Introduction

Enterprises are increasingly dependent on their computer systems to support their business activities. Compromise of these systems in terms of competitors gaining access to it can be extremely costly to the enterprise. Access control is hence needed in any secure computer system that provides for controlled sharing of information and other resources among multiple users. It comprises all system mechanisms that are required to check whether an access request issued by a particular user is allowed or not, and mechanisms that are required to enforce the decision accordingly. Access control models, or authorization models, provide a formalism and framework for specifying, analyzing and evaluating security policies that determine how access is granted and delegated among particular users. The models are desired to be flexible enough to easily accommodate diverse security policies. In this thesis, we pursue this goal by formulating new decentralised authorization models supporting authorization delegation, both positive and negative authorization, and conflict resolution. In particular, we propose new conflict resolution policies in decentralised authorization administration which overcome the drawbacks of existing conflict resolution policies and provide users a flexible way to control authorization delegation.

1.1 Access Control Models and Basic Issues

Various access control models have been published over the years [20, 26, 65]. Some are defined in terms of well known abstraction of subjects, objects and access rights and some in terms of roles, permissions and users. Some models are formulated in matrix form, some in the form of rules, some in graphs, while others use logic representation. Access control models can be broadly classified in two categories:
discretionary and mandatory models.

Discretionary Access Control models govern the access of users to the object on the basis of the user's identity and rules that specify, for each user and object in the system, the types of access the user is allowed for the object.

Mandatory Access Control models govern the access to the object by the individuals on the basis of the classification of subjects and objects in the system. Access of a subject to an object is granted if some relationship, depending on the access mode, is satisfied between the classification of the subject and object.

One form of discretionary access control model is the Role Based Access Control (RBAC). The central notion of RBAC is that accesses are associated with roles and users are assigned to appropriate roles thereby acquiring accesses. In general, this helps to simplify the management of access control. Roles are created according to the job functions in an organization and users are assigned roles based on their qualifications, experience or responsibilities.

Logic based approaches have been developed by many researchers recently for the purpose of formalizing authorization specifications and evaluations. The advantage of this methodology is to separate policies from implementation mechanisms, to give policies precise semantics, and to provide a unified framework that can support multiple policies. We will talk about access control models in more detail in Chapter 2.

There are several fundamental issues which can decide the basic characteristic of an authorization system. First, the type of administration paradigm the model should follow. Centralised administration allows only one central authorization unit to grant access to subjects, while decentralised administration allows several or many subjects to grant other subjects authorizations to access the objects. In general, decentralised administration is more flexible but also harder to manage.

Secondly, the types of authorizations that should be supported by the model. In general, there are two types of authorizations, positive and negative. Positive authorization means permission, whereas negative authorization means prohibition. Many systems consider only positive authorizations whereas some consider only negative authorizations. A more comprehensive system needs to consider both positive
and negative authorizations, and require a policy to resolve conflicts.

Thirdly, should the model allow for implicit authorizations? In a system that supports implicit authorizations, not every kind of access has to be explicitly specified or granted. Authorizations may be propagated, for instance, from a group to users, along a role-hierarchy or along an object-hierarchy. Authorizations may also be derived from explicit logic rules. Allowing implicit authorizations can usually greatly reduce the size of explicit authorization set.

So far, there has not been much work that address both positive and negative authorizations and resolve conflicts properly in a decentralised authorization administration; especially when the administrative privilege can be delegated between subjects. In this thesis, we develop several models for supporting decentralised authorization administration, both positive and negative authorizations, and conflict resolutions. As discretionary access control represents a flexible way to enforce different protection requirements (especially cooperative yet autonomous requirements), and is more widely applicable, our models are mainly based on discretionary access control. However our logic models based on extended logic programs are general enough to support discretionary access control, mandatory access control and role based access control. In addition, our logic based models allow various implicit authorizations to be derived from the explicit authorization set.

1.2 Administration of Authorizations

One basic issue that we mentioned in the last section is concerned with authorization administration policy, which refers to the function of granting and revoking authorizations [20, 26]. Centralised and decentralised administration are two possible approaches to policy management.

With centralised administration, conceptually a single central authorization authority has the privilege to grant and revoke authorizations, as shown in Figure 1.1. It usually reflects the situation in a single enterprise where authorization is controlled by a single authority. But it is rather inflexible, since usually no individual can know what controls are appropriate for every object/system when the number of objects/systems is very large.
With decentralised administration, on the other hand, multiple authorities (subjects) may have the privilege to grant and revoke authorizations, and the ability to manage administrative privilege can be delegated to multiple subjects. Figure 1.2 presents the framework of decentralised administration.

Decentralised authorization usually follows the ownership paradigm; i.e. every creator of an object possesses rights to access them as well as the ability to grant and delegate authorizations on this object to other subjects. It is rather flexible
and apt to the particular requirements of individual subjects. Many commercial systems adopt such a decentralised approach to authorization. Nevertheless, the authorizations become more difficult to control since multiple subjects can grant and revoke authorizations, and the problem of cascading and cyclic authorization may arise.

On the other hand, authorization delegation has been explored by some authors [1, 32, 35, 73]. Some of the work [1, 35, 73] focus on how to represent and believe the delegations between principals (therefore one principal can exercise the access on behalf of the other) which is closely related with authentication. Some focus on how to represent different types of delegations using roles. However, our work is different from theirs as our major concern is on the delegation of administrative privilege, and moreover, once the delegation occurs, how to resolve conflicts properly by taking into account the delegation path which is explicitly recorded in our models. In the authorization delegation setting, a delegator is the entity that delegates his rights to others, and a delegate is the entity to whom the access rights are delegated.

1.3 Conflict Resolution

First, we introduce the so-called open or closed world assumption. In an open system, the default authorization is access. That is, everything is accessible unless explicitly or implicitly (due to, for example, authorization propagation rules or other authorization reference rules) forbidden. The situation is inverse in a closed system in which the default authorization is forbidden. That is, everything is forbidden unless explicitly or implicitly authorised. A closed system guarantees maximum level of security, while an open system guarantees maximum level of accessibility.

As we mentioned before, a fundamental decision concerning basic attitude of the authorization system is about types of authorizations that are allowed in the system. A system may allow for positive, negative or mixed authorizations. A positive system only gives permissions (often in connection with a closed policy), a negative system allows for only prohibitions (frequently used with an open policy) and a mixed system allows for both permissions and prohibitions. In positive systems, it is burdensome to exclude one particular security subject from a particular kind of
access whereas the inverse situation is true for negative systems. Mixed systems are much more flexible than strictly positive and negative ones, although they may be harder to manage and maintain.

In an access model with both positive and negative authorizations, conflicting situations can arise. If a subject is granted both positive and negative authorizations on the same object, then we say that these two authorizations conflict with each other with respect to this subject. For instance, when a subject $s$ is granted both "read" and "not read" rights on a file $F$ from different subjects (grantors), then these two authorizations are in conflict with each other. Solving authorization conflicts in security policy specification is therefore an important issue in a mixed system. Several previous research work have looked at this issue of conflict resolution policy, though in practice the realisation of such schemes has lagged behind the need.

Currently the proposed conflict resolution policies can be summarised as follows (see references [13, 27, 31, 49, 51, 68]):

- **Negative-take-precedence**: If a conflict occurs on some subject, the negative authorization will take precedence over positive one. In other words, a subject is denied an access if the subject receives both negative and positive authorizations. This policy is in favor of security.

- **Positive-take-precedence**: If a conflict occurs on some subject, the positive authorization will take precedence over negative one. In other words, a subject is authorised an access if the subject receives both negative and positive authorizations. This policy is in favor of accessibility.

- **Strong-and-Weak**: Authorizations are classified into two types, strong and weak. Strong authorizations can not be overridden. Conflicts between strong authorizations are not permitted. The strong authorizations will always override the weak ones when conflict occurs between strong and weak authorizations. When conflict occurs between weak authorizations, the negative one will take precedence.

- **More specific-take-precedence**: The authorization granted to a subject will take precedence over the authorizations granted to a group to which the subject
belongs when conflict occurs. This policy is useful for supporting exceptions; for example, when we want to authorise or deny a group of subjects to access an object except for some particular members.

- *Time-take-precedence:* The new authorization will take precedence over the old one. That is, when a subject receives two conflicting authorizations at different times, the newer one will override the older one.

While the existing conflict resolution methods meet the requirements in most centralised administration models, they may cause problems when the delegation of an administrative privilege is taken into consideration.

## 1.4 Problem Statement

In a flexible access control model, it is necessary to have delegation of access rights between subjects especially in a large decentralised system. Therefore, there could be multiple administrators for a specific object in such a decentralised authorization model. When both positive and negative authorizations are allowed in a decentralised authorization model, conflict problem becomes crucial since multiple administrators greatly increase the chance of conflict and cyclic authorizations may lead to unexpected situations. However, despite its significance, the problem of handling conflicts in authorization delegations has not been much explored by researchers. We have observed that most current conflict resolution methods seem limited when applied to delegation situations.

Let us take a closer look at this issue. Consider for instance the following example. This situation typically occurs within a university framework. Suppose the Dean of a college creates a budget file and then delegates the “read” right † to his/her subordinate Heads of Schools (HOS) so that they can grant “read” or “not read” to the members in their schools. Without proper constraints on the access right delegations, consider what happens if a Head of School grants “not read” to the Dean? And what happens if a staff member in the school receives both “read” and

---

†In this thesis, delegating an access right means granting the administrative privilege on this access right as well as granting the access right itself.
"not read" from the Dean and the School Head respectively? By using the existing conflict resolution methods in Section 1.3, it is not difficult to see that the authorization from the Head of School can override the Dean’s. Therefore, for the first case, this results in the Dean not being able to read the file, which may not be reasonable in practice. The second case as well may not be reasonable because the Head of School receives his/her “grant” right for reading of the file from the Dean. The situation is shown in Figure 1.3.

![Diagram](image)

**Figure 1.3**: An Example of Authorizations in a University

Let us take a closer look at how this could happen. Consider the second case as an example. For negative-take-precedence policy and time-take-precedence policy, this is obviously true, since the negative or new authorization will take precedence. For strong and weak policy, suppose that both the “read” and “not read” granted to the staff from the Dean and the HOS are weak authorizations. Since the HOS’s grant is negative, it will dominate the positive one. Consider the more specific take precedence policy: suppose that the Dean’s grant is for a group to which the staff belongs while the HOS’s grant is for the staff specifically, then the HOS’s grant will override the more general one from the Dean.

The reason that most existing conflict resolution policies seem limited when applied to authorization delegations is that the property of delegation is not taken into consideration. For instance, they suffer from the same problem: when a subject $s_1$
delegates some privilege to another subject $s_2$, $s_1$ can lose control of the delegated privilege with respect to further delegations and grants. This situation can lead to unexpected situations; for instance, $s_2$ may then give back to $s_1$ a negative authorization for the same access privilege. Consider another example within the context of a company. Suppose the chairman creates a file and then delegates its “read” right to each member of the executive committee. Let us assume that each member of the executive committee further delegates this “read” right to his (her) subordinate managers so that they can grant the “read” right to the members in their project teams. In this circumstance, this file’s “read” right has multiple administrators, i.e. the chairman, executive committee members and managers. If the policies mentioned above are used to resolve the conflicts, it may not be possible to prevent the following situation from happening: a manager can grant a “not read” right for the file to the member of the executive committee (the manager’s grantor) or even to the chairman (the manager’s grantor’s grantor); furthermore, this negative authorization can dominate the previous positive one that the member or chairman already has. As a result, the member of the executive committee or the chairman can be denied to read the file. This is certainly not reasonable in practice.

We claim that the problem comes from delegation without any control. This can lead to users not exercising the delegation of access rights since this means that they can lose control of the object and therefore risk sacrificing their privileges. We believe that delegation of rights should be supported in a large-scale decentralised access control system for flexibility reason, but at the same time, the delegation must be under some control to prevent undesirable situations from happening. A promising approach to exercising this control is to adopt an appropriate conflict resolution policy. This dissertation makes contributions towards this direction.

1.5 Summary of Contribution

The contribution of this dissertation lies in the area of flexible decentralised authorization model supporting authorization delegation, both positive and negative authorization, and new conflict resolution policies in authorization delegation. Our contribution includes:
A new conflict resolution policy in authorization delegation

In this dissertation, we propose a conflict resolution policy to achieve a well controlled delegation. The main feature of the method is to trace the delegation path explicitly, and give higher priorities to the predecessors whenever the grantors of conflicting authorizations fall in a path. That is, if $s_1$ delegates or transtively delegates to $s_2$ some privilege, then for this privilege, $s_1$ will have higher priority than $s_2$ whenever their authorization grants conflict over some other subject. Moreover $s_2$ is not allowed to grant $s_1$ any further authorizations on this privilege to avoid cyclic authorizations. In other words, the priority of the subject decreases as the privilege delegation moves from one subject to another, and the subject with lower priority cannot grant authorizations to the subject with higher priority. Assuming all the rights on an object are first delegated from the owner of this object, the owner will always have the highest priority for this object and his/her authorizations can never be overridden. The main advantage of this method is to enforce a well-controlled delegation model. The delegators do not lose control of the objects in the sense that their authorizations will override the delegates whenever conflicts occur between them. Thus users do not need to worry about losing control of the objects by using delegation of rights. For the university example given in the last section, the Dean has higher priority than the Heads of Schools. Therefore, through delegation, the Dean can let the Heads of Schools do most of the granting to their school staff (an advantage of decentralised administration). At the same time, the Dean may also issue certain authorizations to some special staff which will be effective no matter whether they have received other authorizations or not (an advantage of centralised administration). For the company example, on the other hand, since the priorities of subjects for "read" right on the file decreases from the chairman to committee members to managers, the unexpected situation discussed there will not occur. Furthermore, if some member of a team gets both "not read" right on the file from his manager and "read" right from the chairman, the chairman's granted privilege will dominate the other.

Many such examples occur in the real world. Consider, for instance, another
real world example in the context of health care application. When it comes to personal health information, it may be necessary to get consent from the various parties concerned before the information can be released. So such an application will have requirements based on consents, denials (negative authorizations), right to information (positive authorization) and the capacity to create a consent (authorization delegation). Various forms of patient consent should be supported. A patient can in general give a health care professional the capacity to further disclose his/her health information (delegation), because of the need for cooperated care; but she/he may deny consent for various reasons such as the disclosure of some specific information or disclosure to a particular party or a set of parties or even disclosure of information for a particular purpose. For example, a patient John can delegate consent to his family GP but may wish to deny disclosure to a doctor in a hospital. In other words, what the patient requires is a controlled delegation on his/her personal health data in the sense that his/her own consents/denials should not be overridden. According to our conflict resolution method, the authorizations from John will have higher priority than the authorizations from his GP, since the GP gets the capacity to grant from John. Therefore, no matter whether John’s GP grants the doctor access to his health data or not, the doctor cannot access the information because of his negative authorization.

We have observed that the policy proposed in [12] has also considered the use of priority information between subjects to resolve conflicts. In their approach, a pre-defined subject (user) hierarchy is taken into account and if one grantor prevails over the other, then the conflict is resolved in favour of the former. However, we argue that this priority information lacks flexibility since the subject hierarchy is a static one. It is also limited when applied to authorization delegations, especially when the subject hierarchy is different from the delegation hierarchy. For instance, in the university example given before, although a Head of School is in a lower position than the Dean in the hierarchy, he/she should have higher priority than the Dean for the files/objects that were created by himself/herself.

On the other hand, in our method, the priority information comes from the delegation path, which is dynamic and is usually different from object to object. In this
way we get the main advantage of decentralised administration - flexibility, as well as the main advantage of centralised administrations - well controlled. We believe that the proposed method helps to address properly a significant class of conflicts in a system with authorization delegations.

A graph based model supporting authorization delegation and conflict resolution

We first present a graph based authorization framework which can support authorization delegation and both positive and negative authorizations. Three types of arcs are used to denote positive, negative and delegation, respectively. Conflicts are classified into comparable and incomparable ones. With comparable conflicts, the conflicts come from the grantors that have grant connectivity relationship with each other, and the predecessor’s authorizations will always take precedence over the successor’s. With incomparable conflicts, the conflicts come from the grantors that do not have grant connectivity relationship with each other. Multiple resolution policies are provided so that users can select the specific one that best suits their requirements. In addition, the overridden authorizations are still preserved in the system and they can be reactivated when other related authorizations are revoked or the policy for resolving conflicts is changed. Furthermore, cyclic authorizations are avoided and cascade overriding are supported when an administrative privilege is overridden. We give a formal description of our model and describe in detail the algorithms to implement the model. Our model is represented using labelled digraphs, which provides a formal basis for proving the semantic correctness of our model.

A weighted graph based model supporting authorization delegation and conflict resolution

The graph model is then extended to allow grantors to express degrees of certainties about their authorizations. The method is represented using a weighted graph. Each authorization is associated with a weight given by the grantor. The weight is a non-negative number, and a smaller number represents a higher certainty. In
general, to resolve conflicts, the delegation path together with the weighted length of the path is considered, and the authorization on the path of the smaller length will win. This method can deal with the comparable conflicts (grantors of authorizations fall in a path) in a more flexible way, and reduce the amount of incomparable conflicts (grantors do not fall in a path) in the graph model. When all the certainties of authorizations are restricted to be positive and equal, the model is similar to the graph model.

A logic based model supporting authorization delegation, rule inheritance and conflict resolution

We further extend our work to consider more complex domains where subjects, objects and access rights are hierarchically structured and authorization inheritance along the hierarchies are taken into account. To take advantage of strong expressive and reasoning power of logic programming, we propose a logic program based formulation that supports delegatable authorizations, where both negation as failure and classical negation are allowable. Rules inheritance is also supported which can usually greatly simplify the security management. A conflict resolution policy has been developed in our approach which can solve both explicit and implicit conflicts and support the controlled delegation and exception. In our framework, authorization rules are specified in a Delegatable Authorization Program (DAP) which is an extended logic program associated with different types of partial orderings on the domain, and these orderings specify various inheritance relationships among subjects, objects and access rights in the domain. The semantics of a DAP is defined based on the stable model semantics and the conflict resolution is achieved in the process of model generation of the DAP. In addition, several important properties of DAP are investigated. The framework provides users a feasible way to express complex security policy.

A logic based model supporting temporal authorization delegation and temporal conflict resolution

In the real world, there are many situations in which users may need to be
granted or delegated some authorizations for limited periods of time. For example, a part time programmer in a company is allowed to work only during his/her permitted working hours, such as 9am to 1pm. This can be achieved using temporal authorizations, where subjects may grant permissions to others for a certain duration; these permissions are automatically revoked on the expiration of the intervals beginning from the instant that they were initially permitted. To contain a suitable time dimension, we propose a temporal decentralised authorization model in which temporal authorization delegations and negations are allowable. The model is also based on extended logic programs. A conflict resolution method based on the underlying temporal delegation relation and various temporal relations is presented, which can support temporal controlled delegation, temporal suspension and the automatic authorization update. For instance, strategies include realization of temporal suspension by giving higher priority to the authorization with smaller interval, whereas automatic authorization update can be achieved by giving higher priority to the authorization with newer interval.

Apply our authorization model to e-consent on health data

As an application, we show how our authorization model can be used in a e-consent system on health data. A system architecture for e-consent is presented and different types of e-consent models are discussed. We show that our model provides users a good framework for representing and evaluating these e-consent requirements in a health care application.

1.6 Outline of Dissertation

This dissertation is organised as follows. Chapter 2 presents background of access control models.

Chapter 3 reviews basic techniques of graph, weighted graph, logic programming and time denotations that are used in this thesis.

Chapter 4 presents our graph based model supporting decentralised authorization administration and conflict resolution. An authorization state is represented
with a digraph. A flexible conflict resolution method is proposed, and conflict resolution and access control are achieved through graph operations. We give a formal description of our model and describe in detail the algorithms to implement the model.

Chapter 5 constructs a weighted graph based scheme supporting decentralised authorizations with certainties. A more flexible conflict resolution policy is proposed, and the algorithm to implement the model is also given.

Chapter 6 describes the proposed logic programming based model. We present the grammar, the precise semantics and the important properties of the language. In particular, three aspects (delegation correctness, authorization inheritance and conflict resolution) are taken into account for the semantics. We show how the model can be used to express complex security policies.

Chapter 7 presents the proposed temporal decentralised authorization model and temporal conflict resolution policy.

Chapter 8 discusses a specific e-consent application in health care, and show how our proposed model can be used in a real world application.

Finally, Chapter 9 summarises our contribution and presents several directions for future research.

Parts of this dissertation have already published [54, 55, 56, 57, 58, 59, 60, 61, 62].
Chapter 2

Overview of Access Control Models

2.1 Discretionary Access Control

Discretionary access control model [20] has long been a widely accepted and used model in the real world secure systems. It governs the access of subjects to the objects on the basis of the subject's identity and of authorizations (or rules) that specify, for each subject and object in the system, the access rights (e.g., read, write, or execute) the subject is allowed on the object. Objects are the passive entities storing information, such as files, records and fields in records. Subjects are active entities that access the objects, such as users or pieces of executable machine code which are invoked by users. Each request of a user to access an object is checked against the specified authorizations. If there exists an authorization stating that the user can access the object in the specific access right, then the access is granted; otherwise, it is denied. Discretionary access control represents a flexible way to enforce different protection requirements, especially cooperative yet autonomous requirements.

2.1.1 Access Matrix Model

The access matrix model represents the milestone of discretionary access control. The model was originally formulated by Lampson [45] in 1971, then extended by Graham and Denning [37], Conway et al. [22] in 1972. Harrison, Ruzzo, and Ullman (HRU) [38] proposed a further version in 1976. The access matrix is a conceptual
model that specifies the rights that each subject possesses for each object. There is a row in this matrix for each subject, and a column for each object. Each cell of the matrix specifies the access authorised for the subject in the row to the object in the column. The task of access control is to ensure that only those operations authorised by the access matrix actually get executed. An example of an access matrix is shown in Figure 2.1. The subjects shown here are John, Alice, and Bob. There are three files. This matrix specifies that, for example, John can read and write File1 but no access to File 2 or File 3. In a large system, the access matrix will be enormous in size, and most of its cells are likely to be empty. Accordingly the access matrix is very rarely implemented as a matrix. Three common approaches to implementing the access matrix are Access Control Lists (ACL), Capabilities, and Authorization Relations. ACL corresponds to storing the matrix by columns. Each object is associated with a list that indicates, for each subject, the accesses the subject is authorised to execute on the object. Capabilities, on the other hand, corresponds to storing the matrix by rows. Each subject is associated with a list that indicates, for each object, the accesses the subject is authorised to execute on the object. Authorization relations store the matrix as a relation with attributes subject, object and access right. Each tuple of this relation specifies one access right of a subject to an object. The three access control mechanisms implementing the matrix in Figure 2.1 are shown in Figure 2.2, Figure 2.3 and Figure 2.4.

2.1.2 Extensions to the Access Matrix Model

Most discretionary models can be considered as an extension of the access matrix model. A main problem with the access matrix model is its weak safety properties. The safety question in access matrix model is, in general, undecidable. The safety question consists of asking whether or not a given subject can ever acquire a particular access right to a specific object. There is an essential conflict between the expressive power of an access control model and tractability of safety analysis. A consequence of the very broad expressive power of the access control matrix would mean that it has very weak safety properties.

Some work has been done to extend the access matrix model to make the safety
problem decidable. The Take-Grant model [46] is deliberately designed with limited expressive power to eliminate the negative safety of HRU model. It uses a graph structure to represent the system authorization state, which focuses on the problem of propagation of privileges. This model was analyzed by many authors [17, 47]. We will talk about this model in more detail in Chapter 4. Other work includes the Schematic Protection Model proposed by Sandhu [63] and its extension to the Extended Schematic Protection Model by Ammann and Sandhu [4]. Another model that extends the access matrix model where the safety question is decidable is the Typed Access Matrix model [64]. The Typed Access Matrix (TAM) model extends the HRU model with the specification of types for subjects and objects. Each subject/object in the system is assigned, at creation time, a type which remains constant for the whole life of the subject/object. Types and access modes are not predefined in the model but are part of the system definition: they are defined when a system is initialised and thereafter remain constant. The restriction of the TAM model to monotonic operations (that is, revocation of authorization is not considered) has been shown to have decidable safety cases identified on the basis of the types of subjects and objects involved in the operations.
2.2 Mandatory Access Control

Mandatory access control models govern the access to the information by the individuals on the basis of the classification of subjects and objects in the system. The milestone is represented by Bell-LaPadula model [7, 8, 9]. Biba proposed a similar model in 1977 [16] for the purpose of safeguarding information integrity. The two models have been combined in Dion model [25]. In mandatory access control, each user and each object in the system is assigned a security level. The security level associated with an object reflects the sensitivity of the information contained in the object. The security level associated with a user, also called clearance, reflects the user's trustworthiness. The security levels are usually partially ordered. In the military and civilian government areas, for example, the security levels are Top Security(TS), Security(S), Confidential(C), and Unclassified(U), where $TS > S > C > U$. Each security level is said to dominate itself and all others below it.

Access classes are associated with every subject and objects in the system, and the access of a subject to an object is granted if some relationship, depending on the access rights, is satisfied between the classifications of the subject and the object. In particular, the following two principles are required to hold.

**Read down:** A subjects clearance must dominate the security level of the object
being read.

**Write up:** A subject's clearance must be dominated by the security level of the object being written.

Satisfaction of these principles prevents information in high-level objects (i.e., more sensitive) to flow to objects at lower levels. Mandatory access control is mainly used in rigid environments, like those of the military.

### 2.3 Role-Based Access Control

Role-based access control is attracting increasing attention as a security mechanism, particularly for commercial applications. Several models have been published [28, 29, 39, 66, 67] and several commercial implementations are becoming available. A comprehensive framework for RBAC models was defined by RBAC96 [66]. Role-based access control governs users' access to the information on the basis of the users' positions in the organization. It is based on the common practice in organizations of assigning duties and responsibilities to the employees on the basis of their role within the organization. Roles are identified in the system. Then, instead of specifying all the accesses each user is allowed to execute, authorizations are specified for roles. Users are given authorizations indirectly through roles. The user playing a role is
allowed to execute all access rights for which the role is authorised. In general, a user can take on different roles on different occasions. Also the same role can be played by several users, perhaps simultaneously. When a user changes role, for example, we need to take away the user’s current role and assign the user a new role. If authorizations are assigned directly to users, it becomes necessary to revoke all the existing access rights of the user and assign new ones. Since roles in an organization are relatively persistent with respect to user turnover and task re-assignment, role-based access control greatly simplifies security management. Roles can be organised into hierarchical structure, based on the generalization or specialization relation for example, so that access rights can be inherited between roles. For example, a senior role in the hierarchy inherit permissions from juniors. Hierarchial roles can further simplify security management. Role-based access control is a useful approach for many commercial and government organizations.

2.4 Logic Based Access Control

Another approach has been the use of logic based techniques. Several schemes based on logic have been proposed with a view to formalizing authorization specifications and evaluations. The advantage of this approach is to separate policies from imple-
mentation mechanisms and give policies a precise semantics. Abadi et al proposed a modal logic based approach for access control in distributed systems [1]. Their work focuses on how to believe that a principal (subject) is making a request, either on his/her own or on someone else’s behalf. The delegation in their model mainly concerns on the access right itself. Jason Crampton et al’s work is also based on modal logic [23], which investigates the ability of representing and reasoning about the implementation of a real-world access control mechanism. Woo and Lam proposed an expressive language to authorization in distributed systems [76]. In their method, they consider structural properties inherent in authorization and provide formal semantics evaluation which is based on an extended logic program. Jajodia et al also proposed a logical language and illustrated how it can specify authorization, conflict resolution, access control and integrity constraint checking [41, 42]. Bertino et al [12] also proposed a logic framework in which they considered hierarchically structured domain of subjects, objects and access rights for authorization, supported both negation as failure and classical negation, and provided a conflict resolution method. However, the above logic models do not address the issue of delegation for administrative privileges and conflict resolution problem in this context, which is the major concern in this thesis.
Chapter 3

Foundations

This chapter reviews basic techniques of graph, weighted graph, logic programming and time denotations that are used in this thesis.

3.1 Directed Graph and Weighted Directed Graph

In this section, we give a brief review of basic concepts of directed graph and weighted directed graph [36, 52, 53].

Definition 3.1 A directed (simple) graph, or digraph, \((V,E)\) consists of a set of vertices \(V\) and a set of arcs \(E\) that are ordered pairs of elements of \(V\).

Note that multiple arcs from a vertex to a second vertex are not allowed in simple digraphs. Instead, they are allowed in directed multigraphs.

For an arc \((a, b)\) in a digraph, \(a\) is called the *initial vertex* of \((a, b)\), and \(b\) is called the *terminal vertex* of \((a, b)\). *In-arc* of a vertex \(v\) is the arc with \(v\) as its terminal vertex and *in-degree* of a vertex \(v\) is the number of its in-arcs. *Out-arc* of a vertex \(v\) is the arc with \(v\) as its initial vertex and *out-degree* of a vertex \(v\), is the number of its out-arcs. A *path of length* \(n\) from \(a\) to \(b\) is a sequence of one or more arcs \((a, x_1)\) \((x_1, x_2), ..., (x_{n-1}, b)\), denoted by \(a, x_1, \cdots, x_{n-1}, b\), and \(a\) is called the *predecessor* of \(b\), while \(b\) is called the *successor* of \(a\). A path that begins and ends at the same vertex is called a cycle.

If a digraph contains no cycles it is termed *acyclic*. An acyclic digraph is called a *rooted acyclic digraph* if it has a particular vertex, called the root, with the property that there is a path from it to any other vertex of the digraph.
Example 1 Figure 3.1 is a digraph and Figure 3.2 is a rooted acyclic digraph rooted at $a$. In Figure 2.1, $(a, b)$ is the in-arc of vertex $b$, $(b, e)$ and $(b, d)$ are two out-arcs of $b$. Therefore, the in-degree of $b$ is 1 and the out-degree of $b$ is 2. In addition, $a, b, e, d$ is a path from $a$ to $b$ and $a$ is $b$'s predecessor and $b$ is $a$'s successor. □

![Figure 3.1: A Digraph.](image)

![Figure 3.2: A Rooted Acyclic Digraph](image)

Definition 3.2 A weighted directed graph is a digraph that has a number assigned to each arc. The number is called the weight of the arc.

A weighted length of a path is the sum of the weights of the arcs of this path. The weights can be used to indicate a variety of things, such as relative lengths,
time spans and costs. The weighted digraph is perhaps the most widely used graph theoretic structure.

Example 2 The following weighted digraph Figure 3.3 represents a street map, where the streets are all one-way, and the weight of each arc is the distance of the two places, measured in kilometers.

