**RC 21492 (96966) 28 May 1999**
**Computer Science**

# IBM Research Report

## A Logic-based Knowledge Representation for Authorization with Delegation

Ninghui Li
Computer Science, New York University
251 Mercer Street, New York, NY 10012, USA
Internet e-mail: `ninghui@cs.nyu.edu`
Web: `http://www.cs.nyu.edu/~ninghui`

Benjamin N. Grosof
IBM Research Division, T.J. Watson Research Center
P.O. Box 704, Yorktown Heights, NY 10598, USA
Internet e-mail: `grosof@us.ibm.com` (alt. `grosof@cs.stanford.edu`)
Web: `http://www.research.ibm.com/people/g/grosof/`

Joan Feigenbaum
AT&T Labs – Research
Room C203, 180 Park Avenue, Florham Park, NJ 07932, USA
Internet e-mail: `jf@research.att.com`
Web: `http://www.research.att.com/~jf`



IBM Research Division
**Almaden · Austin · Beijing · Delhi · Haifa · T.J. Watson · Tokyo · Zurich**

**Abstract**: We introduce Delegation Logic (DL), a logic-based knowledge representation (*i.e.*, language) that deals with *authorization* in large-scale, open, distributed systems. Of central importance in any system for deciding whether requests should be authorized in such a system are *delegation* of authority, *negation* of authority, and *conflicts* between authorities. DL's approach to these issues and to the interplay among them borrows from previous work on delegation and trust management in the computer-security literature and previous work on negation and conflict handling in the logic-programming and non-monotonic reasoning literature, but it departs from previous work in some crucial ways. In this introductory paper, we present the syntax and semantics of DL and explain our novel design choices. This first paper focuses on delegation, including explicit treatment of delegation depth and delegation to complex principals; a forthcoming companion paper focuses on negation.

Compared to previous logic-based approaches to authorization, DL provides a novel combination of features: it is based on logic programs, expresses delegation depth explicitly, and supports a wide variety of complex principals (including but not limited to $k$-out-of-$n$ thresholds). Compared to previous approaches to trust management, DL provides another novel feature: a concept of proof-of-compliance that is not entirely *ad-hoc* and that is based on model-theoretic semantics (just as usual logic programs have a model-theoretic semantics). DL's approach is also novel in that it combines the above features with smooth extensibility to non-monotonicity, negation, and prioritized conflict handling. This extensibility is accomplished by building on the well-understood foundation of DL's logic-program knowledge representation.

**Keywords**: Authorization, delegation, trust management, security policy, non-monotonicity, conflict handling, knowledge representation, logic programs.

**Publication and Copyright Information**: This paper (formatted differently) is an extended version of the paper to appear in the Proceedings of the IEEE Computer Security Foundations Workshop (CSFW-12), `http://www2.csl.sri.com/csfw/index.html`, to be held June 28–30, 1999, in Mordano, Italy. The Limited Distribution Notice on the front page notwithstanding (it is standard boilerplate for all IBM Research Reports), the author(s) and IBM retain unrestricted copyright to distribute this paper.

# 1 Introduction

In today's Internet, there are a large and growing number of scenarios that require authorization decisions. By an *authorization* decision, we mean one in which one party submits a *request*, possibly supported by one or more *credentials*, that must comply with another party's *policy* if it is to be granted. Scenarios that require authorization decisions include content advising [25], mobile-code execution [11], public-key infrastructure [6, 29, 18, 9, 26], and privacy protection [22, 20].

Electronic commerce is one class of services in which authorization decisions play a prominent role. Merchants and customers both have valuable resources at risk and must have appropriate policies in place before authorizing access to these resources. An interesting aspect of e-commerce is that security policies and business policies are not always clearly separable. If a merchant requires that electronic checks for more than a certain amount be signed by at least two members of a set of trusted parties, is that a "security policy" or a "business policy"? It would be desirable for one authorization mechanism to be able to handle both.

