} & \, holds(DBA, \text{Update}, \text{ResPeople}), & (5.42) \\
\text{always} & \, holds(\text{StdAdm}, \text{Read}, \text{Postgraduate}), & (5.43) \\
\text{always} & \, holds(\text{StdAdm}, \text{Update}, \text{Postgraduate}), & (5.44) \\
\text{always} & \, holds(\text{AcaAdm}, \text{Read}, \text{Staff}), & (5.45) \\
\text{always} & \, holds(\text{AcaAdm}, \text{Update}, \text{Staff}), & (5.46) \\
\text{always} & \, holds(T, \text{Read}, \text{Tom}), & (5.47) \\
\text{always} & \, holds(S, \text{Read}, \text{Sue}), & (5.48)
\end{align*}
\]
always holds(F, Read, Faye), \hspace{1cm} (5.49)
always holds(P, Read, Peter), \hspace{1cm} (5.50)
always holds(J, Read, James). \hspace{1cm} (5.51)

(iv) Observing our database structure, it is also clear that Sue belongs to two different classes Postgraduate and Tutor. Due to the membership authorization inheritance property, it turns out that subject StdAdm can read and update all methods of Sue that are inherited from classes Staff and Tutor. Clearly, this is not our expectation since intuitively, a student administrator is not usually allowed to access some staff's information (e.g. salary). This problem can be avoided easily by including the following proposition in A:

\[ o \text{ is a member of Staff implies } \neg \text{holds(StdAdm, } r, o | \text{salary)}. \] \hspace{1cm} (5.52)

(v) Furthermore, if some staff is the supervisor of a postgraduate student, this staff should be able to read all information about his/her student unless there is an explicit declaration denying this. So A also includes one more access proposition:

\[ \text{holds}(s, \text{read}, o') \land o' \text{ is a member of Staff} \land \]
\[ o \text{ is a member of Postgraduate} \land \]
\[ o \text{ has method area}(o') \rightarrow \Pi \]
\[ \text{implies holds}(s, \text{Read}, o) \]
\[ \text{with absence } \neg \text{holds}(s, \text{Read}, o). \] \hspace{1cm} (5.53)

Now we have completed all specifications for this extended research people object oriented database. It can be proved that this databases is well-specified. That is, A has a unique model. The following results show that A indeed presents the desired authorization policies on objects in the database.

THESIS
(a) DBA can read and update every postgraduate student and staff members' objects and all their methods inherited from all of their superclasses, i.e.
\[ \Lambda \models_{\lambda} \text{holds(DBA, Read/Update, O)}^{8}, \text{ and} \]
\[ \Lambda \models_{\lambda} \text{holds(DBA, Read/Update, O}|f), \]
where O is any object constant in \( \Lambda \) and f is any method associated with O.

(b) StdAdm can read and update every student member object, i.e.
\[ \Lambda \models_{\lambda} \text{holds(StdAdm, Read/Update, O),} \]
where O is Tom or Sue, which also implies that StdAdm can also read and update all methods of Tom and Sue inherited from their superclasses ResPeople and Postgraduate, i.e.
\[ \Lambda \models_{\lambda} \text{holds(StdAdm, Read/Update, O}|f).} \]
But StdAdm cannot read and update Sue's salary which is inherited from class Staff:
\[ \Lambda \models_{\lambda} \neg\text{holds(StdAdm, Read/Update, Sue|salary).} \]

(c) AcaAdm can read and update staff members Faye and James, i.e.
\[ \Lambda \models_{\lambda} \text{holds(AcaAdm, Read/Update, Faye), and} \]
\[ \Lambda \models_{\lambda} \text{holds(AcaAdm, Read/Update, James),} \]
which also imply that AcaAdm can also read and update all methods of Faye and James. Note that since there is no explicit description about object Peter, AcaAdm cannot access every method of Peter.

(d) For every individual user (i.e. T,S,F,P and J), he/she can read his/her corresponding object and all its methods inherited from all its superclasses, i.e.
\[ \Lambda \models_{\lambda} \text{holds(T, Read, Tom),} \]
\[ \Lambda \models_{\lambda} \text{holds(T, Read, Tom}|f), \]
\[ \ldots, \]
\[ \Lambda \models_{\lambda} \text{holds(J, Read, James),} \]
\[ \Lambda \models_{\lambda} \text{holds(J, Read, James}|f), \]
where f denotes the any method associated with the corresponding object.

---

8 It denotes holds(DBA, Read, O) and holds(DBA, Update, O).
(e) Every supervisor can read his/her student object’s all methods, i.e.
\[ \Lambda \models_{\lambda} \text{holds}(P, \text{Read}, \text{Tom}|f) \]
\[ \Lambda \models_{\lambda} \text{holds}(J, \text{Read}, \text{Sue}|f') \]
where \( f \) and \( f' \) denotes any method of \text{Tom} and \text{Sue} respectively. Note that
\[ \Lambda \models_{\lambda} \text{holds}(P, \text{Read}, \text{Tom}|f) \] is derived from the fact that
\[ \Sigma \models \text{Peter is a member of Staff}. \]

5.5 Discussions

Here, we briefly review some related work, and discuss the different approaches used in object-oriented database security.

