

# A Logic Program Solution for Conflict Authorizations

YUN BAI

School of Computing and Information Technology  
University of Western Sydney  
Penrith South DC, NSW 1797  
AUSTRALIA

*Abstract:* As a security mechanism, authorization is to provide access control to the system resources according to the policies and rules specified by the security strategies. Either by update or in the initial specification, conflicts in authorization is an issue needs to be solved. In this paper, we propose a new approach to solve conflict by using prioritized logic programs and discuss the uniqueness of its answer set. Addressing conflict resolution from logic programming viewpoint and the uniqueness analysis of the answer set provide a novel, efficient approach for authorization conflict resolution.

*Key-Words:* security, access control, formal authorization specification, conflict resolution, prioritized logic program.

## 1 Introduction

Today's computer systems are increasingly attacked by malicious attempts. Protecting systems from such attacks is becoming an essential issue in computer security. Access control or Authorization provide the ability to control access to computer system, and to limit what entities can do what kind of operations on the information and the resources of the system. To ensure the security of the system, authorization rules need to be specified in such a way that they not only have a powerful expressiveness to accommodate user and system requirements, but also they need to be flexible enough to capture the changing needs of the environment. Logic based specification approaches provide an appropriate level of such requirements. Therefore, more and more researchers are focusing on the issue of logic based authorization specification. Jajodia *et al* [7] proposed a logic language for expressing authorizations. They used predicates and rules to specify the authorizations; their work mainly emphasizes the representation and evaluation of authorizations. The work of Bertino *et al* [2] describes an authorization mechanism based on a logic formalism. It mainly investigates the access control rules and their derivations. In their recent work [3], a formal approach based on C-Datalog language is presented for reasoning about access control models. Li *et al* [8] developed a logical language called *delegation logic* to represent authorization policies, credentials in large-scale, distributed systems. The work emphasizes the delegation depth and a variety of complex delegation

principals. Chomicki *et al* [4] discussed security policy management using logic program approach. Woo and Lam proposed a formal approach using default logic to represent and evaluate authorizations [9].

This paper is to address high level authorization specification and conflict resolution by using prioritized logic programs. We first propose a logic language by using logic programs to specify authorization rules, and then solve its conflict by using the concept and techniques of prioritized logic programs.

The paper is organized as follows. Section 2 describes authorization rules, its specification and evaluation. Section 3 investigates authorization conflict issue and proposes a new approach to solve it. We introduce prioritized logic programs for efficient conflict resolution. We discuss the unique answer set of an authorization domain in section 4 and finally section 5 concludes the paper.

## 2 Authorization specification and evaluation

We define that all the authorizations rules forms an *authorization domain*. The individual rule is specified by a language  $\mathcal{L}$ . Language  $\mathcal{L}$  includes the following six disjoint sorts for *subject*, *group-subject*, *access-right*, *group-access-right*, *object*, *group-object* together with predicate symbols *holds*,  $\in$ ,  $\subseteq$  and logic connectives.

In language  $\mathcal{L}$ , the fact that a subject  $S$  has access right  $R$  for object  $O$  is represented using a ground

atom  $holds(S, A, O)$ . The fact that a subject  $S$  is a member of  $G$  is represented by  $S \in G$ . Similarly, we represent inclusion relationships between subject groups such as  $G_1 \subseteq G_2$  or between object groups such as  $GO_1 \subseteq GO_2$ . In general, we define a *literal* which represents a *fact*  $F$  to be an atomic formula of  $\mathcal{L}$  or its negation, while a *ground fact* is a fact without variable occurrence. We view  $\neg\neg F$  as  $F$ . A *rule* is an expression of the form:

$$F_0 \leftarrow F_1, \dots, F_m, not F_{m+1}, \dots, not F_n, \quad (1)$$

where each  $F_i$  ( $0 \leq i \leq n$ ) is a literal.  $F_0$  is called the *head* of the rule, while  $F_1, \dots, F_m, not F_{m+1}, \dots, not F_n$  are called the *body* of the rule. Obviously, the body of a rule could be empty. In this case, it represents an authorization fact. A rule is *ground* if no variable occurs in it.

