International Journal of Information Security manuscript No.
(will be inserted by the editor)

On the Sequence of Authorization Policy Transformations

Yun Bai[1], **Yan Zhang**[1], **Vijay Varadharajan**[2]

[1] School of Computing and Information Technology, University of Western Sydney, Locked Bag 1797, Penrith South DC, NSW 1797, Australia, Email: {ybai,yan}@cit.uws.edu.au
[2] Department of Computing, Macquarie University, North Ryde, NSW 2109, Australia, Email: vijay@ics.mq.edu.au

Abstract In [2,3], we proposed a model-based approach to specify the transformation of authorizations based on the *principle of minimal change* [10] and its application in database systems. Nevertheless, there were some limitations in this approach. Firstly, we could not represent a sequence of transformations. Secondly, default authorizations could not be expressed. In this paper, we propose two high-level formal languages $\mathcal{L}^s$ and $\mathcal{L}^{sd}$ to specify a sequence of authorization transformations and default authorizations. Our work starts with $\mathcal{L}^s$, a simple, but expressive language to specify certain sequence of authorization transformations. Furthermore, $\mathcal{L}^{sd}$ has more powerful expressiveness than $\mathcal{L}^s$ in the sense that constraints, causal and inherited authorizations, and general default authorizations can be specified.

Key words: Authorization Policy, Policy Transformations, Formal Language, Default Logic

1 Introduction

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

As a security mechanism, authorization or access control is responsible for ensuring that all accesses to the system resources occur exclusively according to the access polices and rules specified by the security strategies. It has been extensively studied in [1,5,6,13–16,18, 22] etc. and a variety of authorization specification approaches such as access matrix, capability list, procedural and logical specifications have been investigated. In [2,3], we proposed a model-based approach to specify the transformation of authorizations based on the *principle of minimal change* [10] and its application in database systems. Nevertheless, there were some limitations in this approach. Firstly, we could not represent a sequence of transformations. For instance, this approach could not handle a problem like: given a policy base and a sequence of transformations, what is the resulting policy base after performing such a sequence of transformations? Secondly, default authorizations could not be expressed. In certain situation, default policies are needed to specify desired authorizations. In this paper, we propose a high-level formal languages $\mathcal{L}^s$ and then extend $\mathcal{L}^s$ to $\mathcal{L}^{sd}$ to specify authorization transformations[1] in secure computer systems. In particular, both $\mathcal{L}^s$ and $\mathcal{L}^{sd}$ can specify sequence of authorization transformations. Though it has a simple syntax and semantics, we show that $\mathcal{L}^s$ is expressive enough to specify some well-known examples of authorization transformations such as separation of duty and Chinese wall security policy. Language $\mathcal{L}^{sd}$ is an augmentation of $\mathcal{L}^s$ which includes *default propositions* into the domain description of authorization policies. However, the semantics of $\mathcal{L}^{sd}$ is not just a simple extension of the semantics of $\mathcal{L}^s$. We show that $\mathcal{L}^{sd}$ has more powerful expressiveness than $\mathcal{L}^s$ in the sense that constraints, causal and inherited authorizations, and general default authorizations can be specified within $\mathcal{L}^{sd}$. For description purpose, we refer policy base as policy domain description in this paper. The two terms are used interchangeably.

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

[1] In this paper, the terms authorization transformation, authorization policy transformation and policy transformation will be used interchangeably.

The rest of the paper is organized as follows. Section 2 describes language $\mathcal{L}^s$ by outlining both its syntax and semantics and gives some well known authorization policies such as separation of duty, Chinese wall security policy and dynamic separation of duty specified by the language. Section 3 specifies the language $\mathcal{L}^{sd}$ which is an augmentation of $\mathcal{L}^s$. $\mathcal{L}^{sd}$ is able to represent more complex authorization policies such as constraints, causal and inherited policies as well as default policies. Section 4 discusses some related work. Finally, section 5 concludes the paper.

2 Syntax and Semantics of Language $\mathcal{L}^s$

In this section, we propose a language $\mathcal{L}^s$ for specifying sequence of authorization policy transformations. We provide both its syntax and semantics.

2.1 Syntax of $\mathcal{L}^s$

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

The seven disjoint sorts and the predicate symbols $\in$ and $\subseteq$ are defined as follows:

1. Sort *subject*: with subject constants $S, S_1, S_2, \cdots$, and subject variables $s, s_1, s_2, \cdots$.
2. Sort *group-subject*: with group subject constants G, G_1, $G_2, \cdots$, and group subject variables $g, g_1, g_2, \cdots$.
3. Sort *access-right*: with access right constants A, A_1, A_2, $\cdots$, and access right variables $a, a_1, a_2, \cdots$.
4. Sort *group-access-right*: with group access right constants $GA, GA_1, GA_2, \cdots$, and group access right variables $ga, ga_1, ga_2, \cdots$.
5. Sort *object*: with object constants $O, O_1, O_2, \cdots$, and object variables $o, o_1, o_2, \cdots$.
6. Sort *group-object*: with group object constants GO, GO_1, $GO_2, \cdots$, and group object variables go, go_1, $go_2, \cdots$.
7. *Transformation* with finite number of transformations $T, T_1, T_2, \cdots$.
8. A ternary predicate symbol *holds* which takes arguments as *subject* or *group-subject, access-right* or *group-access-right* and *object or group-object* respectively.
9. A binary predicate symbol $\in$ which takes arguments as *subject* and *group-subject* or *access-right* and *group-access-right* or *object* and *group-object* respectively.
10. A binary predicate symbol $\subseteq$ whose both arguments are *group-subjects*, *group-access-rights* or *group-objects*.

In language $\mathcal{L}^s$, 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 access right groups such as $GA_1 \subseteq GA_2$.