In [35], a security model for object-oriented databases was proposed. This model consists of a set of policies, a structure for authorization rules and an algorithm to evaluate access requests. The database is composed of objects that include a collection of facts and a collection of relevant rules. An object binds knowledge rules to database facts. The database is specified by the OSAM* [85, 86] model, in which the generic properties are defined through a generalization association and the set of attributes of a class is defined by an aggregation association. Derived classes (subclasses) are viewed as generic. Class inheritance properties suggest that access to some attributes of a class also imply access to the corresponding values in its subclass. Generally, there are three types of access policies:

1. A user who has access to a class is allowed to have similar type of access in the corresponding subclasses to the attributes inherited from that class.

2. Access to a complete class implies access to the attributes defined in that class as well as to attributes inherited from a higher class.

3. An attribute defined for a subclass is not accessible by accessing any of its superclasses.

In our system, we have considered similar access policies via subclass and membership relationship authorization rules. These rules are specified by the theorems of
§5.5 Discussions

subclass authorization, membership authorization, overriding subclass authorization and overriding membership authorization. In particular, our Theorem 1 and Theorem 3 capture the above three types of access policies. Our model is based on our previous work of formal specification for authorization policies and their transformations, and is discussed from authorization specification point of view. However, in practice, the access policies are organizational dependent. They can vary from organization to organization and can also vary depending on the type of applications.

In this chapter, we have proposed a logic formalization for specifying authorizations in object oriented databases. Our work consisted of two steps: the first step involved a formal language $\mathcal{L}^o$ to formalize object oriented databases within a logic framework. We provided a high level language to specify an object oriented database and defined a precise semantics for it. Our semantics of $\mathcal{L}^o$ shares some features of Kifer et al.'s F-logic for specifying object oriented databases. But our database specification is more succinct and intuitive, and hence it has been possible to extend this by combining it with authorization structures. The second step of our work was to extend $\mathcal{L}^o$ to language $\mathcal{L}^{oa}$ by representing different types of authorizations in the database. It has been shown that the types of authorizations in our formalism are quite flexible and can be used to reason about complex authorizations compared with other approaches.
We have presented formal logical models in the areas of specifying and evaluating authorization policies, single step and sequence of authorization policy transformations and their application in object oriented database systems. We used a policy base which is a knowledge base consisting of a finite set of access rights and a finite set of access rules to represent authorization policy. The rules represent the constraints which the system needs to satisfy. The model(s) of the policy base indicates the consistency of the authorization policy. In authorization evaluation, the entailment relation is used to determine the denial or grant of an access request. For the single step authorization transformation, we first defined the structure of the policy transformation and then employed a model-based semantics to perform the transformation and to resolve the conflict of authorizations under the principle of minimal change. We also proposed two high level languages for specifying the sequence of authorization transformation. It answers questions such as: given a sequence of authorization transformations, what is the final policy base after such a sequence of transformations? Finally, in the application part, we applied our formal authorization model which has a clear and declarative semantics to specify the structural features of object oriented databases and authorizations associated with complex data objects into object oriented database systems.

6.1 Contributions

In this section, we conclude our work and summarize our specific contributions in the areas of authorization policy specification, authorization policy transformation
and application in object oriented database systems.

6.1.1 Authorization Specification

We proposed a logical based formal approach to specify authorization policy and applied an entailment relation to evaluate access requests. Our approach provided an organized way for representing authorization policy and evaluating access requests.

Same as other logical approaches for authorization specification, the high level specification separates authorization representation from its implementation. This separation provides flexibility of enforcing different authorization policies into different implementation systems or enforcing more than one authorization policies into the same implementation system. In addition, the implementation system is not necessarily affected by the change of the authorization policy enforced in it.

In contrast to other logic based approaches, our approach uses model based semantics and an entailment relation which can semantically detect inconsistencies in a policy base, and make a decision for an access request.

For instance, for a policy base PB, by computing its model or set of models, we can always tell if PB is consistent and what policies are implicitly represented by it. If PB does not have any model, this indicates that PB is not consistent. Hence it does not represent a valid authorization policy. If an access request α is made against PB, we need to verify if PB |= α. If the entailment relation is satisfied, it indicates that α is a logical consequence of PB, then the access request is granted.

Compared with traditional authorization policy specifications such as access matrix, capability list or some other low level system mechanism, a logic based approach has powerful expressiveness. It can specify not only the access rights as the traditional authorization specification does, but also the access constraint rules, access inheritance rules. With such enhanced abilities, a logic approach is capable of specifying a variety of complex authorization policies.

In practice, authorization policy has a temporal property. That is, as the system, application or user requirement changes, or the authorization policy itself needs upgrading, the update of authorization policy is unavoidable. Therefore, the specifi-
tion of policy update is an issue that needs to be considered. In our authorization specification, we have considered the possible change of an authorization policy and taken into account the formalization of authorization update. Our approach is designed to have not only the general ability of the logic based approach, but also can accommodate policy updates.