An *extended logic program* is a collection of such rules. In a rule, the set  $\{F_1, \dots, F_m\}$  is the literals without weak negation; the set  $\{not F_{m+1}, \dots, not F_n\}$  is the literals with weak negation.

All the rules required to specify the access control of a system or an organization form an *authorization domain*. It is formally defined as:

**Definition 1** An *authorization domain* is a finite set  $D = \{R_i\}$ , ( $i=1,2, \dots,k$ ) where  $R_i$  is a rule of the form  $F_0 \leftarrow$  or  $F_0 \leftarrow F_1, \dots, F_m, not F_{m+1}, \dots, not F_n$  where  $m > 0, n > m$ .

The following is an example of an authorization domain.

**Example 1**  $D = \{R_1, R_2, R_3\}$ , where  
 $R_1: holds(S, R, O) \leftarrow$   
 $R_2: holds(S_1, W, O) \leftarrow \neg holds(S_2, W, O)$   
 $R_3: holds(S_3, R, O) \leftarrow holds(S_3, R, O_1), O \in O_1,$   
 $not \neg holds(S_3, R, O)$

*This domain represents the current authorization information about the system: subject  $S$  has read right on object  $O$ ; if subject  $S_2$  does not have write right on object  $O$ , then  $S_1$  can write on  $O$ ; if  $S_3$  can read  $O_1$ ,  $O$  is a member of  $O_1$  and there is no information stating that  $S_3$  cannot read  $O$ , then  $S_3$  has read right on  $O$ .*

Evaluation is to assess the access requests to the system. Once an authorization domain is properly specified and applied to a system, its task is to make the decision as which access request is to be granted and which is to be denied according to the specification of the authorization domain. That is, given an authorization domain and an access request, how to decide either to grant or deny such an access request?

Authorization evaluation is to answer this question. It first calculates all the authorization facts from the authorization domain. Then when an access query is proposed, it can make proper decision by checking the calculated set of authorization facts.

Some facts are explicitly represented, such as  $holds(S, R, O)$  in example 1. This is the only authorization fact we can get from this domain.

The set of rules of the authorization domain form an extended logic program  $\Pi$ . Here we use the answer set semantics proposed by Gelfond and Lifschitz to evaluate the extended logic program.

To simplify the procedure, we treat a rule  $r$  in  $\Pi$  with variables as the set of all ground instances of  $r$  formed from the set of ground literals of the language of  $\Pi$ . Let  $\Pi$  be an extended logic program not containing *not* and  $Gl$  the set of all ground literals in the language of  $\Pi$ . The *answer set* of  $\Pi$ , denoted as  $Ans(\Pi)$ , is the smallest subset  $S$  of  $Gl$  such that (i) for any rule  $F_0 \leftarrow F_1, \dots, F_m$  from  $\Pi$ , if  $F_1, \dots, F_m \in S$ , then  $F_0 \in S$ ; and (ii) if  $S$  contains a pair of complementary literals, then  $S = Gl$ . Now let  $\Pi$  be an arbitrary extended logic program. For any subset  $S$  of  $Gl$ , let  $\Pi^S$  be the logic program obtained from  $\Pi$  by deleting (i) each rule that has a formula *not*  $F$  in its body with  $F \in S$  (since *not*  $F$  is not true, this rule will no longer take effect), and (ii) all formulas of the form *not*  $F$  in the bodies of the remaining rules (for similar reason, such formulas will no longer take effect). We define that  $S$  is an *answer set* of  $\Pi$ , denoted by  $Ans(\Pi)$ , iff  $S$  is an answer set of  $\Pi^S$ , i.e.  $S = Ans(\Pi^S)$ . An extended logic program may have one, more than one, or no answer set at all.

In the real world, we expect that the authorization domain has a unique answer set. If an authorization domain has no answer set, it indicates that the domain is inconsistent. If more than one answer sets are presented, there may exist conflict in the domain. We need to sort it out and find the answer set we prefer. In the following section, we will investigate conflict resolution to get the preferred answer set.