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

A *policy proposition* in $\mathcal{L}^s$ is an expression of the form

$$\phi \textbf{ after } T_1, \cdots, T_m, \tag{1}$$

where ϕ is a ground fact expression and $T_1, \cdots, T_m$ ($m \geq 0$) are transformations. A policy proposition specifies a static fact expression after a sequence of transformations. Intuitively, this proposition means that after performing transformations $T_1, \cdots, T_m$ sequentially, the ground fact expression ϕ holds. If $m = 0$, we will rewrite (1) as

$$\textbf{initially } \phi, \tag{2}$$

which is called *initial policy proposition.*

A *transformation proposition* is an expression of the form

$$T \textbf{ causes } \phi \textbf{ if } \psi, \tag{3}$$

where T is a transformation, ϕ and ψ are ground fact expressions. A transformation proposition specifies a dynamic transformation procedure. Intuitively, a transformation proposition expresses the following meaning: at a given state, if the pre-condition ψ is true, then after performing the transformation T at this state, the ground fact expression ϕ will be true in the resulting state.

If the set of ψ is empty, we will rewrite (3) as

$$T \textbf{ causes } \phi, \tag{4}$$

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

A proposition is a *ground proposition* if it does not contain variables. A *policy domain description* D in $\mathcal{L}^s$ is a finite set of initial policy propositions and transformation propositions.

Given a domain description D, for a proposition containing variables, we treat it as a set of all ground propositions obtained by substituting each variable with its corresponding constants in D.

Example 1 The following is a domain description:

initially $holds(S_1, Read, O) \wedge holds(S_1, Write, O)$,
initially $holds(S_2, Read, O) \wedge holds(S_2, Write, O)$,
$Delete\text{-}write(s, Write, o)$ **causes**
$\neg holds(s, Write, o)$.

This domain description expresses the following information: initially subjects S_1 and S_2 have *Read* and $Write$ rights for object O. A transformation $Delete\text{-}write(s, Write, o)$ is available in this domain; if this transformation occurs, then the subjects S_1 and S_2 will no longer have the $Write$ right on object O.

Since in this domain the set of subject is (S_1, S_2), the set of object is O, after the transformation, we would expect to get $\neg holds(S_1, Write, O)$ and $\neg holds(S_2, Write, O)$.

Example 2 A domain description D consists of the following propositions:

initially $holds(S, Own, O)$,
$holds(S, Own, O)$ **causes** $holds(S, Write, O)$
if $\neg holds(S, Write, O)$,
$Delete\text{-}own(S, Own, O)$ **causes** $\neg holds(S, Own, O)$).

This domain description states that: initially subject S owns object O. If S cannot write on O is not explicitly specified in the domain, then S has write right on O, and S will no longer owns O if $holds(S, Own, O)$ is removed from the domain.

Example 3 Let us now consider the more general access policy on dynamic separation of duty. In this case, a subject can potentially execute any operation in a given set, though s/he cannot execute all of them. By executing some operation, s/he will automatically rule out the possibility of executing the others. The policy is referred to as dynamic in the sense that which actions a user can execute is determined by the user. For instance, consider the following simple example. Let a group *officer* be represented using a group-subject *G-Officer*. Let this group have access rights to submit, evaluate and approve a budget. Let the budget be represented using an object B. Now if a subject S belongs to *G-Officer*, that is, $S \in$*G-Officer*, then $holds(S, Submitable, B)$, $holds(S, Evaluateable, B)$ and $holds(S, Approveable, B)$. Let the transformations be $Rqst(S, Submit, B)$, $Rqst(S, Evaluate, B)$ and $Rqst(S, Approve, B)$. The domain description D can be represented as follows:

initially $S \in$*G-Officer*,
initially $holds(S, Submitable, B)$,
initially $holds(S, Evaluateable, B)$,
initially $holds(S, Approveable, B)$,

$Rqst(S, Submit, B)$ **causes**
$holds(S, Submit, B) \wedge$
$\neg holds(S, Evaluateable, B) \wedge$
$\neg holds(S, Approveable, B)$
if $S \in$*G-Officer* $\wedge holds(S, Submitable, B)$.

$Rqst(S, Evaluate, B)$ **causes**
$holds(S, Evaluate, B) \wedge$
$\neg holds(S, Approveable, B) \wedge$
$\neg holds(S, Submitable, B)$
if $S \in$*G-Officer* $\wedge holds(S, Evaluateable, B)$.

$Rqst(S, Approve, B)$ **causes**
$holds(S, Approve, B) \wedge$
$\neg holds(S, Evaluateable, B) \wedge$
$\neg holds(S, Submitable, B)$
if $S \in$*G-Officer* $\wedge holds(S, Approveable, B)$.

Example 4 Now we consider the specification of the Chinese wall access policy [8] using our domain description. The Chinese wall access policy can be viewed as a special kind of dynamic separation of duty. In Chinese wall policy, objects are grouped into *company datasets*, for instance Company-1 and Company-2. Company datasets whose organizations are in competition are then grouped together into *conflict of interest classes*. A subject can potentially access an object from either company dataset, but if the subject accesses an object in a company dataset 1, it cannot be allowed anymore to access any object in a company dataset that appear in a conflict of interest class with dataset 1. In our language, company datasets can be represented by a *group-object*. For instance, if $Company_1$ and $Company_2$ are in the same conflict of interest class, a subject who has accessed an object of $Company_1$ will not be allowed to access any object in $Company_2$ and vice versa.

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

initially $O_1 \in Company_1$,
initially $O_2 \in Company_2$,
initially $holds(S, Accessable, O_1)$,
initially $holds(S, Accessable, O_2)$,

$Rqst(S, Access, O_1)$ **causes**
$holds(S, Access, O_1) \wedge$
$\neg holds(S, Accessable, O_2)$
if $O_1 \in Company_1 \wedge O_2 \in Company_2 \wedge holds(S, Accessable, O_1)$,

$Rqst(S, Access, O_2)$ **causes**
$holds(S, Access, O_2) \wedge$
$\neg holds(S, Accessable, O_1)$
if $O_1 \in Company_1 \wedge O_2 \in Company_2 \wedge holds(S, Accessable, O_2)$.