6.1.2 Authorization Transformation

Specifications on authorization policy update, formally called authorization transformation, have been discussed in many research papers. But most of these discussions are based on pseudo-procedures or some low level system mechanism. As far as we know, few of the works are discussed on a formal logic basis.

We proposed a formal logical approach to specify authorization transformations and employed a model based semantics to perform single step authorization transformations. We used preference ordering to solve the possible conflict caused by transformations under the principal of minimal change.

Our transformation approach is based on model semantics of authorization specification. The transformation is performed on every individual model of the policy base. And the union of all the transformed models comprise the model of the transformed policy base, thus we compute the new policy base from its model. To solve the possible conflict during transformation, we defined the preference ordering on these predicates, and used the concepts of strong and weak authorizations. We made the strong authorization to override the weak authorization when conflict occurs.

Model based transformation computes all of the possible states of a policy base represented by its models transformation. It is guaranteed to compute the set of models of the new policy base and then get the new policy base from the computed models.

For the implementation, we used an algorithm called model generator to compute the models of the policy base. The algorithm new policy base finder is used to generate the new policy base from the computed set of models. We also analyzed these implementation algorithms.

To specify sequence of authorization transformations, we proposed another formal
high level language approach. It generates the resulting policy base after a sequence
of authorization transformations, and also can be used to specify many conventional
access control policies [11, 13].

In this approach, we defined some propositions to specify policy base and se-
quence of policy transformations. We further added a default proposition to specify
policy domain constraints, inheritance and causal authorization policies.

6.1.3 Authorization Application

In chapter 5, we addressed authorization in object oriented database systems from
a formal logical point of view. Our logical language has a clear and declarative se-
manics to specify the structural features of object oriented databases and characterize
different types of inheritance properties of authorization policies among complex data
objects in the database. Our formalization characterizes the model-theoretic semantics
of object oriented databases and authorizations associated with them.

Some research work has been done in the area of authorization in database sys-
tems. But most of this work emphasizes authorizations only, the database system
specification and the combination of authorization with the database system are ig-
nored. Our work addresses authorization in object oriented database by specifying
objected oriented database system first, then extends this specification to include au-
thorizations.

In database specification, we used three different kinds of propositions to spec-
ify data object, their relation and constraint among them respectively. Using these
propositions, we can formally specify an object oriented database. Then we added ac-
cess proposition into database specification to address authorizations associated with
object oriented database.

We also discussed reasoning about authorizations and authorization inheritance
in object oriented database. A well specified database with authorizations has the
properties of subclass authorization inheritance and overriding authorization inheri-
tance as well as membership authorization inheritance and overriding authorization
inheritance.
6.2 Future Work

Our research on formal specification of authorization policies and their transformation will continue. In this section, we describe several important issues that we plan to investigate in the near future.

In our authorization specification and single step transformation, we used model based semantics. A model of a policy base can be used to determine the consistency of the policy base and evaluate access request quickly and easily. All models of a policy base consist of the space of possible states of the policy base. Consistency is the condition for a policy base to have models. So to use a model of a policy base to evaluate access rights, firstly we need to make sure the policy base is consistent. Also, the set of all models of a policy base could be too large so that it might be inefficient to evaluate access rights through the models of the policy base. On the other hand, deciding whether an access right can be derived from a policy base, e.g. deciding implicit access rights in a policy base, could be expensive from the computational point of view. Hence, to apply our logic based approach in practice, we need to set some restrictions. In our future work, we plan to use a logic program to represent a policy base, rather than a set of arbitrary formulas. For instance, the policy base described in Example 3 in section 2.2.3 of Chapter 2 can be represented as a logic program:

\[
S_1 \in G \leftarrow,
S_2 \in G \leftarrow,
g\text{-}holds(G, \text{Read}, O) \leftarrow,
s\text{-}holds(s, a, o) \leftarrow s \in g, g\text{-}holds(g, a, o).
\]

The advance of using logic programs to represent policy bases is clear. Since logic programming is a well accepted programming language paradigm for representing rule based systems, by using logic programs we can maintain the rule based feature of our policy bases and also make it easier to evaluate.

The other issue we will further explore is about the access constraint transformation. In our current authorization transformation models, either in the single step transformation or in the sequence of transformations, we only consider the transfor-
mation of specific access rights. It is observed that in our approach, we allow con-
straints of access rights to be presented in our policy bases or domain descriptions.
The role of these constraints is to derive some implicit access rights embedded into
the current policy base or domain description and they are not changeable. Now an
interesting question is: how can these constraints be transformed? For example, sup-
pose in a problem domain, initially we have a constraint saying that if an user is in
group $A$, then this user is allowed to read file $F$. After some time, the situation is
changed and we need a new constraint stating that if an user is in group $A$, this user is
not allowed to read file $F$. Obviously, to keep our policy base (or domain description)
consistent, we need to remove the former constraint and add the latter one in our pol-
icy base (or domain description). How to extend our current formalizations to have
this capability will be addressed in our future work.