Authorization in Internet services is significantly different from authorization in centralized systems or even in distributed systems that are closed or relatively small. In these older settings, authorization of a request is traditionally divided into two tasks: *authentication* and *access control*. Authentication answers the question "who made the request?," and access control answers the question "is the requester authorized to perform the requested action?" Following the "trust-management approach," first put forth by Blaze *et al.* [4, 5], we argue that this traditional view of authorization is inadequate. Reasons include:

- **What to protect?**: In a traditional client/server computing environment, valuable resources usually belong to servers, and it is when a client requests access to a valuable resource that the server uses an authorization procedure to decide whether or not to trust the client. In today's Internet (or any large, open, distributed system), users access many servers, make many different types of requests, and have valuable resources of their own (*e.g.*, personal information, electronic cash); indeed "client" is no longer the right metaphor. Such a user cannot trust all of the servers it interacts with, and authorization mechanisms have to protect the users' resources as well as those of the servers.

- **Whom to protect against?**: In a large, far-flung network, there are many more potential requesters than there are in a smaller, more homogeneous (albeit distributed) system. Some services, *e.g.*, Internet merchants, cannot know in advance who the potential requesters are. Similarly, users cannot know in advance which services they will want to use and which requests they will make. Thus, authorization mechanisms must rely on delegation and on third-party credential-issuers more than ever before.

- **Who stores authorization information?**: Traditionally, authorization information, *e.g.*, an access control list, is stored and managed by the service. Internet services evolve rapidly, and thus the set of potential actions and the users who may request them are not known in advance; this implies that authorization information will be created, stored, and managed in a dynamic, distributed fashion. Users are often expected to gather all credentials needed to authorize an action and present them along with the request. Since these credentials are not always under the control of the service that makes the authorization decision, there is a danger that they could be altered or stolen. Thus, public-key signatures (or, more generally, mechanisms for verifying the provenance of credentials) must be part of the authorization framework.

For these and other reasons, dividing authorization into authentication and access control is no longer appropriate. "Who made this request?" may not be a meaningful question – the authorizer may not even know the requester, and thus the identity or name of the requester may not help in the authorization decision. The goal of a growing body of work on *trust management* [4, 5, 9, 7, 3] is to find a more flexible, more "distributed" approach to authorization. The trust-management literature approaches the basic authorization question directly: "Does the set $C$ of *credentials* prove that the *request* $r$ *complies* with the set of local security *policies* $P$?" The *trust-management engine* is a separate system component that takes $(r, C, P)$ as input and outputs a decision about whether compliance with policy has been proven.

Furthermore, trust-management adopts a "peer model" of authorization. Every entity can be both a requester and an authorizer. To be an authorizer, it maintains policies and is the ultimate source of authority for its authorization decisions. As a requester, it must maintain credentials (*e.g.*, public-key certificates, credit card numbers, and membership certificates) or be prepared to retrieve or obtain them when it wants access to a protected resource. When submitting a request to an authorizer, the requester also submits a set of credentials that purport to justify that the requested action is permissible. An authorizer may directly authorize certain requesters to take certain actions (and may not even try to "authenticate" these requesters by resolving their "identities"), but more typically it will *delegate* this responsibility to credential issuers that it trusts to have the required domain expertise as well as relationships with potential requesters.

Basic issues that must be addressed in the design of a trust-management engine include the definition of "proof of compliance," the extent to which policies and credentials should be programmable, and the language or notation in which they should be expressed.

In this paper, we propose the authorization language Delegation Logic (DL) as a trust-management engine. Its notable features include:

- A definition of "proof of compliance" that is founded on well-understood principles of logic programming and knowledge representation. Specifically, DL starts with the notion of proof embodied in Datalog definite ordinary logic programs [19].[1] DL then extends this with several features tailored to authorization.

- A rigorous and expressive treatment of delegation, including explicit linguistic support for delegation depth and for a wide variety of complex principals.