![Figure 3.3: A Weighted Digraph](image)

3.2 Logic Program

In this section, we introduce basic concepts of logic programs [24, 33, 34, 48]

3.2.1 Extended Logic Program

Traditional logic programming language does not contain classical negation ¬. The declarative semantics of logic programming automatically applies the closed world assumption to all predicates, and each ground atom that does not follow from the facts included in the program is assumed to be false. Procedurally, the query evaluation gives the answer no to every query that does not succeed. A limitation of this method is that logic programming does not allow us to deal directly with incomplete information. Extended logic programs are proposed by M. Gelfond and V. Lifschitz in [3] to overcome this limitation, which contain classical negation ¬ in addition to
negation-as-failure. While traditional logic programs provide negative information implicitly through closed-world reasoning, an extended program can include explicit negative information. In the language of extended programs, we can distinguish between a query which fails in the sense that it does not succeed and a query which fails in the stronger sense that its negation succeeds. In the access control context, for example, we can distinguish between the situation that a person is not granted to read a file (negation-as-failure) and the situation that a person is granted "not read" (classical negation, or strong negation) for a file. Classical negation allows us to explicitly express that some access is forbidden.

An extended logic program (ELP) is a set of rules of the form

$$L_0 \leftarrow L_1, ..., L_m, \neg L_{m+1}, ..., \neg L_n$$

Where $0 \leq m \leq n$, and each $L_i$ is a literal. A literal is a formula of the form $A$ or $\neg A$, where $A$ is an atom. The negation sign in the negative literal $\neg A$ represents classical negation, while "not" in the expression of the form $\neg L$ represents negation-as-failure. Literal $A$ and $\neg A$ are complementary.

The semantics of an extended logic program $P$ is defined in terms of its ground instantiation $\text{ground}(P)$. This contains all ground instances of rules over the Herbrand universe that is generated by the function and constant symbols occurring in $P$.

A interpretation is a set $I$ of ground literals such that no complementary literals belong to $I$ simultaneously.

Given an interpretation $I$ and an extended logic program $P$, the Gelfond-Lifschitz reduction, $P^I$, of $P$ with respect to $I$ is the program obtained from $\text{ground}(P)$ as follows:

1. remove each rule that has a formula $\neg L$ in its body with $L \in I$, and
2. remove all formulas of the form $\neg L$ in the bodies of the remaining rules.

Given a set $R$ of ground rules, we denote by $\text{pos}(R)$ the positive version of $R$, obtained from $R$ by considering each negative literal $\neg p(t_1, ..., t_n)$ as a positive one with predicate symbol $\neg p$.

Let $I$ be an interpretation for $P$. We say that $I$ is an answer set for $P$ if $I$ is a minimal model of the positive version $\text{pos}(P^I)$. 
Example 3 Consider the extended program consisting of just one rule:

\[
\neg \text{write(file)} \leftarrow \neg \text{read(file)}.
\]

Intuitively, this rule means, “write(file) can not be granted if there is no evidence that read(file) is granted”. The only answer set of this program is \(\neg \text{write(file)}\).

In general, an extended logic program may possess none, one, or several answer sets.

3.2.2 Normal and Stratified Logic Program

If no classical negation occurs in \(P\), i.e., the only form of negation is negation-as-failure in rule bodies, then \(P\) is called a normal logic program.

An interpretation \(I\) of a normal logic program \(P\) is a stable model of \(P\) if \(I\) is the minimal model of \(P'\).

Definition 3.3 The dependency graph of a normal program \(P\) has a node for each predicate symbol occurring in \(P\) and a directed edge from the node for predicate \(Q\) to the node for predicate \(R\) whenever predicate \(Q\) is in the body of some rule and \(R\) is in the head of the same rule. An edge from node \(Q\) to node \(R\) is positive iff there is a rule \(r\) in \(P\) in which \(R\) is in the head of \(r\), and \(Q\) is the predicate symbol of a positive literal in the body of \(r\). The edge is negative if \(Q\) is the predicate symbol of a negative literal in the body of \(r\).

Definition 3.4 A normal program is stratified if each cycle of the dependency graph is formed by positive edges only, although the remainder of the graph may contain negative edges.

In general, a normal logic program \(P\) may have zero, one or multiple stable models. However, every stratified \(P\) has a unique stable model.
Example 4 Let $P_1$ be the following program, which means $\text{read(file)}$ and $\text{write(file)}$ can be granted exclusively.

\[
\text{read(file)} \leftarrow \text{not write(file)} \\
\text{write(file)} \leftarrow \text{not read(file)}
\]

Its dependency graph is shown in Figure 3.4. Obviously it is not stratified, and $\{\text{read(file)}\}$ and $\{\text{write(file)}\}$ are two stable models of $P_1$. Let $P_2$ be the following program.

\[
\text{read(file)} \leftarrow \text{not write(file)} \\
\text{write(file)} \leftarrow
\]

Its dependency graph is shown in Figure 3.5. It is stratified and the only stable model for it is $\{\text{write(file)}\}$.

![dependency digraph](image)

**Figure 3.4:** Dependency Digraph of $P_1$

### 3.3 Time Points and Intervals

Two important and often used representations of time are the time point and temporal interval [2, 3]. A time point is a zero-length moment in time, such as “1:00 PM.” By contrast, a temporal interval consists of time duration, such as “5 hours”. Intervals can be represented by modeling their endpoints, or instants such as “9:00
§3.3 Time Points and Intervals

**Figure 3.5: Dependency Digraph of P₂**

AM to 4:00 PM.". Interval and instant-based representation of time are widely investigated in the study of time. Assuming a model consisting of a fully ordered set of points of time, an interval is an ordered pair of points $a$ and $b$, denoted by $[a, b]$, with the first point $a$ less than the second $b$. The length of such an interval is identified by $b - a$. Although relative timing between two intervals can be determined from these endpoints, in many cases, sometimes it is more convenient to specify intervals with respect to each other. That is, decouple the intervals from an absolute time references and capture the temporal relations of intervals. Given any two intervals, there are 13 distinct ways in which they can be related [2]. These relations indicate how two intervals relate in time; whether they overlap, equal or before, etc. The 13 relations can be represented by seven cases because six of them are inverses.

We will use a linear model of time, and instants are represented by integers or rational numbers. The following are the time related predicates and functions that can be used in our framework.

- Time point predicates and functions: $=$, $<$, $\leq$, $+$.  

- Interval predicates and functions: their use and definitions are those developed by Allen in [2].
<table>
<thead>
<tr>
<th>Relation</th>
<th>Symbol</th>
<th>Pictorial Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>x before y</td>
<td>(\prec)</td>
<td>xxx yyy</td>
</tr>
<tr>
<td>x equal y</td>
<td>=</td>
<td>xxx</td>
</tr>
<tr>
<td></td>
<td></td>
<td>yyy</td>
</tr>
<tr>
<td>x meets y</td>
<td>m</td>
<td>xxxyy y</td>
</tr>
<tr>
<td>x overlaps y</td>
<td>o</td>
<td>xxx</td>
</tr>
<tr>
<td></td>
<td></td>
<td>yyy</td>
</tr>
<tr>
<td>x during y</td>
<td>d</td>
<td>xxx</td>
</tr>
<tr>
<td></td>
<td></td>
<td>yyyyy y</td>
</tr>
<tr>
<td>x starts y</td>
<td>s</td>
<td>xxx</td>
</tr>
<tr>
<td></td>
<td></td>
<td>yyyyy y</td>
</tr>
<tr>
<td>x finishes y</td>
<td>f</td>
<td>xxx</td>
</tr>
<tr>
<td></td>
<td></td>
<td>yyyyy y</td>
</tr>
</tbody>
</table>

\(x \text{ dur } y = x d y \lor x s y \lor x f y.\)

We will use \(\subseteq\) to denote \(\text{dur}\), and \(x \subseteq y = x \subseteq y \lor x = y\)

\(\cap\) is the interval intersection function; \(\emptyset\) is the empty interval. Open ended intervals are also allowed. \((-\infty, t] = \{x/x \leq t\}, [t, +\infty) = \{x/x \geq t\}\), and \((-\infty, +\infty) = (-\infty, t] \cup [t, +\infty)\)

- Mixed point-interval predicates: \(\text{in}\) with type \(\mathcal{T} \times \mathcal{I}\). \(\text{in}(t, i)\) is true if \(t\) is inside \(i\), which is formally defined as \(\text{in}(t, i) = [t, t] \subseteq i\).
Chapter 4

A Graph Based Authorization Model

The chapter is organised as follows. In Section 1, we discuss related work that have been previously proposed. In Section 2, we propose a formal model for authorization and conflict resolution. In Section 3, we present the relevant algorithms that implement our model. In Section 4, we briefly consider authorization state transformations based on our model, while in Section 5 we apply the model to a distributed environment. Finally, in Section 6 we summarise the major contributions in this chapter.

4.1 Related Works

In terms of the specification of authorization models, some approaches in the past have used a graph structure to represent the underlying semantics. Perhaps the most famous one is the Take-Grant model. It was first proposed by Jones et al. [43]; subsequently various extensions have been proposed to overcome some of the disadvantages it suffers from [19, 30, 44, 46, 47]. Among these, the Action-Entity model [19, 30] considers a richer set of administrative privileges and supports predicates in authorizations. The Take-Grant model uses a graph structure to denote the system authorization state and to represent the transfer of the privileges. In the graph, the nodes represent the subjects and objects in the system, and the labelled arcs represent the access authorizations. An arc labelled with one or more access rights indicates that the source vertex is authorised to perform the rights on the destination vertex.
The model considers the following four privileges:

Read. An arc labelled with ‘read’ from vertex $A$ to vertex $B$ indicates that the subject (or object) represented by $A$ can read the object represented by $B$.

Write. An arc labelled with ‘write’ from vertex $A$ to vertex $B$ indicates that the subject (or object) represented by $A$ can write into the object represented by $B$.

Take. An arc labelled with ‘take’ from vertex $A$ to vertex $B$ indicates that $A$ is enabled to ‘take’ any right held by $B$ on other subjects or objects in the system.

Grant. An arc labelled with ‘grant’ from vertex $A$ to vertex $B$ indicates that $A$ can ‘grant’ to $B$ any of the rights held by $A$ on other subjects or objects in the system.

Four operations, take, grant, create and remove, are considered in the model. Figure 4.1 is an example of graph denotation and operation in the Take-Grant model.

![Figure 4.1: An Example of Graph Denotation and Operation in Take-Grant Model](image)

However the Take-Grant approach and the extensions did not consider negative authorizations; therefore conflict problem is not an issue, whereas in this thesis, the conflict problem is the major concern. In our model, we allow both positive and negative authorizations, enable the transfer of every specific access right, and provide a flexible way to handle the conflict problem. We also use a graph structure to represent the semantics but, as will be shown later (Section 4.2.2), the denotation
is different from the one used by the Take-Grant model.

### 4.2 The Authorization Delegation and Conflict Resolution Model

In this section, we outline the basic idea and provide a formal description of our authorization conflict resolution model.

#### 4.2.1 Basic Ideas

In our authorization model, we allow both positive and negative authorizations, and permit access rights to be delegated from one subject to another. So, for any access right on any object, there may be multiple administrators that can grant authorizations. Different to the previously proposed conflict resolution policies, we classify conflict authorizations into two categories namely *comparable* and *incomparable* conflicts. Consider the situation where a subject $s_3$ is granted two conflicting authorizations with respect to an access right $r$ on an object $o$ from subjects $s_1$ and $s_2$ respectively. We say that these two conflicting authorizations are comparable if $s_3$’s administrative privilege for $r$ on $o$ is granted (or transitively granted) by $s_1$, or vice versa. In the first case we assign a higher priority to $s_1$’s grant than $s_2$’s grant to solve the conflict occurring over the subject $s_3$. On the other hand, if there is no grant connectivity relationship between $s_1$ and $s_2$, then this conflicting authorization is said to be incomparable. In our model, we support multiple policies to solve incomparable conflicts to meet different user’s requirements. For example, we may use the positive authorization to override the negative authorization or vice versa. We require that all the rights of an object be originally delegated from the owner of the object, so that the owner’s authorization will take precedence over any other conflicting authorizations.

In addition, although some authorizations may be overridden by other authorizations, they are not eliminated from the authorization state. We preserve all the authorizations; they are either revoked explicitly or by cascaded revocation. So conflicting authorizations can be present simultaneously in our model. The main
advantages of this approach include the ability of re-activating the overridden authorizations after the other related authorizations are revoked, and the ability of changing the policy of incomparable conflict resolution.

Now let us consider some examples. Suppose a subject \( s_1 \) has created a file \( F \) and delegated its "read" right to \( s_2 \) and \( s_3 \). After some time, \( s_1 \) grants \( s_4 \) "not read" right but \( s_2 \) grants \( s_4 \) "read" right on the same file \( F \). This is the case that we call comparable conflict, and \( s_1 \)'s grant will take precedence over \( s_2 \)'s since \( s_2 \) gets its delegated "read" right on \( F \) from \( s_1 \). So \( s_4 \) cannot read file \( F \). Now consider another situation that \( s_2 \) and \( s_3 \) grant \( s_5 \) "read" and "not read" on file \( F \) respectively. This is the case we refer to as incomparable conflict, and \( s_5 \) may or may not read the file \( F \) depending on the strategy that is being used with incomparable conflicts. The strategy is selected by \( F \)'s owner \( s_1 \) based on the sensitivity of the object (vis-a-vis the owner). As we mentioned earlier, the overridden authorization still remains in the system, so that after sometime, in the first case, if \( s_1 \) revoked its authorization to \( s_4 \), then \( s_4 \) can read \( F \) because of the grant from \( s_2 \).

In the next section, we specify our authorization and conflict resolution model using digraphs. We believe that such a notation presents an intuitive interpretation of the underlying semantics.

### 4.2.2 Notation And Definitions

Let \( S \) be a finite set of subjects (users), \( O \) be a finite set of objects (files, relations), \( R \) be a finite set of access rights (e.g. read, write, select, etc.), and \( T \) be a finite set of grant types. Then we have the following definition for authorization.

**Definition 4.1 (Authorization)** An authorization is a 5-ary tuple \((s, o, t, r, g)\), where \( s \in S, o \in O, t \in T, r \in R, g \in S \).

Intuitively, an authorization \((s, o, t, r, g)\) states that a grantor \( g \) has granted subject \( s \) the access right \( r \) on object \( o \) with grant type \( t \). We will consider three grant types: \( T = \{*, +, -\} \), where

* \( * \): delegatable, which means that the subject has been granted the access right \( r \) on \( o \) as well as the privilege for administration of \( r \) on \( o \).
+ : positive, which means that the subject has been granted the access right \( r \) on \( o \).

- : negative, which means that the subject has been denied the access right \( r \) on \( o \).

For example, \((\text{user}_1, \text{file}_1, +/-, \text{read}, \text{user}_2)\) states that \(\text{user}_1\) is granted/denied to "read" \(\text{file}_1\) by \(\text{user}_2\), and \((\text{user}_1, \text{file}_1, *, \text{read}, \text{user}_2)\) states that \(\text{user}_1\) is granted by \(\text{user}_2\) not only the privilege to "read" \(\text{file}_1\), but also the privilege to grant authorizations with respect to the "read" right on \(\text{file}_1\) to other subjects.

Note that \(*\) means \(+\) together with administrative privilege on an access. The administrative privilege is related to a specific access right on an object. That is, a subject may have the administrative privilege for "read" but not for "write".

**Definition 4.2 (Authorization State)** An authorization state is the set of all authorizations at a given time.

In this chapter, we will usually use \(\mathcal{A}\) and \(a\) (possibly with subscripts) to denote an authorization set and a single authorization respectively, and use \(a.s, a.o, a.t, a.r, a.g\) to denote the corresponding components of subject, object, type, right and grantor of \(a\) respectively.

In order to formalise our approach, we use a *labelled digraph* to represent an authorization state as follows. For every object \(o\), \(G_o\) is used to represent all the authorizations with respect to the object \(o\). Let \(G_o = (V, E, t, l)\) be a labelled digraph, where \(V\) is a finite set of vertices representing the subjects that hold some authorizations on \(o\), \(E\) is a finite set of arcs such that if there exists an authorization \((s, o, t, r, g)\) in authorization state \(\mathcal{A}\), then \((g, s)\) is in \(E\), \(t\) is a type function from \(E\) to \(T\), which maps every arc in \(E\) to a specific type in \(T\). We will use different types of arcs, denoted as \(t(e)\), to represent different grant types, as shown in Figure 4.2. Suppose \(E_*, E_+, E_-\) denote the sets of \(\ast\) arcs, \(+\) arcs, and \(-\) arcs respectively. Then \(E = E_* \cup E_+ \cup E_-\). \(l\) is a label function from \(E\) to the power set of \(R\), which maps every arc \((g, s)\) of type \(t\) in \(E\) to a set of rights on \(o\) that \(g\) grants to \(s\) and the grant type is \(t\). For instance, if \(t((g, s)) = \ast\), then \(l((g, s)) = \{r \mid \exists (s, o, \ast, r, g) \in \mathcal{A}\}\). In \(G_o\), every arc \(e\) is labelled with \(l(e)\). In the rest of this chapter, we will sometimes
omit $t$ and $l$ and simply write $G = (V, E)$, whenever there is no confusion in the context. Following this, an authorization state $A$ can be represented by a digraph $G$, which is a set of $G_o$ for all objects $o$ in the system. That is, $G = \{G_o \mid o \in O\}$.

**Example 5** Suppose we have following authorizations on object $o$, then $G_o$ is shown in Figure 4.3.

\[(S_2, o, *, R, S_1)\]
\[(S_2, o, *, W, S_1)\]
\[(S_9, o, +, W, S_1)\]
\[(S_3, o, *, R, S_1)\]
\[(S_3, o, *, W, S_1)\]
\[(S_4, o, *, R, S_2)\]
\[(S_6, o, -, R, S_2)\]
\[(S_6, o, -, W, S_2)\]
\[(S_5, o, *, R, S_3)\]
\[(S_5, o, *, W, S_3)\]
\[(S_6, o, *, R, S_4)\]
\[(S_7, o, *, R, S_4)\]
\[(S_7, o, -, R, S_5)\]
\[(S_7, o, -, R, S_6)\]
\[(S_9, o, *, R, S_6)\]
\[(S_8, o, *, R, S_7)\]
Figure 4.3: $G_o$: an example of graph representation of authorizations on an object $o$, where $S_1, ..., S_9$ are subjects, $R$ and $W$ are access rights.

In addition, we will use $G_{o,r}$ to denote all the authorizations with respect to a specific access right $r$ on $o$. That is, $G_{o,r}$ is a subgraph of $G_o = (V, E, t, l)$ that contains all arcs with the label containing $r$ and the corresponding vertices. More formally, $G_{o,r} = (V', E', t')$, where $E' = \{(s_1, s_2) \mid (s_1, s_2) \in E \text{ and } r \in l((s_1, s_2))\}$, $V' = \{v \mid \exists u'(v, v') \in E' \text{ or } (v', v) \in E'\}$, and for any $e' \in E'$, $t'(e') = t(e')$. Note that there is no need for arc labels in $G_{o,r}$ anymore. For example, with reference to the $G_o$ denoted in Figure 4.3, $G_{o,R}$ and $G_{o,W}$ can be illustrated in Figure 4.4 and Figure 4.5 respectively.

Figure 4.4: $G_{o,R}$ of Figure 4.3
Now we can define a binary relation on the set of subjects.

Definition 4.3 (Grant Connectivity Relation $<_\text{o,r}$ on Subjects) Given an authorization state $\mathcal{A}$, for any subjects $s_1, s_2 \in S$, object $o \in O$, and access right $r \in R$, we say that $s_1$ is grant-connected to $s_2$ with respect to $r$ and $o$ in $\mathcal{A}$, denoted by $s_1 <_{\text{o,r}} s_2$, if there exists an authorization $(s_2, o, t, r, s_1)$ for some $t$ in $\mathcal{A}$, or there exists some subject $s_3$ satisfying $s_1 <_{\text{o,r}} s_3$, and $s_3 <_{\text{o,r}} s_2$.

$s_1 <_{\text{o,r}} s_2$ means that there exists a sequence of subjects $s_1, x_1, x_2, \ldots, x_n, s_2$ such that $(x_1, o, t_0, r, s_1)$, $(x_2, o, t_1, r, x_1)$, $\ldots$, $(s_2, o, t_n, r, x_n)$ are all in the authorization state. In terms of our graph notation, $s_1 <_{\text{o,r}} s_2$ if and only if there exists a path from $s_1$ to $s_2$ in $G_{\text{o,r}}$, or in other words, $s_1$ is the predecessor of $s_2$ and $s_2$ is the successor of $s_1$ in $G_{\text{o,r}}$. The grant connectivity relation provides us with an important priority information about the subjects, which will be used later to solve the conflict problem. When the object and right are clear in the context, we sometimes simply write $s_1 < s_2$. For example, in the digraph $G_{\text{o,R}}$ of Figure 4.4, we have:

$S_1 < S_2 < S_4 < S_6 < S_7 < S_8$,  
$S_1 < S_2 < S_4 < S_6 < S_9$, and  
$S_1 < S_3 < S_5 < S_7 < S_8$.  

\[ \text{Figure 4.5: } G_{\text{o,W}} \text{ of Figure 4.3.} \]
4.2.3 Formal Description of the Model

We say that an authorization state $\mathcal{A}$ is delegation correct, if for any subject $s$, object $o$ and right $r$, $s$ can grant $r$ on $o$ to other subjects if and only if $s$ has been granted $r$ on $o$ with delegation type $\ast$, that is, $\exists g, (s, o, \ast, r, g) \in \mathcal{A}$. In our graph representation, this means that in $G_{o,r}$, only the vertices pointed to by at least one $\ast$ arc can have out-arcs, while the vertices pointed to only by $+$ or $-$ arcs must be terminal ones, that is, their out-degrees must be zero. We assume that for every object $o$, only the owner of $o$, denoted by $s_o$, has been implicitly granted all the rights on $o$ with delegatable type by the system when the object is created. So if a state is delegation correct, then there will be a path from $s_o$ to any other vertex in $G_{o,r}$.

We say that an authorization $a_1$ contradicts an authorization $a_2$ if $a_1.s = a_2.s$, $a_1.o = a_2.o, a_1.g = a_2.g, a_1.r = a_2.r$, but $a_1.t \neq a_2.t$. The contradictory authorizations state that a grantor gives the same subject two different types of authorizations over the same object with the same access right. For example, authorizations $(s_2, F_1, \ast, R, s_1), (s_2, F_1, +, R, s_1)$ and $(s_2, F_1, -, R, s_1)$ contradict each other. Figure 4.6 gives the corresponding graph representation. An authorization state $\mathcal{A}$ is not contradictory if for any $a$ and $a'$ in $\mathcal{A}$, $a$ does not contradict $a'$. In our graph representation, this means that in any $G_{o,r}$ there is only one arc from each vertex to another.

![Diagram](image)

**Figure 4.6:** Contradictory grants on object $F_1$.

**Definition 4.4** (Consistent Authorization State) An authorization state is con-
sistent if it satisfies the following three conditions:

1. it is delegation correct,

2. it is not contradictory, and

3. for any object \( o \) and access right \( r \), \( <_{o,r} \) is a strict partial order.

Recall that a strict partial order is transitive and anti-symmetric. In our graph notation, requiring relation \( <_{o,r} \) to be a strict partial order means that the corresponding \( G_{o,r} \) is acyclic. In fact, by considering the properties of delegation correctness and not contradictory together, we have the following theorem:

**Theorem 4.1** Let \( A \) be a consistent state, then for any object \( o \) and access right \( r \), \( G_{o,r} \) in \( A \) is a simple rooted acyclic digraph, with the owner of the object as the root.

Remember that in a simple graph there are no multiple arcs between each pair of vertices. Also in the rooted acyclic graph, from the root one can reach any vertex in the graph. This theorem is easy to prove using the definition of consistent authorization state; so we omit the proof here. Figure 4.7 shows three examples of inconsistent authorization state, where \( G_{1,o,r} \) is not delegation correct because of the arc \( (s_2, s_3) \); \( G_{2,o,r} \) is contradictory because there are two arcs from \( s_5 \) to \( s_6 \); and \( G_{3,o,r} \) is cyclic because of the cycle \( s_8, s_9, s_8 \).

By requiring \( G_{o,r} \) acyclic, we have the following: if a subject \( s \) receives an authorization directly or indirectly from another subject \( s' \) on some object \( o \) and access right \( r \), then \( s \) cannot grant \( s' \) any further authorization on \( o \) and \( r \) later on. In this way, we can solve the problem that exists in most conflict resolution methods discussed in Chapter 1.

For a consistent authorization state \( A \) and a single authorization \( a \), if \( A \cup \{a\} \) is still consistent, then we call \( a \) is consistent with \( A \). In our model, we require that the authorization state should always be consistent.

**Definition 4.5 (Conflicting Authorizations)** For any two authorizations \( a_1 \) and \( a_2 \) in \( A \), \( a_1 \) conflicts with \( a_2 \) if \( a_1.s = a_2.s \), \( a_1.o = a_2.o \), \( a_1.r = a_2.r \), \( a_1.t \neq a_2.t \) and \( a_1.g \neq a_2.g \).
From the definition, two authorizations are in conflict if they have the same subject, object and access right, but have different grant types and grantors. In our graph $G_{o,r}$, this means that the conflicting arcs have the same terminal vertex but different initial vertices and arc types. Since there are three grant types in our model, three kinds of conflicts may arise, as illustrated in Figure 4.8.

Note that type * and + are considered conflicting in the sense that * holds the administrative privilege on an access right while + does not. Conflicts are additionally classified into comparable conflicts and incomparable conflicts as follows.

**Definition 4.6 (Comparable Conflicts)** Suppose $a_1$ and $a_2$ are any two conflicting authorizations on object $o$ and access right $r$. Then $a_1$ and $a_2$ are comparable if $a_1.g <_{o,r} a_2.g$ or $a_2.g <_{o,r} a_1.g$. Otherwise they are incomparable.

In other words, two conflicting authorizations are comparable if their grantors are grant-connected to each other. In our graph $G_{o,r}$, two conflicting arcs are comparable if there is a path between their initial vertices. For example, in Figure 4.4, $(s_2, s_6)$ and $(s_4, s_6), (s_4, s_7)$ and $(s_6, s_7)$ are two pairs of comparable conflicts, while $(s_4, s_7)$ and $(s_5, s_7)$ are pairs of incomparable conflicts.

In fact, the grantors are comparable in comparable conflicts. In the grant relation path, we have higher priorities for the predecessors than the successors. So, when authorizations conflict with each other, the predecessor's grant will take precedence over the successor's. This idea can be formalised by the following overriding rule.
Definition 4.7 (Overriding Rule) For any two authorizations $a_1$ and $a_2$ in $A$, $a_1$ overrides $a_2$ if $a_1.s = a_2.s$, $a_1.o = a_2.o$, $a_1.r = a_2.r$, and $a_1.g <_{o,r} a_2.g$. An authorization is inactive if there exists some authorization that overrides it. Otherwise it is active. We use $\text{Act}(A)$ to denote the set of all active authorizations in an authorization state $A$.

The overriding rule tells us that if two authorizations are about the same subjects, objects and rights, and their grantors are grant-connected to each other, then the authorization from the predecessor will override the one from the successor. Note that this definition does not require the grant types of the two authorizations to be different. Hence the predecessor’s authorization will override the successor’s even though they are not in conflict. This is reasonable since this means that the two authorizations are identical except for the grantor.

Correspondingly in the graph $G_{o,r}$ for some $o$ and $r$, if two arcs point to the same vertex, and there is a path between their initial vertices, then the arc from the predecessor will override the arc from the successor. Let $G$ be the graph corresponding to an authorization state $A$; then active graph of $G$, denoted by $\text{Act}(G)$, is the subgraph of $G$ that contains only active arcs. It is easy to show that for any $G_{o,r}$, $\text{Act}(G_{o,r})$ is still a rooted acyclic graph, since by using the overriding rule, the in-degrees of some vertices may be reduced but not to zero. But $\text{Act}(G_{o,r})$ may become inconsistent. For example, in Figure 4.4, $(s_2, s_6)$ overrides $(s_4, s_6)$ because $s_2$ is $s_4$’s predecessor. For the same reason $(s_4, s_7)$ overrides $(s_6, s_7)$. Figure 4.9 gives the active graph of Figure 4.4. Note that it is inconsistent because the arc $(s_6, s_9)$ is not delegation correct.

Definition 4.8 (Effective State) If an authorization state $A$ is consistent, then
the maximal consistent subset of \( \text{Act}(A) \) forms the effective state of \( A \), denoted by \( \text{Eff}(A) \).

Let \( G \) be the graph corresponding to an authorization state \( A \), then the effective graph of \( G \), denoted by \( \text{Eff}(G) \), corresponds to \( \text{Eff}(A) \). \( \text{Eff}(G) \) is in fact a set of \( \text{Eff}(G_o,r) \) for all objects \( o \) and access rights \( r \) of the system. Note that in the effective state, we have already eliminated all the comparable conflicts, that is, the conflicts in which their grantors are grant-connected to each other. Hence only the incomparable conflicts exist. Figure 4.10 gives effective graph of Figure 3.

**Theorem 4.2** A consistent authorization state \( A \) has a unique effective state \( \text{Eff}(A) \).
Proof. Obviously $\text{Act}(A)$ is unique. So we only need to prove that the maximal consistent subset of $\text{Act}(A)$ is unique.

Let $A_1$ and $A_2$ be two maximal consistent subsets of $\text{Act}(A)$, and $a = (s, o, t, r, g)$ be any authorization in $A_1$. Then there should be a corresponding arc $(g, s)$ in $G''_{o,r}$ of $A_1$. We need to prove that $(g, s)$ is also in $G''_{o,r}$ of $A_2$. Since $A_1$ and $A_2$ are both consistent, $G'_{o,r}$ and $G''_{o,r}$ are both rooted acyclic graph with root $s_o$. Let $\text{maxlen}(g)$ denote the length of the longest paths from the root $s_o$ to $g$ in $G'_{o,r}$. We will prove by induction of the $\text{maxlen}(g)$.

