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.

II. AUTHORIZATION DESCRIPTION

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

In language \( L \), the fact that a subject \( S \) has access right \( R \) for object \( O \) is represented using a ground atom \( \text{holds}(S, A, O) \). 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 \( L \) or its negation, while a ground fact \( \neg \text{F} \).

An extended logic program is a collection of such rules. In a rule, the set \( \{F_1, \ldots, F_m\} \) is the literals without weak negation; the set \( \{\neg \text{F}_1, \ldots, \neg \text{F}_n\} \) 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 \( D = \{R_i\} \), \( i=1,2, \ldots, k \) where \( R_i \) is a rule of the form \( F_0 \leftarrow \text{F}_1, \ldots, \text{F}_n, \neg \text{F}_{n+1}, \ldots, \neg \text{F}_m \) where \( m \geq 0 \) and \( n \geq m \).

The following is an example of an authorization domain.

Example 1: \( D = \{R_1, R_2, R_3\} \), where

\(-R_1: \text{holds}(S, R, O) \leftarrow R_2: \text{holds}(S_1, W, O) \leftarrow \text{holds}(S_2, W, O) \)
\( R_3: \text{holds}(S_3, R, O) \leftarrow \text{holds}(S_3, R, O_1), O \in O_1, \neg \text{holds}(S_3, R, O) \)

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

- $R_1 : holds(S_1, R, O_1) ←$
- $R_2 : ¬holds(S_1, R, O) ←$
- $R_3 : O ∈ O_1 ←$
- $R_4 : holds(S_2, R, O) ← holds(S_1, R, O_1), O ∈ O_1, not ¬holds(S_1, R, O)$

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 $S_1$ cannot 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 among the conflicting 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 $R_4$ 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. ■

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:

- $R_1 : holds(S, A, O) ←$
- $R_2 : holds(S_1, A, O) ←$
- $R_3 : holds(S_2, A, O) ← holds(S_1, A, O)$

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:

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

Now, the new knowledge $R_4 : ¬\text{holds}(S_1, A, O)$ 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 conflict.

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:

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

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 $(\Pi, R, <)$, where $\Pi$ is an extended logic program, $R$ 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 $\mathcal{P}$.

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 $\mathcal{P} = (\Pi, R, <)$. 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 $\mathcal{P} = (\Pi, R, <)$ will be based on its ground instantiation $\mathcal{P}' = (\Pi', R', <')$. 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 $R_0 ← R_1, ⋅ ⋅ ⋅, R_n$, not $R_{n+1}, \ldots, not R_r (r$ does not necessarily belong to $\Pi)$. Rule $r$ is defeated by $\Pi$ if $\Pi$ has an answer set and for any answer set $\text{Ans}(\Pi)$ of $\Pi$, there exists some $R_i ∈ \text{Ans}(\Pi)$, where $m + 1 ≤ i ≤ n$.

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

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

1) $\Pi_0 = \Pi$;
2) $\Pi_i = \Pi_{i−1} \setminus \{r_1, r_2, ⋅ ⋅ ⋅\}$ (a) there exists $r ∈ \Pi_{i−1}$ such that
   - for every $j (j = 1, 2, ⋅ ⋅ ⋅)$, $N(r) < N(r_j) ∈ \mathcal{P}(<)\prec$
   - and $r_1, ⋅ ⋅ ⋅, r_n$ are defeated by $\Pi_{i−1} \setminus \{r_1, r_2, ⋅ ⋅ ⋅\}$, and (b) there
   - does not exist a rule $r′ ∈ \Pi_{i−1}$ such that $N(r′) < N(r′′)$
   - for some $j (j = 1, 2, ⋅ ⋅ ⋅)$ and $r'$ is defeated by $\Pi_{i−1} \setminus \{r'\}$;
3) $\mathcal{P}^\prec = \bigcap_{i=0}^{\infty} \Pi_i$.