- The ability to handle "non-monotonic" policies. These are policies that deal explicitly with "negative evidence" and specify types of requests that do *not* comply. Important

[1]For review of standard concepts and results in logic programming, see [2], for example. "Ordinary" logic programs (LP's) correspond essentially to pure Prolog, but without the limitation to Prolog's particular inferencing procedure. These are also known as "general" LP's (a misleading name, since there are many further generalizations of them) and as "normal" LP's. "Definite" means without negation. "Datalog" means without function symbols of more than zero arity. "Arity" means number of parameters.

examples include hot-lists of "revoked" credentials and resolution of conflicting advice from different, but apparently both trustworthy, sources.

"Non-monotonic" here means in the sense of logic-based knowledge representation (KR).[2]

In combining both of these properties, DL departs sharply from earlier trust-management engines, some key points of which we now review. PolicyMaker, which was introduced in [4] and was the first system to call itself a "trust-management engine," uses an *ad-hoc* (albeit rigorously analyzed [5]) notion of "proof of compliance" and handles only monotonic policies. KeyNote [3] is a second-generation system based on most, but not all, of the same design principles as PolicyMaker; in particular, KeyNote uses an *ad-hoc* notion of proof of compliance (derived from the one used in PolicyMaker), and it does not handle non-monotonic policies. Unlike PolicyMaker, KeyNote takes an integrated approach to the design of the compliance-checking algorithm and the design of the programming language in which credentials and policies are expressed. DL also takes an integrated approach to these two aspects of authorization. REFEREE [7] handles nomonotonic policies, but it uses an *ad hoc* proof system that was never rigorously analyzed. SPKI [9] handles limited forms of non-monotonicity, but the "proof of compliance" notion (to the extent that one is specified in [9]) is *ad-hoc*.

The outline of the rest of the paper is as follows. In section 2, we give an overview of DL. In section 3, we give the syntax and semantics of the monotonic case of DL, called D1LP. In section 4, we give an example of D1LP's usage. In section 5, we give an overview of our expressive extension to handle negation and prioritized conflict, called D2LP. A forthcoming companion paper gives details about D2LP. In section 6, we briefly discuss related work and future work.

# 2 Overview of DL

Our use of a logic-program knowledge representation as the foundation of our authorization language (a.k.a. "trust-management engine") offers several attractions: computational tractability[3], wide practical deployment, semantics shared with other practically important rule systems, relative algorithmic simplicity, yet considerable expressive power.

We chose Datalog definite ordinary logic programs (OLP's) as the starting point for DL. (More generally, however, we could started from other variants of logic-based knowledge representation, e.g., OLP's without the Datalog restriction.) DL extends Datalog definite OLP's along two dimensions that are crucial to authorization: delegation and non-monotonic reasoning. The resulting notion of "proof of compliance" is easier to justify than the *ad-hoc*

[2] A KR $\mathcal{K}$ is (logically) *monotonic* when its entailment relationship (*i.e.*, what it sanctions as conclusions) has the following property: if the set of premises (e.g., rules) $\mathcal{P}_2$ is a superset of the set of premises $\mathcal{P}_1$, then the set of conclusions entailed by $\mathcal{P}_2$ according to $\mathcal{K}$ is a superset of the set of conclusions entailed by $\mathcal{P}_1$. If a KR is not monotonic, it is called *non-monotonic*. Non-monotonicity means that adding premises can lead to retracting previously-sanctioned conclusions.

[3] Under commonly met restrictions (*e.g.*, no logical functions of non-zero arity, a bounded number of logical variables per rule), inferencing, *i.e.*, rule-set execution, in LPs can be computed in worst-case polynomial-time. By contrast, classical logic (*e.g.*, first-order logic), is NP-complete under these restrictions and semi-decidable without these restrictions.

notions used in PolicyMaker, KeyNote, REFEREE, and SPKI, because it is an extension of the well-studied, logic-programming framework.

