International Journal of Engineering, Mathematical and Physical Sciences
ISSN: 2517-9934
Vol:1, No:9, 2007

Logic Program for Authorizations

Yun Bai

Abstract**—** As a security mechanism, authorization is to provide access control to the system resources according to the polices 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.

Keywords**—** authorization, formal specification, conflict resolution, prioritized logic program.

I. Introduction

Authorization plays an essential role to ensure the security of a wide variety of computing and IT systems such as data management systems, e-trading systems, database transaction systems, etc. Research in authorization has long been an important area in computer system security. Specifically, in a computer system, the function of the authorization is to control access to the system. It only allows the authorized users performing authorized operations on the shared data resource of the system. Study on the formal specification of authorization (also called *access control*) has become a major challenge in the current development of secure computing and IT systems. Jajodia *et al* [8] 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* [9] 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* [5] discussed security policy management using logic program approach. Woo and Lam proposed a formal approach using default logic to represent and evaluate authorizations [10].

This paper is to address high level authorization specification and resolution for inconsistent authorizations 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 effective and efficient conflict resolution. We discuss the unique answer set of an authorization domain and its proof in section 4 and section 5 concludes the paper.

Yun Bai is with School of Computing and Mathematics, University of Western Sydney, NSW 1797, Australia, E-mail: ybai@scm.uws.edu.au

II. Authorization description

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:

$$F_0 \leftarrow F_1, \cdots, F_m, notF_{m+1}, \cdots, notF_n, \qquad (1)$$

where each F_i $(0 \leq i \leq n)$ is a literal. F_0 is called the *head* of the rule, while $F_1, \cdots, F_m$,*not* $F_{m+1}, \cdots$, *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 $\{F_1, \cdots, F_m\}$ is the literals without weak negation; the set $\{notF_{m+1}, \cdots, notF_n\}$ 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, ...k) where R_i is a rule of the form $F_0 \leftarrow$ or $F_0 \leftarrow F_1, \cdots, F_m, notF_{m+1}, \cdots, notF_n$ where m>0, n>m.

The following is an example of an authorization domain.

Example 1: $D = \{R_1, R_2, R_3\}$, where
$\neg R_1$: $holds(S, R, O) \leftarrow$
R_2: $holds(S_1, W, O) \leftarrow holds(S_2, W, O)$
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 does not have read right on object O; if subject S_2 has 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. ■

III. Prioritized logic program and evaluation

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: $holds(S_1, R, O) \leftarrow holds(S_1, R, O_1), O \in O_1,$ *not* $\neg holds(S_1, R, O)$

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 $holds(S_1, R, O)$ 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 $\neg holds(S_1, R, O)$ 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:

R_1: $holds(S, A, O) \leftarrow$
R_2: $holds(S_1, A, O) \leftarrow$
R_3: $holds(S_2, A, O) \leftarrow holds(S_1, A, O)$

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:

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

Now, the new knowledge R_4: $\neg holds(S_1, A, O)$ is added to the domain. It conflicts with the existing R_2: $holds(S_1, A, O)$. 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:

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

We call the logic program with partial ordering $<$ on the rules *prioritized logic program* $\mathcal{P}$ [11]. $\mathcal{P}$ is defined to be a triplet $(\Pi, \mathcal{R}, <)$, where Π is an extended logic program, $\mathcal{R}$ is a naming function mapping each rule in Π 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 $\mathcal{P}$.

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 Π 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 Π be a ground extended logic program and r a rule with the form $R_0 \leftarrow R_1, \cdots, R_m,$ not $R_{m+1}, \cdots,$ not R_n (r does not necessarily belong to Π). Rule r is *defeated* by Π iff Π has an answer set and for any answer set $Ans(\Pi)$ of Π, there exists some $R_i \in Ans(\Pi)$, 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 $\{holds(S, R, O_1), O \in O_1, holds(S_1, R, O)\}$.