The access policies that we have considered in chapter 5 are based on subject au-
thorization viewpoint. In practice, both subject and object can be in a hierarchy struc-
ture. From object-oriented system viewpoint, access propagation is also data structure
related. In our model, authorization propagation from object viewpoint can also be
specified. For instance, there is an object folder, the folder contains a number of doc-
uments, each document contains a number of files. We can also specify authorization
propagations based on the hierarchical structure. For example, the documents can be
viewed as subclasses of class folder. If the folder can be accessed, then the document
could also be accessed. This can be specified as:

$$\text{holds}(s, \text{access}, \text{Folder}) \ implies \ holds(s, \text{access}, \text{Document}).$$

In our future work, we intend to consider attributes for other types of associations.
In particular, we will consider in more detail the generalization and aggregation as-
sociations. In addition, the placement of the authorization policies also needs to be
addressed. They may be placed in a special class or a class they refer to, or prop-
agated through the hierarchy structure. Furthermore, we have not investigated the
access to the authorization system itself yet. This will also be considered in our future
work.

To the best of our knowledge, the work presented in Chapter 5 is the first such work trying to build a formal logic framework for specifying object oriented databases with various types of authorizations. In fact, our work can be further extended to handle the problem of transformation of authorizations in object oriented databases. The question we are interested here is how to update some subjects' access rights whenever requests come. For instance, in the requirement 1 discussed in section 5.3 of Chapter 5, suppose after some time, the administrator is allowed to read a postgraduate’s salary record if this postgraduate is also a tutor. In this case, a transformation of the administrator’s access right occurs.
Bibliography


THESIS


THESIS


**THESIS**


**THESIS**


**THESIS**


**THESIS**


THESIS


THESIS

Appendix: Proofs

**Proposition 1** A consistent domain description $D$ has a unique model.

**Proof:** Obviously, a consistent domain description $D$ has a model. From the definition of domain description, $D$ is a finite set of initial policy propositions and transformation propositions. As each initial policy proposition has the form

initially $\phi$,

where $\phi$ is a conjunction of ground atomic formulas or their negations. Then there must be a unique initial state $\sigma_0$ in which each initial policy proposition of $D$ is true. Now consider each transformation proposition in $D$:

$T$ causes $\phi$ if $\Psi$.

Since both $\phi$ and $\Psi$ are conjunctions of atomic formulas or their negations, for any transition $\rho$, the resulting state $\rho(T, \sigma)$ is unique. Therefore, according to Definition 7, it is obvious that the model $(\sigma_0, \rho)$ of $D$ is unique. ■

**Proposition 2** Let $D$ be a static domain description in language $L^{sd}$. We specify a Woo-Lam policy base $B$ in terms of $D$ as the follows:

(i) for each initial policy proposition initially $\phi$ in $D$, we specify a rule $\phi$ in $B$;

(ii) for each default proposition $\phi$ implies $\psi$ with absence $\gamma$ in $D$, we specify a rule $\phi$:

$\neg \gamma / \psi$ in $B$;

(iii) for each special default proposition $\phi$ provokes $\psi$, we specify a rule $\phi \Rightarrow \psi$ in $B$;

(iv) for each special default proposition always $\psi$, we specify a rule $\psi$ in $B$. 

133
Then for any ground fact expression $\psi$, $D \models_{\mathcal{L}_{id}} \text{initially} \ \psi$ if and only if $B \vdash_{wl} \psi$.

Proof: Under the specification defined in Proposition 2, we actually translate a static domain description $D$ into an equivalent Reiter's default theory $B$ [69]. On the other hand, from Definition 10, it can be shown that the model of our domain description is coincident with the answer set of extended logic program [37]. Then from the correspondence between the answer set of an extended logic program and the extension of a default theory, as proved in [37], the result holds. ■

Proposition 3 Let $B$ be a Woo-Lam policy base where for each rule $f : f'/g$ in $B$, $f$, $f'$ and $g$ are expressed in the form of ground fact expressions in language $\mathcal{L}_{id}$. We specify a static domain description $D$ in terms of $B$ as follows:

(i) for each rule $f : f'/g$ in $B$, we specify a default proposition $f$ implies $g$ with absence $-f_1 \land \cdots \land f_k$, where $f' \equiv f'_1 \land \cdots \land f'_{k'}$.

(ii) for each rule $g$ in $B$, we specify an initial policy proposition initially $g$;

(iii) for each rule $f \Rightarrow g$ in $B$, we specify a special default proposition $f$ provokes $g$ in $D$.

Then for each ground fact expression $\psi$, $B \vdash_{wl} \psi$ if and only if $D \models_{\mathcal{L}_{id}} \text{initially} \ \psi$.