In Definition 3, $\mathcal{P}^\prec$ is a ground extended logic program obtained from $\Pi$ by eliminating some less preferred rules from $\Pi$. In particular, if $\mathcal{R}(r) < \mathcal{R}(r_1)$, $\mathcal{R}(r) < \mathcal{R}(r_2)$, ⋅ ⋅ ⋅, and $\Pi_{i−1} \setminus \{r_1, r_2, ⋅ ⋅ ⋅\}$ defeats $\{r_1, r_2, ⋅ ⋅ ⋅\}$, then $r_1, r_2, ⋅ ⋅ ⋅$ are 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 $\mathcal{P} = (\Pi, \vee, \lhd)$ be a PLP and $Gl$ is an answer set of $\mathcal{P}$, denoted as $Ans^r(\mathcal{P})$, iff $S = Ans(\mathcal{P}^r)$ for some reduct $\mathcal{P}^r$ 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:

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

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

$$Ans^r(\mathcal{P}_4) = \{ holds(S_1, R, O_1), O \in O_1 \}$$

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

- $R_1 : holds(S, W), O_3 \lhd$
- $R_2 : holds(S, W), O \lhd \neg notb, holds(S, W), O_1$
- $R_3 : holds(S, W), O_2 \lhd$
- $R_4 : holds(S, W), O_1 \lhd \neg notb, holds(S, W), O$
- $R_1 \ast R_2, R_3 \ast R_4$

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

$$\{ holds(S, W), O_3 \lhd, holds(S, W), O_2 \lhd, \neg notb, holds(S, W), O_1 \}$$

and

$$\{ holds(S, W), O_3 \lhd, \neg notb, holds(S, W), O_1 \lhd \neg notb, holds(S, W), O \}$$

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 $\Pi$ 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 $F_0 \leftarrow F_1, \ldots, F_m$, not $F_{m+1}, \ldots, not F_n$ 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:

$$a \leftarrow not b$$

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(not $a$) = 1, according to condition 2 of the definition stratum($a$) = stratum(not $a$) - 1 = 0, from rule $a \leftarrow not a$ and the condition 3, stratum(not $a$) < 0, so stratum($b$) < stratum(not $b$) < 0. From rule $b \leftarrow not a$ and the condition 3, stratum($b$) > stratum(not $a$) > 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:

$$F_0 \leftarrow F_1, \ldots, F_m, not F_{m+1}, \ldots, not F_n$$

We use pos($r$) to denote the set of literals in the body of $r$ without negation as failure $\{ F_1, \ldots, F_m \}$, and neg($r$) the set of literals in the body of $r$ with negation as failure $\{ F_{m+1}, \ldots, F_n \}$. 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 $\Pi$, e.g. body($\Pi$) = $\bigcup_{r \in \Pi}$ body($r$). If $\Pi$ 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 $\Pi$.

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 $D(r_p)$ of literals with respect to $r_p$ as follows:

$$D_0 = \{ head(r_p) \};$$
$$D_i = D_{i-1} \cup \{ head(r') \mid head(r') \in pos(r) \}$$
$$D(r_p) = \bigcup_{i=1}^{\infty} D_i;$$
$$r \in \Pi \text{ and } r' \text{ are those rules such that } head(r') \in D_{i-1};$$
$$D(r_p) = \bigcup_{i=1}^{\infty} D_i.$$
We say that \( r_q \) is defeasible through \( r_p \) in \( \Pi \) if and only if 
\[ \text{neg}(r_q) \cap D(r_p) \neq \emptyset, \]
and \( r_q \) and \( r_p \) 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 \( r_1, r_2, \ldots, r_k \) such that \( \text{head}(r_p) \) occurs in \( \text{pos}(r_1), \text{head}(r_1) \) occurs in \( \text{pos}(r_i) \) for all \( i = 1, \ldots, k \), and for some \( k, \text{head}(r_k) \) occurs in \( \text{neg}(r_i) \). Under this condition, it is clear that by triggering rule \( r_p \) in \( \Pi \), it is possible to defeat rule \( r_q \) if rules \( r_1, \ldots, r_k \) are triggered as well. As a special case that \( D(r_p) = \emptyset, r_q \) 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 [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 \( \Pi = (\Pi, \mathcal{N}, \prec) \), if \( \Pi \) is locally stratified, then \( \Pi \) will also have a unique answer set. In other words, \( \Pi \)’s local stratification implies that \( \Pi \) has a unique answer set.