As in much of the related literature, *e.g.,* [1, 21, 27], we use the term *principal* to mean an "entity" or "party" to an authorization decision. For example, a principal may make a request, issue a credential, or make a decision. Each authorization decision must involve a distinguished principal that functions as the "trust root" of the decision; this principal is referred to as `Local`.[4] DL supports the specification of sets of principals, via thresholds and lists, as well as *dynamic* sets of the form "all principals that satisfy the following predicate." DL principals express beliefs by making *direct statements* and *delegation statements*.

The DL framework provides a uniform representation for requests, policies, and credentials. Information in DL is represented as rules and facts that are built out of statement expressions. A request in DL corresponds to a query. *E.g.*, a simple query might be to ask whether the ground statement `"Local says is_key(12345,Bob)"` is true. More generally, a request can be a complex expression of statements; these expressions are called *statement formulas* and are defined in the next section. All of the policies and credentials that the receiving principal uses in evaluating the request form a DL program $P$. The DL semantics defines a unique minimal model for $P$, and the request is authorized if and only if it is in this model. The DL semantics provide the definition of "proof of compliance." This use of model-theoretic semantics is a novel feature of DL and a clear departure from the approaches taken by other trust-management engines.

Delegation is one of the two major concepts with which we extended Datalog definite OLP's to form DL, and it is the main technical focus of this paper. Distinguishing features of DL's approach to delegation include:

- Delegations have arbitrary but specified depth. For example, by using a depth-2 delegation statement, a principal $A$ may delegate trust about a certain class of actions to principal $B$ and allow $B$ to delegate to others but *not* allow these others to delegate further.

- Delegations to complex principal structures are allowed. For example, a principal $A$ may delegate trust about a certain class of purchases to all principals that satisfy the predicate `GoodTaste()`.

The other major concept that we added to Datalog definite OLP's to form DL is non-monotonicity. DL uses explicit negation to allow a policy to say what is forbidden, negation-as-failure to allow a policy to draw conclusions when there is no information about something, and priorities to handle conflicts among policies.

We use DL to denote our general approach to trust management. The monotonic version of DL (*i.e.*, Datalog definite OLP's plus our delegation mechanism) is called D1LP, and the non-monotonic version (*i.e.*, with negation and prioritized conflict handling) is called D2LP. This first paper focuses on D1LP, and only gives an overview of D2LP; a forthcoming companion paper focuses on D2LP.

Compared to previous logic-based approaches to authorization, DL provides a novel combination of features: it is based on logic programs, expresses delegation depth explicitly, and

[4] `Local` plays the role that `POLICY` plays in PolicyMaker.

supports a wide variety of complex principals (including but not limited to $k$-out-of-$n$ thresholds). Compared to previous approaches to trust management, DL provides another novel feature: a concept of proof-of-compliance that is not entirely *ad-hoc* and that is based on model-theoretic semantics (just as usual logic programs have a model-theoretic semantics). DL's approach is also novel in that it combines the above features with smooth extensibility to non-monotonicity, negation, and prioritized conflict handling. This extensibility is accomplished by building on the well-understood foundation of DL's logic-program knowledge representation.

# 3 Syntax and Semantics of D1LP

In this section, we formally define D1LP's syntax and semantics.

## 3.1 Syntax

1. The *alphabet* of D1LP consists of three disjoint sets, the *constants*, the *variables*, and the *predicate symbols*. The set of *principals* is a subset of the constants and the set of *principal variables* is a subset of the variables. Variables start with '_' ("underscore").[5] The special variable symbol '_' means a new variable whose name doesn't matter. A *term* is either a variable or a constant. Note that we prohibit function symbols with non-zero arity: this is the *Datalog* restriction. This restriction helps enable finiteness of the semantics and of computing inferences (a.k.a. entailments).

2. A *base atom* is an expression of the form

$$pred(t_1, ..., t_n)$$

where *pred* is a predicate symbol and each $t_i$ is a term.