When $\text{maxlen}(g) = 0$, $g$ is the root of $G'_{o,r}$ representing the owner of object $o$, and hence the result is certainly true. Suppose that the result is true when $\text{maxlen}(g) \leq k$. Consider the case when $\text{maxlen}(g) = k + 1$. Suppose $(g, s)$ is not in $G''_{o,r}$, then since $A_2$ is a maximal consistent subset of $\text{Act}(A)$, $(g, s)$ must be not consistent with $G''_{o,r}$. But $(g, s)$ can not make $G''_{o,r}$ contradictory (i.e. there is more than one arc from $g$ to $s$) or cyclic, since $G''_{o,r} \cup (g, s)$ is still a subgraph of $G_{o,r}$ of $A$ and this will lead to $A$ to be inconsistent. So $(g, s)$ must make $G''_{o,r}$ to delegation incorrect. This means that there is no in-arc of $g$ with $\star$ type in $G''_{o,r}$. From the inductive hypothesis, all the in-arcs of $g$ in $G'_{o,r}$ will be in $G''_{o,r}$ too, and hence this will lead to $G'_{o,r}$ being not delegation correct. This is a contradiction. So $(g, s)$ is also in $G''_{o,r}$. This concludes that $A_1 \subseteq A_2$. For the same reason $A_2 \subseteq A_1$. Thus $A_1 = A_2$. □

Now let us consider the incomparable conflicts. We call an authorization state $A$ is conflict-free if for any $a_1 \in A$ and $a_2 \in A$, $a_1$ is not in conflict with $a_2$.

**Definition 4.9 (Stable State)** If an authorization state $A$ is consistent, then the maximal consistent and conflict-free subset of $\text{Eff}(A)$ forms a stable state of $A$, denoted as $\text{stable}(A)$.

Note that an authorization state may have more than one stable state. In theory, one stable state presents one resolution to incomparable conflicts. Let $G$ be the graph corresponding to an authorization state $A$, then the stable graph of $G$, denoted by $\text{stable}(G)$, corresponds to $\text{stable}(A)$. $\text{stable}(G)$ is in fact a set of $\text{stable}(G_{o,r})$ for all objects $o$ and access rights $r$ in the system. Figure 4.11 and Figure 4.12 are two stable graphs of Figure 4.4.
For incomparable conflicts, we cannot resolve them using their grantor’s priorities, since their priorities are not comparable. In our model we can support different strategies for resolving incomparable conflicts by evaluating different stable states. For example, we can support the following three strategies according to the grant types of authorizations:

1. Pessimistic: the priority sequence is $- > + > *$;

2. Optimistic: the priority sequence is $* > + > -$;

3. Any: the priority sequence is $* = + = -$.
Hence, with multiple grantors when incomparable conflicts occur, these are resolved according to the types of authorizations above since grantors are not comparable. A user can select the appropriate strategy that best suits the needs of his/her application. Even in one application, the strategy can vary from object to object. For example, for some objects that are very confidential, one can select the pessimistic strategy; for other objects that are not that sensitive, one can select the optimistic strategy. One can tell the system which strategy to apply to an object when the object is created, and one can change the strategy later when the sensitivity of the object is changed. The stable state should be recalculated according to the new policy when incomparable conflict resolution strategy is changed.

Another possible strategy for resolving incomparable conflicts is to grant an additional authorization to the subject over whom the conflicts occur by a common predecessor of the grantors of these conflicting authorizations, in particular, by the owner of the object. This technique can be used to change the incomparable conflicts to comparable conflicts, which can then be resolved. In fact, the common predecessor here acts as a judge in the sense that his/her decision has higher priority and hence can resolve the dispute.

Note that in our model, the overridden authorizations are not removed. That is, the inactive paths are kept. This is important as they can become active again, for instance when the overriding authorizations are revoked or the incomparable conflict resolution policy changes. In practice, an inactive authorization can be implemented by a flag bit.

### 4.2.4 Access Control Policy

Now we can define our access control policy. We use 3-ary tuple \((s, o, r)\) to denote an access request to the system, where \(s \in S, o \in O, r \in R\). It states that a subject \(s\) requests to exercise access right \(r\) on object \(o\). Then we have following access control policy.

**Definition 4.10 (Access Control Policy)** Let \(A\) be an authorization state, \((s, o, r)\) be an access request, \(P\) be a policy to resolve the incomparable conflicts on \(o\), and \(\text{stable}(A, P)\) be a stable state of \(A\) when applying \(P\) to \(o\). We say that \((s, o, r)\) is
permitted if there exists some grantor \( g \) such that \((s, o, *, r, g)\) or \((s, o, +, r, g)\) is in \(\text{stable}(A, P)\); \((s, o, r)\) is denied if there is some grantor \( g \) such that \((s, o, -, r, g)\) is in \(\text{stable}(A, P)\); otherwise, \((s, o, r)\) is undecided.

It is worth mentioning that in our model, the system is the implicit grantor of any object's owner for all access rights on this object with delegatable grant type. In practice, the answer \textit{undecided} may be treated as denial too. We prefer to distinguish them here to make the semantics more clear.

According to the access control policy, the procedure of evaluating an access request \((s, o, r)\) can be outlined as follows:

1. If \( s \) is the root of \( G_o \), return \textit{yes};

2. Compute \(\text{stable}(G_{o,r}, P) = (V, E, t)\);

3. If \( s \) is not in \( V \), return \textit{undecided};

4. Let \((g, s)\) be the in-arc of \( s \) in \( E \), if \( t((g, s)) = \ast \) or \(+\) return \textit{yes} else return \textit{no}.

For example, let \( G_o \) be the graph of Figure 4.3, then \((s_4, o, r)\) is allowed, and \((s_6, o, r)\) is denied (see Figure 4.11 or Figure 4.12). \((s_7, o, r)\) is allowed if the optimistic policy is used for incomparable conflicts (see Figure 4.11), or denied if the pessimistic policy is used (see Figure 4.12).

As shown above, to determine an access request \((s, o, r)\), we need to compute \(\text{stable}(G_{o,r})\) from \(\text{Eff}(G_{o,r})\). \(\text{Eff}(G_{o,r})\) is evaluated from \(G_{o,r}\) that is again evaluated from \(G_o\). \(G_{o,r}\) can be easily obtained by selecting all the arcs with \( r \) in their label and corresponding vertices from \(G_o\). For \(\text{Eff}(G_{o,r})\) and \(\text{stable}(G_{o,r})\), we give the detailed algorithms in the next section.

4.3 Algorithms

In this section, we give detailed algorithms to implement our model. We also present the correctness proof and the computational complexity of the algorithms.
4.3.1 Algorithm for Evaluating the Effective Graph

Algorithm 4.1 is used to evaluate the effective graph of a $G_{o,r}$ for some object $o$ and right $r$. The output is a graph $G'$, and $G''$ is a temporary working graph used to construct a topological sorting $^1$ of $<_{o,r}$. The algorithm first copies the root and out-arcs of root in $G_{o,r}$ to $G'$. Then it checks every vertex $v$ of $G_{o,r}$, in terms of the sequence of topological sorting; if its in-arc does not include delegable type $*$ in $G'$, then all of its out-arcs in $G_{o,r}$ will not go to $G'$, in order to keep $G'$ delegation correct. Otherwise, for every out-arc of $v$, say $(v, x)$, if there exists any predecessor $p$ of $v$ in $G_{o,r}$ such that $(p, x)$ is in $G_{o,r}$ (which means that $(v, x)$ is inactive), then $(v, x)$ will not go to $G'$.

Algorithm 4.1: Evaluate_Eff_Graph($G_{o,r}, s_o$)

Input: $G_{o,r} = (V, E, t)$ for some object $o$ and access right $r$, with root $s_o$ and arc type function $t$

Output: $Eff(G_{o,r}) = G' = (V', E', t')$

begin

1. $E' = \{(s_o, x)|(s_o, x) \in E\}$;
2. $V' = \{s_o\} \cup \{x|(s_o, x) \in E'\}$;
3. for all $e' \in E'$ do $t'(e') = t(e')$;
4. $E'' = E - E'$;
5. $V'' = V - \{s_o\}$; (* copy the root and out-arcs of root from $G_{o,r}$ to $G'$ and then copy the remaining part of $G_{o,r}$ to $G''$)

6. for each $v \in V''$ with 0 in-degree do begin
7. if the in-arcs of $v$ in $E'$ include $*$ type then begin
8. $P = \{x|(x, v) \in E\}$;
9. for each $p \in P$ do $P = P \cup \{x|(x, p) \in E\}$; (* compute all predecessors of $v$ in $G_{o,r}$)
10. for each arc $(v, x)$ that goes out from $v$ in $E$ do begin
11. if for each $p \in P$, $(p, x) \notin E$ then begin

\(^1\text{A topological sorting is an extension of a given partial order } \leq \text{ to a linear order } < \text{ in that if } a \leq b \text{ then } a < b.\)
\( E' = E' \cup \{(v, x)\} \);
\( V' = V' \cup \{x\} \);
\( t'(e') = t(e') \);
end
end
\( E'' = E'' - \{(v, x) | (v, x) \in E''\} \);
\( V'' = V'' - \{v\} \);
end

**Theorem 4.3** Algorithm Evaluate\_Eff\_Graph is correct, and \( \text{Eff}(G_{o,r}) \) can be computed by Evaluate\_Eff\_Graph\((G_{o,r}, s_o)\) in \( O(N^3) \) time, where \( N \) is the number of vertices in \( G_{o,r} \).

**Proof.**

To prove the correctness of the algorithm, we first point out the fact that, in the algorithm, the vertices are selected and operated from \( G \) in terms of the topological sorting of the partial order \( <_{o,r} \). This is easy to see from the for loop condition in line 6 and statements in lines 18 and 19. This means that for every vertex \( v \) in \( G \), all its in-arcs that can go to \( G' \) will go to \( G' \) before \( v \) is selected in line 6.

1. \( G' \) is active: The out-arcs of the root are obviously active; for any other arc \((v, x)\), its activeness can be inferred from the statements in line 8 and 9, by evaluating all the predecessors of \( v \), and if statement in line 11, preventing \((v, x)\) from being added to \( G' \) if there is any arc from \( v \)'s predecessors to \( x \).

2. \( G' \) is consistent: Since \( G_{o,r} \) is consistent, \( G' \subseteq G_{o,r} \), we only need to prove \( G' \) is delegation correct. And this is true because for any vertex \( v \), all its in-arcs that can go to \( G' \) will go into \( G' \) before \( v \) is selected in line 6, as we mentioned above, and the if statement in line 7, which prohibits an arc \((v, x)\) from being added to \( G' \), if \( v \) does not have an in-arc of * type in \( G' \).

3. \( G' \) is maximal: Suppose \( s \) is any vertex in \( G_{o,r} \), then we only need to prove that sometime \( s \) will be selected by the for loop in line 6. Let maxlen(s) denote the length of the longest paths from the root \( s_o \) to \( s \) in \( G_{o,r} \). We will prove this by
induction on $\text{maxlen}(s)$. First, when $\text{maxlen}(s) = 0$, then $s$ is the root and it is obviously true. Now suppose that it is true when $\text{maxlen}(s) \leq k$. Consider the case when $\text{maxlen}(s) = k + 1$. Clearly, for all $g, (g, s) \in E$, $g$ will be selected sometime by the induction hypothesis. According to line 18, these arcs will be deleted from $E''$, and hence the in-degree of $s$ will be reduced to 0 sometime in $G''$. Since $V''$ is a finite set and every selected vertex will be deleted from $V''$ in line 19, $s$ will certainly be selected sometime by the for loop in line 6. So, $G'$ is $\text{Eff}(G_{o,r})$.

Let us now determine the complexity of the proposed algorithm. It is easy to see that the execution of statements in line 1 to line 5 occur at most $N$ times respectively. The for loop in line 6 has at most $N$ runs, where checking the condition in line 7 also results in at most $N$ runs. Now we analyse the complexity of line 8 and line 9 - computing all predecessors of $v$ in $G_{o,r}$. Remember $G_{o,r}$ is a rooted acyclic digraph, and so it has at most $(N - 1) + (N - 2) + ... + 1 = 1/2(N^2 - N)$ arcs. Since for any arc we will at most go through it once in line 9, computing all predecessors of $v$ in $G_{o,r}$ (i.e., lines 8 and 9) can be done in $O(N^2)$. It is also easy to observe that the execution of lines 10 and 11 happens at most $N$ times. So the complexity is $O(N^3)$.

4.3.2 Algorithm for Evaluating the Stable Graph

Algorithm 4.2 evaluates a stable graph according to the policy for incomparable conflicts. Its input includes an effective graph $G$ of $G_{o,r}$ for some object $o$ and right $r$, the root $s_o$, and the policy $P$ to be used. Its output is $G'$, the stable graph of $G_{o,r}$ corresponding to $P$. $G''$ is a temporary working graph used to construct a topological sorting of $<_{o,r}$. The algorithm begins from copying the root and out-arcs of root from $G$ to $G'$. Then, it checks for every vertex $v$ in $G$, in terms of the sequence of topological sorting; if it has more than one in-arcs in $G'$, then only one remains and the others are deleted from $G'$ according to the policy $P$. In addition, if its in-arc in $G'$ is of delegatable type $\ast$, then put all of its out-arcs into $G'$, although they may be deleted from $G'$ later by using policy $P$ if they conflict with other authorizations.

Algorithm 4.2: $\text{Evaluate\_Stable\_Graph}(G, P, s_o)$
§4.3 Algorithms

Input: $G = (V, E, t) - G$ is a effective graph of $G_{o,r}$ for some object $o$ and access right $r$, with root $s_o$ and arc type function $t$,

$P$ - the policy of solving incomparable conflicts over $o$

Output: $\text{stable}(G_{o,r}, P) = G' = (V', E', t')$

begin

1. $E' = \{ (s_o, x) | (s_o, x) \in E \}$;
2. $V' = \{ s_o \} \cup \{ x | (s_o, x) \in E' \}$;
3. for all $e' \in E'$ do $t'(e') = t(e')$;
4. $E'' = E - E'$;
5. $V'' = V - \{ s_o \}$; (* copy the root and out-arcs of root in $G$ to $G'$ and then copy the remaining part in $G$ to $G''$ *)

6. for each $v \in V''$ with 0 in-degree do begin

7. if the in-degree of $v$ is greater than 1 in $G'$

8. then select any in-arc that has the highest priority according to policy $P$ for incomparable conflicts and delete other in-arcs from $G'$

9. if the type of $v$'s in-arc in $E'$ is $*$ then begin

10. $E' = E' \cup \{ (v, x) | (v, x) \in E \}$;
11. $V' = V' \cup \{ x | (v, x) \in E' \}$;
12. $t'((v, x)) = t((v, x))$ for all $(v, x) \in E'$;

13. end

14. $E'' = E'' - \{ (v, x) | (v, x) \in E'' \}$;
15. $V'' = V'' - \{ v \}$;

end

Theorem 4.4 Algorithm $\text{Evaluate\_Stable\_Graph}$ is correct, and $\text{stable}(G_{o,r})$ can be computed by $\text{Evaluate\_Stable\_Graph}(G, P, \prec)$ in $O(N^2)$ time, where $N$ is the number of vertices in $G$.

Proof. To begin with, note that the vertices are selected and operated from $G$ in terms of the topological sorting of the partial order $\prec_{o,r}$. This can be easily seen from the for loop condition in line 6 and statements in line 14 and 15.
To prove the correctness of the algorithm, we need to show that (1) $G'$ is conflict-free, (2) $G'$ is consistent, and (3) $G'$ is maximal. The proof of (3) is almost the same as the proof for Theorem 2. So here we only give the proofs of (1) and (2).

(1) $G'$ is conflict-free. The arcs from the root are obviously not in conflict with any other arcs since $G$ is effective and therefore no comparable conflicts exist. On the other hand, since predecessors go in $G'$ first, all $v$'s in-arcs will go in $G'$ before $v$ is selected in line 6. From lines 7 and 8 in the algorithm, it is clear that for any vertex $v$ other than the root in $G'$, the in-degree of $v$ is exactly 1. This means that no conflicts exist in $G'$.

(2) $G'$ is consistent. Since $G$ is consistent, $G' \subseteq G$. So we only need to show that $G'$ is delegation correct. This is true because of the for loop condition in line 6, which means that the predecessors are selected into $V'$ first, and the if statement in line 9 within the loop means that an arc $(v, x)$ cannot go to $G'$, except when $v$ has an in-arc of $*$ type.

Let us now determine the complexity of the algorithm. It can be easily seen that the execution of statements from line 1 to line 4 occurs at most $N$ times respectively. The for loop in line 6 contains at most $N$ runs, where every statement within the loop also happens at most $N$ times. So the time complexity of the algorithm is $O(N^2)$.

\[\square\]

4.4 Authorization State Transformation

In a dynamic environment, an authorization state is not static since users may need to add, update, or revoke certain authorizations. In this section, we consider how our proposed authorization state can be changed. Since an update can be implemented by revoking and adding, here we only consider the addition and revocation of authorizations in our model.

In the case of addition, an authorization $a = (s, o, t, r, g)$ can be added to the authorization state $\mathcal{A}$ if and only if in $G_{a,r} = (V, E)$ of $\mathcal{A}$, $g$ is the root, or it satisfies the following three conditions:
A1 There exists $g', (g', g) \in E$, and $t((g, g)) = \ast$;

A2 $(g, s) \notin E$; and

A3 There is no path from $s$ to $g$.

**Theorem 4.5** Let $A$ be a consistent state, and $A'$ is the resulting state after an authorization $a = (s, o, t, r, g)$ is added to $A$ through the above procedure. Then $A'$ is also consistent.

*Proof.* It is easy to see that $A'$ is delegation correct and not contradictory according to conditions A1 and A2, respectively. There are no cycles in $G_{o,r}$ of $A'$ containing no $a$ since $A$ is consistent. There are also no cycles containing $a$ since that will lead to a path from $s$ to $g$ in $G_{o,r}$ of $A$ which contradicts condition A3. Therefore $A'$ is consistent. \qed

For revocation, on the other hand, we adopt a cascading revocation approach to implement this operation. An authorization $a = (s, o, t, r, g)$ can be revoked from the system if the requester is $g$. The detailed procedure can be outlined as follows: in $G_{o,r}$,

V1 Delete arc $(g, s)$;

V2 If vertex $g$ or $s$ is isolated, then delete $g$ or $s$ respectively;

V3 If the type of $(g, s)$ is $\ast$, and, after the deletion, there is no in-arc of $\ast$ type for $s$, recursively revoke all the out-arcs of $s$.

**Theorem 4.6** Let $A$ be a consistent state, and $A'$ is the resulting state after an authorization $a = (s, o, t, r, g)$ is revoked from $A$ through the above procedure. Then $A'$ is also consistent.

*Proof.* Since $A$ is consistent, it is easy to see that $A'$ is not contradictory and $<_{o,r}$ in $A'$ is still a strict partial order, since removing an arc does not affect those properties. Suppose there exists an arc $(g, s)$ in $G_{o,r}$ of $A'$ such that $g$ has no in-arc of $\ast$ type. Since $A$ is consistent, its in-arc of $\ast$ type must be revoked or recursively
revoked. But, according to V3, \((g, s)\) should also be revoked. Therefore \(A'\) is also delegation correct. 

For example, for the \(G_{\omega, R}\) shown in Figure 4.4, \((s_3, o, -, R, s_7)\) or \((s_8, o, -, R, s_7)\) cannot be added to \(G_{\omega, R}\) because adding \((s_3, o, -, R, s_7)\) will result in a cyclic graph while adding \((s_8, o, -, R, s_7)\) generates a graph which represents a contradictory authorization state. But adding \((s_5, o, +, R, s_4)\) is allowed. On the other hand, consider the situation where 2 requests to revoke \((s_4, o, *, R, s_2)\). This will lead to the deletion of arc \((s_2, s_4)\) and cascading deletion of arcs \((s_4, s_6)\), \((s_6, s_9)\), \((s_6, s_7)\) \((s_4, s_7)\), and \((s_7, s_8)\) from the graph. The resulting \(G_{\omega, R}\) after the addition and revocation is shown in Figure 4.13.

According to the above theorems, if the authorizations are added/revoked incrementally, the resulting state would always be consistent. Note that adding and revoking authorizations will update both the effective and stable state. In the above example, the update effective state is the same as the update stable state as there is no incomparable conflict in Figure 4.13, which is shown in Figure 4.14. Users are also allowed to change the policy of resolving incomparable conflicts for an object. This would also lead to an update of the stable state.

![Figure 4.13: G_{\omega, R} after update.](image)
4.5 Apply Our Model to a Distributed Environment

In this section, we discuss the issue of applying our model to a distributed system [10, 50, 74] where authorizations may be physically stored in several computers connected by a network. We will refer to a computer in a network to be a node.

Definition 4.11 A local authorization state is the set of all authorizations at one node at a given time. A global authorization state is the set of all authorizations at all nodes in the network at a given time.

Basically we have two methods to apply our model to a distributed environment: keep global authorization state consistent or only keep local authorization state consistent.

Remark 1: The global authorization state may not be consistent even though all local authorization states are consistent.

Example 6 Suppose Figure 4.15 denotes two local authorization states for access right R on object O. It is easy to see that they are both consistent. However the global state for R on O shown in Figure 4.16 is not consistent.
Remark 2: The global authorization state may be consistent even though some local authorization states are not consistent.

Example 7 Suppose Figure 4.17 denotes another two local authorization states for R on O. It is easy to see that the local state on node 2 is not consistent since S1 has no right to grant R on O there. However the global state for R on O shown on Figure 4.18 is consistent.

4.5.1 Global Access Control Model

In some distributed processing frameworks, information is logically related but physically distributed to different computers. Physical distribution of data and autho-
Corporations are usually transparent to users. From users’ point of view, the system behaves like all data and authorizations are stored in the same computer. There is a global access control to process distributed queries, as is the case with (homogeneous) distributed database systems. Thus, our objective is to keep global authorization state consistent and evaluate effective state and stable state based on it.

Recall that a graph $G$ corresponding to a global state consists of a union of disjoint subgraphs $G_o$ for all objects $o$, and the consistent graph (corresponding to a consistent state), effective graph (corresponding to an effective state) and stable graph (corresponding to a stable state) are all evaluated on the basis of $G_o$. Thus, we have the following obvious but important theorem.

**Theorem 4.7** Let $G$ denote a graph corresponding to an authorization state, $G_o$ denote its subgraph corresponding to all authorizations on an object $o$. Then, $G$ is consistent if and only if $G_o$ is consistent for each object $o$.

**Proof.** This theorem is obtained in a straightforward manner from the definition of a consistent authorization state, which puts constraints only on the authorizations for the same object. \qed
This theorem tells us, the graph \( G \) for an authorization state can be naturally distributed on the network for processing based on its subgraphs \( G_o \).

We say an authorization \( grant(s, o, t, r, g) \) is \textit{local} if it is stored at the node where the object \( o \) is stored. Otherwise it is \textit{distributed}. Then, we have following theorem.

**Theorem 4.8** If all authorizations are local, and there is a single copy for each object in the network, then the global authorization state is consistent if and only if all local authorization states are consistent.

\textbf{Proof.} The computer where the object is stored contains all the authorizations on this object, \( G_o \), since there is only a single copy of the object in the network. \( \Box \)

When there exist distributed authorizations or multiple copies of the objects, we need to collect and combine authorizations over the network to evaluate the global authorization state. The following are possible solutions.

- Select one node in the network as the centralised authorization server, and send all authorizations to this node.

In this case, the authorization server exercises access control for the whole network and all queries need to be sent to the server. This policy for example is suitable for Client-Server network structure.
- Broadcast each authorization to all nodes in the network.

In this case, every node in the network maintains a global state and all queries are processed locally. The advantages of this method include fast query processing and state reliability (if one node fails for some reason, other nodes still keep a complete copy of an authorization state), while the disadvantage is the heavy communication amount and big storage space. For instance, this is suitable for local area networks where broadcasting is available.

- Send authorizations to the node where the object is stored.

When there are multiple copies of objects, we can:

1. Send authorizations to all the nodes where the object is stored.
2. Select one copy of the object as the primary copy and send all authorizations to the node where the primary copy is stored.

This method is in between the previous two methods. In this case, the global state is partitioned and each node where the object or primary object is stored maintains part of the global state. All queries need to be sent to the nodes where the related objects (or primary objects) reside. Each node may act as an authorization server (for objects that are stored on it) as well as a client (for objects that are not stored on it).

- Divide objects into different classes and select an authorization server for each class.

The classes may have intersections, which means that multiple servers may exist for authorizations on a single object. All the authorizations should be sent to the server(s) for the related objects. This method is usually application oriented. We can classify the objects, for example, based on ownership, application or organization. That is, select an authorization server for all the objects owned by a subject, or an application, or an organization. The reason of doing this is that these objects are logically close related and thus can be accessed simultaneously.
4.5.2 Completely Local Access Control Model

In some network architecture, each node of the network is completely autonomous and does not rely on the global data access control to process distributed queries. And each node can join or leave the system at will without affecting the other nodes, as is the case with federated databases. In this case, we only need to keep the local authorization state consistent and evaluate local effective and stable state based on it. Queries are processed locally based on the local stable state. It is rather flexible. However, since conflicting authorizations may exist at different nodes, the decision to a query may be different when it is submitted at different nodes.

4.5.3 Mixed Access Control Model

In some networks, there are both some degree of global control and some degree of autonomy for each node in the network. In this case, we still need to keep local authorization state consistent, and evaluate local effective and stable stable state based on it. But, since conflicting authorizations may still exist at different local states, we need to consider all the local stable states for access control.

Suppose $S_1, S_2, \ldots, S_n$ are $n$ local stable states, we can use following policies for distributed access control.

1. (1) Negative take precedence

   A request is refused if a related negative authorization exists at any local stable state. In this case, when a node receives a query request, it will check its own local stable state first. If there is negative authorization about this request, then the request is refused. Otherwise, it needs to send the query to other nodes. If there is a negative authorization in any node, the query is refused. Otherwise, if a positive or delegatable authorization exists at any node then the query is granted. Otherwise, the query is “undecided”.

2. (2) Positive take precedence

   A request is granted if a related positive authorization exists at any local stable state. In this case, when a node receives a query request, it will check its own local
stable state first. If there exists a positive or delegatable authorization about this
request, then the request is granted. Otherwise, it needs to send the query to other
nodes. If there is a positive or delegatable authorization in any node, the query is
granted. Otherwise, if a negative authorization exists at any node, then the query
is denied. Otherwise, the query is “undecided”.

(3) All negative required
A request is refused if the related negative authorization exists at all local stable
states. In this case, each query will be sent to all the nodes and denied only when a
negative authorization about this query exists at all sites. Otherwise, if a positive or
delegatable authorization exists at any node then the query is granted. Otherwise,
the query is “undecided”.

(4) All positive required
A request is granted if the related positive or delegatable authorization exists
at all local stable states. In this case, each query will be sent to all the nodes and
granted only when a positive or delegatable authorization about this query exists at
all sites. Otherwise, if a negative authorization exists at any node then the query is
denied. Otherwise, the query is “undecided”.

(5) Hierarchical structure of nodes
In this case, we consider priority information of nodes. That is, nodes are organ-
ised into a hierarchy (partial order) where lower level means higher priority. Thus,
for the conflicting authorizations from nodes at different levels, the lower level one
will override the higher level one. For the conflicting authorizations from nodes at
the same level, we can use negative or positive take precedence policy. When a node
receives a query, it only needs to send it to the nodes which are lower or equal to it
and make a decision accordingly.
4.6 Summary

In this chapter, we have proposed a graph based authorization model and a conflict resolution policy to resolve conflicts that can occur when access rights are delegated. A major feature of our approach is that we classified conflicts into comparable and incomparable ones and this classification is useful not only in the control of access right delegation but also for supporting multiple policies to resolve conflicts. Our model also provides a flexible framework to preserve conflicting authorizations so that it is possible to re-activate some early overridden access rights if proper authorizations are revoked or the policy of conflict resolution is changed. Application of our model in a distributed environment is also discussed.
A Weighted Graph Based Authorization Model

In this chapter, we extend the model developed in last chapter to allow grantors to express certainties about their authorizations. This model gives subjects more flexibility to control their delegations of access rights on objects. A new conflict resolution policy based on weighted lengths of authorization paths is proposed. As we will see, it deals with the comparable conflicts in a more flexible way and may reduce the amount of incomparable conflicts significantly.

5.1 Motivation

Recall that, in the graph model of Chapter 4, conflicts are classified into comparable and incomparable ones based on the underlying delegation relation, and the predecessor's authorizations will override the successor's for comparable conflicts. Incomparable conflicts are resolved based on other properties, such as the type of authorizations. In this chapter, this model is further improved to achieve more flexible delegation control.

Firstly, in the graph model, the delegates of an access right from the same delegator would have the same priority, and hence their conflicting grants of authorizations would be incomparable. This may not apply in some applications, and may result in many incomparable conflicts. Let us revisit the example within the university framework. The Dean of a college creates a budget file and then delegates the "read" right to his/her subordinate Heads of Schools (HOS) and Directors of Centers (DC). The Heads of Schools and the Directors of Centers receive the same priority from the
§5.1 Motivation

Dean and their conflicting grants to a specific staff would become incomparable, as shown in Figure 5.1. However, the Dean may wish to give the Heads of Schools higher priorities than Directors of Centers, which may result in their conflicting authorization grants becoming comparable.

![Figure 5.1: An Incomparable Conflict](image)

Secondly, the comparable conflict resolution policy in the graph model is not general enough for some applications. Take the above university example again. Consider the situation where a specific staff receives a “read” from the Dean and a “not read” from his/her Head of School for the budget file, as shown in Figure 5.2.

![Figure 5.2: A Comparable Conflict](image)

There may exist two possibilities:

1. The Dean is quite confident about his/her authorization to the staff and would not want it to be overridden. This situation is well supported by the graph model.
2. The Dean is not too sure about his/her authorization to the staff and would not mind if it is overridden by a more confident authorization from the Head of School. This situation could not be supported by graph model.

Thirdly, a disadvantage of the conflict resolution method in the graph model is that, a subject can increase his/her relative priority by issuing a new authorization. Look at the example illustrated by Figure 5.3 and Figure 5.4. First, HOS1 and HOS2 have the same priority (Figure 5.3). After issuing a new authorization to HOS2, HOS1 has achieved higher priority than HOS2 (Figure 5.4).

![Figure 5.3: HOS1 and HOS2 Have the Same Priority](image1)

![Figure 5.4: HOS1 Has Higher Priority Than HOS2](image2)

To solve all the above limitations, we extend the graph model to a weighted
graph model.

5.2 Basic Idea

In general, the idea is that each grantor can add a weight to his/her authorization grant which expresses his/her degree of certainty about this grant. The weight is a non-negative integer, with the smaller number denoting the higher priority. Then, we can solve conflicts based on the weighted lengths of authorization paths, and let the authorization with the smaller weighted length override the others.

Let us see how it works for the above examples. In Figure 5.1, the Dean can assign weight 1 to the grant to HOS, and weight 2 to the grant to DC to distinguish the two grants, as shown in Figure 5.5. In this way, the two conflicting grants on the staff from HOS and DC, with the same weight, become comparable, and the grant from HOS will override since it has a shorter weighted length of 2.