## 3 Conflict resolution

### 3.1 Prioritized logic program

Now, let's consider the following authorization domain.

**Example 2**  $D = \{R_1, R_2, R_3, R_4\}$ , where  
 $R_1: holds(S_1, R, O_1) \leftarrow$   
 $R_2: \neg holds(S_1, R, O) \leftarrow$   
 $R_3: O \in O_1 \leftarrow$   
 $R_4: holds(S_1, R, O) \leftarrow holds(S_1, R, O_1), O \in O_1,$

$\neg \text{holds}(S_1, R, O)$

This domain states that currently  $S_1$  can read  $O_1$ ;  $S_1$  cannot read  $O$ ;  $O$  is a member of  $O_1$ ; if  $S_1$  can read  $O_1$  and  $O$  is a member of  $O_1$  and it is not specified that  $S_1$  can not read  $O$ , then  $S_1$  has the right to read  $O$ .

Obviously, rules  $R_2$  and  $R_4$  conflict with each other as their heads are complementary literals, and applying  $R_2$  will defeat  $R_4$  and vice versa. However, we can assign preference ordering among the conflict rules. If we define  $R_2 < R_4$ , we expect that rule  $R_4$  is preferred to apply first and then defeat rule  $R_2$  after applying  $R_4$  so that the solution  $\text{holds}(S_1, R, O)$  can be derived. On the other hand, if we define  $R_4 < R_2$ , we expect that rule  $R_2$  is preferred to apply first and then defeat rule  $R_4$  after applying  $R_2$  so that the solution  $\neg \text{holds}(S_1, R, O)$  can be derived.

The above example shows an inconsistent authorization domain. In some other situation, the authorization domain is consistent initially, but after certain update, it becomes inconsistent.

For instance, we initially have the following authorization domain:

$$\begin{aligned} R_1: & \text{holds}(S, A, O) \leftarrow \\ R_2: & \text{holds}(S_1, A, O) \leftarrow \\ R_3: & \text{holds}(S_2, A, O) \leftarrow \text{holds}(S_1, A, O) \end{aligned}$$

It says that currently both  $S$  and  $S_1$  can access  $O$ ; if  $S_1$  can access  $O$  then  $S_2$  can also access  $O$ . The answer set for this domain is:

$$\{\text{holds}(S, A, O), \text{holds}(S_1, A, O), \text{holds}(S_2, A, O)\}$$

Now, the new knowledge  $R_4: \neg \text{holds}(S_1, A, O)$  is added to the domain. It conflicts with the existing  $R_2: \text{holds}(S_1, A, O)$ . We need to define a preference order to solve this conflicts. Suppose we prefer the update, that is we set the newly added  $R_4$  higher preference than the existing conflicting  $R_2$ . After the update, the new authorization domain has the following answer set:

$$\{\text{holds}(S, A, O), \neg \text{holds}(S_1, A, O)\}$$

We call the logic program with partial ordering  $<$  on the rules *prioritized logic program*  $\mathcal{P}$  [10].  $\mathcal{P}$  is defined to be a triplet  $(\Pi, \mathcal{R}, <)$ , where  $\Pi$  is an extended logic program,  $\mathcal{R}$  is a naming function mapping each rule in  $\Pi$  to a name, and  $<$  is a strict partial ordering on names. The partial ordering  $<$  in  $\mathcal{P}$  plays an essential role in the evaluation of  $\mathcal{P}$ . We also use  $\mathcal{P}(<)$  to denote the set of  $<$ -relations of  $\mathcal{P}$ . Intuitively  $<$  represents a preference of applying rules during the evaluation of the program. In particular, if  $\mathcal{R}(r) < \mathcal{R}(r')$

holds in  $\mathcal{P}$ , rule  $r'$  would be preferred to apply over rule  $r$  during the evaluation of  $\mathcal{P}$ .