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

Similarly, if S accesses O_2, it will not be able to access O_1.

2.2 Semantics of $\mathcal{L}^s$

Now we define the semantics of language $\mathcal{L}^s$. A *state* is a finite set of ground facts. Given a ground fact F (i.e. F is $holds(S, A, O)$ or $\neg holds(S, A, O)$) and a state σ, we say F is *true* in σ iff $F \in \sigma$, and F is *false* in σ iff $\neg F \in \sigma$. A ground fact expression $\phi \equiv F_1 \wedge \cdots \wedge F_k$, where each F_i $(1 \leq i \leq k)$ is a ground fact, is *true* in σ iff each F_i $(1 \leq i \leq k)$ is in σ. Furthermore, a fact expression with variables is true in σ iff each of its ground instances is true in σ. A state σ is *complete* if for any ground fact F of $\mathcal{L}^s$, F or $\neg F$ is in σ. Otherwise σ is called a *partial state*. An *inconsistent state* σ is a state containing a pair of complementary ground facts F and $\neg F$.

A *transition function* ρ maps a set (T, σ) into a state, where T is a transformation and σ is a state. Intuitively, $\rho(T, \sigma)$ denotes the resulting state caused by performing transformation T in σ. A *structure* M is a pair (σ, ρ), where σ is a state, and ρ is a transition function. For any structure M and any set of transformations $T_1, \cdots, T_m$, the notation $M^{T_1,\cdots,T_m}$ denotes the state

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

where ρ is the transition function of M, and σ is the state of M.

We denote a policy proposition ϕ *is satisfied* in a structure M as $M \models_{\mathcal{L}^s} \phi$ **after** $T_1, \cdots, T_m$. This is true iff ϕ is true in the state $M^{T_1,\cdots,T_m}$. Given a domain description D, we say that a state σ_0 is the *initial state* of D iff (i) for each initial policy proposition **initially** ϕ of D, ϕ is true in σ_0; (ii) if there is another state σ satisfying condition (i), then $\sigma_0 \subseteq \sigma$ (i.e. σ_0 is the least state satisfying all initial policy propositions of D).

Furthermore, we restrict that the domain description D does not contain transformations such as T **causes** ϕ **if** ψ_1 and T **causes** $\neg\phi$ **if** ψ_2. This restriction ensures that inconsistent state will not occur after transformations.

Definition 1 *A structure (σ_0, ρ) is a* model *of a domain description D iff σ_0 is a consistent initial state of D, and for any transformation T and state σ, the following conditions hold:*

1. *if D includes a transformation proposition T* **causes** *ϕ* **if** *ψ, and ψ is true in σ, then ϕ is true in $\rho(T, \sigma)$;*
2. *for each F in D and $F \in \sigma$, $F \in \rho(T, \sigma)$ iff $\neg F \notin \rho(T, \sigma)$.*

Condition 1 says that if precondition ψ for transformation T is true in the current state σ, then after performing the transformation, the effect of the transformation ϕ should be true in the resulting state $\rho(T, \sigma)$. Condition 2 states that for any existing fact F in current state, F still holds in the resulting state after transformation T if and only if $\neg F$ is not generated from the transformation T.

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

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

Proof Obviously, a consistent domain description D has a model. From the definition of domain description, D is a finite set of initial policy propositions and transformation propositions. As each initial policy proposition has the form

initially ϕ,

where ϕ is a conjunction of ground atomic formulas or their negations. Then there must be a unique initial state σ_0 in which each initial policy proposition of D is true. Now consider each transformation proposition in D:

T **causes** ϕ **if** Ψ.

Since both ϕ and Ψ are conjunctions of atomic formulas or their negations, for any transition ρ, the resulting state $\rho(T, \sigma)$ is unique. Therefore, according to Definition 1, it is obvious that the model (σ_0, ρ) of D is unique.

Example 5 Continuation of Example 3. For the dynamic separation of duty example, the initial state of D is:

$$\begin{aligned}\sigma_0 = \{&S \in \textit{G-Officer}, holds(S, Submitable, B),\\ &holds(S, Evaluateable, B),\\ &holds(S, Approveable, B)\}.\end{aligned}$$

It can be easily shown that the following results hold:

$$\begin{aligned}D \models_{\mathcal{L}^s}\ &S \in \textit{G-Officer} \wedge holds(S, Submit, B) \wedge\\ &\neg holds(S, Evaluateable, B) \wedge\\ &\neg holds(S, Approveable, B)\\ &\textbf{after } Rqst(S, Submit, B),\\ D \models_{\mathcal{L}^s}\ &S \in \textit{G-Officer} \wedge\\ &holds(S, Evaluate, B) \wedge\\ &\neg holds\ (S, Submitable, B) \wedge\\ &\neg holds(S, Approveable, B)\\ &\textbf{after } Rqst(S, Evaluate, B),\\ D \models_{\mathcal{L}^s}\ &S \in \textit{G-Officer} \wedge holds(S, Approve, B) \wedge\\ &\neg holds(S, Submitable, B) \wedge\\ &\neg holds(S, Evaluateable, B)\\ &\textbf{after } Rqst(S, Approve, B).\end{aligned}$$

Example 6 Continuation of Example 4. For the Chinese wall policy, the initial state of D is:

$$\begin{aligned}\sigma_0 = \{&O_1 \in Company_1, O_2 \in Company_2,\\ &holds(S, Accessable, O_1),\\ &holds(S, Accessable, O_2)\}.\end{aligned}$$