![Diagram](image)

*Figure 5.5: Weighted Authorizations For Figure 5.1*

In Figure 5.2, the Dean can assign weight 1 to the grant to HOS and weight 5 to the grant to the staff. Thus, when HOS assign weight 1 to his/her grant to the staff, this grant will override the Dean's (see Figure 5.6).

In Figure 5.4, suppose the weights of grants from the Dean to HOS1 and HOS2 are both 1. An additional grant from HOS2 to HOS1 will not decrease the relative priority of HOS2 since the shortest weighted path from the Dean to it is not changed
5.3 Formal Description of the Model

Definition 5.1 (Weighted Authorization) A weighted authorization is of the form \( a : w \), where \( a \) is an authorization and \( w \) is a non-negative integer. \( a \) is called the base authorization of this weighted authorization.

Definition 5.2 (Weighted Authorization State) A weighted authorization state, denoted by \( A^w \), is the set of all weighted authorizations at a given time. All base authorizations at a given time form its base authorization state.

We use a weighted digraph to represent a weighted authorization state. Suppose \( a : w \) is a weighted authorization where \( a = (s, o, t, r, g) \), then a label \( w \) is added.
to the arc corresponding to $a$ in $G_{\alpha,r}$ of the base authorization state. We can define a weight function $w$ which matches each arc in $G_{\alpha,r}$ to its weight. That is $w((g,s)) = w$. We use $G_{\alpha,r}^w$ to denote the corresponding weighted graph.

**Example 8** Suppose we have following weighted authorizations, then the corresponding $G_{\alpha,r}^w$ is shown in Figure 5.8.

$\begin{align*}
(s_1, F, *, R, s_0) & : 1 \\
(s_2, F, *, R, s_0) & : 1 \\
(s_3, F, -, R, s_2) & : 2 \\
(s_3, F, *, R, s_1) & : 3 \\
(s_4, F, *, R, s_3) & : 1 \\
(s_4, F, -, R, s_0) & : 6 \\
(s_4, F, *, R, s_1) & : 5
\end{align*}$


![Figure 5.8: An Example $G_{\alpha,r}^w$.](image)

**Definition 5.3** (Consistent Weighted Authorization State) A weighted authorization state is consistent if its base authorization state is consistent.

Suppose all the authorizations on an object are first granted from the owner of the object, then we have the following theorem:
§5.3 Formal Description of the Model

Theorem 5.1 Let $A^w$ be a consistent weighted authorization state, then for any object $o$ and access right $r$, $G_{o,r}^w$ in $A^w$ is a rooted acyclic simple weighted digraph, and rooted at the owner of the object.

This theorem is immediately obtained from the definition of a consistent weighted authorization state and the Theorem 4.1.

Definition 5.4 (Conflicting Weighted Authorizations) Two weighted authorizations are conflicting if their base authorizations are conflicting.

Now we are in a position to define our conflict resolution method. A path in $G_{o,r}^w$ is called a delegation path if every arc in the path is of type $\ast$.

Definition 5.5 (Useful Path) Let $G_{o,r}^w$ be a weighted graph for any object $o$ and access right $r$ rooted at $s_o$; $s$ be any vertex in $G_{o,r}^w$. Then we have following recursive definition of a useful path:

1. A useful path to $s_o$ is from $s_o$ to itself.
2. A useful path to any vertex $s$ is a path: $s_0, s_1, \ldots, s_{k-1}, s_k, s$ ($k \geq 0$), where $s_0, s_1, \ldots, s_{k-1}, s_k$ is a shortest useful path to $s_k$, and $(s_{k-1}, s_k)$ has delegatable type $\ast$ when $k > 0$.

From the definition, it is easy to see that any useful path in $G_{o,r}^w$ is from $s_o$. Only arcs on useful paths can compete with each other, and the arc in the shorter path will override the other.

Example 9 In Figure 5.8, $(s_o, s_1)$ is the only useful path to $s_1$ and therefore is the shortest useful path to $s_1$. For the same reason, $(s_o, s_2)$ is the shortest useful path to $s_2$. There are two paths to $s_3$, $(s_o, s_1, s_3)$ and $(s_o, s_2, s_3)$, and they are both useful paths with $(s_o, s_2, s_3)$ being the shortest. Therefore $(s_2, s_3)$ will override $(s_1, s_3)$. There are four paths to $s_4$. $(s_o, s_1, s_3, s_4)$ is not a useful path since $(s_o, s_1, s_3)$ is not a shortest useful path to $s_3$. $(s_o, s_2, s_3, s_4)$ is also not a useful path since $(s_2, s_3)$ is not of type $\ast$, although $(s_o, s_2, s_3)$ is the shortest useful path to $s_3$. $(s_o, s_1, s_4)$ and $(s_o, s_4)$ are two useful paths to $s_4$ with the same shortest length 6.
Definition 5.6 (Active Authorization) An authorization \((s, o, t, r, g)\) is active if
the corresponding arc \((g, s)\) in \(G_{o,r}^w\) is in a shortest useful path to \(s\).

Definition 5.7 (Effective Weighted Authorization State) Let \(A^w\) be a weighted
authorization state, then the subset of all its active arcs forms its effective weighted
authorization state, denoted by \(Eff(A^w)\).

Theorem 5.2 If a weighted authorization state \(A^w\) is consistent, then its effective
weighted authorization state \(Eff(A^w)\) is also consistent.

Proof. Since \(A^w\) is consistent, its base authorization state is not contradictory,
and for any object \(o\) and access right \(r\), \(<_{o,r}\) is a strict partial order. These two
conditions are also satisfied in the base state of \(Eff(A^w)\), since \(Eff(A^w)\) is a
subset of \(A^w\). On the other hand, according to the definition, an active arc is either
from the root or from a subject that has an active delegatable in-arc to it, which
follows that the base state of \(Eff(A^w)\) is delegation correct. \(\Box\)

Please note that, an active path to a vertex \(s\) may be not the shortest path from
\(s_o\) but is the shortest useful path from \(s_o\). An active path to a vertex \(s\) may be not
unique. That is, there may exist multiple active paths to a vertex \(s\) if they have the
same shortest length. In addition, for each active path \(s_o, s_1, s_k, s, s_o, ..., s_k\) must be
an active delegation path.

Example 10 The effective weighted authorization state of Figure 5.8 is shown
in Figure 5.9. There are two active paths to \(s_4\), \((s_o, s_1, s_4)\) and \((s_o, s_4)\). Neither
\((s_o, s_1, s_4)\) nor \((s_o, s_4)\) is the shortest path from \(s_o\) to \(s_4\). In fact, \((s_o, s_2, s_3, s_4)\) is the
shortest path to \(s_4\). However it is not a useful path to \(s_4\) and therefore not an active
path. \(\blacksquare\)

Please note that inactive authorizations are not removed from the system. They
are still kept in the system for the purpose of possible re-activation when the au-
 thorizations that cause them inactive are revoked.

Conflicts may still exist in the effective weighted state when the two active paths
have the same smallest weighted length, as is the case in Figure 5.9. In this case,
we say these two conflicting authorizations are incomparable.
For incomparable conflicts, as in the graph model, we support multiple strategies for resolving them. A user can select the appropriate strategy that best suits the needs of his/her application. In our framework, we support the following three strategies according to the grant types of authorizations:

1. Pessimistic: the priority sequence is \(- > + > *\);
2. Optimistic: the priority sequence is \(* > + > -\);
3. Any: the priority sequence is \(* = + = -\).

This leads us to the situation where incomparable conflicts occur in the graph model addressed in Chapter 4. We would not address this issue again, but just give the following definition of stable weighted authorization state.

**Definition 5.8 (Stable Weighted Authorization State)** If a weighted authorization state \(A^w\) is consistent, then the maximal consistent and conflict-free subset of \(Eff(A^w)\) forms a stable weighted authorization state of \(A^w\), denoted as \(stable(A^w)\).

**Example 11** Figure 5.10 and Figure 5.11 are two stable weighted authorization states of Figure 5.9.
5.4 Access Control Policy

Now we can define our access control policy. We use 3-ary tuple \((s, o, r)\) to denote an access request to the system, where \(s \in S, o \in O, r \in R\). It states that a subject \(s\) requests to exercise access right \(r\) on object \(o\). Then we have following access control policy.

**Definition 5.9 (Access Control Policy)** Let \(A^w\) be a weighted authorization state, \((s, o, r)\) be an access request, \(P\) be a policy to resolve the incomparable conflicts on \(o\), \(\text{stable}(A^w, P)\) be a weighted stable state of \(A^w\) when applying \(P\) to \(o\) and \(\text{stable}(A, P)\) be its base state. We say that \((s, o, r)\) is permitted if there exists some grantor \(g\) such that \((s, o, *, r, g)\) or \((s, o, +, r, g)\) is in \(\text{stable}(A, P)\); \((s, o, r)\) is denied if there is some grantor \(g\) such that \((s, o, -, r, g)\) is in \(\text{stable}(A, P)\); otherwise, \((s, o, r)\) is undecided.

Please note that the system is the implicit grantor of any object’s owner for all access rights on this object with delegatable grant type.

According to the access control policy, the procedure of evaluating an access request \((s, o, r)\) can be outlined as follows:

1. If \(s\) is the root of \(G^w_o\), return yes;
2. Compute $\text{stable}(G^w_{o,r}, P) = (V, E, t)$;

3. If $s$ is not in $V$, return \textit{undecided};

4. Let $(g, s)$ be the in-arc of $s$ in $E$, if $t((g, s)) = \ast$ or $+$ return \textit{yes} else return \textit{no}.

5.5 Algorithm

In this section, we present the algorithm to evaluate the effective weighted authorization state. The algorithm is based on Dijkstra's distance algorithm.

\textbf{Algorithm 5.1: \textit{Evaluate\_Weighted\_Eff\_Graph}}($G^w_{o,r}$, $s_o$)

\textbf{Input:} $G^w_{o,r} = (V, E, t, w)$ for some object $o$ and access right $r$, with root $s_o$, arc function $t$, and weight function $w$

\textbf{Output:} $\text{Eff}(G^w_{o,r}) = G^e = (V', E', t', w')$

\textbf{Method:} Label each vertex $v$ with $l(v)$, which is the length of a shortest useful path from $s_o$ to $v$ that has been found at that instant.

\begin{verbatim}
begin
01 \hspace{1em} l(s_o) = 0;
02 \hspace{1em} \text{for all } v \neq s_o \text{ set } l(s_o) = \infty;
03 \hspace{1em} T = V; V' = \emptyset; E' = \emptyset;
04 \hspace{1em} \text{while } T \neq \emptyset
05 \hspace{2em} \text{...}
end
\end{verbatim}
begin
  Find $v \in T$ with finite minimum label $l(v)$;
  if such a $v$ does not exist then exit;
  $V' = V' \cup v$;
  if $v \neq s_o$ then
    For every $v' \in p(v)$
      begin
        $E' = E' \cup \{(v', v)\}$;
        $t'((v', v)) = t((v', v))$;
        $w'((v', v)) = w((v', v))$;
      end
    If $v = s_o$ or there exists $v' \in p(v)$ such that $t((v', v)) = \ast$ then
      begin
        For every $e = (v, x)$
          if $l(x) > l(v) + w(e)$ then
            begin
              $l(x) = l(v) + w(e)$;
              $p(x) = \{v\}$;
            end
          else
            if $l(x) = l(v) + w(e)$ then $p(x) = p(x) \cup \{v\}$
      end
      $T = T - \{v\}$;
  end
end

Example 12 We now apply the algorithm to the digraph of Figure 5.8, finding its $Eff(G^m_{s_o})$. The major steps performed and the actions taken are as follows:

1-3. $l(s_o) = 0$, and $l(v) = \infty$ for all other vertices, $T = \{s_o, s_1, s_2, s_3, s_4\}, V' = \emptyset, E' = \emptyset$. 
4. Since $T$ is not empty, we continue

6. Select $s_2$, since it has minimum label.

8. $V' = \{s_0\}$

16-26. We examine the arcs out of $s_0$ and set $l(s_1) = 1, l(s_2) = 1, l(s_4) = 6, p(s_1) = p(s_2) = p(s_4) = \{s_0\}$

27. $T = \{s_1, s_2, s_3, s_4\}$

6. Select $s_1$ (or $s_2$), since it has minimum label.

8. $V' = \{s_0, s_1\}$

12. $E' = \{(s_0, s_1)\}$

16-26. We examine the arcs out of $s_1$ and set $l(s_3) = 4, p(s_3) = \{s_1\}, p(s_4) = \{s_0, s_1\}$

27. $T = \{s_2, s_3, s_4\}$

6. Select $s_2$, since it has minimum label.

8. $V' = \{s_0, s_1, s_2\}$

12. $E' = \{(s_0, s_1), (s_0, s_2)\}$

16-26. We examine the arcs out of $s_2$ and set $l(s_3) = 3, p(s_3) = \{s_2\}$

27. $T = \{s_3, s_4\}$
6. Select \( s_3 \), since it has minimum label.

8. \( V' = \{s_0, s_1, s_2, s_3\} \)

12. \( E' = \{(s_0, s_1), (s_0, s_2), (s_2, s_3)\} \)

16-26. Since there exists no \( v' \in p(s_3) \) such that \( t((v', s_3)) = * \), nothing changes.

27. \( T = \{s_4\} \)

6. Select \( s_4 \), since it has minimum label.

8. \( V' = \{s_0, s_1, s_2, s_3, s_4\} \)

12. \( E' = \{(s_0, s_1), (s_0, s_2), (s_2, s_3), (s_0, s_4), (s_1, s_4)\} \)

16-26. We examine the arcs out of \( s_4 \) and nothing changes.

27. \( T = \emptyset \), and the algorithm stops.

\[\]

**Theorem 5.3** Algorithm Evaluate Weighted Eff Graph is correct, and \( Eff(G_{o,r}) \) can be computed by Evaluate Weighted Eff Graph\( (G_{o,r}, s_o) \) in \( \mathcal{O}(N^2) \) time, where \( N \) is the number of vertices in \( G_{o,r} \).

**Proof.** We prove by induction on the order in which vertices are deleted from \( T \) and entered \( V' \). Take as the induction hypothesis the following assertion: At the \( k \)th iteration

(1) the label of a vertex \( v \) in \( V' \) is the length of the shortest useful path from \( s_o \) to this vertex, and
(2) the label of a vertex not in $V'$ is the length of the shortest useful path from $s_o$ to this vertex that contains only (besides the vertex itself) vertices in $V'$.

When $k = 0$, $V' = \{s_o\}$, so the length of the shortest useful path from $s_o$ to itself is 0 (here we are allowing a path to have no arcs in it), and the length of the shortest useful path from $s_o$ to a vertex other than $s_o$ is $\infty$.

Assume that the inductive hypothesis holds for the $k$th iteration. Let $v$ be the vertex added to $V'$ at the $(k+1)$st iteration so that $v$ is a vertex not in $V'$ at the end of the $k$th iteration with the smallest label (in case of ties, any vertex with smallest label may be used). From the inductive hypothesis we see that vertices in $V'$ before the $(k+1)$st iteration are labelled with the length of the shortest useful path from $s_o$. Also $v$ must be labelled with the length of the shortest useful path to it from $s_o$. If this were not the case, at the end of the $k$th iteration there would be a useful path of length less than $l_k(v)$ containing a vertex not in $V'$ (because $l_k(v)$ is the length of the shortest useful path from $s_o$ to $v$ containing only vertices in $V'$ after the $k$th iteration). Let $u$ be the first vertex not in $V'$ in such a useful path. There is a useful path with length less than $l_k(v)$ from $s_o$ to $u$ containing only vertices of $V'$. This contradicts the choice of $v$. Hence (1) holds at the end of the $(k+1)$st iteration.

Let $u$ be a vertex not in $V'$ after $k+1$ iterations. A shortest useful path from $s_o$ to $u$ containing only elements of $V'$ either contains $v$ or it does not. If it does not contain $v$, then by the inductive hypothesis its length is $l_k(u)$. If it does contain $v$, then it must be made up of a useful path from $s_o$ to $v$ of shortest possible length containing elements of $V'$ other than $v$, and with the in-arc of $v$ being of type $*$ (if the length is greater than 0), followed by the arc from $v$ to $u$. In this case its length would be $l_k(v) + w(v, u)$. This shows that (2) is true, since $l_{k+1} = min\{l_k(u), l_k(v) + w(v, u)\}$.

For each vertex $v$, it is not difficult to see that we have used $p(v)$ to record all parents of $v$ that make its label $l(v)$ (when $l(v)$ is finite), and entered the corresponding in-arcs to $E'$ when $v$ is selected. Thus we have entered all the useful paths that make $l(v)$ for each vertex $v$ to $E'$, this concludes the proof of the first part of the theorem.

We now determine the time complexity of the algorithm. Note that in line 6 the minimum label of $T$ must be found. This can certain be done $|T| - 1$ comparisons.
Initially, $T = V$, and line 27 reduces $T$ one vertex at a time. Thus the process is repeated $N$ times. The time required in line 6 is then on the order of $\sum_{i=1}^{N} i$ and therefore is $O(N^2)$. In Line 10, $p(v)$ has at most $N$ elements. So the for loop requires at most $O(N^2)$. Line 18 uses each arc once at most, so it requires at most $O(|E|) = O(N^2)$. The entire algorithm thus has time complexity $O(N^2)$.

\section{Authorization State Transformation}

As mentioned earlier, in a dynamic environment, an authorization state is not static since users may need to add, update, or revoke certain authorizations. In this section, we consider how our proposed weighted authorization state can be changed.

In the case of addition, a weighted authorization $a = (s, o, t, r, g) : w$ can be added to a weighted authorization state $A^w$ if and only if it is consistent with the current weighted state $A^w$. This means that in its base state $A$, $g$ must get the delegatable right for $r$ on $o$, $a$ can not contradict with any other authorization in $A$ and $<_{o,r}$ must still be a partial order after the addition. That is, $a : w$ can be added to $A^w$, if and only if in $G_{o,r} = (V, E)$ of $A$, $g$ is the root, or it satisfies the following three conditions:

A1 There exists $g', (g', g) \in E$, and $t((g', g)) = *$;

A2 $(g, s) \notin E$; and

A3 There is no path from $s$ to $g$.

For revocation, on the other hand, we adopt a cascade revocation approach to implement this operation. An authorization $a = (s, o, t, r, g) : w$ can be revoked from the system if the requester is $g$, and the authorization state must remain consistent after the revocation. The detailed procedure can be outlined as follows: in $G_{o,r}^w$.

V1 Delete arc $(g, s)$ with its weight;

V2 If vertex $g$ or $s$ is isolated, then delete $g$ or $s$ respectively;
V3 If the type of \((g, s)\) is \(*\), and, after the deletion, there is no in-arc of \(*\) type for \(s\), recursively revoke all the out-arcs of \(s\) with their weights.

Based on our algorithms, if the authorizations are added/revoked incrementally, the resulting state would always be consistent. Note that adding and revoking authorizations will update both the effective and stable weighted state. Users are also allowed to change the policy of resolving incomparable conflicts for an object. This would also lead to an update of the stable state.

### 5.7 Several Remarks of the Model

- **Remark 1:** In Algorithm 5.1, the label of each vertex \(l(v)\) indicates the priority of the subject. Lesser the value, the higher priority the subject has.

- **Remark 2:** A delegator can give a delegate the same priority as itself by assigning 0 to the weight of the delegation.

- **Remark 3:** Although weights are non-negative, a delegate's priority could be higher than its delegator's due to the possible multiple useful paths to it. However, in an effective weighted state, a delegator's priority must be greater or equal to its delegate's; or, in general, a predecessor's priority must be greater or equal to its successor's.

- **Remark 4:** The priority of an authorization depends not only on the priority of the grantor but also on the weight of the authorization assigned by the grantor. In fact, when a conflict occurs, each sum of the grantor's label and weight of its granted authorization on the useful path is considered, and the authorization with the smallest sum will be granted. This means that an authorization from a grantor with higher priority could be overridden by an authorization from a grantor with lower priority, as long as the sum of the former is less than that of the latter. This gives a grantor more flexibility to control his/her authorization delegations.

- **Remark 5:** When all the authorizations have the same positive weight, the model reduces to a model similar to the graph model addressed in Chapter 4.
When all the authorizations have 0 as their weights, the model reduces to a model where all the grantors have the same priority.
A Logic Based Authorization Model

6.1 Motivation

In Chapter 4 and 5, we developed digraph based approaches, which allow both positive and negative authorizations, and support delegation of access rights between subjects. Especially, we have proposed conflict resolution methods for delegatable authorizations that are based on the underlying delegation relations. In Chapter 4, intuitively speaking, if $s_1$ delegates $s_2$ a right to grant others read on file $F$, then when it happens that $s_1$ and $s_2$ grant $s_3$ read and not read on $F$ respectively, then the grant by $s_1$ will override the grant by $s_2$ (since it is $s_1$ that delegates $s_2$ the right to grant). We believe that this controlled delegation can take the advantage of both decentralised and centralised administration of access rights. If users want to distribute the administration of rights without further control, they can just delegate the rights to other persons and let them grant thereafter. Whenever users want to take some control of the access rights, their grants will always have higher priorities.

In Chapter 5, a more flexible conflict resolution method is proposed. Users can assign weights to their authorizations, and the authorization on the shortest useful path will override the others.

In the proposed graph models, we did not consider relationships between subjects, objects and access rights. In this chapter, we extend our work to consider more complex domains where subjects, objects and access rights are hierarchically structured and authorization inheritance along the hierarchies are taken into account. For example, a member of a group usually can inherit all the access rights...
§6.1 Motivation

granted to the group. If someone is granted to write to a directory, then it may be that he/she should also be able to read the directory and all files in that directory. Supporting inheritance of authorizations can often effectively simplify the specification and evaluation of authorizations, especially in some application domains where inheritance is an important feature, such as object-oriented databases. When authorization inheritance is under consideration, the problem of conflict becomes more complex since a lot of implicit conflicts among different types of authorizations may arise.

To take advantage of strong expressive and reasoning power of logic programming, in this chapter, we develop our framework based on extended logic programs [34], which can support both negation as failure and classical negation. The extended logic programs, which is formalised based on nonmonotonic reasoning semantics, has strong expressive power in the sense that it can deal directly with incomplete information in reasoning. Since the incomplete information is a common issue in the security world, many access control policies are easier to specify in extended logic programs. For example, if we want to express negation by default, like s is denied to read the file F if s is not granted to read it, the negation as failure (weak negation) is often the most direct way to express this intention. On the other hand, in many situations, classical negation (strong negation) is useful to explicitly specify that something is forbidden.

In our framework, authorization rules are specified in a delegatable authorization program (DAP) which is an extended logic program associated with different types of partial orderings on the domain. These orderings specify various inheritance relationships among subjects, objects and access rights in the domain. The semantics of a DAP is defined based on the stable model concept and the conflict resolution is achieved in the model generation of the DAP. The chapter is organised as follows. Section 2 describes the syntax of the delegatable authorization program (DAP), while Section 3 defines the semantics of the program. Section 4 investigates some important properties of DAP, and finally Section 5 discusses the related work and concludes the chapter with some remarks.
§6.2 Syntax of DAP

Our language $L$ is a many-sorted first order language, with four disjoint sorts for subject, object, access right and authorization type respectively. Let $L$ has the following vocabulary:

1. **Sort subject:** with subject constant poset $(S, <_S)$: $\#, s, s', s'', s_1, s_2, ...$, and subject variables $s, -s_1, -s_2, ...$

2. **Sort object:** with object constant poset $(O, <_O)$: $o, o', o'', o_1, o_2, ...$, and object variables $o_1, -o_2, -o_3, ...$

3. **Sort access right:** with access right constant poset $(A, <_A)$: $\text{read}, \text{write}, a, a', a'', a_1, a_2, ...$, and access right variables $a, -a_1, -a_2, ...$

4. **Sort authorization type:** with authorization type constant set $T = \{-, +, \cdot\}$, and authorization type variables $\cdot, t_1, t_2, ...$

We suppose $\#$ in $S$ denotes the security administrator, and it is not comparable to any subjects in $S$ w.r.t. $<_S$. In the constant set of authorization types $T = \{-, +, \cdot\}$, $-$ means negative, $+$ means positive, and $\cdot$ means delegatable. A negative authorization specifies the access that must be forbidden, while a positive authorization specifies the access that must be granted. A delegatable authorization specifies the access that must be delegated as well as granted. That is, $\cdot$ means $+$ together with administrative privilege on the access. The partial orders $<_S, <_O, <_A$ represent inheritance hierarchies of subjects, objects and access rights respectively.

5. **Predicate Symbol set $P$**

$P$ consists of a set of ordinary predicates defined by users, and one built-in predicate symbol for delegatable authorization, $\text{grant}$. $\text{grant}$ is a 5-term predicate symbol with type $S \times O \times T \times A \times S$. The first argument is the grantee, the second argument is the object, the third argument is the authorization type, the fourth argument is the access right, and the fifth argument is the grantor of this authorization. Intuitively, $\text{grant}(s, o, t, a, g)$ means $s$ is granted by $g$ the access right $a$ on object $o$ with authorization type $t$. $\text{grant}$ is called authorization predicate.

A term is either a variable or a constant. Note that we prohibit function symbols in our language.

An atom is a construct of the form $p(t_1, ..., t_n)$, where $p$ is a predicate of arity $n$. 
in $P$ and $t_1, \ldots, t_n$ are terms.

A literal is either an atom $p$ or the negation of the atom $\neg p$, where the negation sign $\neg$ represents classical negation. Correspondingly, a literal that has an authorization predicate is called an authorization literal. Two literals are complementary if they are of the form $p$ and $\neg p$, for some atom $p$.

A rule $r$ is a statement of the form:

$$b_0 \leftarrow b_1, \ldots, b_k, \text{not } b_{k+1}, \ldots, \text{not } b_m, m \geq 0$$

where $b_0, b_1, \ldots, b_m$ are literals, and not is the negation as failure symbol. $b_0$ is the head of $r$, while the conjunction of $b_1, \ldots, b_k, \text{not } b_{k+1}, \ldots, \text{not } b_m$ is the body of $r$. Obviously, the body of $r$ could be empty. We sometimes use $\text{Head}_r$ and $\text{Body}_r$ to denote the head and body of $r$ respectively. Correspondingly, when $b_0$ is an authorization literal, the rule is called authorization rule.

A Delegatable Authorization Program, DAP, consists of a finite set of rules.

A term, an atom, a literal, a rule or program is ground if no variable appears in it.

Example 13 Let $S = \{\#, s_1, s_2; s_1 \leq_S s_2\}, O = \{o_1, o_2; o_1 \leq_O o_2\}, A = \{\text{write, read; write} <_A \text{ read}\}$, then the following is an example DAP $\Pi$:

$$r_1 : \text{dba}(s_1) \leftarrow$$

($s_1$ is a dba.)

$$r_2 : \neg \text{dba}(s_2) \leftarrow$$

($s_2$ is not a dba.)

$$r_3 : \neg \text{secret}(o_1) \leftarrow$$

($o_1$ is not secret.)

$$r_4 : \text{secret}(o_2) \leftarrow$$

($o_2$ is secret.)

$$r_5 : \text{grant}(s_1, o_2, *, \text{write, #}) \leftarrow$$

(The security administrator delegates the access right 'write' on object $o_2$ to $s_1$.)

$$r_6 : \text{grant}(s_2, o_2, -, \text{write, s}_1) \leftarrow$$

($s_1$ denies $s_2$ to write on object $o_2$.)


\[ r_7 : \text{grant}(s, o, -, write, \#) \iff \text{secret}(o), \: \text{not} \: \text{dba}(s) \]

(Given an object \(o\) and a subject \(s\), if \(o\) is secret, and there is no information saying that \(s\) is a dba, then, the administrator denies subject \(s\) to write on the object \(o\).) ■

### 6.3 Semantics of DAP

In this section, we first discuss various aspects that are taken into account for the semantics of a DAP, and then present the formal definitions for semantics which is based on the stable model semantics. Finally we specify our access control policy.

#### 6.3.1 Considerations of the Semantics

To develop a formal semantics of a DAP, we should consider three aspects that will affect the process of evaluating a DAP: delegation correctness, authorization propagation along the hierarchies of subjects, objects and access rights, and conflict resolution.

**Delegation correctness**

Informally, we say that an authorization set is *delegation correct* if it satisfies the following two conditions: (a) subject \(s\) can grant other subjects an access right \(a\) over object \(o\) if and only if \(s\) is the security administrator \(\#\) or \(s\) has been granted \(a\) over \(o\) with a delegatable type \(*\); (b) if subject \(s\) receives a delegatable authorization directly or indirectly from another subject \(s'\) on some object \(o\) and access right \(a\), then \(s\) cannot grant \(s'\) any further authorization on the same \(o\) and \(a\) later on.

Intuitively, the condition (a) comes from the system assumption and the feature of the delegatable authorization type; condition (b) says that, for a given object and an access right, a subject is forbidden from giving authorizations to its grantors, the grantors' grantors, and so on. This restriction makes delegation more reasonable and can solve the problem that exists in most conflict resolution methods when authorization delegation is considered. This issue has been explored in detail in previous chapters. Hence, we will require the authorization set derived by a DAP
be delegation correct.

**Authorization propagations**

Rule based authorization specification allows implicit authorizations to be derived from the authorization set, and hence this can greatly reduce the size of explicit authorization set. In particular, we support the implicit authorizations by permitting rules inheritance. From the object-oriented feature, rules defined for a given object are automatically inherited by its sub-objects. In particular, our model supports three dimensional inheritance along hierarchies of subjects, objects and access rights. For instance, if a group is authorised to write a directory $D$, then by inheritance, we can derive the following implicit authorizations: all members of the group can write $D$ (inherited from subject hierarchy), the group can write all files and subdirectories under $D$ (inherited from object hierarchy), the group can also read $D$ and therefore can read all files and subdirectories under $D$ (inherited from both object and access right hierarchies), and all members of the group can read $D$ as well and therefore can read all files and subdirectories under $D$ (inherited from subject, object and access right hierarchies), and so on.

**Conflict resolution**

Since both positive and negative authorizations are acceptable in our framework, conflicts among authorizations may arise. And since we also allow implicit authorizations, implicit conflicts may occur, which make the problem more complicated. The basic idea of our method for solving conflicts is outlined as follows.

- Solving conflicts using delegation relation. According to the delegation relation, if subject $s$ delegates to subject $s'$ directly or indirectly an authorization on object $o$ and access right $a$, then when a conflict w.r.t $o$ and $a$ occurs, the authorization from $s$ (i.e. $s$ is the grantor) will always override the one from $s'$. Given a DAP, the delegation relations for different objects and access rights with respect to this will be dynamically derived by the system.