3. A *direct statement* is an expression of the form

$$X \texttt{ says } p$$

where $X$ is either a principal or a principal variable, "`says`" is a keyword, and $p$ is a base atom. $X$ is called the *subject* of this direct statement. A base atom encodes a trust belief or a security action, and a direct statement represents a belief of the subject.

4. A *threshold structure* takes one of the following forms:

   - $\texttt{threshold}(k,\ \{(A_1, w_1), \ldots, (A_n, w_n)\})$

     where "`threshold`" is a keyword, $k$ and the $w_i$'s are positive integers, the $A_i$'s are principals, and $A_i \neq A_j$ for $i \neq j$. The $w_i$'s are called *weights*. The set

$$\{(A_1, w_1), \ldots, (A_n, w_n)\}$$

[5]In Prolog, variables can also start with upper-case letters, and all constants start with lower-case letters. We want to allow constants to start with upper-case letters, and we restrict variables to start with underscore.

is called a *principal-weight pair set* (abbreviated *P-W set*). If $w_i = 1$, then $(A_i, w_i)$ can be written as $A_i$. A threshold structure supports something if the sum of all the weights of those principals that support it is greater than or equal to $k$.

- `threshold`($k$, $Prin$ `says` $pred/x$)

  where "`threshold`" and $k$ are the same as above, $Prin$ is a principal, $pred$ is a predicate symbol, and $x$ is the arity (number of parameters) of $pred$. The arity $x$ must be either 1 or 2. ($x$ is *not* a logical variable.) When $x = 1$, "$Prin$ `says` $pred/1$" defines a P-W set that gives weight 1 to all principals $A$ such that "$Prin$ `says` $pred(A)$" is true. When $x = 2$, "$Prin$ `says` $pred/2$" defines a P-W set, where the corresponding weight for any principal $A$ is the greatest positive integer $w$ such that "$Prin$ `says` $pred(A, w)$" is true. These are called *dynamic threshold structures*.

5. A *principal structure* takes one of the following forms:

   - $A$ where $A$ is a principal
   - $TS$ where $TS$ is a threshold structure
   - $PS_1, PS_2$ where $PS_1$ and $PS_2$ are principal structures. This is the conjunction of two principal structures. If both $PS_1$ and $PS_2$ support a base atom $p$, then $PS_1, PS_2$ also supports $p$.
   - $PS_1; PS_2$ where $PS_1$ and $PS_2$ are principal structures. This is the disjunction of two principal structures. If either $PS_1$ or $PS_2$ supports a base atom $p$, then $PS_1; PS_2$ also supports $p$.
   - $\{PS\}$ where $PS$ is a principal structure

   In a principal structure, conjunction(',') takes precedence over disjunction(';'). A *principal list* is the special case of a principal structure that has the form $\{A_1, \ldots, A_n\}$, where each $A_i$ is a principal. A *principal set* is the special case of a principal list in which there are no repetitions, *i.e.*, in which $A_i \neq A_j$ for $i \neq j$.

6. A *delegation statement* takes the form

   $X$ `delegates` $p$^$d$ `to` $PS$

   where X is either a principal or a principal variable, `delegates` and `to` are keywords, $p$ is a base atom, $d$ is either a positive integer or the asterisk symbol '$*$', and $PS$ is a principal structure. $X$ is called the *subject*, $d$ is called the *delegation depth*, and $PS$ is called the *delegatee*. For example,

```
Alice delegates is_key(_,_)^2 to Bob
```

is a delegation statement. Intuitively, it means:

```
Alice says is_key(_Key_X,_X)  if  Bob says is_key(_Key_X,_X).
```

In this example, Alice trusts Bob in making direct statements about the predicate `is_key`. Alice may also trust Bob in judging other people's ability to make direct statements about `is_key`, *i.e.*, Alice trusts anyone Bob trusts. In this case, the delegation depth is 2. Similarly, delegation depth can also be greater than 2. A delegation depth '$*$' means unlimited depth.