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

$D \models_{\mathcal{L}^s} O_1 \in Company_1 \wedge O_2 \in Company_2 \wedge$
$holds(S, Access, O_1) \wedge$
$\neg holds(S, Accessable, O_2)$
after $Rqst(S, Access, O_1)$,
$D \models_{\mathcal{L}^s} O_1 \in Company_1 \wedge O_2 \in Company_2 \wedge$
$holds(S, Access, O_2) \wedge$
$\neg holds(S, Accessable, O_1)$
after $Rqst(S, Access, O_2)$,
$D \models_{\mathcal{L}^s} O_1 \in Company_1 \wedge O_2 \in Company_2 \wedge$
$holds(S, Access, O_1) \wedge$
$\neg holds(S, Accessable, O_2)$
after $Rqst(S, Access, O_1), Rqst(S, Access, O_2)$,
$D \models_{\mathcal{L}^s} O_1 \in Company_1 \wedge O_2 \in Company_2 \wedge$
$holds(S, Access, O_2) \wedge$
$\neg holds(S, Accessable, O_1)$
after $Rqst(S, Access, O_2), Rqst(S, Access, O_1)$.

2.3 Document Release Example

In this section, we consider a slightly modified version of the well-known document release example [24] and specify the authorization transformations using our language $\mathcal{L}^s$.

Example 7 The following is cited from [24]:

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

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

initially $holds(Sci, Own, Doc)$,
initially $holds(Sci, Read, Doc)$,
initially $holds(Sci, Write, Doc)$,

$Rqst(Sci, Doc, PO)$ **causes**
$holds(PO, Review, Doc) \wedge \neg holds(Sci, Write, Doc)$
if $holds(Sci, Own, Doc) \wedge holds(Sci, Write, Doc)$,

$Get\text{-}approval(Sci, Doc, PO)$ **causes**
$holds(Sci, Pat\text{-}ok, Doc) \wedge \neg holds(PO, Review, Doc)$
if $holds(PO, Review, Doc) \wedge holds(Sci, Own, Doc)$,

$Get\text{-}rejection(Sci, Doc, PO)$ **causes**
$holds(Sci, Pat\text{-}reject, Doc) \wedge \neg holds(PO, Review, Doc)$,
if $holds(PO, Review, Doc) \wedge holds(Sci, Own, Doc)$,

$Release\text{-}doc(Sci, Doc)$ **causes**
$holds(Sci, Release, Doc) \wedge \neg holds(Sci, Pat\text{-}ok, Doc)$
if $holds(Sci, Pat\text{-}ok, Doc)$,

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

The initial state of D is

$\sigma_0 = \{holds(Sci, Own, Doc), holds(Sci, Read, Doc),$
$holds(Sci, Write, Doc)\}$.

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

$D \models_{\mathcal{L}^s} holds(PO, Review, Doc) \wedge$
$\neg holds(Sci, Write, Doc)$
after $Rqst(Sci, Doc, PO)$,
$D \models_{\mathcal{L}^s} holds(Sci, Pat\text{-}ok, Doc) \wedge$
$\neg holds(PO, Review, Doc)$
after $Rqst(Sci, Doc, PO)$,
$Get\text{-}approval(Sci, Doc, PO)$,
$D \models_{\mathcal{L}^s} holds(Sci, Pat\text{-}reject, Doc) \wedge$
$\neg holds(PO, Review, Doc)$
after $Rqst(Sci, Doc, PO)$,
$Get\text{-}rejection(Sci, Doc, PO)$,
$D \models_{\mathcal{L}^s} holds(Sci, Release, Doc) \wedge$
$holds(Sci, Pat\text{-}ok, Doc)$
after $Rqst(Sci, Doc, PO)$,
$Get\text{-}approval(Sci, Doc, PO)$,
$Release\text{-}doc(Sci, Doc)$,
$D \models_{\mathcal{L}^s} holds(Sci, Write, Doc)$
after $Rqst(Sci, Doc, PO)$,
$Get\text{-}rejection(Sci, Doc, PO)$,
$Revise\text{-}doc(Sci, Doc)$.

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

The following results hold.

$D \models_{\mathcal{L}^s} holds(Sci, Own, Doc)$
after $Rqst(Sci, Doc, PO)$,
$D \models_{\mathcal{L}^s} holds(Sci, Own, Doc)$

after $Rqst(Sci, Doc, PO)$,
$Get\text{-}approval(Sci, Doc, PO)$,
$D \models_{\mathcal{L}^s} holds(Sci, Own, Doc)$
after $Rqst(Sci, Doc, PO)$,
$Get\text{-}rejection(Sci, Doc, PO)$,
$D \models_{\mathcal{L}^s} holds(Sci, Own, Doc)$
after $Rqst(Sci, Doc, PO)$,
$Get\text{-}approval(Sci, Doc, PO)$,
$Release\text{-}doc(Sci, Doc)$,
$D \models_{\mathcal{L}^s} holds(Sci, Own, Doc)$
after $Rqst(Sci, Doc, PO)$,
$Get\text{-}rejection(Sci, Doc, PO)$,
$Revise\text{-}doc(Sci, Doc)$.

2.4 Properties of Language $\mathcal{L}^s$

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

- *Incomplete information is allowed.* In $\mathcal{L}^s$, the state of the authorization policies can be specified to be *incomplete* in the sense that some authorizations may not be represented in the state. For instance, in Example 7, the initial state σ_0 is incomplete because it does not include the facts $holds(PO, Review, Doc)$ or $\neg holds(PO, Review, Doc)$.
- *Denials are expressed explicitly.* As incomplete information is allowed in the state of authorization policies, denials (negations of authorization policies) must be explicitly represented in the state.
- *The entailment relation $\models_{\mathcal{L}^s}$ of $\mathcal{L}^s$ is nonmonotonic with respect to transformation propositions.* Recall that a domain description D is a finite set of policy propositions and transformation propositions. The nonmonotonicity of $\models_{\mathcal{L}^s}$ with respect to transformation propositions states that adding more transformation propositions into D may result in a policy proposition being no longer entailed in the domain description. This is because a new transformation proposition may change an authorization policy to become negative. For example, consider a domain description D consisting of the following policy and transformation propositions:
  initially $holds(S, Read, File)$,
  $Assign\text{-}write(S, Write, File)$ **causes**
  $holds(S, Write, File)$.