### 3.2 Evaluation of prioritized logic program

The evaluation of a PLP will be based on its ground form. It is to find the answer set of the authorization domain. Given a PLP  $\mathcal{P} = (\Pi, \mathcal{R}, <)$ . We say  $\mathcal{P}$  is *well formed* if there does not exist a rule  $r'$  that is an instance of two different rules  $r_1$  and  $r_2$  in  $\Pi$  and  $\mathcal{R}(r_1) < \mathcal{R}(r_2) \in \mathcal{P}(<)$ . In the rest of this paper, we will only consider well formed PLPs in our discussions, and consequently, the evaluation for an arbitrary program  $\mathcal{P} = (\Pi, \mathcal{R}, <)$  will be based on its ground instantiation  $\mathcal{P}' = (\Pi', \mathcal{R}', <')$ . Therefore, in our context a ground prioritized (or extended) logic program may contain infinite number of rules. In this case, we will assume that this ground program is the ground instantiation of some program that only contains finite number of rules.

**Definition 2** Let  $\Pi$  be a ground extended logic program and  $r$  a rule with the form  $R_0 \leftarrow R_1, \dots, R_m, \text{not } R_{m+1}, \dots, \text{not } R_n$  ( $r$  does not necessarily belong to  $\Pi$ ). Rule  $r$  is defeated by  $\Pi$  iff  $\Pi$  has an answer set and for any answer set  $\text{Ans}(\Pi)$  of  $\Pi$ , there exists some  $R_i \in \text{Ans}(\Pi)$ , where  $m + 1 \leq i \leq n$ .

Let us consider program example 2 once again. If we choose  $R_2 < R_4$  and  $R_2$  is defeated by  $\mathcal{D} - \{R_2\}$ , rule  $R_2$  should be ignored during the evaluation of  $\mathcal{D}$ . We will get the unique answer set  $\{\text{holds}(S, R, O_1), O \in O_1, \text{holds}(S_1, R, O)\}$ .

To calculate the set of access facts of an authorization domain, we need to evaluate its corresponding extended logic program. That is, to find the answer set of prioritized logic program  $\mathcal{P}$ . Now, we present the procedure for finding the answer set. We start from a reduced set or the reduct of  $\mathcal{P}$ .

**Definition 3** Let  $\mathcal{P} = (\Pi, \mathcal{N}, <)$  be a prioritized extended logic program.  $\mathcal{P}^<$  is a reduct of  $\mathcal{P}$  with respect to  $<$  if and only if there exists a sequence of sets  $\Pi_i$  ( $i = 0, 1, \dots$ ) such that:

1.  $\Pi_0 = \Pi$ ;
2.  $\Pi_i = \Pi_{i-1} - \{r_1, r_2, \dots \mid (a) \text{ there exists } r \in \Pi_{i-1} \text{ such that for every } j (j = 1, 2, \dots), \mathcal{N}(r) < \mathcal{N}(r_j) \in \mathcal{P}(<) \text{ and } r_1, \dots, \text{ are defeated by } \Pi_{i-1} - \{r_1, r_2, \dots\}, \text{ and (b) there does not exist a rule } r' \in \Pi_{i-1} \text{ such that } \mathcal{N}(r_j) < \mathcal{N}(r') \text{ for some } j (j = 1, 2, \dots) \text{ and } r' \text{ is defeated by } \Pi_{i-1} - \{r'\}\}$ ;

$$3. \mathcal{P}^< = \bigcap_{i=0}^{\infty} \Pi_i.$$

In Definition 3,  $\mathcal{P}^<$  is a ground extended logic program obtained from  $\Pi$  by eliminating some *less preferred rules* from  $\Pi$ . In particular, if  $\mathcal{R}(r) < \mathcal{R}(r_1)$ ,  $\mathcal{R}(r) < \mathcal{R}(r_2)$ ,  $\dots$ , and  $\Pi_{i-1} - \{r_1, r_2, \dots\}$  defeats  $\{r_1, r_2, \dots\}$ , then rules  $r_1, r_2, \dots$  will be eliminated from  $\Pi_{i-1}$  if no less preferred rule can be eliminated (i.e. conditions (a) and (b)). This procedure is continued until a fixed point is reached. It is worth to note that the generation of a reduct of a PLP is based on the ground form of its extended logic program part. Furthermore, if  $\mathcal{R}(r_1) < \mathcal{R}(r_2)$  holds in a PLP where  $r_1$  or  $r_2$  includes variables, then  $\mathcal{R}(r_1) < \mathcal{R}(r_2)$  is actually viewed as the set of  $<$ -relations  $\mathcal{R}(r'_1) < \mathcal{R}(r'_2)$ , where  $r'_1$  and  $r'_2$  are ground instances of  $r_1$  and  $r_2$  respectively.