- Solving conflicts from grantee inheritance. If the grantors of two conflicting
authorizations are identical, we consider grantees of the authorizations next. We use the more specific-take-precedence principle. According to the inheritance hierarchy of subjects, if \( s <_S s' \), then the authorization with \( s' \) as grantees will override the inherited one with \( s \) as grantees.

- Solving conflicts from object inheritance. In two conflicting authorizations, if both the grantors and grantees are identical, we then consider the object inheritance relation. We still use the more specific-take-precedence principle. According to the inheritance hierarchy of objects, if \( o <_O o' \), then the authorization on \( o' \) will override the inherited one on \( o \).

- Solving conflicts from access right inheritance. In two conflicting authorizations, if all grantors, grantees and objects are identical, we finally consider the access right inheritance relation. Again, we apply the more specific-take-precedence principle to resolve the conflict. According to the inheritance hierarchy of access rights, if \( a <_A a' \), then the authorization on \( a' \) will override the inherited one on \( a \).

- Conflicts that are unsolvable. If all above policies fail to resolve the conflict between the two authorizations, we treat this conflict as unsolvable in our framework.

**Example 14** Suppose \( s_1 < s_2 \), \( o_1 < o_2 \). Consider two authorizations

\[
l_1 : grant(s_1, o_1, *, read, \#) \quad \text{and} \quad l_2 : grant(s_2, o_1, -, read, s_1).
\]

From inheritance of subjects, we can get \( l_3 : grant(s_2, o_1, *, read, \#) \). \( l_1 \) and \( l_2 \) are conflicting and \( l_3 \) will override \( l_2 \) because the grantor \( s_1 \) in \( l_2 \) gets its grant right from grantor \( \# \) in \( l_3 \), although \( l_2 \) is more specific than \( l_3 \).

Consider another two authorizations

\[
l_1 : grant(s_1, o_2, +, read, \#) \quad \text{and} \quad l_2 : grant(s_2, o_1, -, read, \#).
\]

From inheritance of subjects and objects, we can get two conflicting authorizations \( l_3 : grant(s_2, o_2, +, read, \#) \) and \( l_4 : grant(s_2, o_2, -, read, \#) \). Since their grantors are identical, we consider grantees next. Since \( l_3 \) and \( l_4 \) come from \( l_1 \) and \( l_3 \) which have \( s_1 \) and \( s_2 \) as their grantees respectively, and \( s_1 < s_2 \), \( l_4 \) will override \( l_3 \).
Now we consider the third case. Let \( l_1 : grant(s_1, o_1, +, read, \#) \) and 
\( l_2 : grant(s_1, o_2, -, read, \#) \) be two authorizations. From inheritance of objects, we 
can get authorization \( l_3 \) from \( l_1, l_3 : grant(s_1, o_2, +, read, \#) \), which conflicts with \( l_2 \). 
Since \( l_3 \) comes from \( l_1 \) that has the same grantor and grantee with \( l_2 \), we consider 
objects next. Since \( l_1 \) has \( o_1 \) as object, \( l_2 \) has \( o_2 \) as object and therefore more specific, 
\( l_2 \) will override \( l_3 \). ■

**Stable model semantics**

There are two leading semantics for extended logic programs: well-founded sem-
antics \([71, 72]\) and stable model semantics \([33, 34]\). In our approach, we will develop 
a stable model based semantics for our DAPs because it provides a more flexible 
manner to deal with contradictory and incomplete information. Hence the stable 
model semantics seems to be more suitable for our purpose of handling authorization 
conflicts.

### 6.3.2 Formal Definition of the Semantics

Let \( \Pi \) be a DAP, the **Base** \( B_\Pi \) of \( \Pi \) is the set of all possible ground literals con-
structible from the predicates appearing in the rules of \( \Pi \) and the constants occurring 
in \( S, O, A, \) and \( T \). Two ground literals are **conflicting** on subject \( s \), object \( o \) and 
access right \( a \) if they are of the form \( grant(s, o, t, a, g) \) and \( grant(s, o, t', a, g') \) and 
\( t \neq t' \). Note that types * and + are considered conflicting in the sense that * holds 
the administrative privilege while + does not. A **ground instance** of \( r \) is a rule ob-
tained from \( r \) by replacing every variable \( x \) in \( r \) by \( \delta(x) \), where \( \delta(x) \) is a mapping 
from the variables to the constants in the same sorts. Two ground rules \( r \) and \( r' \) 
are **conflicting** if \( Head_r \) and \( Head_r' \) are complementary or conflicting literals. Let 
\( G(\Pi) \) denote all ground instances of the rules occurring in \( \Pi \).

A **subset** of the Base of \( B_\Pi \) is **consistent** if no pair of complementary or conflicting 
literals is in it. An **interpretation** \( I \) is any consistent subset of the Base of \( B_\Pi \).

Given an interpretation \( I \subseteq B_\Pi \), a ground literal \( L \) is **true** in \( I \) if \( L \in I \). \( L \) is 
**false** in \( I \) if \( \neg L \in I \); otherwise \( L \) is **unknown** in \( I \). Given a ground rule \( r \in G(\Pi) \) of 
the form
\[ b_0 \leftarrow b_1, \ldots, b_k, \neg b_{k+1}, \ldots, \neg b_m, m \geq 0 \]

the body of \( r \) is true in \( I \) if every literal \( b_i, 1 \leq i \leq k \) is true in \( I \) and every literal \( b_i, k+1 \leq i \leq m \) is not true in \( I \). Rule \( r \) is satisfied in \( I \) if either the head of \( r \) is true in \( I \) or the body of \( r \) is not true in \( I \).

Next are the definitions of delegation relation on subjects and delegation correctness of an authorization set.

**Definition 6.1 (Delegation Relation \(<_{o,a} \) on Subjects)** Let \( G \) be a set of ground authorization literals, \( S, O, A \) are subject, object and access right constant sets respectively. For any subjects \( s, s' \in S \), object \( o \in O \), and access right \( a \in A \), we say \( s \) is delegation-connected to \( s' \) in \( G \) w.r.t. \( a \) and \( o \), denoted by \( s <_{o,a} s' \), if there exists an authorization \( grant(s', o, *, a, s) \in G \), or there exists some subject \( s'' \) satisfying \( s <_{o,a} s'' \) and \( s'' <_{o,a} s' \).

\( s <_{o,a} s' \) means there exists a sequence of subjects \( s, s_1, s_2, \ldots, s_n, s' \) such that \( grant(s_1, o, *, a, s), grant(s_2, o, *, a, s_1), \ldots, grant(s', o, *, a, s_n) \) are all in \( G \).

**Definition 6.2** Let \( G \) be a set of ground authorization literals, \( S, O, A \) are subject, object and access right constant sets respectively. \( G \) is delegation correct, if

1. For any subject \( s \in S \), object \( o \in O \) and access right \( a \in A \), \( s \) can grant \( a \) on \( o \) to any other subject \( s' \) with any type \( t \) if \( s \) is the security administrator \( s \), or \( s \) has been granted \( a \) on \( o \) with delegatable type \( * \). That is, if \( grant(s', o, t, a, s) \in G \) then \( s = \# \) or for some \( g \in S \), \( grant(s, o, *, a, g) \in G \), and

2. For any authorization literal \( grant(s, o, t, a, s') \in G \), \( s <_{o,a} s' \)

Let \( \Pi \) be a DAP, its semantics is formally defined by four steps. First, transform \( \Pi \) into \( \Pi^D \) according to the requirement of delegatable correctness. Second, for the ground version of \( \Pi^D \), \( G(\Pi^D) \), evaluate its inheritance propagation \( G^*(\Pi^D) \) induced by the partial orders on the subjects, objects and access rights. Third, define the conflict resolution rules to solve the possible contradictions. Fourth, extend the classical concept of an answer set to take into account of the presence of possible contradictions.
Transformation of a DAP

We first introduce two new predicates \textit{can-grant} and \textit{delegate}. \textit{can-grant} has a
type of \( S \times O \times A \). The first argument is \textit{subject}, the second is \textit{object} and the third is \textit{access right}. Intuitively, \textit{can-grant}(s, o, a) means that subject \( s \) has the right to
grant access \( a \) on object \( o \) to other subjects. \textit{delegate} has a type of \( S \times S \times O \times A \).
The arguments are \textit{grantor}, \textit{subject}, \textit{object} and \textit{access right} respectively from left
to right. Intuitively, \textit{delegate}(g, s, o, a) means subject \( g \) has directly or indirectly
granted subject \( s \) access \( a \) on object \( o \) with type \( * \). The two predicates are derived
by the system using logical rules of inference.

Let \( \Pi \) be a DAP, we transform \( \Pi \) to \( \Pi^D \) through the following steps.

\textbf{Step 1}

Adding the following four rules into \( \Pi^D \) to derive the delegation relation for any
object \( o \) and access right \( a \) w.r.t. \( \Pi \).

\begin{align*}
d_1. & \textit{can-grant}(\#, \_, o, \_, a) \leftarrow \\
d_2. & \textit{can-grant}(s, \_, o, \_, a) \leftarrow \textit{grant}(s, \_, o, *, \_, a, g) \\
d_3. & \textit{delegate}(g, s, \_, o, \_, a) \leftarrow \textit{grant}(s, \_, o, *, \_, a, g) \\
d_4. & \textit{delegate}(s, \_, s_1, \_, o, \_, a) \leftarrow \textit{delegate}(s, \_, s_2, \_, o, \_, a), \textit{delegate}(s_2, \_, s_1, \_, o, \_, a)
\end{align*}

We denote \( D = \{d_1, d_2, d_3, d_4\} \).

\textbf{step 2.}

To guarantee the authorization set to be delegation correct, we need to add the
following three conditions to any authorization rule:

- The grantor \( g \) has the right to grant authorizations on the object \( o \) and access
right \( a \) to other subjects.

- The grantor \( g \) and grantees \( s \) can not be identical.

- The grantees \( s \) has not directly or transitively delegated \( a \) on \( o \) to the grantor \( g \).
Hence, for any authorization rule \( r \) in \( \Pi \), it is transformed into \( \hat{r} \) by the following, and added to \( \Pi^D \):

\[
\hat{r}: \text{Head}_r \leftarrow \text{Body}_r, \text{can-grant}(\text{Head}_r.g, \text{Head}_r.o, \text{Head}_r.a),
\]

\( \text{Head}_r.g \neq \text{Head}_r.s, \text{not delegate}(\text{Head}_r.s, \text{Head}_r.g, \text{Head}_r.o, \text{Head}_r.a) \)

where \( \text{Head}_r.s, \text{Head}_r.g, \text{Head}_r.o, \text{Head}_r.a \) denote the grantee, grantor, object and access right argument of the predicate \( \text{grant} \) of \( \text{Head}_r \) respectively.

Rule \( \hat{r} \) means that if \( \text{Body}_r \) is true, \( \text{delegate}(\text{Head}_r.s, \text{Head}_r.g, \text{Head}_r.o, \text{Head}_r.a) \) is not known to be true (that is, \( \text{Head}_s \) does not delegate the access on the object to \( \text{Head}_g \)), \( \text{can-grant}(\text{Head}_r.g, \text{Head}_r.o, \text{Head}_r.a) \) is true (that is, \( \text{Head}_r.g \) has the privilege to grant), and \( \text{Head}_r.g \) is not equal to \( \text{Head}_r.s \), then \( \text{Head}_r \) is true.

**Step 3**

Copy the remaining rules of \( \Pi \) to \( \Pi^D \).

**Example 15** (Example 13 continued) For the DAP \( \Pi \) in Example 13, the \( \Pi^D \) is \( \{ r_1, r_2, r_3, r_4, \hat{r}_5, \hat{r}_6, \hat{r}_7 \} \cup D \), where \( \hat{r}_5, \hat{r}_6 \) and \( \hat{r}_7 \) are:

\( \hat{r}_5: \text{grant}(s_1, o_2, \ast, \text{write}, \#) \leftarrow \text{can-grant}(\#, o_2, \text{write}), \# \neq s_1, \)

\( \text{not delegate}(s_1, \#, o_2, \text{write}) \)

\( \hat{r}_6: \text{grant}(s_2, o_2, -, \text{write}, s_1) \leftarrow \text{can-grant}(s_1, o_2, \text{write}), s_1 \neq s_2, \)

\( \text{not delegate}(s_2, s_1, o_2, \text{write}) \)

\( \hat{r}_7: \text{grant}(\_, o, -, \text{write}, \#) \leftarrow \text{secret}(\_), \text{not dba}(\_), \text{can-grant}(\#, o, \text{write}), \)

\( \# \neq \_, \text{not delegate}(\_, \#, o, \text{write}) \)

**Authorization Propagation**

Next we consider the authorization propagations along the hierarchies of subjects, objects and access rights represented by the corresponding partial orders. From now on, a rule with variables is treated as shorthand for the set of all its ground instances. That is, we consider the ground version of \( \Pi^D, G(\Pi^D) \). The basic idea is
to give every rule in $G(\Pi^D)$ a three-ary identifier which indicates what this rule is about in terms of subject, object and access right. Then, the propagations of a rule along the underlying hierarchies can be evaluated by replacing each occurrence of the proper coordinate of its identifier in the rule by any greater subject, object or access right w.r.t. the corresponding partial order. The identifiers of rules will also be used for solving conflict, which will be illustrated later.

Now we explain how to identify a rule. First, let us look at the authorization as it is the most important property that needs to be inherited. For any authorization rule, the grantees, object and access right of predicate grant in its head is used as its identifier. Second, to achieve the inheritance as much as possible, for any rule that is not an authorization rule, select the minimal subject, object, and access right w.r.t. the corresponding partial order as its identifier. $\bot$ is used instead when there is no subject or object or access right in the rule. Third, for any rule in $G(D)$, $(\bot, \bot, \bot)$ will be used as its identifier, since the predicates can-grant and delegate do not appear in user programs and therefore no contradiction w.r.t them will occur. Also since they are instantiated for all the possible constants, no inheritance is needed for them.

We say a constant $c$ is minimal in a rule $r$ w.r.t. some partial order $<$ if $c$ is in $r$ and there is no constant $c'$ in $r$ such that $c' < c$. Add $\bot$ to the constant sets $S$, $O$ and $A$, and suppose $\bot$ is not comparable to any element in the sets w.r.t $<_S$, $<_O$ and $<_A$ respectively. Then, we have

**Definition 6.3** Function $\text{rid}: G(\Pi^D) \rightarrow S \times O \times A$, is defined by the following:

$$\text{rid}(r) = \begin{cases} 
(\bot, \bot, \bot) & \text{if } r \in G(D) \\
(\text{Head}_r.s, \text{Head}_r.o, \text{Head}_r.a) & \text{if } r \text{ is an authorization rule} \\
(s, o, a) & \text{otherwise, where} \\
\text{s is minimal in } r \text{ w.r.t } <_S \text{ or } \bot \text{ if } \exists s(s \in S \wedge s \text{ is in } r), \\
\text{o is minimal in } r \text{ w.r.t } <_O \text{ or } \bot \text{ if } \exists o(o \in O \wedge o \text{ is in } r), \\
\text{a is minimal in } r \text{ w.r.t } <_A \text{ or } \bot \text{ if } \exists a(a \in A \wedge a \text{ is in } r). 
\end{cases}$$

There may exist more than one minimal subject, object or access right in a rule; in this situation, simply select any one.
Example 16 (Example 15 continued) $G(\Pi^D) = \{r_1, \ldots, r_6, \hat{r}_{71}, \hat{r}_{72}, \hat{r}_{73}, \hat{r}_{74}, \hat{r}_{75}, \hat{r}_{76}\} \cup G(D)$, where $\hat{r}_{71}, \hat{r}_{72}, \ldots, \hat{r}_{76}$ are ground instances of rule $\hat{r}_7$ in $\Pi^D$ obtained by replacing $*$ and $\#$ with $s_1$ and $o_1, s_1$ and $o_2, s_2$ and $o_1, s_2$ and $o_2, \#$ and $o_1$, or $\#$ and $o_2$, respectively.

$\hat{r}_{71} : grant(s_1, o_1, -, write, \#) \leftarrow secret(o_1), not dba(s_1), can-grant(\#, o_1, write), \# \neq s_1, not delegate(s_1, \#, o_1, write)$

$\hat{r}_{72} : grant(s_1, o_2, -, write, \#) \leftarrow secret(o_2), not dba(s_1), can-grant(\#, o_2, write), \# \neq s_1, not delegate(s_1, \#, o_2, write)$

$\hat{r}_{73} : grant(s_2, o_1, -, write, \#) \leftarrow secret(o_1), not dba(s_2), can-grant(\#, o_1, write), \# \neq s_2, not delegate(s_2, \#, o_1, write)$

$\hat{r}_{74} : grant(s_2, o_2, -, write, \#) \leftarrow secret(o_2), not dba(s_2), can-grant(\#, o_2, write), \# \neq s_2, not delegate(s_2, \#, o_2, write)$

$\hat{r}_{75} : grant(\#, o_1, -, write, \#) \leftarrow secret(o_1), not dba(\#), can-grant(\#, o_1, write), \# \neq \#, not delegate(\#, \#, o_1, write)$

$\hat{r}_{76} : grant(\#, o_2, -, write, \#) \leftarrow secret(o_2), not dba(\#), can-grant(\#, o_2, write), \# \neq \#, not delegate(\#, \#, o_2, write)$

The following rules are in $G(D)$:

$d_{11} : can-grant(\#, o_2, write) \leftarrow$

$d_{21} : can-grant(s_1, o_2, write) \leftarrow grant(s_1, o_2, *, write, \#)$

$d_{31} : delegate(\#, s_1, o_2, write) \leftarrow grant(s_1, o_2, *, write, \#)$

...

The rid values for rules in $G(\Pi^D) - G(D)$ are:

$\text{rid}(r_1) = (s_1, \bot, \bot)$

$\text{rid}(r_2) = (s_2, \bot, \bot)$

$\text{rid}(r_3) = (\bot, o_1, \bot)$

$\text{rid}(r_4) = (\bot, o_2, \bot)$

$\text{rid}(\hat{r}_5) = (s_1, o_2, write)$

$\text{rid}(\hat{r}_6) = (s_2, o_2, write)$

$\text{rid}(\hat{r}_{71}) = (s_1, o_1, write)$
Let $R_{rid}$ be a relation on $G(\Pi^D)$ such that $r R_{rid} r'$ iff $\text{rid}(r) = \text{rid}(r')$. It is easy to see that $R_{rid}$ is an equivalence relation (reflexive, symmetric, and transitive) on $G(\Pi^D)$. Hence the equivalence classes of $R_{rid}$ form a partition of $G(\Pi^D)$. Let $x = \text{rid}(r)$ for some $r$, we use $G_x(\Pi^D)$ to denote the equivalence class of $R_{rid}$ that contains all the rules with the same rid value $x$. That is: $G_x(\Pi^D) = \{ r | r \in G(\Pi^D) \land \text{rid}(r) = x \}$.

Rule propagations along hierarchies of subjects, objects and access rights are then evaluated by replacing the corresponding coordinate of $x$ with any greater constant for every rule in $G_x(\Pi^D)$, and for every equivalence class $G_x(\Pi^D)$. There is an exception for the authorization rule when the grant type of the predicate grant in its head is negative. In this case, the propagation will go in the opposite direction. That is, if subject $s$ has been granted the access $a$ on object $o$ with type $\ast$ or $+$, then, implicitly, $s$ has also been granted any access right $a'$ such that $a < a'$ on $o$ with the same type. If subject $s$ has been denied for the access $a$ on object $o$, then, implicitly, it has also been denied any access right $a'$ such that $a' < a$.

Let $r_{c/c'}$ mean to replace every occurrence of $c$ in $r$ by $c'$, then we have the following definition.

**Definition 6.4** Let $\Pi$ be a DAP, $G_{(s,o,a)}(\Pi^D)$ be any equivalence class of $G(\Pi^D)$ w.r.t. $R_{rid}$, then we define:

$G_{(s,o,a)}^*(\Pi^D) = G_{(s,o,a)}(\Pi^D) \cup$

$(r_{s/s',o/o',a/a'} | r \in G_{(s,o,a)}(\Pi^D) \land (s \leq_S s') \land (o \leq_O o') \land ((\text{Head}_r \text{ is not an authorization literal and } a \leq_A a') \lor (\text{Head}_r \text{ is an authorization literal and } \text{Head}_r.t \neq - \text{ and } a \leq_A a'))$
or \((\text{Head}_r\text{.t} = - \text{ and } a' \leq_A a)\))

\[
G^*(\Pi^D) = \bigcup G^*_e(\Pi^D) \text{ where } G^*_e(\Pi^D) \text{ is an equivalence class of } R_{rid}.
\]

The denotation \(c \leq c'\) denotes \(c < c'\) or \(c = c'\). Note that \(G^*(\Pi^D)\) contains all the rules implied by the hierarchies of subjects, objects and access rights, as well as the rules to guarantee the authorization set derived by \(\Pi\) to be delegation correct. It is worth mentioning that a rule may be derived by propagation from several equivalence classes. We treat each one as distinct. Thus all the \(G^*_e(\Pi^D)\), where \(G^*_e(\Pi^D)\) is an equivalence class of \(R_{rid}\), also form a partition of \(G^*(\Pi^D)\).

**Example 17** (Example 16 continued) The partition of \(G(\Pi^D)\) w.r.t. \(R_{rid}\) is:

\[
\begin{align*}
G_{s_1, l, l} &= \{r_1\}, \\
G_{s_2, l, l} &= \{r_2\}, \\
G_{l, o_1, l} &= \{r_3\}, \\
G_{l, o_2, l} &= \{r_4\}, \\
G_{s_1, o_1, write} &= \{\hat{r}_{71}\}, \\
G_{s_1, o_2, write} &= \{\hat{r}_{5}, \hat{r}_{72}\}, \\
G_{s_2, o_1, write} &= \{\hat{r}_{73}\}, \\
G_{s_2, o_2, write} &= \{\hat{r}_{6}, \hat{r}_{74}\}, \\
G_{l, o_1, write} &= \{\hat{r}_{75}\}, \\
G_{l, o_2, write} &= \{\hat{r}_{76}\}, \\
G_{l, l, l} &= G(D)
\end{align*}
\]

Then \(G^*(\Pi^D)\) is the union of the following sets:

\[
\begin{align*}
G^*_{s_1, l, l} &= \{r_1, r'_1\} \\
G^*_{s_2, l, l} &= \{r_2\} \\
G^*_{l, o_1, l} &= \{r_3, r'_3\} \\
G^*_{l, o_2, l} &= \{r_4\} \\
G^*_{s_1, o_1, write} &= \{\hat{r}_{71}, \hat{r}'_{72}, \hat{r}_{73}, \hat{r}'_{74}\} \\
G^*_{s_1, o_2, write} &= \{\hat{r}_5, \hat{r}'_{5}, \hat{r}_5', \hat{r}_3', \hat{r}_{73}, \hat{r}_{74}'\} \\
G^*_{s_2, o_1, write} &= \{\hat{r}_{73}, \hat{r}_{74}'\} \\
G^*_{s_2, o_2, write} &= \{\hat{r}_6, \hat{r}_{74}\}
\end{align*}
\]
\[ G^*_{(s_0, \text{write})} = \{ \hat{r}_{75}, \hat{r}_{76} \} \]
\[ G^*_{(s_1, \text{read})} = \{ \hat{r}_{76} \} \]
\[ G^*_{(\perp, \perp, \perp)} = G(D), \text{ where} \]

\[ r^1_1 : \text{dba}(s_2) \leftarrow \]
\[ r^1_3 : \neg \text{secret}(o_2) \leftarrow \]
\[ \hat{r}^1_5 : \text{grant}(s_1, o_2, \ast, \text{read}, \#) \leftarrow \text{can-grant}(\#, o_2, \text{read}), \# \neq s_1, \]
\[ \quad \text{not delegate}(s_1, \#, o_2, \text{read}) \]
\[ \hat{r}^2_5 : \text{grant}(s_2, o_2, \ast, \text{write}, \#) \leftarrow \text{can-grant}(\#, o_2, \text{write}), \# \neq s_2, \]
\[ \quad \text{not delegate}(s_2, \#, o_2, \text{write}) \]
\[ \hat{r}^3_5 : \text{grant}(s_2, o_2, \ast, \text{read}, \#) \leftarrow \text{can-grant}(\#, o_2, \text{read}), \# \neq s_2, \]
\[ \quad \text{not delegate}(s_2, \#, o_2, \text{read}) \]

Note that \( \hat{r}_{72} \) means an occurrence of \( \hat{r}_{72} \) in another equivalence class, and so on.

Conflict Resolution

The partial orders on \( S, O \) and \( A \) are extended to \( S \times O \times A \) by using lexicographical ordering [53].

Definition 6.5 A relation \( <_{S,O,A} \) on \( S \times O \times A \) is defined by: \( (s, o, a) <_{S,O,A} (s', o', a') \) if \( s < s' \lor (s = s' \land o <_O o') \lor (s = s' \land o = o' \land a <_A a') \).

From the definition, we know that \( <_S \) dominates, except for equality, in which case we consider \( <_O \). If equality holds again, we pass to \( <_A \). It is easy to see that \( <_{S,O,A} \) is a partial order on \( S \times O \times A \) [53].

To achieve our purpose of resolving conflicts, we now define a relation \( <_r \) on \( G^*(\Pi^P) \) in terms of the classes the rules belong to. Basically, the class that a rule belongs to tells us where this rule comes from. For example, if \( r \in G^*_{x}(\Pi^P) \), then we can infer that \( r \) is either in \( G^*_{x}(\Pi^P) \) or derived from \( G^*_{x}(\Pi^P) \) through inheritance of subjects, objects or access rights.

Definition 6.6 Let \( \Pi \) be a DAP, a relation \( <_r \) on \( G^*(\Pi^P) \) is defined by: \( r <_r r' \) if \( r \in G^*_{x}(\Pi^P) \land r' \in G^*_{y}(\Pi^P) \land x <_{S,O,A} y \). \( <^*_r \) is the transitive closure of \( <_r \).
Example 18 (Example 17 continued) According to the definition of $<_{s,o,a}$, the following holds:

$(s_1, \bot, \bot) < (s_2, \bot, \bot), (s_2, o_1, \text{write}), (s_2, o_2, \text{write});$

$(\bot, o_1, \bot) < (\bot, o_2, \bot);$  

$(s_1, o_2, \text{write}) < (s_2, o_1, \text{write}), (s_2, o_2, \text{write});$

$(s_1, o_1, \text{write}) < (s_1, o_2, \text{write}), (s_2, o_1, \text{write}), (s_2, o_2, \text{write});$

$(s_2, o_1, \text{write}) < (s_2, o_2, \text{write}).$

Therefore, we have:

$r_1, r_1^1 < r_2, \hat{r}_{73}, \hat{r}_{74}^m, \hat{r}_6, \hat{r}_{74};$

$r_3, r_3^1 < r_4;$

$r_5, r_5^1, r_5^2, r_5^3, \hat{r}_{72}, \hat{r}_{74}^m < \hat{r}_{73}, \hat{r}_{74}^m, \hat{r}_6, \hat{r}_{74};$

...

Note that we omit the subscript for $<$, which is easy to see from the context. ■

Clearly $<_r$ is a partial order. Intuitively speaking, $r <_r r'$ means that $r'$ is more specific than $r$. Now we are ready to define the conflict resolution rules.

Definition 6.7 Suppose $I$ is an interpretation for $G^*(\Pi^D)$, $r$ and $r'$ are conflicting ground rules, we say that $r$ overrides $r'$ in $I$ if:

1. body$_r$ is true in $I$, and

2. at least one of the following conditions holds:

(a). Head$_r$ and Head$_{r'}$ are conflicting literals on object $o$ and access right $a$, and delegate(Head$_r.g$, Head$_{r'}g$, o, a) is true in $I$.

(b). Head$_r$ and Head$_{r'}$ are conflicting literals on object $o$ and access right $a$, Head$_{r}.g =$ Head$_{r'}g$ and $r' <_r r$

(c). Head$_r$ and Head$_{r'}$ are complementary literals and $r' < r$

Note that Head$_r.g$ denotes the grantor of predicate grant in Head$_r$, and so does Head$_{r'}g$. When there are multiple occurrences of a rule $r$, $r$ overrides $r'$ if one occurrence of $r$ overrides $r'$. $r$ is overridden by $r'$ if all the occurrences of $r$ are overridden by $r'$.

As we will prove later, all the access rights must be first delegated from the administrator $\S$. Thus, according to our conflict resolution method, the authorizations
granted by \( \# \) can never be overridden by authorizations granted by other grantors. In other words, \( \# \) has the highest priority, while the subjects that directly receive delegatable access rights from \( \# \) have the second highest priorities, and so on. This means that the priorities of grantors decrease along the delegation path. In this way we can realise the controlled delegation of authorizations. When grantors are the same, the more specific rule will dominate, which can support exception of inheritance.

Example 19 (Example 18 continued) Suppose \( I \) is an interpretation for \( G^*(\Pi^D) \), and \( \text{body}_{\#r_2}, \text{body}_{r_4}, \text{body}_{\#r_3}, \) and \( \text{body}_{r_7} \) are all true in \( I \). The conflicting rules in \( G^*(\Pi^D) \) are \( r_1^1 \) and \( r_2^1 \) and \( r_4^1 \); \( r_5^2 \) and \( r_6^2 \) and \( r_7^4 \). The heads of \( r_1^1 \) and \( r_2^1 \) are complementary literals, since \( r_1^1 < r_2^1 \), \( r_2 \) will override \( r_1^1 \). For the same reason, since \( r_4^1 < r_4 \), \( r_4 \) will override \( r_3^1 \).

The heads of \( r_5^2, r_6 \) and \( r_7^4 \) are conflicting literals, \( r_6 \) is overridden by \( r_5^2 \) since \( \text{delegate}(\#, s_1, a_2, \text{write}) \) is true as long as \( r_5 \) is satisfied, although \( r_5^2 < r_6 \), which means that \( r_5 \) is more specific than \( r_6 \). \( r_5^2 \) is overridden by \( r_7^4 \), since the grantors of the predicate \( \text{grant} \) in the heads of both rules are \( \# \), and \( r_5^2 < r_7^4 \).

The Stable Model Semantics

The answer set semantics is based on the so-called stable model semantics. We need to extend the traditional definition of answer set to take into account the explicit contradictions.