Proof: The proof is similar to that of Proposition 2. ■

Proposition 4 (Method Projection) Let $I$ be a structure of $\mathcal{L}_o$ and $\phi$ a ground object proposition with the form:

$O$ has method $f_1(\cdots) \hookrightarrow \Pi_1,$

$\cdots,$

$f_n(\cdots) \hookrightarrow \Pi_n$,

where $\hookrightarrow$ is a symbol for $\Rightarrow$ or $\hookrightarrow$. Then $I \models \phi$ implies that $I \models O$ has method $f_1(\cdots) \hookrightarrow \Pi_1, \cdots, I \models O$ has method $f_n(\cdots) \hookrightarrow \Pi_n$.

Proof: Given a database $\Sigma = (\Gamma, \Delta, \Omega)$, let $M$ be a model of $\Sigma$. Then according to the condition 2 in Definition 15, the result holds directly. ■
Proposition 5 (Method Aggregation) Let $I$ be a structure of $L^0$ and $\phi_1$ and $\phi_2$ ground object propositions with forms:

$$O \text{ has method } f_1(\cdots) \Leftrightarrow \Pi_1,$$

$$\cdots,$$

$$f_n(\cdots) \Leftrightarrow \Pi_n,$$

and

$$O \text{ has method } f_1'(\cdots) \Leftrightarrow \Pi'_1,$$

$$\cdots,$$

$$f'_n(\cdots) \Leftrightarrow \Pi'_m,$$

respectively, where $\Leftrightarrow$ is a symbol for $\Rightarrow$ or $\iff$. Then $I \models \phi_1$ and $I \models \phi_2$ imply that $I \models \phi$, where $\phi$ is ground object proposition with the form:

$$O \text{ has method } f_1(\cdots) \Leftrightarrow \Pi_1,$$

$$\cdots,$$

$$f_n(\cdots) \Leftrightarrow \Pi_n,$$

$$f_1'(\cdots) \Leftrightarrow \Pi'_1,$$

$$\cdots,$$

$$f'_m(\cdots) \Leftrightarrow \Pi'_m.$$

Proof: Let $\phi_1$ has the form:

$$O \text{ has method } f_1(\cdots) \Rightarrow \Pi_1,$$

$$\cdots,$$

$$f_k(\cdots) \Rightarrow \Pi_k,$$

$$f_{k+1}(\cdots) \Leftrightarrow \Pi_{k+1}.$$
..., 
\[ f_n(\cdots) \mapsto \Pi_n, \]
and \( \phi_2 \) has the form:

\[ O \text{ has method } f'_1(\cdots) \Rightarrow \Pi'_1, \]
\[ \cdots, \]
\[ f'_l(\cdots) \Rightarrow \Pi'_l, \]
\[ f'_{l+1}(\cdots) \Rightarrow \Pi'_{l+1}, \]
\[ \cdots, \]
\[ f'_m(\cdots) \Rightarrow \Pi'_m. \]

Since \( I \models \phi_1 \) and \( I \models \phi_2 \), according to the condition 2 of Definition 14, we have

- for each \( f(O_1, \ldots, O_p) \), where \( f \) is in \( \{f_1, \cdots, f_k, f'_1, \cdots, f'_l\} \),
  \[ \Rightarrow^{(p)}_I (O')(O_1, \cdots, O_p, O) = \Pi, \text{ where } f_I(O_1, \cdots, O_p) = O', \text{ and} \]

- for each \( f'(O'_1, \cdots, O'_q) \) where \( f' \) is in \( \{f'_{k+1}, \cdots, f'_m, f'_1, \cdots, f'_l\} \),
  \[ \Rightarrow^{(q)}_I (O'')(O'_1, \cdots, O'_q, O) = \Pi', \text{ where } f'_I(O'_1, \cdots, O'_q) = O''. \]

This follows that \( I \models \phi \), where \( \phi \) is a ground object proposition with the form:

\[ O \text{ has method } f_1(\cdots) \Rightarrow \Pi_1, \]
\[ \cdots, \]
\[ f_k(\cdots) \Rightarrow \Pi_k, \]
\[ f_{k+1}(\cdots) \Rightarrow \Pi_{k+1}, \]
\[ \cdots, \]
\[ f_n(\cdots) \Rightarrow \Pi_n, \]
\[ f'_1(\cdots) \Rightarrow \Pi'_1, \]
\[ \cdots, \]
\[ f'_l(\cdots) \Rightarrow \Pi'_l, \]

THESIS
\[ f'_{i+1}(\cdots) \Rightarrow \Pi'_{i+1}, \]
\[ \cdots, \]
\[ f'_m(\cdots) \Rightarrow \Pi'_m. \]

So the result holds. ■

**Theorem 1 (Subclass Inclusion)** If \( I \models O \) is a member of \( C_1 \), and \( I \models C_1 \) is a subclass of \( C_2 \), then \( I \models O \) is a member of \( C_2 \).