**Definition 4** Let  $\mathcal{P} = (\Pi, \nabla, <)$  be a PLP and  $Gl$  the set of all ground literals in the language of  $\mathcal{P}$ . For any subset  $S$  of  $Gl$ ,  $S$  is an answer set of  $\mathcal{P}$ , denoted as  $Ans^P(\mathcal{P})$ , iff  $S = Ans(\mathcal{P}^<)$  for some reduct  $\mathcal{P}^<$  of  $\mathcal{P}$ . Given a PLP  $\mathcal{P}$ , a ground literal  $L$  is entailed from  $\mathcal{P}$ , denoted as  $\mathcal{P} \models L$ , if  $L$  belongs to every answer set of  $\mathcal{P}$ .

Using Definitions 3 and 4, it is easy to conclude that in example 2, if we assign  $R_2 > R_4$ ,  $\mathcal{P}$  has a unique reduct as follows:

$$\mathcal{P}^< = \{holds(S_1 R, O_1) \leftarrow, \\ \neg holds(S_1, R, O) \leftarrow, O \in O_1 \leftarrow\}$$

from which we obtain the following answer set of  $\mathcal{P}$ :

$$Ans^P(\mathcal{P}_1) = \{holds(S_1, R, O_1), \\ \neg holds(S_1, R, O), O \in O_1\}$$

If the preference ordering is  $R_2 < R_4$ ,  $\mathcal{P}$  has a unique reduct as follows:

$$\mathcal{P}^< = \{holds(S_1 R, O_1) \leftarrow, O \in O_1 \leftarrow, \\ holds(S_1, R, O) \leftarrow holds(S_1, R, O_1), \\ O \in O_1, not \neg holds(S_1, R, O)\}$$

from which we obtain the following answer set of  $\mathcal{P}$ :

$$Ans^P(\mathcal{P}_1) = \{holds(S_1, R, O_1), O \in \\ O_1, holds(S_1, R, O)\}$$

**Example 3** Now we consider another authorization domain  $D$ , it's corresponding program  $\mathcal{P}$  is:

$$R_1 : holds(S, W, O_3) \leftarrow, \\ R_2 : holds(S, W, O) \leftarrow \\ not holds(S, W, O_1), \\ R_3 : holds(S, W, O_2) \leftarrow, \\ R_4 : holds(S, W, O_1) \leftarrow \\ not holds(S, W, O), \\ R_1 > R_2, R_3 > R_4.$$

According to Definition 3, it is easy to see that  $\mathcal{P}$  has two reducts:

$$\{holds(S, W, O_3) \leftarrow, \\ holds(S, W, O_2) \leftarrow, \\ holds(S, W, O_1) \leftarrow not holds(S, W, O)\}, \\ and \\ \{holds(S, W, O_3) \leftarrow, \\ holds(S, W, O) \leftarrow not holds(S, W, O_1), \\ holds(S, W, O_2) \leftarrow\}.$$

From Definition 4, it follows that  $\mathcal{P}$  has two answer sets:

$$\{holds(S, W, O_3), holds(S, W, O_1), holds(S, W, O_2)\} \\ and \\ \{holds(S, W, O_3), holds(S, W, O), holds(S, W, O_2)\}.$$

## 4 Unique answer set of an authorization domain

Example 4 has two answer sets. If an access request  $holds(S, W, O_1)$  is presented, according to one answer set, the access request is granted; according to the other answer set, the same request will be denied.

Now we investigate the unique answer set of an authorization domain. To investigate this issue, we first extend the concept of local stratification for general logic programs [1] to extended logic programs.