Definition 6.8 Let \( I \) be an interpretation for a DAP \( \Pi \), the reduction of \( \Pi \) w.r.t \( I \), denoted by \( \Pi^I \), is defined as the reduction of \( G^*(\Pi^D) \) w.r.t \( I \). That is the set of rules obtained from \( G^*(\Pi^D) \) by deleting

1. every rule overridden in \( I \), and
2. every rule that has a formula not \( L \) in its body with \( L \in I \), and
3. all formulas of the form not \( L \) in the bodies of the remaining rules.

Given a set \( R \) of ground rules, we denote by \( \text{pos}(R) \) the positive version of \( R \), obtained from \( R \) by considering each negative literal \( \neg p(t_1, ..., t_n) \) as a positive one with predicate symbol \( \neg p \).
Definition 6.9 Let $M$ be an interpretation for $\Pi$. We say that $M$ is an answer set for $\Pi$ if $M$ is a minimal model of the positive version $\text{pos}(\Pi^M)$. If $M$ is an answer set for $\Pi$, then its subset of all the authorization literals $A$ is called an authorization answer set for $\Pi$.

Example 20 (Example 19 continued) For the DAP $\Pi$ in Example 13, the only answer set for it is: $\text{dba}(s_1), -\text{dba}(s_2), -\text{secret}(o_1), \text{secret}(o_2), \text{grant}(s_1, o_2, *, write, \#), \text{grant}(s_1, o_2, *, read, \#), \text{grant}(s_2, o_2, -, write, \#), \text{grant}(s_2, o_2, *, read, \#), \text{can-grant}(\#, o_1, write), \text{can-grant}(\#, o_1, read), \text{can-grant}(\#, o_2, write), \text{can-grant}(\#, o_2, read), \text{can-grant}(s_1, o_2, write), \text{can-grant}(s_1, o_2, read), \text{delegate}(\#, s_1, o_2, write), \text{delegate}(\#, s_1, o_2, read), \text{delegate}(\#, s_2, o_2, read)$.

6.3.3 Access Control Policy

A query is a three-ary tuple $(s, o, a)$ in $S \times O \times A$, which indicates a subject $s$ requesting access $a$ over object $o$. The access control policy is a function $f$ from $S \times O \times A$ to $\{\text{true, false, undecided}\}$. Given a request $(s, o, a)$, if $f(s, o, a) = \text{true}$ then it is granted. If $f(s, o, a) = \text{false}$ then it is denied. Otherwise, $f(s, o, a) = \text{undecided}$; it is left to decide by the access control mechanism system. Although most access control mechanisms may treat $\text{undecided}$ as denial, the distinction between $\text{false}$ and $\text{undecided}$ is important. $\text{false}$ means denial in the stronger sense that its negation exists, while $\text{undecided}$ means denial in the weaker sense that it does not succeed.

According to stable model semantics, there may exist several authorization answer sets for a given DAP; they may be not consistent with each other in the sense that they may contain conflicting literals. We will adopt an optimistic approach to deal with this problem. Let $\Pi$ be a DAP, $A_1, ..., A_m$ be its authorization answer sets. For any query $(s, o, a)$, $f(s, o, a) = \text{true}$ if there exists $\text{grant}(s, o, *, a, g)$ or $\text{grant}(s, o, +, a, g)$ for some $g$ in some $A_i$, $1 \leq i \leq m$. Otherwise, $f(s, o, a) = \text{false}$, if there exists $\text{grant}(s, o, -, a, g)$ for some $g$ in some $A_i$, $1 \leq i \leq m$. Otherwise, $f(s, o, a) = \text{undecided}$.

On the other hand, there may exist no authorization answer set for a given DAP $\Pi$. In this case, we say $\Pi$ is not $\text{well-defined}$. 
6.4 Properties of DAP

Definition 6.10 A DAP $\Pi$ is a well-defined DAP if there exists an authorization answer set for it.

Proposition 6.1 Let $\Pi$ be a well-defined DAP, $A$ be an authorization answer set for it, then $A$ is delegation correct.

Proof. Let $M$ be the corresponding answer set of $A$. According to the definition of answer set, for any rule in $G^*(\Pi^D)$, it is either satisfied or overridden in $M$. For rules in $G(D)$, they should be satisfied since predicates $\text{can-grant}$ and $\text{delegate}$ do not appear in user programs and therefore no conflicting rules exist for them.

For any authorization rule $r$, it is changed to $\hat{r}$ in $\Pi^D$. According to $\hat{r}$, for any subjects $s, s'$, object $o$, access right $a$ and authorization type $t$, if $\text{grant}(s', o, t, a, s)$ is true in $M$, then $\text{can-grant}(s, o, a)$ must be true in $M$. From rule $d_1$ and $d_2$, if $\text{can-grant}(s, o, a)$ is true in $M$, then $s$ is $\|$ or there exists $g$ such that $\text{grant}(s, o, *, a, g)$ is true in $M$. Since $A$ contains all the authorization literals in $M$, condition 1 is satisfied.

For the second condition, for any subjects $s, s'$, object $o$ and access right $a$, it is easy to see that $s <_{o,a} s'$ is true in $M$ iff $\text{delegate}(s, s', o, a)$ is true in $M$. Since any authorization rule $r$ is transformed to $\hat{r}$, it follows immediately from the definition of $\hat{r}$.

The conclusion holds for $A$ too, since $A$ contains all the authorization literals in $M$. Therefore $A$ is delegation correct. □

Proposition 6.2 Let $\Pi$ be a well-defined DAP, and $A$ be an authorization answer set for it. For any object $o \in O$ and access right $a \in A$, the delegation relation $<_{o,a}$ is a strict partial order (irreflexive, antisymmetric and transitive) in $G$.

Proof. For any subjects $s, s'$, object $o$ and access right $a$, it is easy to see that $s <_{o,a} s'$ is true in $M$ iff $\text{delegate}(s, s', o, a)$ is true in $M$. From rule $d_4$, we know that $<_{o,a}$ is transitive in $M$. Next we prove $<_{o,a}$ is irreflexive and antisymmetric. For any $s, s'$ in $S$ such that $s <_{o,a} s'$ holds, if $\text{delegate}(s, s', o, a)$ is derived from rule $d_3$, then we know that $\text{grant}(s', o, *, a, s)$ is true in $M$. Thus, according to
\( r, s \neq s' \) is true and delegate \((s', s, o, a)\) is not known to be true in \( M \). That is, \( s' \not<_{o,a} s \). If delegate \((s, s', o, a)\) is derived from rule \( d_4 \), then without loss of generality, suppose it is derived from delegate \((s, s'', o, a)\)...(1) and delegate \((s'', s', o, a)\), and delegate \((s'', s', o, a)\) is derived from rule \( d_3 \). Therefore grant \((s', o, *, a, s'')\) is true in \( M \), and then \( s'' \neq s' \) is true and delegate \((s', s'', o, a)\)...(2) is not known to be true in \( M \), according to rules \( d_3 \) and \( r \). Now if \( s = s' \), then from (1), delegate \((s', s', o, a)\) is true, which contradicts (2). Again if delegate \((s', s, o, a)\) is true, then combined with (1), we can get that delegate \((s', s'', o, a)\) is true from rule \( d_4 \), which contradicts (2). So \( s \neq s' \) and \( s' \not<_{o,a} s \). So \( <_{o,a} \) is irreflexive and antisymmetric. So \( <_{o,a} \) is a strict partial order w.r.t. \( M \). The conclusion holds for \( A \) too, since \( A \) contains all the authorization literals in \( M \). \( \square \)

Proposition 6.3 Let \( \Pi \) be a well-defined DAP, \( A \) be an authorization answer set for it, \( S \) is the subject constant set. For any delegation relation \( <_{o,a} \) on \( S \) w.r.t \( A \), let \( S' = \{ s | \exists s', s, s' \in S \land (s <_{o,a} s' \lor s' <_{o,a} s) \} \). Then the security administrator \( \# \) is the least subject in \( S' \) w.r.t. to \( <_{o,a} \).

Note that \( c \) is the least element of a poset \((C, \leq)\) if \( c < c' \) for all \( c' \in C \). The set \( S' \)
contains all the subjects in \( S \) that are comparable to some subjects w.r.t \( <_{o,a} \).

Proof. Let \( M \) be the corresponding answer set of \( A \). First note that every finite nonempty poset \((Z, \leq)\) has a minimal element [53]. If \( s \in S' \) is minimal, since \( s \) is comparable to some subjects, there exists some \( s' \in S' \) such that \( s <_{o,a} s' \), which means delegate \((s, s', o, a)\) is true in \( M \). Without loss of generality, suppose delegate \((s, s', o, a)\) is not obtained by transitivity, since we can always find such \( s' \).

So, grant \((s', o, *, a, s)\) is true according to \( d_3 \). So can-grant \((s, o, a)\) is true. From \( d_1 \) and \( d_2 \), we know that either \( s = \# \) or grant \((s, o, *, a, g)\) is true for some \( g \). However, from grant \((s, o, *, a, g)\) we can get delegate \((g, s, o, a)\) from \( d_3 \), that is, \( g <_{o,a} s \), which means that \( s \) is not minimal. Hence we get \( s = \# \).

Since all the subjects in \( S' \) are comparable w.r.t \( <_{o,a} \), and \( \# \) is the only minimal subject, \( \# \) is the least in \( S' \) w.r.t \( <_{o,a} \). The conclusion holds for \( A \) too, since \( A \) contains all the authorization literals in \( M \). \( \square \)
§6.5 Related Work and Summary

The proposition says if a subject $s$ is comparable to some subject $s'$ w.r.t $<_{o,a}$, which means $s$ is delegated by $s'$ or delegate to $s'$ access $a$ on $o$, then $s <_{o,a} s$. Therefore all access rights are delegated initially from security administrator $\hat{a}$.

6.5 Related Work and Summary

Logic based approaches have been developed by many researchers recently for the purpose of formalizing authorization specifications and evaluations. The advantage of this methodology is that it separates policies from implementation mechanisms, gives policies precise semantics, and provides a unified framework that can support multiple policies.

In this chapter, we have proposed a logic program based formulation that supports delegatable authorizations, negation as failure and classical negation, and rules inheritance. Since the administration of access rights can be delegated, our model suits large-scale systems where decentralised administration of access rights is needed. The expressive power and nonmonotonic reasoning of extended logic programs provide users a feasible way to express complex security policy. On the other hand, allowing rule inheritance can usually greatly reduce the complexity of authorization specification. We have also presented a conflict resolution method which supports the controlled delegation and exception. Exception is supported by using the more specific-take-precedence principle, while controlled delegation is supported by giving higher priorities to the predecessors w.r.t the delegation relation path.

Abadi et al proposed a modal logic based approach for access control in distributed systems [1]. Their work focuses on how to believe that a principal (subject) is making a request, either on his/her own or on someone else's behalf. The delegation in their model mainly concerns on the access right itself. Whereas our work focuses on handling delegation of access right administration and resolving conflicts when multiple administrators exist. Jason Crampton et al's work is also based on modal logic [23], which investigates the ability of representing and reasoning about the implementation of a real-world access control mechanism. Woo and Lam proposed an expressive language to authorization in distributed systems [76]. In their method, they consider structural properties inherent in authorization
and provide formal semantics evaluation which is based on extended logic program. But they did not deal with authorization delegation and conflict resolution problem. Jajodia et al also proposed a logical language and illustrated how it can specify authorization, conflict resolution, access control and integrity constraint checking [41]. However they also did not consider delegation of access right administration and conflict resolution problem.

Bertino et al also proposed a logic framework for authorizations [12]. In their model, they considered hierarchically structured domain of subjects, objects and access rights for authorization, supported both negation as failure and classical negation, and provided a conflict resolution method. The main differences between these two approaches are: (a) They did not consider delegation, and hence their model mainly suits for centralised control of access rights. In fact, for specification, we can show that Bertino et al’s model can be a special case of ours when we restrict the authorization type only to + and −. (b) The conflict resolution method is different. In our model, the delegation relation is considered first, which is a dynamic one and often different from object to object. In their model, the user (grantor) hierarchy is first considered, which is a static one and the same for all objects. We argue that, in many cases especially in a decentralised authorization administration, the grantor’s priorities are usually different for different objects, and usually the owner of the object should have the highest priority. (c) The underlying specifications in these two methods are different. In their model, two different literal forms are used: ordinary literal (they refer to them as simple literal) and referential literal, which makes their formalism relatively complex. We use only the ordinary literal form and therefore our syntax is much simpler.
Chapter 7

A Temporal Authorization Model

7.1 Motivation

In the real world, there are many situations in which users may need to be granted some authorizations for limited periods of time. For example, a contract staff in a university can access the university network for the length of his/her contract. A part time programmer in a company is allowed to work only during his/her permitted working hours, such as 9AM to 1PM. There are also many situations in which users may need to be delegated some administrative privileges for a certain duration. For example, a contract database administrator can grant, revoke or possibly delegate authorizations for the database to other users only within his period of contract. Temporal authorizations have been used for this purpose. By using temporal authorizations, subjects may grant permissions to others for a certain duration, and these permissions are automatically revoked on the expiration of the time intervals, beginning from the instant they were initially permitted.

Temporal authorizations have been recognised as an important practical security policy to adopt. Temporal aspects have been studied for Discretionary Access Control (DAC) and Role-based Access Control Model (RBAC) models. In [14], Bertino et al presented an authorization model with temporal capabilities. Both positive and negative authorizations are supported in their model, and derivation rules can be expressed in which four temporal operators can be used. However it doesn’t support authorization delegations and thus is mainly suitable for a centralised authorization model. In [15], Bertino et al introduced a temporal RBAC model, which supports both periodic activations and deactivations of roles, and temporal dependencies among such actions. In [6], Barker also represented a temporal RBAC model. The
model is based on logic programs which incorporate the Simplified Event Calculus and can support time-constrained permissions and membership of roles.

In this chapter, we propose a temporal decentralised authorization model for DAC, which allows temporal authorization delegations and negations. When temporal authorizations are delegated in a model, not only the facts such as who owns the authorizations but also the facts such as who owns the administrative privileges in a period of time are derived from the system. A conflict resolution method based on the underlying delegation relation and temporal relation is proposed, which can support controlled delegation, temporal suspension and the automatic authorization update. The controlled delegation is realised by giving higher priorities to the predecessors than the successors along the delegation path. The temporal suspension is realised by giving higher priority to the authorization with smaller interval, while automatic authorization update is achieved by giving higher priority to the authorization with newer interval. In addition, cyclic authorizations are avoided and cascade overriding is supported when an administrative privilege is overridden. Authorization inheritances are also supported in our model.

To take advantage of strong expressive and reasoning power of logic programming, we will develop our temporal authorization framework based on extended logic programs [34], which supports both negation as failure and classical negation. In fact, there are a number of important reasons why such logic programs are of particular value for representing security models with temporal authorizations: they enable a wide range of security requirements to be represented in a high-level language and as an executable specification which may be formally verified as satisfying organizational, administrative, user and technical requirements prior to implementation; they have well-defined semantics; and sound, complete, terminating and efficient proof methods are known to exist for classes of logic programs in which realistic temporal security technologies may be specified.

In this chapter, we show how extended logic programs may be used to specify complex security policies which support time constrained permissions. In our framework, authorization rules are specified in a temporal delegatable authorization program (TDAP). This is an extended logic program associated with different
types of partial orderings on the domain. These orderings specify various inheritance relationships among subjects, objects and access rights in the domain. A set of domain-independent temporal rules are given to achieve the property of temporal delegation correctness, temporal authorization propagation and temporal conflict resolution. The semantics of a TDAP is defined based on the stable model. We also investigate the condition under which a unique answer set for a TDAP exists.

The rest of this chapter is organised as follows. Section 2 describes the syntax of the temporal delegatable authorization program (TDAP). Section 3 explains the general ideas. Section 4 defines general temporal rules, the semantics of the program, and the access control policy. Section 5 investigates some important properties of TDAP, and finally Section 6 concludes the chapter with some remarks.

### 7.2 syntax of TDAP

Our language $\mathcal{L}$ is a many-sorted first order language, with six disjoint sorts $S, O, A, T, TI, I$ for subjects, objects, access rights, authorization types, time points and time intervals respectively. Variables are denoted by strings starting with lower case letters, and constants by strings starting with upper case letters.

In addition, three partial orders $<_S, <_O$ and $<_A$ are defined on sorts $S, O$ and $A$ respectively, which are used to represent the inheritance hierarchical structures of subjects, objects and access rights. There are three authorization types denoted by $-, +$ and $\ast$, where $-$ means negative, $+$ means positive, and $\ast$ means delegatable. A negative authorization specifies that the access must be forbidden, while a positive authorization specifies that the access must be granted. A delegatable authorization specifies that the administrative privilege for the access as well as the access itself must be granted. In other words, $\ast$ means $+$ plus administrative privilege on the access. The functions are user-defined functions plus time and interval functions given in the Section 3.3.

The predicates are typed with fixed arity. It consists of a set of ordinary predicates defined by users, the time and interval predicates introduced in Section 3.3 and two built-in predicate symbols for delegatable authorizations, $grant$ and $own$. $grant$ is a 6-term predicate symbol with type $S \times O \times T \times A \times S \times I$. The first
argument is the grantees, the second is the object, the third is the authorization type, the fourth is the access right, the fifth is the grantor and the sixth is the time interval of this authorization. Intuitively, grant(s, o, t, a, g, i) means that s is granted by g the access right a on object o with authorization type t for a period of i. grant is called authorization predicate. own is a 2-term predicate symbol with type $S \times O$. Intuitively, own(s, o) means that s is the owner of o. A variable or a constant is a term. If f is a n-ary function symbol and $t_1,...,t_n$ are terms then $f(t_1,...,t_n)$ is a term. An atom is a construct of the form $p(t_1,...,t_n)$, where p is a predicate of arity n in P and $t_1,...,t_n$ are terms. A literal is either an atom p or the negation of the atom $\neg p$, where the negation sign $\neg$ represents classical negation. Specially, we forbid the negation form of the authorization predicate grant, since we can use the argument of authorization type in grant to express the opposite meaning. Two literals are complementary if they are of the form p and $\neg p$, for some atom p. A rule r is a statement of the form:

$$b_0 \leftarrow b_1, ..., b_k, \neg b_{k+1}, ..., \neg b_m, m \geq 0$$

where $b_0, b_1, ..., b_m$ are literals, and $\neg$ is the negation as failure symbol. A Temporal Delegatable Authorization Program, denoted as TDAP, consists of a finite set of rules. A term, an atom, a literal, a rule or program is ground if no variable appears in it.

Example 21 Information security concerns that arise in the context of consent in health care systems relate to personal health data, consent to the usage of the health data for a period of time and denials given by patients, the capacity to create a temporal consent (i.e. temporal consent delegation). Various forms of temporal patient consents can be supported in our model. Consider the following situation. A patient John delegates a temporal consent for reading and writing of his Health Data (HD) to his hospital Doctor (Dr) and his Family GP (FGP) from Jan. 01, 2002 to Dec. 31, 2002, but denies disclosure to his Immediate Family (IF)(because, for example, HD contains information about STD conditions) during this period. This situation can be represented by the following TDAP II.
(r₁) own(John, HD) ←

(r₂) grant(Dr, HD, *, Write, John, [01/01/2002, 31/12/2002]) ←

(r₃) grant(FGP, HD, *, Write, John, [01/01/2002, 31/12/2002]) ←

(r₄) grant(IF, HD, −, Read, John, [01/01/2002, 31/12/2002]) ←

Please note that, in order to have more expressive examples, we introduce user-defined intervals such as hours and dates. The authorizations to John’s hospital doctor and family GP for “reading” of his health data are implied by rules (r₂) and (r₃) respectively, which will be explained in more detail in the next section.

7.3 Basic Ideas

In this section, we give basic considerations for semantics of TDAP. Three major aspects are taken into consideration: temporal delegation correctness, temporal authorization propagation and temporal conflict resolution.

7.3.1 Temporal Delegation Correctness

To make the authorization delegation more reasonable, specific constraints are needed to be enforced on it. First, it is natural to require that authorization grantors own the relative administrative privileges for the effective periods of those authorizations. In our model, the privileged grantors include the owners of objects and the subjects that hold * type of authorizations during those periods. Second, cyclic authorizations should be avoided at any time, which are usually unnecessary. To illustrate the problem, consider Example 21 again. John has delegated his hospital doctor the capacity to further create a temporal consent on his health data. What happens if the doctor denies disclosure to John by assigning him a negative authorization on his health data? This is clearly undesirable. Also it does not make much sense if the doctor grants John to read his health data. The constraints are formally specified in the following definition.
Definition 7.1 *(Temporal delegation correctness)* An authorization set is temporal delegation correct if it satisfies the following two conditions: (a) subject $s$ can grant other subjects an access right $a$ over object $o$ for an interval $i$ if and only if $s$ is the owner of $o$ or $s$ has been granted $a$ over $o$ with a delegatable type $*$ for an interval $i'$ such that $i \subseteq i'$; (b) if a subject $s$ receives a delegatable authorization directly or indirectly from another subject $s'$ on some object $o$ and access right $a$ with time interval $i$, then $s$ cannot grant $s'$ any further authorization on the same $o$ and $a$ for any time period $i'$ such that $i' \cap i \neq \emptyset$.

7.3.2 Temporal Authorization Propagations

Next we consider the temporal authorization propagations along hierarchies of subjects, objects and access rights represented by the corresponding partial orders. It has been widely recognised that these propagations can greatly reduce the amount of authorizations that need to be explicitly specified, and thus can simplify the work by a great deal.

Example 22 In Example 21, usually we need to give every person in John’s immediate family an explicit negative authorization. However, if authorization inheritance along the subject hierarchy is supported, then we just need to indicate who belongs to his immediate family (through “<” relation in our model). The negative authorization defined by $(r_4)$ will automatically propagate to each member of his immediate family. On the other hand, John’s health data may consist of personal information and health information. Health information may further consist of clinical information and treatment information. By supporting authorization inheritance along the object hierarchy, rules $(r_2)$ to $(r_4)$ will automatically propagate to all these data. In addition, with the support of authorization inheritance along the access right hierarchy, and by defining $\text{Write} < \text{Read}$, rule $(r_2)$ and $(r_3)$ will propagate to “Read” automatically.
7.3.3 Temporal Conflict Resolution

Since both positive and negative authorizations are acceptable in our framework, conflicts among authorizations may arise. As we mentioned before, allowing authorization delegation greatly increases the chance of conflict since there may exist multiple administrators for an access right on an object. For example, in Example 21, the patient John, his hospital doctor and his family GP are all administrators for his health data. Furthermore, since the authorizations have time intervals associated with them, the conflicts are also time related. They may conflict all the time, or only some time in the spans of their intervals. Two temporal authorizations are considered to be conflicting if they are on the same object and access right, and their intervals are intersected, and they have different authorization types. Note that types * and + are considered to be conflicting in the sense that * holds the administrative privilege while + does not. The basic idea of our method of resolving conflicts is outlined in terms of the following five principles.

Principle 1: Solving conflicts based on the underlying temporal delegation relation by giving higher priorities to the predecessors

Along the temporal delegation path, we give higher priorities to the predecessors. In particular, if a subject $s$ delegates to a subject $s'$ directly or indirectly an authorization on object $o$ and access right $a$ at time $ti$, then, when a conflict w.r.t $o$ and $a$ occurs at $ti$, the authorization from $s$ (i.e. $s$ is the grantor) will override the one from $s'$ at $ti$. As we will show later, all the access rights on an object $o$ must be first delegated from the owner of $o$ at any time $ti$. Hence, according to our conflict resolution method, the owner has the highest priority at any time $ti$, while the subjects that directly receive delegable access rights from the owner at time $ti$ have the second highest priorities at that time, and so on. In other words, the priorities of grantors decrease along the temporal delegation path. In this way, the authorizations granted by owners will never be overridden by authorizations granted by other grantors. Therefore, despite delegation, the owners can still take control of the objects. As we said before, we believe that this controlled delegation can take advantage of both centralised and decentralised authorization administration.
Example 23 In Example 21, John has the highest priority on his health data and his authorizations can never be overridden. Suppose John’s family GP tries to disclose his health data to his immediate family by giving them a positive authorization as follows:

\[(r_5) \text{grant}(IF, HD, +, Read, FGP, [01/01/2002, 31/12/2002]) \leftarrow \]

\[(r_5) \text{conflicts with rule (r_4) and is overridden by (r_4) since the grantor of (r_5), FGP, receives his/her capacity to grant from the grantor of (r_4), the patient John.} \]

Principle 2: Smaller time intervals take precedence

If the grantors of two conflicting authorizations are identical, then we consider time intervals of the authorizations. If one interval is contained within the other, then we use the smaller time interval-take-precedence principle, which means that the authorization with smaller time interval will override the one with the larger interval. This principle can be used to support temporary suspension or exception of an authorization.

Example 24 In Example 21, suppose John does not want his family GP to be involved in his treatment during the period from June 01, 2002 to Oct. 31, 2002. Then he may issue the following authorization:

\[(r_6) \text{grant}(FGP, HD, -, Write, John, [01/06/2002, 31/10/2002]) \leftarrow \]

\[(r_6) \text{conflicts with (r_3) and will override (r_3) during the period from June 01, 2002 to Oct. 31, 2002 according to Principle 2.} \]

Principle 3: Newer time intervals take precedence

If the grantors of two conflicting authorizations are identical, and the two intervals are overlapping, then we will let the authorization with newer interval dominate.
This principle can support automatic update of authorizations without revoking the old ones.

**Example 25** In Example 21, suppose John moved to another hospital later and therefore refuses his previous doctor to write on his health data from Sep. 01, 2002 to Sep. 30, 2003. The following rule expresses his denial:

(\(r_7\) grant(Dr, HD, -, Write, John, [01/09/2002, 30/09/2003]) \(\leftarrow\)

(\(r_7\)) conflicts with (\(r_2\)) during the period from Sep. 01, 2002 to Dec. 31, 2002 and will override it according to Principle 3. ■

**Principle 4: Solving conflicts according to the types of authorizations**

If all the above policies fail to solve the conflict between two authorizations, then we will solve the conflicts in a pessimistic manner by using the negative-take-precedence principle. This will help to achieve the maximum degree of security. We solve the conflict based on the authorization type, by giving the authorization type – the highest priority followed by + and then by *.

**Example 26** In Example 21, suppose John's hospital doctor allows a Health Consultant (HC) to see his health data while John's family GP denies the disclosure from Jan. 01, 2002 to June 30, 2002. The following rules express this situation.

(\(r_8\) grant(HC, HD, -, Read, FGP, [01/01/2002, 30/06/2002]) \(\leftarrow\)

(\(r_9\) grant(HC, HD, +, Read, Dr, [01/01/2002, 31/12/2002]) \(\leftarrow\)

(\(r_8\)) conflicts with (\(r_9\)) during the period from Jan. 01, 2002 to June 30, 2002 and will override it according to Principle 4. ■

**Principle 5: Cascade overriding**

When a delegatable authorization is overridden in a period, the authorizations granted by the grantee of that authorization should also be overridden in that period.
In other words, we support cascade overriding.

**Example 27** For the previous examples, rule \((r_9)\) will be cascade overridden during the period from Sep. 01, 2002 to Dec. 31, 2002 due to the overriding of rule \((r_2)\) by \((r_7)\) during this time. ■

### 7.4 Formal semantics of TDAP

In this section, we give a set of domain-independent rules to capture the features of temporal delegation correctness, temporal conflict resolution and temporal authorization propagation along the hierarchies of subjects, objects and access rights. The basic idea is to combine these general rules with a set of domain-specific rules defined by user through TDAP to derive the authorizations holding in a period of time.

#### 7.4.1 Temporal Rules for Delegation

To capture the feature of temporal delegation correctness, we introduce several new auxiliary predicates and treat them as system reserved words. `delegater` has a type of \(S \times S \times O \times A \times I\). The arguments are `grantor`, `subject`, `object`, `access right` and `time interval` respectively. Intuitively, \(delegater(g, s, o, a, i)\) means that a subject \(g\) has directly or indirectly granted a subject \(s\) an access \(a\) on object \(o\) with the delegatable type \(*\) for the period of \(i\). `exist-delegater(g, s, o, a, i)` has the same type as `delegater` and is used to avoid the existential quantifier to be used in a rule. This is because in extended logic programs all the variables in clauses are considered to be universally quantified. `exist-delegater(g, s, o, a, i)` is true if there is any delegation from \(g\) to \(s\) on \(o\) and \(a\) within time interval \(i\). `grant` has the same type and meaning as `grant` except that `grant` is derived from the system which has passed through the temporal delegation correctness check. The following five rules are used to deal with the temporal delegation correctness.

\[(D1) \, grant(s, o, t, a, g, i) \leftarrow grant(s, o, t, a, g, i), own(g, o)\]
(D2) grant1(s, o, t, a, g, i) ← grant(s, o, t, a, g, i), grant1(g, o, *, a, g', i'),  
i ⊆ i', g ≠ s, not exist-delegate(s, g, o, a, i)

(D3) delegate(g, s, o, a, i) ← grant1(s, o, *, a, g, i)

(D4) delegate(s, s1, o, a, i ∩ i') ← delegate(s, s2, o, a, i),  
delegate(s2, s1, o, a, i')

(D5) exist-delegate(s, g, o, a, i) ← grant(s, o, t, a, g, i),  
delegate(s, g, o, a, i'), i' ∩ i ≠ ∅

Rules (D1) and (D2) define predicate grant1, which represents an authorization that satisfies the temporal delegation correctness requirement. Rule (D1) means that any grant from the owner will be accepted. Rule (D2) states that for any grant(s, o, t, a, g, i), if grant1(g, o, *, a, g', i') is true and i ⊆ i', meaning that g has the privilege to grant in i; g is not equal to s, meaning that g cannot grant to itself; and exist-delegate(s, g, o, a, i) is not known to be true (negation as failure), meaning that s does not delegate a on o to g at any time within i (to avoid cyclic authorization); then grant1(s, o, t, a, g, i) is true. As stated earlier, predicate exist-delegate is introduced to avoid the existential quantifier to be used in (D2), since in extended logic programs all the variables in clauses are considered to be universally quantified.

Rule (D3) and (D4) derive the temporal delegation relation. Rule (D3) says, if g grants s an access a on o with type * for a period i, then g delegates access a on o to s for a period i. Note that grant1 is used here so that the delegation relation is based on the system derived authorizations other than the arbitrary authorizations defined by the users, which may be not temporal delegation correct. Rule (D4) states that, if s delegates to s2 and s2 delegates to s1 an access a on o for the periods i and i' respectively, then, by transitivity, s delegates to s1 the access a on o for the period i ∩ i'.

Rule (D5) defines exist-delegate. It expresses that for a given grant(s, o, t, a, g, i), if there exists delegate(s, g, o, a, i') such that i ∩ i' ≠ ∅, then exist-delegate(s, g, o, a, i) is true; this means that s has delegated a to g at some time within i. It should be noted that predicates delegate and grant1 are mutually defined from each other,
and (D1) is the base.