  Clearly, we have $D \models_{\mathcal{L}^s} holds(S, Write, File)$ **after** $Assign\text{-}write(S, Write, File)$. However, if we add another transformation proposition into D:
  $Delete\text{-}write(S, Write, File)$ **causes**
  $\neg holds(S, Write, File)$
  if $holds(S, Write, File)$,
  Then we have $D' \models_{\mathcal{L}^s} \neg holds(S, Write, File)$ **after**
  $Assign\text{-}write(S, Write, File)$,
  $Delete\text{-}write(S, Write, File)$.
  where D' is the new domain description with the above transformation proposition added into D.
- $\models_{\mathcal{L}^s}$ *is also nonmonotonic with respect to policy propositions.* This can be observed from the following example. Suppose a domain description D consists of the following policy and transformation propositions:
  initially $holds(S, Own, File)$,
  $Delete\text{-}own(S, Own, File)$ **causes**
  $\neg holds(S, Own, File)$ **if** $S \in G$.
  As the pre-condition $S \in G$ of $Delete\text{-}own(S, Own, File)$ does not hold in the initial state, we have $D \models_{\mathcal{L}^s} holds(S, Own, File)$ **after** $Delete\text{-}own(S, Own, File)$. However, if we add a policy proposition $S \in G$ into D to form a new domain description D', then we have
  $D' \models_{\mathcal{L}^s} \neg holds(S, Own, File)$ **after**
  $Delete\text{-}own(S, Own, File)$.

  Now let us discuss some limitations of $\mathcal{L}^s$:

- $\mathcal{L}^s$ cannot express *inherited* and *causal* authorization policies. For instance, in many situations, a subject can inherit an access right from its group's access right. Unfortunately, this kind of fact cannot be expressed by $\mathcal{L}^s$. Also an authorization policy may have a causal relationship with other policies. For example, we may need to express the fact that if subject S has $Write$ right on object O, then subject S' should also have $Read$ right on object O. $\mathcal{L}^s$ is not able to express this kind of causal relation.
- $\mathcal{L}^s$ cannot express *default* authorization policies. For instance, the *closed world assumption* on the state of authorization policies can be viewed as a general default authorization: any policy that is not explicitly represented in the state will be assumed to be its negation. Therefore, if $holds(S, Write, O)$ is not explicitly specified in the domain, by default, we assume that S cannot write on O.
- $\mathcal{L}^s$ cannot express *constraints*. Sometimes we need to specify some restrictions on authorization policies. For instance, we are unable to specify that group G_1 remains a subgroup of G in any state in the domain. Such restriction is represented by *constraints* which should be satisfied by any state.

3 Syntax and Semantics of Language $\mathcal{L}^{sd}$

Because of its simple syntax and semantics, $\mathcal{L}^s$ has relatively low cost for implementation. With its expressive power, $\mathcal{L}^s$ is ideal for some information systems where extensive authorization control is not necessary.

For some systems where more flexible authorizations are needed, $\mathcal{L}^s$ can be extended to $\mathcal{L}^{sd}$ to overcome these limitations mentioned above, to satisfy more complex authorization requirements.

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

The language $\mathcal{L}^{sd}$ has the same sorts and types of propositions as the language $\mathcal{L}^s$ but in addition has one more

type of proposition that we refer to as *default proposition* of the form:

$$\phi \textbf{ implies } \psi \textbf{ with absence } \gamma, \tag{5}$$

where ϕ, ψ and γ are fact expressions. Note that ϕ, ψ and γ may contain variables. In this case, the default proposition (5) will be treated as a set of default propositions obtained from (5) by replacing ϕ, ψ and γ with their ground instances respectively.

Intuitively, the default proposition says that if ϕ is true in a state σ, and it can not be derived that γ is true in σ, then we will infer that ψ is true in σ.

When ϕ is tautology, that is ϕ is always *true*, (5) becomes:

$$\psi \textbf{ with absence } \gamma, \tag{6}$$

This can be used to specify *default authorization* such as *closed world assumption*.

Another special form of the default proposition when the set γ is empty in (5). In this case, we rewrite (5) as

$$\phi \textbf{ provokes } \psi, \tag{7}$$

which is viewed as a *causal* or *inheritance* relation between ϕ and ψ. For example, in many situations, a system should satisfy the following relation:

$holds(s, Own, o)$ **provokes** $holds(s, Read, o) \wedge holds(s, Write, o)$.

Furthermore, when the set ϕ is empty in (7), we rewrite (7) as

$$\textbf{always } \psi, \tag{8}$$

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

always $holds(Root, a, o)$.

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

Example 8 The following is an example of domain description D specified by $\mathcal{L}^{sd}$:

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

3.2 Semantics of $\mathcal{L}^{sd}$

The semantics of $\mathcal{L}^{sd}$ is not just a simple extension of the semantics of $\mathcal{L}^{s}$ although the syntax of $\mathcal{L}^{sd}$ is a simple augmentation of that of $\mathcal{L}^{s}$. The reason is that to define a proper semantics of the default proposition (5), we have to employ a *fix-point semantics* that shares the spirit of fix-point semantics used in *logic programs* [17].

Given a domain description D, we first define the initial state of D. Suppose that a domain description D_p only contains initial policy propositions, default propositions of the special form (7) and transformation propositions.