**Definition 5** Let  $\Pi$  be an extended logic program and  $Gl$  be the set of all ground literals of  $\Pi$ .

1. A local stratification for  $\Pi$  is a function *stratum* from  $Gl$  to the countable ordinals.
2. Given a local stratification *stratum*, we extend it to ground literals with negation as failure by setting *stratum*(not  $F$ ) = *stratum*( $F$ ) + 1, where  $F$  is a ground literal.
3. A rule  $F_0 \leftarrow F_1, \dots, F_m, not F_{m+1}, \dots, not F_n$  in  $\Pi$  is locally stratified with respect to *stratum* if

$$stratum(F_0) \geq stratum(F_i), \\ where 1 \leq i \leq m, and \\ stratum(F_0) > stratum(not F_j), \\ where m + 1 \leq j \leq n.$$

4.  $\Pi$  is called locally stratified with respect to stratum if all of its rules are locally stratified.

For an extended logic program which represents certain authorization domain, if such rules exist:

$$\begin{aligned} a &\leftarrow \text{not}b \\ b &\leftarrow \text{not}a \end{aligned}$$

We will have two answer sets  $\{a\}$  and  $\{b\}$ .

The above definition is to ensure that in an extended logic program, there does not exist such rules resulting in multiple answer sets.

For instance, if we assign  $\text{stratum}(\text{nota})=1$ , according to condition 2 of the definition  $\text{stratum}(a)=\text{stratum}(\text{nota}) - 1 = 0$ . from rule  $a \leftarrow \text{not}b$  and the condition 3,  $\text{stratum}(\text{not}b) < 0$ , so  $\text{stratum}(b) < \text{stratum}(\text{not}b) < 0$ . From rule  $b \leftarrow \text{not}a$  and the condition 3,  $\text{stratum}(b) > \text{stratum}(\text{nota}) > 1$ . So a domain consists the above rules does not satisfy the definition, it is not locally stratified.

Let  $\Pi$  be a ground extended logic program and  $r$  be a rule in  $\Pi$  of the form:

$$F_0 \leftarrow F_1, \dots, F_m, \text{not } F_{m+1}, \dots, \text{not } F_n.$$

We use  $\text{pos}(r)$  to denote the set of literals in the body of  $r$  without negation as failure  $\{F_1, \dots, F_m\}$ , and  $\text{neg}(r)$  the set of literals in the body of  $r$  with negation as failure  $\{F_{m+1}, \dots, F_n\}$ . We specify  $\text{body}(r)$  to be  $\text{pos}(r) \cup \text{neg}(r)$ . We also use  $\text{head}(r)$  to denote the head of  $r$ :  $\{F_0\}$ . Then we use  $\text{Gl}(r)$  to denote  $\text{head}(r) \cup \text{body}(r)$ . By extending these notations, we use  $\text{pos}(\Pi)$ ,  $\text{neg}(\Pi)$ ,  $\text{body}(\Pi)$ ,  $\text{head}(\Pi)$ , and  $\text{Gl}(\Pi)$  to denote the unions of corresponding components of all rules in  $\Pi$ , e.g.  $\text{body}(\Pi) = \bigcup_{r \in \Pi} \text{body}(r)$ . If  $\Pi$  is a non-ground program, then notions  $\text{pos}(\Pi)$ ,  $\text{neg}(\Pi)$ ,  $\text{body}(\Pi)$ ,  $\text{head}(\Pi)$ , and  $\text{Gl}(\Pi)$  are defined based on the ground instantiation of  $\Pi$ .

The following definition is to specify under which conditions an extended logic program is locally stratified.