**Proof:** This result directly follows from the interplay between the partial ordering \( \subseteq_U \) and the binary relation \( \in_U \) on \( U \) (see the condition 3 of Definition 13). ■

**Theorem 2 (Subclass Type Inheritance)** The following results hold.

(i) If \( I \models O \) has method \( f(Q_1, \cdots, Q_k) \Rightarrow S \) and \( I \models R \) is a subclass of \( O \), then \( I \models R \) has method \( f(Q_1, \cdots, Q_k) \Rightarrow S \).

(ii) If \( I \models O \) has method \( f(Q_1, \cdots, Q_n, \cdots, Q_k) \Rightarrow S \) and \( I \models Q'_l \) is a subclass of \( Q_k \), then \( I \models O \) has method \( f(Q_1, \cdots, Q'_l, \cdots, Q_k) \Rightarrow S \).

(iii) If \( I \models O \) has method \( f(\cdots) \Rightarrow S \) and \( I \models S \) is a subclass of \( R \), then \( I \models O \) has method \( f(\cdots) \Rightarrow R \).

**Proof:** See the description in section 5.2 of Chapter 5, immediately after the presentation of Theorem 2. ■

**Proposition 6 (Subclass Value Inheritance)** If \( I \models O \) is a subclass of \( C \) and \( I \models C \) has method \( f(\cdots) \Rightarrow \Pi \), then \( I \models O \) has method \( f(\cdots) \Rightarrow \Pi \).

**Proof:** This result directly follows from the condition 4 in Definition 15. ■

**Proposition 7 (Membership Type and Value Inheritance)** The following results hold.

(i) If \( I \models O \) is a member of \( C \) and \( I \models C \) has method \( f(\cdots) \Rightarrow \Pi \), then \( I \models O \) has method \( f(\cdots) \Rightarrow \Pi \).
Appendix: Proofs

(ii) If \( I \models O \text{ is a member of } C \) and \( I \models C \text{ has method } f(\cdots) \to \Pi \), then \( I \models O \text{ has method } f(\cdots) \to \Pi \).

Proof: This result directly follows from the condition 3 of Definition 15. 

**Theorem 3 (Subclass Authorization Inheritance)** Let \( \Lambda \) be a well-specified extended database and \( S, R, C \) and \( O \) are arbitrary subject constant, access right constant and object constants respectively. Then the following results hold.

(i) If \( \Lambda \models_\chi holds(S, R, C) \land O \text{ is subclass of } C \) and \( \Lambda \not\models_\chi \neg holds(S, R, O) \), then \( \Lambda \models_\chi holds(S, R, O) \).

(ii) If \( \Lambda \models_\chi holds(S, R, C, f) \land O \text{ is subclass of } C \) and \( \Lambda \not\models_\chi \neg holds(S, R, O, f) \), then \( \Lambda \models_\chi holds(S, R, O, f) \).

Proof: We only prove (i) while (ii) can be proved in a similar way. Let \( \Lambda = (\Sigma, \Xi) \) and \( I^\Lambda = (I^\Sigma, I^\Xi) \) be a model of \( \Lambda \). Then from the fact \( \Lambda \models_\chi holds(S, R, C) \land O \text{ is subclass of } C \), we have \( holds(S, R, C) \in I^\Xi \) and \( I^\Xi \models O \text{ is subclass of } C \). Since \( \Lambda \not\models_\chi \neg holds(S, R, O) \), we have \( holds(S, R, O) \notin I^\Xi \). So from Definition 18, the form of proposition

\[
holds(S, R, C) \land O \text{ is subclass of } C
\]

implies \( holds(S, R, O) \)

with absence \( \neg holds(S, R, O) \)

is translated into the form

\[
holds(S, R, C) \land O \text{ is subclass of } C
\]

provokes \( holds(S, R, O) \).

Then from Definition 19, we know that \( I^\Lambda \) satisfies the condition: if \( I^\Lambda \models holds(S, R, O) \) and \( I^\Lambda \models O \text{ is subclass of } C \), then \( I^\Lambda \models holds(S, R, O) \). This concludes that \( I^\Lambda \models holds(S, R, O) \). As \( I^\Lambda \) is an arbitrary model of \( \Lambda = (\Sigma, \Xi) \), we then have \( \Lambda \models_\chi holds(S, R, O) \).
Theorem 4 (Membership Authorization Inheritance) Let \( \Lambda \) be a well-specified complex database, and \( S, R, C \) and \( O \) are arbitrary subject constant, access right constant and object constants respectively. Then the following results hold.

(i) If \( \Lambda \models_{\chi} holds(S, R, C) \land O \text{ isa member of } C \) and \( \Lambda \not\models_{\chi} \neg holds(S, R, O) \), then \( \Lambda \models_{\chi} holds(S, R, O) \).

(ii) If \( \Lambda \models_{\chi} holds(S, R, C[f]) \land O \text{ isa member of } C \) and \( \Lambda \not\models_{\chi} \neg holds(S, R, O[f]) \), then \( \Lambda \models_{\chi} holds(S, R, O[f]) \).