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 Π_i $(i = 0, 1, \cdots)$ such that:

1) $\Pi_0 = \Pi$;
2) $\Pi_i = \Pi_{i-1} - \{r_1, r_2, \cdots \mid$ (a) there exists $r \in \Pi_{i-1}$ such that
   for every j ($j = 1, 2, \cdots$), $\mathcal{N}(r) < \mathcal{N}(r_j) \in \mathcal{P}(<)$ and
   $r_1, \cdots,$ are defeated by $\Pi_{i-1} - \{r_1, r_2, \cdots\}$, and (b) there
   does not exist a rule $r' \in \Pi_{i-1}$ such that $N(r_j) < N(r')$
   for some j ($j = 1, 2, \cdots$) and r' 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 Π by eliminating some *less preferred rules* from Π. In particular, if $\mathcal{R}(r) < \mathcal{R}(r_1)$, $\mathcal{R}(r) < \mathcal{R}(r_2)$, $\cdots$, and $\Pi_{i-1} - \{r_1, r_2, \cdots\}$ defeats $\{r_1, r_2, \cdots\}$, then rules $r_1, r_2, \cdots$ will be eliminated from Π_{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:

$\mathcal{P}^< = \{holds(S_1R, O_1) \leftarrow,$
$\neg holds(S_1, R, O) \leftarrow, O \in O_1 \leftarrow\}$

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:

$\mathcal{P}^< = \{holds(S_1R, O_1) \leftarrow, O \in O_1 \leftarrow,$
$holds(S_1, R, O) \leftarrow holds(S_1, R, O_1),$
$O \in O_1,$ *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
$\{holds(S, W, O_3), holds(S, W, O), holds(S, W, O_2)\}$. ■

IV. 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 Π be an extended logic program and Gl be the set of all ground literals of Π.

1) A *local stratification* for Π 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 $stratum(\text{not } F) = stratum(F) + 1$, where F is a ground literal.
3) A rule $F_0 \leftarrow F_1, \cdots, F_m,$ not $F_{m+1}, \cdots,$ not F_n in Π is *locally stratified* with respect to $stratum$ if
$stratum(F_0) \geq stratum(F_i)$, where $1 \leq i \leq m$, and
$stratum(F_0) > stratum(notF_j)$, where $m+1 \leq j \leq n$.
4) Π 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:

$a \leftarrow notb$
$b \leftarrow nota$

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

Let Π be a ground extended logic program and r be a rule in Π of the form:

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

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

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

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

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

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

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

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

Proposition 2: Let Π be a ground extended logic program. If Π is locally stratified, then Π has a unique answer set.

The above result is easy to prove from the corresponding result for general logic programs showed in [6] based on Gelfond and Lifschitz's translation from an extended logic program to a general logic program [7]. It is observed that for a PLP $\mathcal{P} = (\Pi, \mathcal{N}, <)$, if Π is locally stratified, then $\mathcal{P}$ will also have a unique answer set. In other words, Π's local stratification implies that $\mathcal{P}$ has a unique answer set.

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

$a \leftarrow b...$
$c \leftarrow a...$
$d \leftarrow c...$
$e \leftarrow ...notd$

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 $\{e\}$.

Theorem 1: Every prioritized logic program has a $<$-partition.

Theorem 2: (**Unique Answer Set Theorem**) Let $\mathcal{P} = (\Pi, \mathcal{N} <)$ be a ground PLP and $\{\Pi_1, \cdots, \Pi_k\}$ be a $<$-partition of Π in $\mathcal{P}$. $\mathcal{P}$ has a unique reduct if there does not exist two rules r_p and r_q in Π_i and Π_j $(i, j > 1)$ respectively such that r_p and r_q are mutually defeasible in Π. $\mathcal{P}$ has a unique answer set if $\mathcal{P}$ has a unique locally stratified reduct.

Proof: According to Proposition 3, it is sufficient to only prove the first part of this theorem: $\mathcal{P}$ has a unique reduct if there does not exist two rules r_p and r_q in Π_i and Π_j $(1 < i, j)$ respectively such that r_p and r_q are mutually defeasible in Π.