**Definition 6** Let  $\Pi$  be an extended logic program and  $r_p$  and  $r_q$  be two rules in  $\Pi$ . We define a set  $\mathcal{D}(r_p)$  of literals with respect to  $r_p$  as follows:

$$\begin{aligned} \mathcal{D}_0 &= \{\text{head}(r_p)\}; \\ \mathcal{D}_i &= \mathcal{D}_{i-1} \cup \{\text{head}(r) \mid \text{head}(r') \in \text{pos}(r) \\ &\text{where } r \in \Pi \text{ and } r' \text{ are those rules such that} \\ &\text{head}(r') \in \mathcal{D}_{i-1}\}; \\ \mathcal{D}(r_p) &= \bigcup_{i=1}^{\infty} \mathcal{D}_i. \end{aligned}$$

We say that  $r_q$  is defeasible through  $r_p$  in  $\Pi$  if and only if  $\text{neg}(r_q) \cap \mathcal{D}(r_p) \neq \emptyset$ .  $r_p$  and  $r_q$  are called mutually defeasible in  $\Pi$  if  $r_q$  is defeasible through  $r_p$  and  $r_p$  is defeasible through  $r_q$  in  $\Pi$ .

Intuitively, if  $r_q$  is defeasible through  $r_p$  in  $\Pi$ , then there exists a sequence of rules  $r_1, r_2, \dots, r_l, \dots$  such that  $\text{head}(r_p)$  occurs in  $\text{pos}(r_1)$ ,  $\text{head}(r_i)$  occurs in  $\text{pos}(r_{i+1})$  for all  $i = 1, \dots$ , and for some  $k$ ,  $\text{head}(r_k)$  occurs in  $\text{neg}(r_q)$ . Under this condition, it is clear that by triggering rule  $r_p$  in  $\Pi$ , it is possible to defeat rule  $r_q$  if rules  $r_1, \dots, r_k$  are triggered as well. As a special case that  $\mathcal{D}(r_p) = \emptyset$ ,  $r_q$  is defeasible through  $r_p$  iff  $\text{head}(r_p) \in \text{neg}(r_q)$ . The following proposition simply describes the relationship between local stratification and mutual defeasibility.

**Proposition 1** Given a ground extended logic program  $\Pi$ . If  $\Pi$  is locally stratified, then there are no mutually defeasible pairs of rules in  $\Pi$ .

**Proposition 2** Let  $\Pi$  be a ground extended logic program. If  $\Pi$  is locally stratified, then  $\Pi$  has a unique answer set.

The above result is easy to prove from the corresponding result for general logic programs showed in [5] based on Gelfond and Lifschitz's translation from an extended logic program to a general logic program [6]. It is observed that for a PLP  $\mathcal{P} = (\Pi, \mathcal{N}, <)$ , if  $\Pi$  is locally stratified, then  $\mathcal{P}$  will also have a unique answer set. In other words,  $\Pi$ 's local stratification implies that  $\mathcal{P}$  has a unique answer set.

For instance, if these rules are in an authorization domain:

$$\begin{aligned} a &\leftarrow b... \\ c &\leftarrow a... \\ d &\leftarrow c... \\ e &\leftarrow \dots \text{not}d \end{aligned}$$

Let the first rule be  $r_p$  and the last one be  $r_q$ . Once rule  $r_p$  takes effect, we have  $a$  in the answer set. Suppose all other conditions for rest of the rules taking effect are satisfied, we will have  $c, d$  in the answer set as well. This will prevent  $r_q$  from taking effect. So rule  $r_q$  is defeasible through  $r_p$ . Similarly, if rule  $r_q$  is triggered first, then the other rules cannot take effect since the result contradicts with the condition triggering rule  $r_q$ . So we say rule  $r_p$  and rule  $r_q$  are mutually defeasible. The domain has two answer sets:  $\{a, c, d\}$  and  $\{e\}$ .