Example 28 In Example 21, suppose we add two authorizations. A receptionist (Recp) grants a Nurse (Nrs) to “read” John’s health data; and John’s doctor grant John “not read” of his own data; both of them are from 10/01/2002 to 31/12/2002.

\[(r_{10}) grant(Nrs, HD, +, Read, Recp, [10/01/2002, 31/12/2002]) \leftarrow\]
\[(r_{11}) grant(John, HD, -, Read, Dr, [10/01/2002, 31/12/2002]) \leftarrow\]

Combined with rules (D1) to (D5), it is not difficult to see that the corresponding predicates grant1 for \(r_{10}\) and \(r_{11}\) will not be derived from the system. 

\[grant1(Nrs, HD, +, Read, Recp, [10/01/2002, 31/12/2002])\] will not be derived because Recp is neither the owner of HD nor holder of \(*\) type of authorization on “read” of HD (\(grant1(Recp, HD, *Read, g, i)\) is false for any \(g\) and \(i\)). In other words, Recp is not qualified to grant to others.

\[grant1(John, HD, -, Read, Dr, [10/01/2002, 31/12/2002])\] is also not derived, as we expect, because \(exist-delegate(John, Dr, HD, Read, [10/01/2002, 31/12/2002])\) is true; this means that Dr gets the administrative privilege for HD from John. ■

7.4.2 Temporal Rules for Propagation

The following rules are used to realise the temporal propagation.

\[(H1) grant(s, o, t, a, g, i) \leftarrow grant(s’, o, t, a, g, i), s’ <_S s\]
\[(H2) grant(s, o, t, a, g, i) \leftarrow grant(s, o’, t, a, g, i), o’ <_O o\]
\[(H3) grant(s, o, t, a, g, i) \leftarrow grant(s, o, t, a’, g, i), a’ <_A a, t \neq -\]
\[(H4) grant(s, o, t, a, g, i) \leftarrow grant(s, o, t, a’, g, i), a <_A a’, t = -\]

Note that unlike other propagations that are downward along the hierarchies, when the grant type is -, the propagation is upward along the access right hierarchy. In Example 21, John’s immediate family is forbidden to read, which implies that the family members are also forbidden to write.
7.4.3 Temporal Rules for Conflict Resolution

We will use a set of domain-independent temporal rules to realise the conflict resolution policies defined in Section 3. First, we introduce four more system reserved predicates $\text{override}1$, $\text{override}2$, $\text{grant}2$ and $\text{hold}$ with the same type $S \times O \times T \times A \times S \times TT$. $\text{override}1(s, o, t, a, g, ti)$ means that the authorization $\text{grant}1(s, o, t, a, g, i)$ was overridden at time $ti$ by some other authorizations according to the Principle 1,2 or 3. $\text{grant}2(s, o, t, a, g, ti)$ means that the authorization $\text{grant}1(s, o, t, a, g, i)$ was not overridden at time $ti$ by any other authorization according to the Principle 1,2,3 or 5. $\text{override}2(s, o, t, a, g, ti)$ means that $\text{grant}2(s, o, t, a, g, i)$ was overridden at time $ti$ by some other authorizations according to Principle 4. $\text{hold}(s, o, t, a, g, ti)$ means an authorization that actually holds at time $ti$ (not overridden by any other authorization). We define $- < + <$ $*$ and the rules are as follows:

(C1) $\text{override}1(s, o, t, a, g, ti) \leftarrow \text{grant}1(s, o, t, a, g, i), \text{grant}1(s, o, t', a, g', i'), \text{delegate}(g', g, o, a, i''), \text{in}(ti, i), \text{in}(ti, i'), \text{in}(ti, i''), t \neq t'$

(C2) $\text{override}1(s, o, t, a, g, ti) \leftarrow \text{grant}1(s, o, t, a, g, i), \text{grant}1(s, o, t', a, g', i'), i' \sqsubset i, \text{in}(ti, i'), t \neq t'$

(C3) $\text{override}1(s, o, t, a, g, ti) \leftarrow \text{grant}1(s, o, t, a, g, i), \text{grant}1(s, o, t', a, g', i'), i \circ i', \text{in}(ti, i), \text{in}(ti, i'), t \neq t'$

(C4) $\text{grant}2(s, o, t, a, g, ti) \leftarrow \text{grant}1(s, o, t, a, g, i), \text{own}(g, o), \text{in}(ti, i), \text{not override}1(s, o, t, a, g, ti)$

(C5) $\text{grant}2(s, o, t, a, g, ti) \leftarrow \text{grant}1(s, o, t, a, g, i), \text{in}(ti, i), \text{not override}1(s, o, t, a, g, ti), \text{grant}2(g, o, *, a, g', ti)$

(C6) $\text{override}2(s, o, t, a, g, ti) \leftarrow \text{grant}2(s, o, t, a, g, ti), \text{grant}2(s, o, t', a, g', ti), t' < t$

(C7) $\text{hold}(s, o, t, a, g, ti) \leftarrow \text{grant}2(s, o, t, a, g, ti), \text{own}(g, o), \text{not override}2(s, o, t, a, g, ti)$

(C8) $\text{hold}(s, o, t, a, g, ti) \leftarrow \text{grant}2(s, o, t, a, g, ti), \text{not override}2(s, o, t, a, g, ti), \text{hold}(g, o, *, a, g', ti)$
Rules (C1), (C2) and (C3) correspond to the conflict resolution principles 1, 2 and 3. Rule (C1) states that, given an authorization \( \text{grant}1(s, o, t, a, g, i) \), if there exists another authorization \( \text{grant}1(s, o, t', a, g', i') \) such that \( \text{delegate}(g', g, o, a, i'') \) is true (meaning that \( g' \) delegates to \( g \) the privilege \( a \) on \( o \) for an interval \( i'' \)), \( t \) is not equal to \( t' \) (meaning conflicting), and \( i, i' \) and \( i'' \) are all intersected, then for any time \( t_i \) that is inside the intersection of \( i, i' \) and \( i'' \), \( \text{overridden}1(s, o, t, a, g, ti) \) is true; this means that \( \text{grant}1(s, o, t, a, g, i) \) is overridden at time \( t_i \) according to Principle 1. Rule (C2) states that, given an authorization \( \text{grant}1(s, o, t, a, g, i) \), if there exists another authorization \( \text{grant}1(s, o, t', a, g, i') \) such that interval \( i' \) during \( i \) is true, \( t \) is not equal to \( t' \) (meaning conflicting), then for any time \( t_i \) that is inside \( i' \), \( \text{overridden}1(s, o, t, a, g, ti) \) is true; this means that \( \text{grant}1(s, o, t, a, g, i) \) is overridden at time \( t_i \) according to Principle 2. Rule (C3) states that, given an authorization \( \text{grant}1(s, o, t, a, g, i) \), if there exists another authorization \( \text{grant}1(s, o, t', a, g, i') \) such that interval \( i' \) overlaps \( i \) is true, \( t \) is not equal to \( t' \) (meaning conflicting), then for any time \( t_i \) that is inside \( i \) and \( i' \), \( \text{overridden}1(s, o, t, a, g, ti) \) is true; this means that \( \text{grant}1(s, o, t, a, g, i) \) is overridden at time \( t_i \) according to Principle 3.

Rules (C4) and (C5) are used to derive the authorizations that are not overridden or cascade overridden by using the Principles 1, 2, 3 and 5. Rule (C4) states that, given an authorization \( \text{grant}1(s, o, t, a, g, i) \), if \( \text{own}(g, o) \) is true, and \( \text{overridden}1(s, o, t, a, g, ti) \) is not known to be true, where \( t_i \) is in \( i \), then \( \text{grant}2(s, o, t, a, g, ti) \) is true. Rule (C5) states that, given an authorization \( \text{grant}1(s, o, t, a, g, i) \), if \( \text{grant}2(g, o, *, a, g', ti) \) is true which means that \( g \) still owns the right to grant \( a \) on \( o \), and \( \text{overridden}1(s, o, t, a, g, ti) \) is not known to be true, where \( t_i \) is in \( i \), then \( \text{grant}2(s, o, t, a, g, ti) \) is true.

Rule (C6) corresponds to conflict resolution Principle 4. It states that, given an authorization \( \text{grant}2(s, o, t, a, g, ti) \), if there exists another conflicting authorization \( \text{grant}2(s, o, t', a, g', ti) \) such that the type \( t' < t \), then \( \text{overridden}2(s, o, t, a, g, ti) \) is true; this means that \( \text{grant}2(s, o, t, a, g, ti) \) is overridden according to Principle 4.

Rules (C7) and (C8) are used to derive the authorizations that are not overridden or cascade overridden by any other authorizations. Rule (C7) states that, given an
authorization $\text{grant}_2(s,o,t,a,g,ti)$, if $\text{own}(g,o)$ is true and $\text{overridden}_2(s,o,t,a,g,ti)$ is not known to be true, then $\text{hold}(s,o,t,a,g,ti)$ is true. Rule (C8) states that, given an authorization $\text{grant}_2(s,o,t,a,g,ti)$, if $\text{hold}(g,o,*,a,g',ti)$ is true, which means that $g$ still owns the right to grant $a$ on $o$, and $\text{overridden}_2(s,o,t,a,g,ti)$ is not known to be true, then $\text{hold}(s,o,t,a,g,ti)$ is true.

It should be noted that we always use two rules to derive authorizations, such as (C4) and (C5), or (C7) and (C8), in order to capture the temporal delegation correctness and cascade overriding. We have used several predicates to denote authorizations. $\text{grant}$ is for users to specify the temporal authorizations; $\text{grant}_1$ and $\text{grant}_2$ are intermediate predicates derived by the system, whereas $\text{hold}$ denotes the final authorization derived by the system. In addition, $\text{grant}$ and $\text{grant}_1$ are based on time intervals while $\text{grant}_2$ and $\text{hold}$ are based on time instants. Let $X$ denote all the general rules, i.e. $X = \{D1,...,D5,H1,...,H4,C1,...,C8\}$.

### 7.4.4 Formal Semantics

There are two leading semantics for extended logic programs: well-founded semantics and stable model semantics [34]. We select stable model semantics for TDAPs because stable model semantics provides a more flexible manner to deal with contradicted or incomplete information.

Let $\Pi$ be a TDAP, the Base $B_\Pi$ of $\Pi$ is the set of all possible ground literals constructed from the system reserved predicates, predicates appearing in the rules of $\Pi$, the constants occurring in $S,O,A,T,T',TT$ and function symbols in $\Pi$. A ground instance of $r$ is a rule obtained from $r$ by replacing every variable $x$ in $r$ by $\delta(x)$, where $\delta(x)$ is a mapping from the variables to the constants in the same sorts. Let $G(\Pi)$ denote all ground instances of the rules occurring in $\Pi$. Two ground literals are conflicting on subject $S$, object $O$ and access right $A$ if they are of the form $\text{hold}(S,O,T,A,G,TT)$ and $\text{hold}(S,O,T',A,G',TT)$, and $T \neq T'$. A subset of the Base of $B_\Pi$ is consistent if no pair of complementary or conflicting literals is in it. An interpretation $I$ is any consistent subset of the Base of $B_\Pi$.

**Definition 7.2** Given a TDAP $\Pi$, an interpretation for $\Pi$ is any interpretation of $\Pi \cup X$. 
**Definition 7.3** Let \( I \) be an interpretation for a TDAP \( \Pi \), the reduction of \( \Pi \) w.r.t \( I \), denoted by \( \Pi^I \), is defined as the set of rules obtained from \( G(\Pi \cup X) \) by deleting (1) each rule that has a formula not \( L \) in its body with \( L \in I \), and (2) all formulas of the form not \( L \) in the bodies of the remaining rules.

Given a set \( R \) of ground rules, we denote by \( pos(R) \) the positive version of \( R \), obtained from \( R \) by considering each negative literal \( \neg p(t_1, \ldots, t_n) \) as a positive one with predicate symbol \( \neg p \).

**Definition 7.4** Let \( M \) be an interpretation for \( \Pi \). We say that \( M \) is an answer set for \( \Pi \) if \( M \) is a minimal model of the positive version \( pos(\Pi^M) \). If \( M \) is an answer set for \( \Pi \), then its subset of all the literals with predicate name \( hold \) is called authorization answer set for \( \Pi \), denoted by \( A \).

A query is a four-ary tuple \( (s, o, a, ti) \) in \( S \times O \times A \times T \), which denotes a subject \( s \) requests access \( a \) over object \( o \) at time \( ti \). The access control policy is a function \( f \) from \( S \times O \times A \times T \) to \{true, false, undecided\}. Given a request \( (s, o, a, ti) \), if \( f(s, o, a, ti) = true \) then it is granted. If \( f(s, o, a, ti) = false \) then it is denied. Otherwise, \( f(s, o, a, ti) = undecided \), and it is left to decide by the access control system. Although most access control mechanisms may treat \( undecided \) as denial, the distinction between \( false \) and \( undecided \) is important. \( false \) means denial in the stronger sense in that its negation exists, while \( undecided \) means denial in the weaker sense in that it does not succeed.

According to stable model semantics, there may exist several authorization answer sets for a given TDAP, and they may not consistent with each other in the sense that they may contain conflicting literals. We will adopt optimistic approach to deal with this problem. Let \( \Pi \) be a TDAP, \( A_1, \ldots, A_m \) be its authorization answer sets. For any query \( (s, o, a, ti) \), \( f(s, o, a, ti) = true \) if there exists \( hold(s, o, *, a, g, ti) \) or \( hold(s, o, +, a, g, ti) \) for some \( g \) in some \( A_i \), \( 1 \leq i \leq m \). Otherwise, \( f(s, o, a, ti) = false \) if there exists \( hold(s, o, -, a, g, ti) \) for some \( g \) in some \( A_i \), \( 1 \leq i \leq m \). Otherwise \( f(s, o, a, ti) = undecided \).

On the other hand, there may exist no authorization answer set for a given TDAP \( \Pi \). In this case, we say \( \Pi \) is not well-defined.
7.5 Properties of TDAP

In this section, we will investigate some important properties of TDAP. First we show that the answer set is temporal delegation correct as we expected. Then, we show that all the authorizations on an object are first delegated from the owner at any time. Therefore, according to our conflict resolution method, the owner's authorizations will not be overridden by other users' authorizations. Finally, we study the syntactical property of TDAP which will guarantee the existence of the unique answer set.

Definition 7.5 A DAP $\Pi$ is a well-defined TDAP if there exists an authorization answer set for it.

We next define a delegation relation on subjects.

Definition 7.6 (Delegation Relation $<_{o,a,ti}$ on Subjects) Let $G$ be a set of ground authorization literals with predicate name hold. For any subjects $s, s' \in S$, object $o \in O$, access right $a \in A$, and time $ti \in T_I$, we say $s$ is delegation-connected to $s'$ in $G$ w.r.t $a$ and $o$ at time $ti$, denoted by $s <_{o,a,ti} s'$, if there exists an authorization $hold(s', o, *, a, s, ti) \in G$, or there exists some subject $s''$ satisfying $s <_{o,a,ti} s''$, and $s'' <_{o,a,ti} s'$.

$s <_{o,a,ti} s'$ means there exists a sequence of subjects $s, s_1, \cdots, s_n, s'$ such that $hold(s_1, o, *, a, s, ti), hold(s_2, o, *, a, s_1, ti), \cdots, hold(s', o, *, a, s_n, ti)$ are all in $G$.

Proposition 7.1 Let $\Pi$ be a well-defined TDAP, $A$ be an authorization answer set for it, then $A$ is temporal delegation correct.

Proof. Our proof is based on the definition of temporal delegation correctness. Note that a time point $t$ can be considered as a special interval $[t, t]$. Condition (a) is easily seen from general rules (C7) and (C8). To prove condition (b), we only need to prove that for any $hold(s', o, t, a, s, ti) \in A$, $s' <_{o,a,ti} s$. We define another relation $<_{o,a,ti}$ on subjects as follows. Suppose $M$ is the corresponding answer set for $\Pi$, then $s <_{o,a,ti} s'$, if there exists an authorization $grant1(s', o, *, a, s, i) \in M$ such that $ti \in i$ or there exists some subject $s''$ satisfying $s <_{o,a,ti} s''$, and $s'' <_{o,a,ti} s'$. 
According to general rule (D2), it is not difficult to see that $s' \not<_{o,a,ti} s$. Since $<_{o,a,ti}$ is a subset of $<_{o,a,ti}'$ (hold derived from grant1), it follows that $s' \not<_{o,a,ti} s$. \hfill \Box

**Proposition 7.2** Let $\Pi$ be a well-defined TDAP, $A$ be an authorization answer set for it. Then, for any $o \in O$, $a \in A$, $ti \in T_I$, the delegation relation $<_{o,a,ti}$ w.r.t $A$ is a strict partial order.

**Proof.** We define another relation $<_{o,a,ti}'$ on subjects as follows. Suppose $M$ is the corresponding answer set for $\Pi$, then $s <_{o,a,ti}' s'$, if there exists an authorization $\text{grant1}(s', o, *, a, s, i) \in M$ such that $ti \in i$ or there exists some subject $s''$ satisfying $s <_{o,a,ti} s''$, and $s'' <_{o,a,ti}' s'$. According to general rule (D2), it is not difficult to see that $<_{o,a,ti}'$ is a strict partial order. Since $<_{o,a,ti}$ is a subset of $<_{o,a,ti}'$ (hold derived from grant1), it follows that $<_{o,a,ti}$ is also a strict partial order. \hfill \Box

Next, we show that all the authorizations on an object are first delegated from the owner. Let $\text{owner}(o) = s$ such that $\text{own}(o, s)$ is true.

**Proposition 7.3** Let $\Pi$ be a well-defined TDAP, $A$ be an authorization answer set for it. For any delegation relation $<_{o,a,ti}$ on $S$ w.r.t $A$, let $S' = \{s | \exists s'(s, s' \in S \land (s <_{o,a,ti} s' \vee s' <_{o,a,ti} s))\}$. Then $\text{owner}(o)$ is the least subject in $S'$ w.r.t. $<_{o,a,ti}$.

**Proof.** First note that every finite nonempty poset $(Z, \leq)$ has a minimal element. Suppose $s \in S'$ is minimal w.r.t. $<_{o,a,ti}$. Since $s$ is comparable to some subjects, there exists some $s' \in S'$ such that $s <_{o,a,ti} s'$. Without loss of generality, suppose $s <_{o,a,ti} s'$ is not derived by transitivity, since we can always find such $s'$. Hence, $\text{hold}(s', o, *, a, s, ti)$ is true. According to rules (C7) and (C8), $s = \text{owner}(o)$ or $\text{hold}(s, o, *, a, s'', ti)$ is true for some $s'' \in S'$. However, the second condition follows $s'' <_{o,a,ti} s$, which means that $s$ is not minimal. So we get $s = \text{owner}(o)$.

Since all the subjects in $S'$ are comparable w.r.t $<_{o,a,ti}$, and $\text{owner}(o)$ is the only minimal subject, $\text{owner}(o)$ is the least in $S'$ w.r.t $<_{o,a,ti}$. \hfill \Box

The proposition states that if a subject $s$ is comparable to some other subject $s'$ w.r.t $<_{o,a,ti}$, which means that $s$ is delegated by $s'$ or delegates to $s'$ access $a$ on $o$ at time $t$, then $\text{owner}(o) <_{o,a,ti} s$. Therefore all access rights are delegated initially from the owner.
Finally we discuss under what conditions, a TDAP can have exactly one answer set. Without loss of generality, consider a TDAP without variables.

Definition 7.7 The delegation graph on an object o and an access right a of a grounded TDAP Π has a node for each subject symbol occurring in Π and a directed arc labelled with i from the node for subject g to the node for subject s and all subjects $s <_{S} s'$ whenever there is an authorization grant$(s,o,a,*,g,i)$ in the head of a rule.

Proposition 7.4 (Unique Answer Set Theorem) Given a well-defined TDAP Π, if pos(Π) is stratified, and there is no cycle in its delegation graph for any object o and access a such that all the labels are intersected, then Π has unique answer set.

Proof. We only need to prove that pos(Π ∪ X) is also stratified [34]. It is easy to see that no cycles containing “not” will involve user defined predicates, since Π is stratified, and all the predicates in X are system reserved except for grant. Thus we only need to prove that pos(X) is stratified. It is easy to check that only one cycle containing “not” exists in pos(X), that is:

$$grant_1 \leftarrow not\, exist-delegate \leftarrow delegate \leftarrow grant_1.$$

Here $a \leftarrow b$ denotes the directed arc from node b to a. Since there is no cycle in Π’s delegation graph such that all the labels are intersected, according to rule (D5), for any grant$(s,o,t,a,g,i)$, exist-delegate$(s,g,o,a,i)$ will not be derived. Hence not exist-delegate$(s,g,o,a,i)$ in the body of rule (D2) is always true and can be removed. It follows that pos(X) is also stratified.

7.6 Summary

In this chapter, we have proposed a logic program based formulation that supports temporal delegatable authorizations, negation as failure and classical negation, and authorization inheritance. The basic idea is to combine domain-dependent rules defined by users with domain-independent rules, which are used to enforce temporal delegation correctness, resolve conflicts and derive inherited authorizations.
Since the administration of access rights can be delegated, our model is suitable for large decentralised systems where distributed administration of access rights are needed. The expressive power and nonmonotonic reasoning of extended logic programs provide users a feasible way to express complex security policy with temporal constraints. We have also presented a conflict resolution method which supports the controlled delegation, temporal suspension of authorizations and automatic authorization update. A condition under which unique stable model exists is also given.
Chapter 8

An Application in Health Care

This chapter considers how our proposed authorization models can support the requirement in a health care application. The application being considered is the e-consent system for electronic patient records. It outlines an architecture for e-consent system within health care and examines the different types of e-consent. Then, we will use the logic model discussed in Chapter 6 to develop a means for representing and evaluating patient consent to the disclosure of their data, with particular emphasis on security. The choice behind the selection of logic model is that its language and conflict resolution policy best suit the needs of various types of e-consent.

8.1 E-consent on Health Data

Computer information processing and electronic communication technologies play an increasingly important role in the area of health care. More and more coordination of health care relies on the electronic transmission of confidential information about patients between different health care and community services. This evolving health care system brings new uses and disclosures of patient data. However, since the patient data is confidential, the need for electronic forms of patient consent, referred to as e-consent [21] has to be considered. Patients should be able to give or withhold ‘e-consent’ to those who wish to access their electronic health information. That is, the health information technology infrastructure needs to support confidential consumer and service provider interactions.

For example, electronic patient records are usually considered as an essential prerequisite for health care and should be opened to the whole clinical team involved
in patient care. The presence of an electronic distributed environment means that patient information will be able to be distributed over the network. More clinical workers in a greater diversity of locations can access it more often and more easily. Consequently, without the existence of some e-consent mechanism, such widespread information could be accessed by unauthorised individuals, or used for purposes not originally consented to by the patient, which can lead to substantial breaches of personal privacy.

The main application areas that need e-consent are those that support coordinated health care. This is typified by data-sharing among multiple teams of health care professionals and institutions, and uses and disclosures of patient data that are additional to those that have hitherto occurred.

The aim is to ensure that appropriate health care consumer authorization is recognised and obtained in any access to patient information, which would not normally be accessible to a practitioner. In this way, the consumers are able to actively participate in an informed manner, in the decision making and in the governance of the health services they need.

At present, individuals working in the health system are responsible for obtaining consent from a patient, or determining whether consent exists for using or passing on a patient's information. In an electronic environment, the existence of patient consent is determined by automatic processes. For example, a set of computer rules can be defined to determine whether clinical staff working in a hospital might have their right to access electronic patient records. This is actually the so-called access control in computer information security.

A number of characteristics of consent are important, including: the explicit and implicit consents; whether consent or denial may be presumed; conflict resolution policy when consent and denial exist at the same time for an individual; the specificity and boundedness of consent; consent delegation; consent update and revocation. Therefore, a flexible e-consent system is needed which is capable of representing and evaluating a complex set of consent instructions with inclusions and exclusions conditions to access information.

Another special issue about the translation of consent rules to the electronic envi-
Electronic Patient Records

8.2 Electronic Patient Records

The patient record is an account of a patient's health and disease after he or she has sought medical help. The record contains findings, considerations, test results and treatment information related to the disease process. The increasing demand for well-structured and accessible patient data, in combination with developments in computer science, sparked the development of an electronic patient record. Computers have the potential to improve legibility, accessibility, and structure management. Electronic patient record may be available to the clinician at any point where electronic access is provided to the record data, and they allow simultaneous access by several clinicians at different sites, whereas paper records have to be physically present at the point of use. Data retrieval from electronic patient record is easier than from paper - not just because electronic records are physically more accessible to their users than paper records - but also because the ability to interrogate the content of electronic records for audit and analysis purposes.

Sophisticated information management is critical to the practice of medicine. The existence of an up-to-date, complete database of the medical records and com-
plete dependable patient data not only enhances the quality of medical care but also improves the teaching and continuing medical education and research.

The following is an example of electronic patient record format.

A. Essential Personal and Contact Details

A1. Name—surname, first name
A2. Address—street, suburb, state, postcode
A3. Date of birth
A4. Phone number—home and/or work and/or mobile
A5. Payment method—private fee (cash or credit card or Eftpos), bulk billing, workers' compensation or 3rd party fee

B. Optional Personal and Contact Details

B1. Title (Miss, Ms, Mrs, Mr or Dr)
B2. Alias or preferred name
B3. Email address
B4. Fax number
B5. Occupation
B6. Gender (male, female or unknown)
B7. Marital status (divorced, married, never married, separated or widowed)
B8. Ethnicity (African, Aboriginal, etc)
B9. Country of birth
B10. Next of kin—name, address and phone number
B11. Employer—name, address and phone number
B12. Family members—name and address

C. Clinical Related Personal Details

C1. Status (regular, casual, transferred or visitor patient)
C2. Provider—name of doctor examining the patient
C3. Location of provider—(medical center, specialist center, physiotherapy, dental center or radiology department)
C4. Procedures/treatment code (for billing purposes)
C5. Pathology and radiology results (in/out)—in or not in
C6. Visit history—all visits (past, current and future), to this location of provider
C7. Next appointment (initiated by patient)—name of preferred doctor, date and time

D. Health Details

D1. Consultation

D11. Date of consultation
D12. Current health problems and complains
D13. Examination and notes taking (blood pressure, temperature, pulse, weight, height...)
D14. Management plan (procedures undertaken, prescriptions, issued, preventive advice, advice concerning treatment risks, costs and options)
D15. Pathology and radiology requests
D16. Referrals
D17. Follow up and recall arrangements

D2. Medical History (all consultation information)

D3. Sensitive Medical History

D31. STD (HIV/AIDS) (all consultation information)
D32. Gynaecological conditions (all consultation information)

D4. Other History

D41. Smoking and alcohol history
§8.3 Role-Based Access Control And E-consent

D42. Employment details
D43. Operation history
D44. Family history (health status of 1st degree relatives and/or cause of their deaths)
D5. Allergies and Sensitivities
D6. Immunization Record
D7. Medication History
D8. Pathology and Radiology Results

8.3 Role-Based Access Control And E-consent

Classic access control is based on the individual (subject) accessing a resource (object): subjects → objects. Sometimes privileges are associated with posts other than individuals. Individuals get their privileges because their posts or positions in the organization. In other words, whoever gets the post would get the privileges of the post. When people leave the organization or change the positions, their privileges are revoked and/or changed. This happens in many organizations from the viewpoint of organization administration. For example, a doctor in a hospital can access the patients’ information in the hospital. When the doctor leaves the hospital, he/she usually lose the capability to access the patients’ information. If the number of subjects and objects is large, individual access control becomes difficult. Each individual needs to be assigned each access right when they get a position in the organization and revoked each access right when the person changes the post or leaves the organization. When privileges are indeed assigned to roles other than individual subjects, this can greatly simplify the access management.

In role-based access control, roles are placed between the user and the resource and subjects get their access rights indirectly by assigning access rights to roles and roles to subjects. Roles describe rights, duties and tasks that people have to perform. When people leave or change roles, only the mapping from subjects to roles need to be revoked or changed. On the other hand, if the duties of the roles
change, only the mapping from roles to objects need to be changed. Roles provide a more abstract viewpoint on access control.

\[ \text{subjects} \rightarrow \text{roles} \rightarrow \text{objects} \]

The concept of role also applies to the provision of patient data in health care contexts. Some consents may be given by patients in relation to roles ("yes, doctor, I consent to have a pathology test done on those samples"). Multiple individuals may perform particular roles at different times, e.g. because of the need for shift-work in both intensive-care and extensive-care.

Roles can be organised into hierarchies so that consents can be inherited, which could greatly reduce the amount of explicit consent specification. Roles can also be delegated. One entity may act on behalf of another. Attorney is an example of such a relationship. Other examples include guardians of minors and of people who are not psychologically capable of managing their own affairs.

8.4 A System Architecture for E-consent

In this section, we present a system architecture that enables various models for e-consent to be developed.

Dimensions of Consent

Consents may involve subjects to whom the consents are given, information (data) to be protected, access rights allowed or prohibited on the information, and subjects who issue the consent.

Subjects: roles, individuals

In the context of e-consent for health care, the consent may be assigned on the basis of an individual's identity or clinical role within an organization. Some consents may be given by patients in relation to identified individuals, e.g., "yes, I consent to send those details to Dr Smith"). In other circumstances, the consent is for a role, such as, "yes, I consent to have a pathology test done on those samples".

Roles can be organised into different hierarchies so that the consent can be in-
herited. The supervision role hierarchy, for example, is based on the organization hierarchy; the isa role hierarchy is based on generalization; and the activity hierarchy is based on aggregation. For example, Figure 8.1 is a possible role hierarchy based on aggregation.

![Role Hierarchy Diagram]

**Figure 8.1**: A Role Hierarchy.

**Data**

In general, the data needed for recording information about a patient are: individual identifiers, personal and contact details, clinic related details, and health details and summaries of a health care event, episode, condition, procedure, or a medication.

To allow consent inheritance along the data dimension, data could be organised into hierarchies. For example, the electronic patient record given in Section 8.2 can be organised into the hierarchy shown in Figure 8.2. A consent to read health details (D) means a consent to read all data associated with the episodes of care (D1,D2,D3 and D4), and all the data captured in various events of care.(D5, D6, D7 and D8).
Access Rights

Usual access rights such as read, write, and update apply to the patient data. Access rights can also be organised into hierarchies to allow inheritance along this dimension. A consent to updating for example, may also imply a consent to reading and writing.