Proof: This theorem is proved in a similar way as to that of Theorem 3. ■

Theorem 5 (Overriding of Subclass Authorization Inheritance) Let \( \Lambda \) be a well-specified complex database, and \( S, R, C \) and \( O \) are arbitrary subject constant, access right constant and object constants respectively. Then the following results hold.

(i) If \( \Lambda \models_{\chi} holds(S, R, C) \land O \text{ isa subclass of } C \) and \( \Lambda \not\models_{\chi} holds(S, R, O) \), then \( \Lambda \models_{\chi} \neg holds(S, R, O) \).

(ii) If \( \Lambda \models_{\chi} holds(S, R, C[f]) \land O \text{ isa subclass of } C \) and \( \Lambda \not\models_{\chi} holds(S, R, O[f]) \), then \( \Lambda \models_{\chi} \neg holds(S, R, O[f]) \).

Proof: Here we only prove (i) and (ii) is proved in a similar way. Let \( \Lambda = (\Sigma, \Xi) \) and \( I^\Lambda = (I^S, I^R) \) be a model of \( \Lambda \). Then from \( I^\Lambda \models holds(S, R, C) \land O \text{ isa subclass of } C \) and \( I^\Lambda \not\models holds(S, R, O) \), we know that in Definition 18, the form of proposition

\[
holds(S, R, C) \land O \text{ isa subclass of } C \\
\text{ implies } holds(S, R, O) \\
\text{ with absence } \neg holds(S, R, O)
\]

can not be translated into the form

\[
holds(S, R, C) \land O \text{ isa subclass of } C \\
\text{ provokes } holds(S, R, O).
\]

THESIS
So it must be the case that the above proposition is deleted from $\Xi$ with $\neg holds(S,R,O) \in I^\Xi$. On the other hand, since $\Lambda = (\Sigma, \Xi)$ is well-specified, that is, it has a unique model, it follows that $\Lambda \models_{\lambda} \neg holds(S,R,O)$. 

**Theorem 6 (Overriding of Membership Authorization Inheritance)** Let $\Lambda$ be a well-specified complex database, and $S$, $R$, $C$ and $O$ are arbitrary subject constant, access right constant and object constants respectively. Then the following results hold.

(i) If $\Lambda \models_{\lambda} holds(S,R,C) \land O$ is a member of $C$ and $\Lambda \not\models_{\lambda} holds(S,R,O)$, then $\Lambda \models_{\lambda} \neg holds(S,R,O)$.

(ii) If $\Lambda \models_{\lambda} holds(S,R,C|f) \land O$ is a member of $C$ and $\Lambda \not\models_{\lambda} holds(S,R,O|f)$, then $\Lambda \models_{\lambda} \neg holds(S,R,O|f)$.

**Proof:** This theorem is proved in a similar way as to that of Theorem 5. 

---

THESIS

THESIS

Yun Bai

A thesis submitted for the degree of Doctor of Philosophy at University of Western Sydney - Nepean

September 2000
PLEASE NOTE

The greatest amount of care has been taken while scanning this thesis,

and the best possible result has been obtained.
Except where otherwise indicated, this thesis is my own original work.

Yun Bai
8 September 2000
To my family.
Acknowledgements

I am deeply grateful to my supervisor, Professor Vijay Varadharajan, for his excellent guidance and constant advice throughout my Ph.D study. Every discussion with him was inspiring and encouraging. I am also very grateful to my co-supervisor, Dr. Yi Mu, for his valuable suggestions and comments.

I thank University of Western Sydney Nepean for supporting me with Nepean Postgraduate Research Award which financially made this work possible.

Finally, I thank my parents, for their encouragement and patience, and my husband Yan, for his constant support, caring and understanding which made this work possible. Also I thank my daughters Feili and Feiya for bringing so much joy and appreciation of life.
Abstract

Most of today's information systems are quite complex and often involve multi-user resource-sharing. In such a system, authorization policies are needed to ensure that the information flows in the desired way and to prevent from illegal access to the system resource. Overall, authorization policies provide the ability to limit and control accesses to systems, applications and information. In the real world, such a policy has a temporal property. That is, it needs to be updated to capture the changing requirements of applications, systems and users. These updates are implemented through the transformation of authorization policies. In this thesis, we propose a logic based formal approach to specifying authorization policies and to reason about the transformation and sequence of transformations of authorization policies and its application in object oriented databases. Firstly, the authorization policy is specified by a logic language using a policy base which comprises a finite set of facts and a finite set of access constraints. The facts represent explicitly the access rights the subjects hold for the objects. The access constraints, on the other hand, are rules which the authorization policy should satisfy. We define the structure of the policy transformation and employ a model-based semantics to perform the transformation under the principle of minimal change. Furthermore, we extend model-based semantics by introducing preference ordering to resolve possible conflicts during transformation of policies. We also discuss the implementation of the model based transformation approach and analyze the complexity of the algorithms introduced. Secondly, we modify the language to consider a sequence of authorization policy transformations. It handles more complex transformations and solves the problem like: given a policy base and a sequence of transformations, what is the resulting policy base after performing the sequence of transformations? The language is able to represent incomplete information, default authorizations and allows denials to be expressed explicitly. We also use the proposed language to specify a variety of well known access control policies such as static sep-
aration of duty, dynamic separation of duty and Chinese wall security policy. Finally, we apply our authorization formalization to object oriented databases. The formalization has a high level language structure to specify object oriented databases and allows various types of authorizations to be associated with. Formal semantics are also provided for the formalization.
Contents