**Example 4** An authorization domain  $D = \{R_1, R_2, R_3\}$ , where  
 $R_1: \text{holds}(S, A, O) \leftarrow$   
 $R_2: \text{holds}(S_1, A, O) \leftarrow \text{holds}(S, A, O)$   
 $R_3: \text{holds}(S_2, A, O) \leftarrow \neg \text{holds}(S_1, A, O)$   
 This domain does not have a unique answer set since  $R_1$  and  $R_3$  are mutually defeasible. It has two answer

sets:  
 $\{holds(S, A, O), holds(S_1, A, O)\}$  and  
 $\{holds(S_2, A, O)\}$

**Example 5** Here is another domain  $D = \{R_1, R_2, R_3, R_4, R_5\}$ , where

$R_1: holds(S, A, O) \leftarrow$

$R_2: holds(S_1, A, O) \leftarrow$

$R_3: holds(S_2, A, O) \leftarrow \neg holds(S_3, A, O)$

$R_4: holds(S_4, A, O) \leftarrow holds(S, A, O)$

$R_5: holds(S_5, A, O) \leftarrow holds(S_1, A, O),$   
 $\neg holds(S_3, A, O)$

This domain does not contain any pair of defeasible rules. It is locally stratified. It has a unique answer set:

$\{holds(S, A, O), holds(S_1, A, O), holds(S_2, A, O),$   
 $holds(S_4, A, O), holds(S_5, A, O)\}$

## 5 Conclusion

In this paper, we proposed a new approach to solve conflicts in authorizations. So far, certain research has been done using logic in authorizations as mention in introduction. These works either focus on authorization representation, or delegation. Little has been done in conflict resolution of authorization.

In our work, we employed a prioritized logic program to resolve authorization conflicts in an authorization domain specified by a logic language. By assigning each rule a name representing its preference ordering, using a fixed point semantics to delete those less preferred rules (the rules will not take effect under current state), then using answer set theory to evaluate the authorization domain to get the preferred authorizations. We also investigated the uniqueness of the answer set of an authorization domain and discussed the conditions under which the domain has a unique answer set. In our future work, we will consider the implementation issue with authorization evaluation and dynamic policy update. A related work using logic programs for conflict resolution in reasoning has been implemented in (removed for blind review) It is our future work to use logic programs(stable model semantics) to implement the approach for authorization conflict resolution presented in this paper.

### References:

- [1] K.R. Apt and R.N. Bol, Logic programming and negation: A survey. *Journal of Logic Programming*, **19,20** (1994) 9-71.
- [2] E. Bertino, F. Buccafurri, E. Ferrari and P. Rullo, "A Logic-based Approach for Enforcing Ac-

cess Control". *Computer Security*, vol.8, No.2-2, pp109-140, 2000.

- [3] E. Bertino, B. Catania, E. Ferrari and P. Perlasca, "A Logical Framework for Reasoning about Access Control Models". *ACM Transactions on Information and System Security*, Vol.6, No.1, pp71-127, 2003.
- [4] J. Chomicki, J. Lobo and S. Naqvi, "A Logical Programming Approach to Conflict Resolution in Policy Management". *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning*, pp121-132, 2000.
- [5] M. Gelfond and V. Lifschitz, The stable model semantics for logic programming. In *Proceedings of the Fifth Joint International Conference and Symposium*, pp 1070-1080. MIT Press, 1988.
- [6] M. Gelfond and V. Lifschitz, Classical negation in logic programs and disjunctive databases. *New Generation Computing*, **9** (1991) 365-386.
- [7] S. Jajodia, P. Samarati, M.L. Sapino and V.S. Subrahmanian, "Flexible Support for Multiple Access Control Policies". *ACM Transactions on Database Systems*, Vol.29, No.2, pp214-260, 2001.
- [8] N. Li, B. Grosz and J. Feigenbaum, "Delegation Logic: A Logic-based Approach to Distributed Authorization". *ACM Transactions on Information and System Security*, Vol.6, No.1, pp128-171, 2003.
- [9] T.Y.C. Woo and S.S. Lam, "Authorization in Distributed systems: A Formal Approach". *Proceedings of IEEE Symposium on Research in Security and Privacy*, pp33-50, 1992.
- [10] Y. Zhang and Y. Bai, "The Characterization on the Uniqueness of Answer Set for Prioritized Logic Programs". *Proceedings of the International Symposium on methodologies on Intelligent Systems*, pp349-356, 2003.
- [11] Y. Zhang, C.M. Wu and Y. Bai Implementing Prioritized Logic Programming, *AI Communications*, Vol.14, No. 4, pp183-196, 2001.