Consent and Denial

Both consents and denials are needed in a flexible e-consent system. Denials are useful when patients want to express explicitly that some disclosure is forbidden. In some e-consent models where the presumption is consent, the explicit denial will override the default consent. For example, some European countries establish default consent in relation to organ donation that are overridden by an explicit denial. In other e-consent models where denials are established as default, the explicit denial could prevent further assigning of consent accidentally or intentionally.

In addition, allowing both consent and denial would enable patients to express more complex consent instructions. A consent applying to a general class can be added qualifications, in order to deny a particular subclass, say, a particular condition, episode or a group of people. Conversely, the patient may wish to deny some
broadly-defined class, but specifically consent to particular exceptions (e.g. deny in general for organ transplant, but permit use for a specific person or for any family-member).

Role Delegation

In other circumstances, a patient may wish to delegate the capability to grant consent to nominated representatives or medical practitioners, who may further wish to delegate the power to consent to other health professionals. This is usually done for flexibility, for cooperation, and for convenience of the carer. Note that the patient still keeps the capability to issue the consent after the delegation.

Consent Delegation

In role delegation, a patient delegates all of his/her privileges of consent to the nominated entity. However, in some situations, a patient may only wish to partially delegate his/her capability of creating consent to others, e.g., delegate the consent on the information regarding the treatment details but not the personal details.

Conflict Resolution

Because of role delegation and consent delegation, multiple grantors may exist for a specific consent and conflicts may arise. For example, a patient may wish to deny all information relating to HIV to his/her immediate family, but his/her family GP, to whom he/she has delegated the privilege of granting consent, may want to disclose the information to the patient’s immediate family. In this case, patient’s immediate family may receive two conflicting authorizations, consent and denial. A proper conflict resolution policy is then needed.

Control of Delegation

As we said before, to achieve high quality of treatment, a patient may wish to give the carer more flexibility by delegating them the required rights. On the other hand, to protect his/her privacy, a patient may not wish to lose control on
his/her health data. One solution to this problem is to give a patient higher priority than his/her delegate, so that once conflict occurs, the patient’s consent grant will override the other.

In addition, proper constraints on delegation are needed to avoid undesirable situations. For example, a doctor receiving the right to grant for a patient's health data should not be able to deny the patient to read his/her own health data by issuing the patient a negative authorization.

Consent Inheritance and Exceptions

As in many information systems, allowing consent inheritance would greatly reduce the amount of consents that need to be explicitly specified. When consents can be inherited, it is important to support exceptions to avoid undesirable inheritance. For example, a consent to the health care professional means a consent to every health carer. A consent to the health care professional followed by a denial to a particular GP means that the GP could not be exposed to the patient's information.

Consent Capability

Usually a patient has the capability to grant consent on his/her own health data. However, in some circumstances a person is physically or legally incapable of giving consent; for instance children or persons in coma, seriously incapacitated or frail people. These cases are usually subject to a variety of health-specific laws, e.g. laws relating to guardianship.

Purposes of a Consent

Sometimes, a consent is assigned on the basis of specific use of information. Common purposes include treatment, notification (requests by persons closely associated with the person concerned such as guardians, partners and immediate family), training, research, and getting advices from specialists.
Context

Sometimes, consent is assigned based on the current context. A doctor may not be allowed to read the patient's health data in a normal situation, but allowed to do so in an emergency situation.

Implicit Consents and Inference Rules

The inherited consents belong to implicit consents. In general, rule based consent specification allows implicit authorizations to be derived from the authorization set. Hence this can greatly reduce the size of explicit authorization set.

A system architecture for e-consent is shown in Figure 8.3.

![Figure 8.3: A System Architecture for E-consent.](image-url)
8.5 Basic Types of E-consent

[21] has presented four basic types of e-consent in health care context. Here we have extended it to the following six types with consent delegation:

(1) Consent/Delegation by Default

Consent to exposure or delegation of patient information is assumed for any person, any data and any access except when explicit denial exists.

(2) General Consent/Delegation

This is a general consent given by a patient for an individual or a particular party to access/delegate some or all of his/her health information without exception. For example, in a blanket consent, a patient may wish to give any health care professional working within a specified health context to access any or all of their health information. In this case, a health worker is allowed to look at the details of the past episodes of care in any future episode of care.

(3) General Consent/Delegation with Specific Denial

In this case, a patient provides a general consent/delegate to a broadly-defined class but with exceptions to some of its subclasses, such as denying consent/delegate to the disclosure of particular information; and/or to the disclosure to a particular party or category of parties. Thus the general consent is modified by specific conditions in which consent is to be withheld. For example, a patient provides a general consent to a health care professional, but expressly precludes the disclosure of information about a STD condition, gynaecological procedure; or disclosure to their family GP, or their immediate family.

(4) General Denial with Specific Consent/Delegation

In this case, a patient generally denies access to their health data but with exceptions, such as consent/delegate to the disclosure of particular information; and/or to the disclosure to a particular party or category of parties. That is, the general denial is modified by specific identified conditions in which denial
is to be withheld. For example, a patient provides a general denial but authorises their primary treating professional to provide a sample of body fluids, accompanied only by relevant data, to a diagnostic service, for the purpose of conduction of tests and reporting the results back to the GP.

(5) General Denial

In this case, a patient generally denies all or part of the information to be used by a particular party or anyone without exception. The extreme case is the blanket denial, where anyone is denied to access any information. Thus, a patient is requested for consent to access information each time there is a new request by a clinician. Contexts in which consumers would be likely to use this are treatments for STD, drug rehabilitation and psychiatric treatment.

(6) Denial by Default

Denial is assumed for any person, any data and any purpose except when explicitly expressed consent exists.

It should be noted that, from (1) to (6), the degree to support accessibility of health carer decreased but the degree of privacy increased. The above basic consent models could be combined to form more complex consent models where different hierarchies of qualifications are allowed. For example, a patient may consent to all information to all health carers; but within that can deny all information relating to HIV; but consents to information relating to HIV to STD clinics; and finally denies all information to a specific STD clinic (e.g. where say patient’s mother works)[21].

8.6 Using the Logic Model to Support E-consent

In this section, we show how to use the Delegatable Authorization Program (DAP) presented in Chapter 6 to support the authorizations in the e-consent model.

Recall that our language $\mathcal{L}$ is a many-sorted first order language, with four disjoint sorts $S, O, A, T$ for subject, object, access right and authorization type respectively. Three partial orders are defined on subject, object and access right
respectively to represent the inheritance hierarchies. Variables are denoted by a string preceded by an underscore. A Delegatable Authorization Program, DAP, consists of a finite set of rules which are statements of the form:

\[ b_0 \leftarrow b_1, ..., b_k, not b_{k+1}, ..., not b_m, m \geq 0 \]

where \( b_0, b_1, ..., b_m \) are literals, and \( not \) is the negation as failure symbol.

Our examples in this section are mainly based on the electronic patient record given in Section 8.2, its hierarchical structure in Figure 8.2 and the subject hierarchical structure in Figure 8.1.

8.6.1 Supporting E-consent System Architecture

Consent and Denial

We use the authorization predicate \textit{grant} to define consent and denial in the context of health care.

\textbf{Definition 8.1} A subject(individual/role) \( s \) consents (denies) another subject \( s' \) to exercise access right \( a \) on patient data \( o \), which is defined by \( \text{grant}(s', o, +, a, s) \)

\((\text{grant}(s', o, -, a, s))\).

For example, \( \text{grant}(Bob, health-data, +/-, read, Alice) \) means that Alice consents/denies for Bob to read her health-data.

Consent Delegation and Role Delegation

We first use the authorization predicate \textit{grant} to define consent delegation.

\textbf{Definition 8.2} A subject(individual/role) \( s \) delegates another subject \( s' \) the privilege to grant consent for access right \( a \) on patient data \( o \), which is defined by \( \text{grant}(s', o, *, a, s) \).

For example, \( \text{grant}(familyGP, health-data, *, read, Alice) \) means that Alice delegates her family GP the right to further disclose her health data for reading.

Note that, in our formulation, * means + plus the right to grant. This means that in the above example, the family GP can read the data as well as disclose the data.
Next, the role delegation from \(s\) to \(s'\) can be expressed by the following rule, which means that \(s\) will delegate any of his/her rights on any object to \(s'\).

\[
\text{grant}(s', o, *, a, s) \leftarrow \text{grant}(s, o, *, a, g)
\]

Capability to Give Consent

We define a guardian relation to denote the capability to give consent. We introduce a new predicate, \(\text{guardian}(s, s')\) with type \(S \times S\), which means that \(s\) is a guardian of \(s'\). Usually patients are their own guardians. For patients who are physically or legally incapable of giving consent, their guardians are different persons (subject to law). Let \(\text{own}(s, o)\), with type \(S \times O\), represent that \(s\) is the owner of the data \(o\). Patients are considered to be owners of their health data.

Next, let the system administrator \(\#\) delegate all the access rights on health data to the guardians of the patients.

\[
\text{grant}(s, o, *, a, \#) \leftarrow \text{own}(s', o), \text{guardian}(s, s')
\]

Hence, at the beginning, only guardians of patients can give consent to patients' data. However, through consent delegation or role delegation, other persons may receive capability to give consent.

Conflict Resolution

Let us have a look again at the conflict resolution policy proposed in Chapter 6 and see how it works here in this application. First we solve the conflicts in terms of the delegation relations and give higher priorities to predecessors. In the e-consent context, a patient certainly wishes to hold higher priorities than his/her delegates, so that his/her consent instructions will override the others in the case of conflicts. This is also true for other delegators. In fact, to achieve high quality of treatment, a patient may wish to give the carer as much flexibility as possible. On the other hand, to protect his/her privacy, a patient may not wish to lose control on his/her health data. Giving a patient or, in general, a delegator higher priority than his/her
delegate would best suit this situation. For example, a patient may wish to deny all information relating to HIV to his/her immediate family, but his/her family GP, to whom he/she has delegated the privilege of granting consent, may want to disclose the information to his/her immediate family. In this case, the patient’s authorization will override his/her family GP’s and therefore his/her immediate family would not be able to access his/her HIV related information.

When two conflicting authorizations have the same grantors, the hierarchies of subjects, objects, and access rights are considered and the more-specific take precedence principle will be used. This can support the exceptions in consent inheritance. For example, a patient provides a general consent to a health care professional on the health data, but specifically precludes the disclosure of information about a STD condition or a gynaecological procedure or disclosure to their family GP. By using the more specific-take-precedence policy, the general consent will be overridden by the more specific denials.

If all the above policies don’t apply, a patient can simply select denial-take-precedence method, which favours privacy; or consent-take-precedence method, which favours convenience.

**Delegation Control**

Cyclic authorization is prohibited in DAP, as they usually do not make sense in this situation and may even cause undesirable situation. Consider for example a patient delegating the “read” right on his/her health data to his/her family GP. It is thus meaningless that the GP grants back to the patient to read his/her data. Moreover, it is undesirable if the GP could deny the patient to read his/her own data. On the other hand, as we mentioned before, giving predecessors higher priorities than successors provides users further control on consent delegation.

**Purposes and Contexts**

We extend the language further by adding another sort *other* to the language, which contains constants and variables other than subjects, objects, access rights and authorization types. For example, we can put different purposes and contexts
in this sort.

We further introduce two predicates, \textit{purpose} and \textit{context}, both with one argument in sort \textit{other}. The following rule means that a patient Alice consents to all health professionals to exercise any access on her health data if their purpose is for treatment.

\[
\text{grant(all-health-carer, all-data, *, update, Alice)} \leftarrow \text{purpose(treatment)}
\]

The following rule states that a patient Bob gives all health professionals consent to access his health data if it is in an \textit{emergency} situation.

\[
\text{grant(all-health-carer, all-data, *, update, Alice)} \leftarrow \text{context(emergency)}
\]

### 8.6.2 Supporting Basic Types of E-consent

We now show how different types of consent can be supported by DAP.

1. Consent/Delegation by Default

The following rule expresses that a patient Alice wants to use "consent by default model" for her health data. That is, a subject can access her data if the explicit denial does not exist.

\[
\text{grant(_s, all-data, +, _a, Alice)} \leftarrow \neg \text{grant(_s, all-data, -, _a, Alice)}
\]

Intuitively, the rule says, \text{grant(_s, all-data, +, _a, Alice)} is true if no information indicates that \text{grant(_s, all-data, -, _a, Alice)} is true.

Similarly, the following rule expresses that a patient Bob wishes to use "delegation by default model" for his health data. That is, a subject can disclose Bob’s data to others if the explicit opposite does not exist.
\[ grant(s, all\text{-}data, *, a, Bob) \leftarrow not\ grant(s, all\text{-}data, -, a, Bob) \]
\[ not\ grant(s, all\text{-}data, +, a, Bob) \]

Intuitively, the rule says, \( grant(s, all\text{-}data, *, a, Bob) \) is true if no information indicates that \( grant(s, all\text{-}data, -, a, Bob) \) or \( grant(s, all\text{-}data, +, a, Bob) \) is true. Note that \( grant(s, all\text{-}data, +, a, Bob) \) means a subject can exercise access \( a \) on all-data, but cannot further grant access \( a \) to another subject.

2. General Consent/Delegation

Suppose a patient Alice wants to give a general consent (delegation) to all the health carers for all the access rights on her health data. This can be represented by the following rule.

\[ grant(all\text{-}health\text{-}carer, all\text{-}data, +(*), a, Alice) \leftarrow \]

Another patient Bob may only wish to disclose his health details (D) to all health carers for reading. The following rule can express this.

\[ grant(all\text{-}health\text{-}carer, D, +, read, Bob) \leftarrow \]

3. General Consent/Delegation with Specific Denial

Suppose a patient Alice provides a general delegation to a health care professional, but expressly precludes the disclosure of information about sensitive medical history (D3), such as a STD condition or gynaecological procedure, to her family GP. The following rules express this situation.

\[ grant(all\text{-}health\text{-}carer, all\text{-}data, *, a, Alice) \leftarrow \]
\[ grant(familyGP, D3, -, a, Alice) \leftarrow \]

Let us have a closer look at how this works. The first rule is a general rule which will propagate downward the subject and object hierarchies defined by Figure 8.1 and Figure 8.2, since consent inheritance is supported. Hence the family GP will
be able to access D3. However the second rule states the opposite. Since Alice is
the grantor of both rules, and the second rule is more specific than the first one,
the second one will override the first one on D3 according to our conflict resolution
policy.

On the other hand, since Alice has delegated the grant right to all-health-carer,
the health professionals can grant consent on her health data thereafter. Suppose
a doctor Bob has given her familyGP a consent on D3, as shown in the following rule:

\[ \text{grant(familyGP, D3, +, a, Bob)} \leftarrow \]

This rule is conflicting with the second rule granted by Alice. In this situation,
as Bob was given his privilege to grant consent by Alice, his positive grant to the
familyGP will be overridden by the negative grant from Alice, according to our con-
flict resolution policy. In fact, except for the administrator, Alice’s grant on her own
data will not be overridden by any other person’s grant.

4. General Denial with Specific Consent/Delegation

For example, a patient Alice provides a general denial to a health care profes-
sional, but consents to the disclosure to her family GP. This could be expressed by
the following two rules.

\[ \text{grant(all-health-carer, all-data, -, a, Alice)} \leftarrow \]
\[ \text{grant(familyGP, all-data, +, a, Alice)} \leftarrow \]

The second “more-specific” rule will override the first one on familyGP. The rea-
son is similar to the one given in 3.

5. General Denial

The following rule expresses that a patient Alice gives a general denial to all the
health carers for all the access rights on her data.
grant(all-health-carer, all-data, -, a, Alice) ←

The following rule expresses that a patient Alice give a general denial to a doctor Bob for all the access rights on her data.

grant(Bob, all-data, -, a, Alice) ←

6. Denial by Default

The following rule expresses a patient Alice wants to use "denial by default" model for her health data. That is, a subject cannot access her data if the explicit consent or delegation does not exist.

grant(s, all-data, -, a, Alice) ← not grant(s, all-data, +, a, Alice),
not grant(s, all-data, *, a, Alice)

For more complex model where qualifications are nested, see the following example [21]. A patient Alice consents to allow access to all information to all health carers; but within that denies all information relating to HIV (D3) and consents to information relating to HIV to STD clinics; and finally denies all information to a specific STD clinic-1 (where her mother works). The following rules plus our conflict resolution policy can achieve these requirements.

grant(all-health-carer, all-data, +, a, Alice) ←
grant(all-health-carer, D3, -, a, Alice) ←
grant(STD - clinics, D3, +, a, Alice) ←
grant(STD - clinic - 1, D3, +, a, Alice) ←

As Alice is the grantor of all the rules, the more specific-take-precedence principle will be used here; hence the inherited consent instruction from the general rules will be overridden by the more specific consent rules. On the other hand, since Alice is the owner, her instructions cannot be overridden by any other person. The above
rules thus meet Alice’s requirements.

8.7 Summary

In this chapter, we apply our logic authorization model in Chapter 6 to a real world application, e-consent on health data. We chose to use the Delegatable Authorization Program (DAP) as it can support different types of e-consent in a straightforward manner. A flexible architecture for e-consent is presented. Then it is shown that the architecture and different types of e-consent can be well supported by delegatable authorization model.
Chapter 9

Conclusion

Authorization delegation and negation are two important features to achieve a flexible access control model. While there have been several efforts on authorization delegation and negation respectively, very little work exists to support both of them. Furthermore, there has been very little work exploring conflict resolution policy in authorization delegations. In this thesis, we have proposed several models which can support authorization delegations and both positive and negative authorizations. We have also investigated solutions to the conflict resolution problem within this context. We have proposed a new conflict resolution policy that enables to achieve a well controlled delegation. This allows us to trace the delegation path explicitly and give higher priorities to the predecessors whenever the grantors of conflicting authorizations fall in a path. Assuming all the rights on an object are first delegated from the owner of this object, the owner will always have the highest priority for this object. Therefore the delegators do not lose control of the objects in the sense that their authorizations can override the delegates' whenever conflicts occur between them.

We have developed a graph based framework to support authorization delegations and both positive and negative authorizations. A major feature of our approach is that we classified conflicts into comparable and incomparable ones and this classification is useful not only in the control of access right delegation but also for supporting multiple policies to resolve conflicts. Our framework also provides a flexible way to preserve conflicting authorizations so that it is possible to re-activate some early overridden access rights if proper authorizations are revoked or the policy of conflict resolution is changed. The detailed algorithms to implement the model were provided and the application of the model in a distributed environment was
also presented. Our framework is represented using labelled digraphs, which provide a formal basis for proving the semantic correctness of our model.

We have further extended the graph model and introduced a weight based scheme that allows users to further express degrees of certainties about their authorization and delegation assignments. The weight is a non-negative number, and a smaller number represents a higher certainty. A delegator can give a delegate the same priority as itself by assigning 0 to the weight of the delegation. This provides users a more flexible way to control their authorization delegations. A promising conflict resolution policy has been proposed which takes into account not only the priorities of grantors but also the certainties of the authorizations. The framework further enhances the existing work on delegation with negation.

Then we proposed a logic model that is based on extended logic programs. Delegable authorizations, derived authorizations and authorization inheritance are all supported. We have also presented resolutions to both explicit and implicit conflicts, which can support well controlled delegation and exceptions in authorization inheritance. Since the administration of access rights can be delegated, our model is suitable for large-scale decentralised systems where distributed administration of access rights are needed. The expressive power and nonmonotonic reasoning of extended logic programs provide users a feasible way to express complex security policy.

The proposed temporal authorization model allows users to express time constraints on their grants for authorizations and delegations. The model is also based on extended logic programs. We have demonstrated that our model is able to support temporal delegation, temporal authorization inheritance and temporal conflict resolution. In particular, the conflict resolution policy, which is based on the underlying temporal delegation relations and various temporal temporal relations, can support well controlled temporal delegation, temporal suspension of authorizations and automatic authorization update. Several important semantic properties are also investigated.

Finally we have demonstrated the application of our proposed authorization models by using our model to specify the authorization requirements in a health
care application. A system architecture for e-consent was presented and different types of e-consent models were discussed. We have shown that our model provides users a flexible framework to support the different types of e-consent for electronic patient record system.

We conclude this thesis by identifying some possible further avenues that can be pursued in the future. On the policy side, we believe that it will be useful to develop a dynamic prioritised conflict resolution method that can support different policies in terms of various properties of subjects, objects and access rights. On the model side, it may be useful to explore extensions of our graph model by supporting structured subjects, objects and access rights as well as implicit authorizations. On the logic side, it could be worthwhile to explore the development of disjunctive logic program based authorization models which would allow disjunctive information to be used in authorizations and delegations. Finally, another direction to pursue would be to consider the application of the proposed models in data warehouse system implementations.
Bibliography


Models for Authorization and Conflict Resolution

Chun Ruan

A thesis submitted for the Degree of
Doctor of Philosophy
University of Western Sydney

March 2003
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 solely my own original work. I certify that no part of this thesis has been submitted as a part of any other degree.

Chun Ruan
25 March 2003
To my parents.
Acknowledgements

I would like to express my sincere gratitude and appreciation to my PhD principle supervisor Professor Vijay Varadharajan for all his valuable guidance, encouragement and support in every aspect during the realisation of this research.

I sincerely thank my PhD supervisor Associate Professor Yan Zhang for his valuable guidance and time.

I am particularly thankful to my family for their unlimited support and patience.

My appreciation also goes to many others who assisted me during my academic pursuit.
Abstract

Access control is a significant issue in any secure computer system. Authorization models provide a formalism and framework for specifying and evaluating access control policies that determine how access is granted and delegated among particular users. The aim of this dissertation is to investigate flexible decentralized authorization model supporting authorization delegation, both positive and negative authorization, and conflict resolution.

A graph based authorization framework is proposed which can support authorization delegations and both positive and negative authorizations. In particular, it is shown that the existing conflict resolution methods are limited when applied to decentralized authorization models and cyclic authorizations can even lead to undesirable situations. A new conflict resolution policy is then proposed, which can support well controlled delegation by giving predecessors higher priorities along the delegation path. We give a formal description of our model and describe in detail the algorithms to implement the model. Our model is represented using labelled digraphs, which provides a formal basis for proving the semantic correctness of our model.

A weighted graph based model is then presented which allows grantors to further express degrees of certainties about their granting of authorizations. The authorization state is represented as a weighted graph, with weights denoting different degrees of certainties. A more flexible conflict resolution policy based on weighted lengths of paths is proposed, in which both the priorities of grantors and the weights of authorizations are taken into account. This provides users a flexible way to control their delegations of access rights. Detailed algorithm and correctness proof are given.

The work is further extended to consider more complex domains where subjects, objects and access rights are hierarchically structured and authorization inheritance along the hierarchies are taken into account. To take advantage of strong expressive and reasoning power of logic programming, we propose a logic program based
formulation that supports delegatable authorizations, both negation as failure and classical negation. Inheritance of rules is supported which help to simplify the security management. A conflict resolution policy has been developed which can solve both explicit and implicit conflicts and support the controlled delegation and exception. A precise semantics is given which is based on stable model semantics. In addition, several important properties of delegatable authorization programs are investigated. The framework provides users a reasonable method to express complex security policy.

In the real world, there are many situations in which users may need to be granted or delegated some authorizations for limited periods of time. To address the time dimension, we propose a temporal decentralized authorization model in which temporal authorization delegations and negations are allowable. A temporal conflict resolution method based on the underlying temporal delegation relation and various temporal relations is presented, which can support temporal controlled delegation, temporal suspension and the automatic authorization update. For instance, strategies include realization of temporal suspension by giving higher priority to the authorization with smaller interval, whereas automatic authorization update can be achieved by giving higher priority to the authorization with newer interval. Proper semantic properties are further investigated.

Finally, as an application, we show how our authorization model can be used in a e-consent system on health data. A system architecture for e-consent is presented and different types of e-consent models are discussed. We show that our model provides users a good framework for representing and evaluating these models.
## Contents

Acknowledgements iv

Abstract v

1 Introduction 1
1.1 Access Control Models and Basic Issues 1
1.2 Administration of Authorizations 3
1.3 Conflict Resolution 5
1.4 Problem Statement 7
1.5 Summary of Contribution 9
1.6 Outline of Dissertation 14

2 Overview of Access Control Models 16
2.1 Discretionary Access Control 16
  2.1.1 Access Matrix Model 16
  2.1.2 Extensions to the Access Matrix Model 17
2.2 Mandatory Access Control 19
2.3 Role-Based Access Control 20
2.4 Logic Based Access Control 21

3 Foundations 23
3.1 Directed Graph and Weighted Directed Graph 23
3.2 Logic Program 25
  3.2.1 Extended Logic Program 25
  3.2.2 Normal and Stratified Logic Program 27
3.3 Time Points and Intervals 28
4 A Graph Based Authorization Model

4.1 Related Works ........................................... 31
4.2 The Authorization Delegation and Conflict Resolution Model .... 33
  4.2.1 Basic Ideas ........................................ 33
  4.2.2 Notation And Definitions .......................... 34
  4.2.3 Formal Description of the Model ................... 39
  4.2.4 Access Control Policy ................................ 46
4.3 Algorithms .............................................. 47
  4.3.1 Algorithm for Evaluating the Effective Graph .......... 48
  4.3.2 Algorithm for Evaluating the Stable Graph .......... 50
4.4 Authorization State Transformation .......................... 52
4.5 Apply Our Model to a Distributed Environment ................. 55
  4.5.1 Global Access Control Model ....................... 56
  4.5.2 Completely Local Access Control Model .......... 60
  4.5.3 Mixed Access Control Model ....................... 60
4.6 Summary ............................................... 62

5 A Weighted Graph Based Authorization Model .................... 63

5.1 Motivation .............................................. 63
5.2 Basic Idea ............................................. 66
5.3 Formal Description of the Model .......................... 67
5.4 Access Control Policy ................................... 72
5.5 Algorithm ................................................ 73
5.6 Authorization State Transformation ........................ 78
5.7 Several Remarks of the Model .............................. 79

6 A Logic Based Authorization Model ............................. 81

6.1 Motivation .............................................. 81
6.2 Syntax of DAP .......................................... 83
6.3 Semantics of DAP ....................................... 85
  6.3.1 Considerations of the Semantics .................... 85
  6.3.2 Formal Definition of the Semantics ................. 88
Contents

6.3.3 Access Control Policy ........................................... 99
6.4 Properties of DAP .................................................. 100
6.5 Related Work and Summary ....................................... 102

7 A Temporal Authorization Model 104
7.1 Motivation ............................................................ 104
7.2 Syntax of TDAP ....................................................... 106
7.3 Basic Ideas ........................................................... 108
  7.3.1 Temporal Delegation Correctness .......................... 108
  7.3.2 Temporal Authorization Propagations ..................... 109
  7.3.3 Temporal Conflict Resolution ............................... 110
7.4 Formal semantics of TDAP ......................................... 113
  7.4.1 Temporal Rules for Delegation ............................. 113
  7.4.2 Temporal Rules for Propagation ............................ 115
  7.4.3 Temporal Rules for Conflict Resolution .................. 116
  7.4.4 Formal Semantics .............................................. 118
7.5 Properties of TDAP .................................................. 120
7.6 Summary .............................................................. 122

8 An Application in Health Care 124
8.1 E-consent on Health Data .......................................... 124
8.2 Electronic Patient Records ........................................ 126
8.3 Role-Based Access Control And E-consent ...................... 129
8.4 A System Architecture for E-consent ............................ 130
8.5 Basic Types of E-consent .......................................... 136
8.6 Using the Logic Model to Support E-consent ................... 137
  8.6.1 Supporting E-consent System Architecture ................ 138
  8.6.2 Supporting Basic Types of E-consent ...................... 141
8.7 Summary .............................................................. 145

9 Conclusion ............................................................. 146

Bibliography ............................................................. 149
List of Figures

1.1 Centralised Administration ........................................ 4
1.2 Decentralised Administration ..................................... 4
1.3 An Example of Authorizations in a University .................... 8

2.1 An Access Matrix .................................................. 18
2.2 Access Control List for Files in Figure 2.1 ....................... 19
2.3 Capability Lists for Files in Figure 2.1 .......................... 20
2.4 Authorization Relation for Files in Figure 2.1 ................. 21

3.1 A Digraph. .......................................................... 24
3.2 A Rooted Acyclic Digraph ......................................... 24
3.3 A Weighted Digraph ................................................. 25
3.4 Dependency Digraph of $P_1$ ....................................... 28
3.5 Dependency Digraph of $P_2$ ....................................... 29

4.1 An Example of Graph Denotation and Operation in Take-Grant Model 32
4.2 Different Arc Types for Different Grant Types .................... 36
4.3 $G_o$: an example of graph representation of authorizations on an object
   $o$, where $S_1,\ldots,S_9$ are subjects, R and W are access rights. 37
4.4 $G_{o,R}$ of Figure 4.3 ............................................. 37
4.5 $G_{o,W}$ of Figure 4.3 ............................................. 38
4.6 Contradictory grants on object $F_1$. ........................... 39
4.7 Examples of Inconsistent Authorization State. .................... 41
4.8 Three Kinds of Conflicting Grants. ............................... 42
4.9 Active Graph of Figure 4.4. ....................................... 43
4.10 Effective Graph of Figure 4.4. ................................... 43
4.11 One Stable Graph of Figure 4.4. ................................ 45
4.12 Another Stable Graph of Figure 4.4. ............................. 45
LIST OF FIGURES

4.13 $G_{o,R}$ after update. ........................................... 54
4.14 Effective and Stable Graph of Figure 4.13. ....................... 55
4.15 Two Consistent Local States for $R$ on $O$ ......................... 56
4.16 An Inconsistent Global State for $R$ on $O$ ....................... 56
4.17 Two Inconsistent Local States for $R$ on $O$ ....................... 57
4.18 A Consistent Global State for $R$ on $O$ ......................... 58

5.1 An Incomparable Conflict ....................................... 64
5.2 A Comparable Conflict ........................................... 64
5.3 HOS1 and HOS2 Have the Same Priority ........................ 65
5.4 HOS1 Has Higher Priority Than HOS2 ............................ 65
5.5 Weighted Authorizations For Figure 5.1 .......................... 66
5.6 Weighted Authorizations for Figure 5.2 .......................... 67
5.7 Weighted Authorizations For Figure 5.4 .......................... 67
5.8 An Example $G_{o,r}^{w}$ .......................................... 68
5.9 $Eff(G_{o,r}^{w})$ of Figure 5.8. .................................... 71
5.10 One Stable State of $G_{o,r}^{w}$ in Figure 5.9 ......................... 72
5.11 Another Stable State of $G_{o,r}^{w}$ in Figure 5.9 ..................... 73

8.1 A Role Hierarchy .................................................. 131
8.2 An Object Hierarchy ............................................. 132
8.3 A System Architecture for E-consent. ............................ 135