Definition 2 *A state σ_0 is an* initial state *of D_p iff σ_0 is the smallest state that satisfies the following conditions:*

1. *for each initial policy proposition* **initially** ϕ, *ϕ is true in σ_0;*
2. *for each default proposition with the form ϕ* **provokes** *ψ, if ϕ is true in σ_0, then ψ is also true in σ_0.*

This definition describes how to get the initial state (initial policy base) from a domain description which only consists of initial policy propositions, default propositions of the special form (7) and transformation propositions. Condition 1 states that part of the facts of the initial state is obtained from these *initial policy propositions*, while Condition 2 states that part of the facts of the initial state is obtained from these *default propositions* ϕ **provokes** ψ. For these propositions, if ϕ is already in the initial state σ_0, ψ will also be included in σ_0. Since we require that the state is the smallest one which satisfy both conditions, this makes the initial state to be unique.

For instance, if the domain description is as follows:

initially a,
initially b,
initially c, and
c **provokes** f,
d **provokes** e.

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

Now we consider a domain description D containing default propositions with the general form (5). To define the initial state of D, we first translate D to domain description D_p described above.

Definition 3 *Let σ_0 be a state. Suppose domain description D_p is obtained from D as follows:*

1. *by deleting each default proposition ϕ* **implies** *ψ* **with absence** *γ from D if for every F_i in γ, F_i is true in σ_0*[2]*;*

[2] Recall that $\gamma \equiv F_1 \wedge \cdots \wedge F_i \wedge \cdots \wedge F_k$, each F_i $(1 \leq i \leq k)$ is a ground fact.

2. *by translating all other default propositions* ϕ **implies** ψ **with absence** γ *to the form* ϕ **provokes** ψ.

Now if this state σ_0 *is an initial state of* D_p*, then we also define it to be an* initial state *of* D.

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

Taking default propositions into account, it turns out that the initial state of a domain description may be not unique, or may not even exist. This is shown by the following example.

Example 9 A domain description D consists of the following propositions:

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

Clearly, D has two initial states:

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

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

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

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

Similarly to that defined in the language $\mathcal{L}^s$, we define that a *structure* of D to be a pair (σ, ρ), where σ is a state, and ρ is a transition function introduced in section 1.2. A policy proposition (1) *is satisfied* in a structure M, denoted as $M \models_{\mathcal{L}^{sd}} \phi$ **after** $T_1, \cdots, T_m$, if ϕ is true in state $M^{T_1,\cdots,T_m}$.

Now, we define the model of a domain description.

Definition 4 *Given a domain description* D*, let* (σ_0, ρ) *be a structure of* D*, where* σ_0 *is a consistent initial state of* D*, and* ρ *is a transition function introduced in section 1.2.* (σ_0, ρ) *is a* model *of* D *iff for any transformation* T *and state* σ*, the following conditions hold:*

1. *if* D *includes a transformation proposition* T **causes** ϕ **if** ψ*, and* ψ *is true in* σ*, then* ϕ *is true in* $\rho(T, \sigma)$*;*
2. *for each* F *in* D *and* $F \in \sigma$*,* $F \in \rho(T, \sigma)$ *iff* $\neg F \notin \rho(T, \sigma)$.

The above definition is similar to Definition 1, since after Definition 2 and 3, the initial state of a general domain description with *default proposition* is defined. So the model of a general domain description is defined the same way as the model of a domain description without *default proposition* does.

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

Example 10 A credit union divides its customers into two classes G_1 and G_2. The member of G_1 has a credit limit of up to \$5000. The member of G_2 has a credit limit of up to \$10000. The credit union reviews the credits of its customers and upgrade or downgrade their credit limits accordingly. A, B and C are three customers of this credit union. Suppose he information we know is that B belongs to G_2, A has a credit limit of up to \$5000 and C belongs to G_1 and has a credit limit of \$5000. The domain description D for this example is:

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

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

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

The initial state of D is:

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

Obviously, the following holds:

$D \models_{\mathcal{L}^{sd}} holds(A, Credit, \$10000) \wedge holds(B, Credit, \$10000) \wedge holds(C, Credit, \$5000) \wedge A \in G_2 \wedge B \in G_2 \wedge C \in G_1$ **after** $Upgrade(A)$,
$D \models_{\mathcal{L}^{sd}} holds(A, Credit, \$5000) \wedge holds(B, Credit, \$5000) \wedge holds(C, Credit, \$5000) \wedge A \in G_1 \wedge B \in G_1 \wedge C \in G_1$ **after** $Downgrade(B)$.

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

$D \models_{\mathcal{L}^{sd}} holds(A, Credit, \$10000) \wedge holds(B, Credit, \$5000) \wedge holds(C, Credit, \$5000) \wedge A \in G_2 \wedge B \in G_1 \wedge C \in G_1$ **after** $Upgrade(A), Downgrade(B)$.

3.3 Properties of Language $\mathcal{L}^{sd}$

Now we summerize the properties of Language $\mathcal{L}^{sd}$.

- *Incomplete information is still allowed.* Some authorizations may not be specified in the domain if they are not so relevant to the current situation. In example 8, neither $holds(S, Read, O)$ nor $\neg holds(S, Read, O)$ is stated in the domain.
- *Negations of authorization are specified explicitly.* Since incomplete information is permitted, negative authorizations must be specified clearly. If S cannot have $Write$ right on O, **initially** $\neg holds(S, Write, O)$ must be clearly specified in the domain.
- *Nonmonotonicity of the entailment relation.* From example 8 and 9, we can see that the transformations cause some authorizations no longer valid.
- *Inherited and causal authorization policies can be specified.* For instance, the policy that a subject can inherit an access right from its group's access right can be specified as:
  $holds(G, Access, O) \wedge S \in G$
  implies $holds(S, Access, O)$
  with absence $\neg holds(S, Access, O)$
  If for some reason, $holds(G, Access, O)$ is no linger a valid authorization, this will cause $holds(S, Access, O)$ invalid. This can be represented as:
  $\neg holds(G, Access, O)$ **provokes** $\neg holds(S, Access, O)$
- *Representation of default authorizations.* Default authorization can be properly specified now. The *closed world assumption* stating that any authorization ϕ which is not explicitly represented in the state will be assumed to be its negation is specified as:
  $\neg\phi$ **with absence** ϕ
- *Representation of constraint authorizations.* For instance, we can use **always** $G_1 \subseteq G$ to represent that G_1 is always a subgroup of G in the domain.