We assume that $\mathcal{P}$ has two different reducts, say $\mathcal{P}^{<(1)}$ and $\mathcal{P}^{<(2)}$. This follows that there exist at least two different rules r_p and r_q such that (1) $r_p \in \Pi_i$ and $r_q \in \Pi_j$, where $1 < i, j$; (2) $r_q \in \mathcal{P}^{<(1)}$, $r_q \notin \mathcal{P}^{<(2)}$, and $r_p \notin \mathcal{P}^{<(1)}$; and (3) $r_p \in \mathcal{P}^{<(2)}$, $r_p \notin \mathcal{P}^{<(1)}$, and $r_q \notin \mathcal{P}^{<(2)}$. According to Definition 2, $\mathcal{P}^{<(1)}$ and $\mathcal{P}^{<(2)}$ are generated from two reduct chains $\{\Pi_0^{(1)}, \Pi_1^{(1)}, \cdots\}$ and $\{\Pi_0^{(2)}, \Pi_1^{(2)}, \cdots\}$ respectively.

Without loss of generality, suppose that for all $0 \leq i < k$, $\Pi_i^{(1)} = \Pi_i^{(2)}$, and

$$\Pi_k^{(1)} = \Pi_{k-1}^{(1)} - \{r_1, \cdots, r_l, r_p, \cdots\},$$
$$\Pi_k^{(2)} = \Pi_{k-1}^{(2)} - \{r_1, \cdots, r_l, r_q, \cdots\},$$

where we set $\Pi_{k-1} = \Pi_{k-1}^{(1)} = \Pi_{k-1}^{(2)}$ and the only difference between $\Pi_k^{(1)}$ and $\Pi_k^{(2)}$ is due to rules r_p and r_q. Let r_p and r_q have the following forms:

$r_p : L_p \leftarrow \cdots$, *not* $L'_p, \cdots$,
$r_q : L_q \leftarrow \cdots$, *not* $L'_q, \cdots$.

Comparing $\Pi_k^{(1)}$ and $\Pi_k^{(2)}$, it is clear that the only difference between these two programs is about rules r_p and r_q. Since $\Pi_k^{(1)}$ defeats r_p and $\Pi_k^{(2)}$ defeats r_q, it follows that $L'_q \in S_k^{(1)}$ and $L'_p \in S_k^{(2)}$, where $S_k^{(1)}$ and $S_k^{(2)}$ are answer sets of $\Pi_k^{(1)}$ and $\Pi_k^{(2)}$ respectively. Then there must exist some rule in $\Pi_k^{(1)}$ of the form:

$r^{(1)} : L'_p \leftarrow \cdots$,

and some rule in $\Pi_k^{(2)}$ of the form:

$r^{(2)} : L'_q \leftarrow \cdots$.

Furthermore, since $\Pi_k^{(1)} - \{r_p, r_q\}$ does not defeat rule r_p and $\Pi_k^{(2)} - \{r_p, r_q\}$ does not defeat rule r_q (otherwise $\Pi_k^{(1)} = \Pi_k^{(2)}$), it is observed that rule r_q triggers rule $r^{(1)}$ in $\Pi_k^{(1)}$ that defeats r_p, and rule r_p triggers rule $r^{(2)}$ in $\Pi_k^{(2)}$ that defeats r_q. This follows that r_p and r_q are mutually defeasible in Π. ■ ■

Example 4: An authorization domain $D = \{R_1, R_2, R_3\}$, where
R_1: $holds(S, A, O) \leftarrow$
R_2: $holds(S_1, A, O) \leftarrow holds(S, A, O)$
R_3: $holds(S_2, A, O) \leftarrow \neg holds(S_1, A, O)$
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:
$\{holds(S, A, O), holds(S_1, A, O), holds(S_2, A, O),$
$holds(S_4, A, O), holds(S_5, A, O)\}$ ■

V. 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 Access 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] V. Crescini and Y. Zhang, "A logic Based Approach for Dynamic Access Control". *Proceedings of 17th Australian Joint Conference on Artificial Intelligence (AI 2004)*, pp623-635, 2004.
[6] 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.
[7] M. Gelfond and V. Lifschitz, Classical negation in logic programs and disjunctive databases. *New Generation Computing*, **9** (1991) 365-386.
[8] 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.
[9] N. Li, B. Grosof 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.
[10] 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.
[11] 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.
[12] Y. Zhang, C.M. Wu and Y. Bai Implementing Prioritized Logic Programming, *AI Communications*, Vol.14, No. 4, pp183–196, 2001.