Acknowledgements v

Abstract vii

1 Introduction 1

1.1 Motivation ............................... 1

1.2 Security Policy and Access Control System ............... 3

1.2.1 Security Policy ....................... 3

1.2.2 Access Control System ................ 3

1.3 Authorization Specifications ..................... 5

1.4 Authorization Transformations .................... 10

1.5 Authorization Application in Database Systems .......... 12

1.6 About This Thesis ........................ 14

2 A Formal Representation for Authorization Policy 17

2.1 General Background .......................... 18

2.1.1 Access Matrix Model ................... 18

2.1.2 Capability-based Model .................. 20

2.1.3 Logic Model ............................ 23

2.2 A Formal Representation of Authorization Policy ........ 27

2.2.1 Preliminary ............................. 27

2.2.2 Language $L$ for Authorization Policy ........... 30

2.2.3 Representation of Policy Base ................ 32

2.2.4 Evaluation of Authorizations ................ 36

2.3 Summary of The Authorization Model ................. 38
3 Authorization Policy Transformation 41
  3.1 Background 42
  3.2 Policy Base Transformations 44
    3.2.1 Definition of Transformation 45
    3.2.2 Examples 47
  3.3 Conflict Resolutions 49
    3.3.1 The Problem 49
    3.3.2 An Approach 51
  3.4 Implementation Issues 60
    3.4.1 The Basic Idea 60
    3.4.2 The Algorithms 62
      3.4.2.1 The Model Generator 62
      3.4.2.2 Resulting Models and New Policy Base Finders 64
  3.5 Analysis of Implementation Algorithms 65
  3.6 Conclusions of The Transformation Model 67

4 Sequence of Policy Transformations 69
  4.1 Syntax and Semantics of Language $L^i$ 70
    4.1.1 Syntax of $L^i$ 70
    4.1.2 Semantics of $L^i$ 74
  4.1.3 Document Release Example 77
  4.1.4 Properties of Language $L^i$ 80
  4.2 Syntax and Semantics of Language $L^{rd}$ 82
    4.2.1 Syntax of $L^{rd}$ 82
    4.2.2 Semantics of $L^{rd}$ 83
  4.3 Related Work 87
    4.3.1 Woo and Lam's Approach: A Review 87
    4.3.2 Comparison 90
  4.4 Conclusions of The Sequence Transformation Model 91

5 Object Oriented Database with Authorization Policy 93
  5.1 Background 94

THESIS
<table>
<thead>
<tr>
<th>Contents</th>
<th>xi</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2 A Formal Specification for Object Oriented Databases</td>
<td>97</td>
</tr>
<tr>
<td>5.2.1 Syntax of $L^o$</td>
<td>97</td>
</tr>
<tr>
<td>5.2.2 Semantics of $L^o$</td>
<td>103</td>
</tr>
<tr>
<td>5.2.3 Inheritance Properties</td>
<td>106</td>
</tr>
<tr>
<td>5.3 Authorization Specification in Object Oriented Database</td>
<td>109</td>
</tr>
<tr>
<td>5.3.1 Syntax of $L^{oa}$</td>
<td>110</td>
</tr>
<tr>
<td>5.3.2 Databases with Associated Authorizations</td>
<td>112</td>
</tr>
<tr>
<td>5.3.3 Semantics of $L^{oa}$</td>
<td>114</td>
</tr>
<tr>
<td>5.4 Reasoning about Authorizations in Object Oriented Database</td>
<td>116</td>
</tr>
<tr>
<td>Database</td>
<td>116</td>
</tr>
<tr>
<td>5.4.1 Authorization Inheritance</td>
<td>116</td>
</tr>
<tr>
<td>5.4.2 Research People Database Revisited</td>
<td>118</td>
</tr>
<tr>
<td>5.5 Discussions</td>
<td>122</td>
</tr>
<tr>
<td>6 Conclusion</td>
<td>125</td>
</tr>
<tr>
<td>6.1 Contributions</td>
<td>125</td>
</tr>
<tr>
<td>6.1.1 Authorization Specification</td>
<td>126</td>
</tr>
<tr>
<td>6.1.2 Authorization Transformation</td>
<td>127</td>
</tr>
<tr>
<td>6.1.3 Authorization Application</td>
<td>128</td>
</tr>
<tr>
<td>6.2 Future Work</td>
<td>129</td>
</tr>
<tr>
<td>A Appendix: Proofs</td>
<td>133</td>
</tr>
<tr>
<td>Bibliography</td>
<td>141</td>
</tr>
</tbody>
</table>