4 Background and Some Related Work

In a logical approach, authorizations are specified by using a high level independent semantics and are separated from their implementation in system specific mechanisms. Furthermore, with its precise syntax and semantics and expressive power, logical approaches in authorization specification have drawn a great interests of many researchers. Jajodia *et al* [19,20] proposed a logic language for expressing authorizations. Similarly to ours, they use predicates and rules to specify the authorizations; their work mainly emphasizes the representation and evaluation of authorizations, not authorization transformations. The work of Bertino *et al* [4,7] described an authorization mechanism based on a logic formalism. It mainly investigates the access control rules and their derivations. In their recent work [5], a formal approach based on C-Datalog language was presented for reasoning about access control models. The general framework can be used to model a large variety of access control models. Li *et al* [21] 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* [9] discussed security policy management using logic programming approach. Woo and Lam proposed a formal approach using default logic to represent and evaluate authorizations [25].

Our work described in this paper is different from most of the related works mentioned above. In summary, most of their works emphasize enchancing the expressive power of the authorization policies, their propagations, reasoning and delegations, while our work is to address the transformation of authorization policies. Our focus is on the policy state after a sequence of changes been performed. Though we all use logic-based approaches, the issues addressed are quite different.

In Woo and Lam's [25] logic-based approach for authorizations, the set of authorizations is specified by a policy base. The language used is a many-sorted first order language with a rule construct. The rule construct is similar to the default construct in default logic [23] but with a different semantics. In terms of evaluation, the policy based is translated to an extended logic program first, then use the idea of stable model to evaluate the extended logic program. Since among these works, Woo and Lam's approach is closest to ours in the sense of both having default reasoning features, we will study and compare our work with their's in the next section.

4.1 Woo and Lam's Approach: A Review

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

Basically, in Woo and Lam's system, a set of authorizations is specified by a policy base. The language used is a many sorted first order language with rule con-

struct[3]. The rule construct is similar to the default construct in default logic [23].

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

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

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

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

Example 11 A policy base B_1 is specified as:

$$\begin{aligned} B_1 = \{&: \neg holds(S, Write, O_1) \wedge holds(S, Write, O_2) \\ &/ holds(S, Write, O_3), \\ &holds(S, Read, O_1) \Rightarrow holds(S, Read, O_2), \\ &holds(S, Read, O_1)\}. \end{aligned}$$

These three rules in this policy base represent the following knowledge respectively: (i) if it is consistent to assume that subject S cannot write to object O_1 and can write to object O_2, then it is believed that S can write to object O_3; (ii) if S can read O_1, then S can read O_2 as well; (iii) subject S has *Read* right to object O_1.

The semantics of a policy base in Woo and Lam's approach is based on a concept called *extension*, which is actually an analogy of the fixed point semantics of Reiter's default logic [23]. Instead of describing Woo and Lam's original definition of extension, here we give its alternative which is based on Reiter's definition [23] but is semantically equivalent to Woo and Lam's original definition.

Definition 5 *Let B be a policy base and E be a set of formulas. We say that E is a* r-extension *(i.e. Reiter's extension) for B if it is one of the smallest deductively closed sets of formulas E' satisfying the condition: For any rule with the form $f : f'/g$ from B, if $f \in E'$ and $\neg f' \notin E$, then $g \in E'$. A set of ground literals Σ is called a* wl-extension *(i.e. Woo-Lam's extension) of B if and only if there exists a r-extension E of B such that Σ is the smallest set satisfying $F \in \Sigma$ for every formula $f \in E$. A ground literal l is* derivable *from B, denoted as $B \vdash_{wl} l$, if and only if l is in every wl-extension of B.*

Example 12 Example 11 continued. According to Definition 4, it is not difficult to see that the policy base B in Example 11 has a unique wl-extension:

$$\begin{aligned} \Sigma = \{&holds(S, Read, O_1), holds(S, Read, O_2), \\ &holds(S, Write, O_3)\}. \end{aligned}$$

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

Obviously, as Woo and Lam's policy base is based on Reiter's default logic, a policy base may have one or more than one or even no extension at all [25][5].

4.2 Comparison

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

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

initial policy proposition: **initially** ϕ,
default proposition: ϕ **implies** ψ **with absence** γ, or its special forms:

ϕ **provokes** ψ, and
always ψ.

Note that here all occurrences of ϕ, ψ and γ are *ground fact expressions* in our language because we assume that any fact expression including variables is viewed as a set of all its ground instances.

[3] Note that Woo and Lam's language is a restricted first order language as no quantifiers are considered in their language.

[4] Note that Woo and Lam's P^- and N^- which record those rights that cannot be explicitly granted and denied respectively cannot be expressed as fact expressions in our language. But it would not be difficult to extend our language to capture this capability. For instance, we can add one more predicate like $holds^-(s, a, o)$ in the language to achieve this purpose.

[5] This is similar to the situation of the state for a domain description in our language $\mathcal{L}^{sd}$.

From our previous discussion, we can assume that Woo and Lam's policy base B only includes ground rules[6]. Furthermore, for each rule $f : f'/g$ in the policy base B, we restrict f, f' and g to be in the form of ground fact expressions in our language $\mathcal{L}^{sd}$. This means that f, f' and g should only be of the form $[\neg]holds(S_1, A_1, O_1) \wedge \cdots \wedge [\neg]holds(S_k, A_k, O_k)$, where notation $[\neg]$ means that the negation sign "$\neg$" may or may not occur.

Under these assumptions, we have the following results to illustrate the relationship between these two approaches.