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

\[
\begin{align*}
& a \leftarrow b \\
& c \leftarrow a \\
& d \leftarrow c \\
& e \leftarrow \ldots \text{notd}
\end{align*}
\]

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 \( r_p \) is defeasible through \( r_q \). Similarly, if rule \( r_q \) is triggered first, then the other rules cannot take effect since the result contradicts with the condition triggering rule \( r_p \). So we say rule \( r_p \) and rule \( r_q \) are mutually defeasible. The domain has two answer sets: \{a, e, d\} and \{e\}.

**Theorem 1:** Every prioritized logic program has a \(<\)-partition.

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

**Proof:** According to Proposition 3, it is sufficient to only prove the first part of this theorem: \( \Pi \) has a unique reduct if there does not exist two rules \( r_p \) and \( r_q \) in \( \Pi_i \) and \( \Pi_j (1 < i, j) \) respectively such that \( r_p \) and \( r_q \) are mutually defeasible in \( \Pi_i \).

We assume that \( \Pi \) 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) \( r_p \in \mathcal{P}^{<(1)} \), \( r_q \not\in \mathcal{P}^{<(2)} \), and \( r_p \not\in \mathcal{P}^{<(1)} \); and (3) \( r_p \in \mathcal{P}^{<(2)} \), \( r_q \not\in \mathcal{P}^{<(1)} \), and \( r_q \not\in \mathcal{P}^{<(2)} \). According to Defn 2, \( \mathcal{P}^{<(1)} \) and \( \mathcal{P}^{<(2)} \) are generated from two reduct chains \( \{\Pi_0^{(1)}, \Pi_1^{(1)}, \ldots\} \) and \( \{\Pi_0^{(2)}, \Pi_1^{(2)}, \ldots\} \) respectively.

Without loss of generality, suppose that for all \( 0 \leq i < k, \Pi_i^{(1)} = \Pi_i^{(2)} \), and
\[
\begin{align*}
\Pi_i^{(1)} &= \Pi_{k+1}^{(1)} - \{r_1, \ldots, r_i, r_p, \ldots\}, \\
\Pi_i^{(2)} &= \Pi_{k+1}^{(2)} - \{r_1, \ldots, r_i, r_q, \ldots\},
\end{align*}
\]
where we set \( \Pi_{k+1}^{(1)} = \Pi_{k+1}^{(2)} = \Pi^{(1)} \) 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:
\[
\begin{align*}
r_p : L_p &\leftarrow \ldots, \text{not} L'_p, \ldots, \\
r_q : L_q &\leftarrow \ldots, \text{not} L'_q, \ldots,
\end{align*}
\]
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 \( L'_p \in S_k^{(1)} \) and \( L'_q \in S_k^{(2)} \), 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:
\[
r : L_p \leftarrow \ldots,
\]
and some rule in \( \Pi_k^{(2)} \) of the form:
\[
r : L_q \leftarrow \ldots,
\]
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 \( r' \) in \( \Pi_k^{(1)} \) that defeats \( r_p \), and rule \( r_p \) triggers rule \( r' \) in \( \Pi_k^{(2)} \) that defeats \( r_q \). This follows that \( r_p \) and \( r_q \) are mutually defeasible in \( \Pi \).

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

**Example 5:** Here is another domain \( D = \{R_1, R_2, R_3, R_4, R_5\} \), where
\[
\begin{align*}
R_1 : &\text{holds}(S, A, O) \\
R_2 : &\text{holds}(S_1, A, O) \\
R_3 : &\text{holds}(S_2, A, O) \leftarrow \neg\text{holds}(S_1, A, O) \\
R_4 : &\text{holds}(S_3, A, O) \leftarrow \text{holds}(S, A, O) \\
R_5 : &\text{holds}(S_4, A, O) \leftarrow \text{holds}(S_1, A, O), \\
&\neg\text{holds}(S_3, A, O)
\end{align*}
\]
This domain does not contain any pair of defeasible rules. It is locally stratified. It has a unique answer set:
\{\text{holds}(S, A, O), \text{holds}(S_1, A, O), \text{holds}(S_2, A, O), \\
\text{holds}(S_3, A, O), \text{holds}(S_4, 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