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

(i) for each initial policy proposition **initially** ϕ *in D, we specify a rule ϕ in B;*
(ii) for each default proposition ϕ **implies** ψ **with absence** γ *in D, we specify a rule $\phi : \neg\gamma/\psi$ in B;*
(iii) for each special default proposition ϕ **provokes** ψ*, we specify a rule $\phi \Rightarrow \psi$ in B;*
(iv) for each special default proposition **always** ψ*, we specify a rule ψ in B.*

Then for any ground fact expression ψ, $D \models_{\mathcal{L}^{sd}}$ **initially** ψ *if and only if $B \vdash_{wl} \psi$.*

Proof Under the specification defined in Proposition 2, we actually translate a static domain description D into an equivalent Reiter's default theory B [23]. On the other hand, from Definition 4, it can be shown that the model of our domain description is coincident with the answer set of extended logic program [17]. Then from the correspondence between the answer set of an extended logic program and the extension of a default theory, as proved in [17], the result holds.

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

(i) for each rule $f : f'/g$ in B, we specify a default proposition f **implies** g **with absence** $\neg f'_1 \wedge \cdots \wedge f'_k$*, where $f' \equiv f'_1 \wedge \cdots \wedge f'_k$;*
(ii) for each rule g in B, we specify an initial policy proposition **initially** g*;*
(iii) for each rule $f \Rightarrow g$ in B, we specify a special default proposition f **provokes** g *in D.*

Then for each ground fact expression ψ, $B \vdash_{wl} \psi$ if and only if $D \models_{\mathcal{L}^{sd}}$ **initially** ψ.

Proof The proof is similar to that of Proposition 2.

[6] Recall that a rule including ordinary variables can be viewed as a set of all instances of this rule.

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

5 Conclusions

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

In [11,12], an access control system has been implemented with policy evaluation and dynamic policy update. It is realized by first translating the policy base which is specified by a logic-based language to a logic program, then using stable model semantics [17] to evaluation the underlying logic program. A related work using logic programming for conflict resolution in reasoning has also been implemented in [26]. It is our future work to use logic programming (stable model semantics) to implement language $\mathcal{L}^{sd}$ presented in this paper.

References

1. V. Atluri and A. Gal, An Authorization Model for Temporal and Derived Data: Securing Information Protals, *ACM Transactions on Information and System Security*, Vol.5, No.1, pp62–94, 2002.
2. Y. Bai and V. Varadharajan, Object Oriented Database with Authorization Policies, *Journal of fundamenta Informaticae*, Vol.53, Nos.3-4, pp229-250, 2002.

3. Y. Bai and V. Varadharajan, On Transformation of Authorization Policies, *Data and Knowledge Engineering*, Vol.45, No.3, pp333-357, 2003.
4. 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.
5. 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.
6. E. Bertino, S. Jajodia and P. Samarati, Supporting Multiple Access Control Policies in Database Systems, *Proceedings of IEEE Symposium on Research in Security and Privacy*, pp94–107, 1996.
7. E. Bertino, A. Mileo and A. Provetti, Policy Monitoring with User-preferences in PDL, *Proceedings of IJCAI-03 Workshop for Nonmonotonic Reasoning, Action and Change*, pp37–44, 2003.
8. D.F.C. Brewer and M.J. Nash, The Chinese Wall Security Policy, *Proceedings of IEEE Symposium on Research in Security and Privacy*, pp215–228, 1989.
9. 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.
10. T.S.C. Chou and M. Winslett, Immortal: A Model-based Belief Revision System, *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning*, pp99–110, 1991.
11. V. Crescini and Y. Zhang, Web Server Authorization with Policy Updater: A Logical Based Access Control System, To appear in the *Proceedings of IADIS International Conference on WWW/Internet*, 2004.
12. V. Crescini and Y. Zhang, A Logical Based Approach for Dynamic Access Control, To appear in the *Proceedings of 17th Australian Joint Conference on Artificial Intelligence*, 2004.
13. M. Dacier and Y. Deswarte, Privilege Graph: an Extension to the Typed Access Matrix Model, *Proceedings of European Symposium on Research in Computer Security*, pp319–334, 1994.
14. D.E. Denning, A Lattice Model of Secure Information Flow, *Communication of ACM*, vol.19, pp236–243, 1976.
15. E.B. Fernandez, E. Gudes and H. Song, A Security Model For Object-Oriented Databases, *Proceedings of IEEE Symposium on Research in Security and Privacy*, pp110–115, 1989.
16. E.B. Fernandez, R.B. France and D. Wei, A Formal Specification of an Authorization Model for Object-oriented Databases, *Database Security, IX: Status and Prospects*, pp95–109, 1995.
17. M. Gelfond and V. Lifschitz, Classical Negation in Logic Programs and Disjunctive Databases, *New Generation Computing*, vol.9, pp365–385, 1991.
18. L. Gong, A Secure Identity Based Capability System, *Proceedings of IEEE Symposium on Research in Security and Privacy*, pp56–63, 1989.
19. 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.
20. S. Jajodia, P. Samarati and V.S. Subrahmanian, A Logical Language for Expressing Authorizations, *Proceedings of IEEE Symposium on Research in Security and Privacy*, pp31–42, 1997.
21. 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.
22. C. Meadows, Policies for Dynamic Upgrading, *Database Security, IV: Status and Prospects*, pp241–250, 1991.
23. R. Reiter, A Logic for Default Reasoning, *Artificial Intelligence*, vol. 13, pp81–132, 1980.
24. R.S. Sandhu and S. Ganta, On the Minimality of Testing for Rights in Transformation Models, *Proceedings of IEEE Symposium on Research in Security and Privacy*, pp230–241, 1994.
25. 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.
26. Y. Zhang, C.M. Wu and Y. Bai Implementing Prioritized Logic Programming, *AI Communications*, Vol.14, No. 4, pp183–196, 2001.