7. A *statement* is either a direct statement or a delegation statement. In the semantics of D1LP, the role of "statement" is similar to the role of "atom" in ordinary LP's.

8. A *statement formula* takes one of the following forms:

   - $S$ where $S$ is a statement.
   - $F_1, F_2$ meaning ($F_1$ and $F_2$), where $F_1$ and $F_2$ are statement formulas,
   - $F_1; F_2$ meaning ($F_1$ or $F_2$), where $F_1$ and $F_2$ are statement formulas,
   - $(F)$ where $F$ is a statement formula.

   In a statement formula, the operator ',' (and) takes precedence over the operator ';' (or).

9. A *clause*, also known as a *rule*, takes the form:

$$S \quad \texttt{if} \quad F.$$

   where $S$ is a statement and $F$ is a statement formula in which no dynamic threshold structures appear. $S$ is called the *head* of the clause, and $F$ is called the *body* of the clause. The body may be empty; if it is, the "`if`" part of the clause may be omitted. A clause with an empty body is also called a *fact*. Permitting dynamic threshold structures in the body in effect introduces logical non-monotonicity, which is why we prohibit it in D1LP. However, when we introduce negation-as-failure in D2LP, this restriction will be dropped.

   There are two special principals that can be used in the body of a clause: "`I`" and "`Local`". "`I`" refers to the subject of the head. It is the default subject for all statements in the body and may optionally be omitted. For example, when Alice believes

```
Bob says p if  q,  I says r.
```

   this is shorthand for Alice believing

```
Bob says p if  Bob says q,  Bob says r.
```

   "`Local`" refers to the principal that is using this statement and trying to make an authorization decision, *i.e.*, the current trust root. For example, when Alice believes "`Bob says p if Local says q.`", and Alice believes (that `Alice says`) "`q`", then Alice can conclude that "`Bob says p.`"

Multi-agent logics of belief (or of knowledge) express beliefs from the viewpoints of multiple agents. DL can be viewed as expressing beliefs from the viewpoints of multiple agents. However, in DL, there is a single, distinguished viewpoint: that of the principal `Local`. Every DL rule or statement is implicitly regarded as a belief of `Local`. In other words, DL is *used* from one principal's viewpoint: *i.e.*, `Local`'s. Let `Local` be the agent Alice. When Alice believes

```
Bob says is_key(Key_M, M)  if  CA says is_key(Key_M, M).
```

this means that "If I (Alice) believe that `CA says is_key(Key_M, M)`, then I (Alice) can believe that `Bob says is_key(Key_M, M)`." The direct statements "`CA says is_key(Key_M, M)`" and "`Bob says is_key(Key_M, M)`" actually mean "Alice believes `CA says is_key(Key_M, M)`" and "Alice believes `Bob says is_key(Key_M, M).`" It doesn't matter whether "`Bob says is_key(Key_M, M)`" is believed by other principals, even Bob himself.

10. A *program* is a finite set of clauses. This is also known as a *logic program (LP)* or as a *rule set*.

As usual, an expression (*e.g.*, term, base atom, statement, clause, or program) is called *ground* if it does not contain any variables.

## 3.2 Semantics

In this subsection, we define the set of statements that are sanctioned as conclusions by a D1LP. Formally, this set of conclusions is defined as the *minimal model* of the D1LP. This model assigns a truth value ($true$ or $false$) to each ground statement. The value $true$ means that the statement is an entailed conclusion; $false$ means that it is not an entailed conclusion. These conclusions represent the beliefs of the principal that is the trust root, *i.e.*, `Local`.

Let $\mathcal{P}$ be a given D1LP.

Our semantics is defined via a series of steps. First, we define a language $\mathcal{LO}_\mathcal{P}$ that expresses definite OLP's (definite logic programs [19]). By contrast, we write the original input (D1LP) language of $\mathcal{P}$ as $\mathcal{LI}_\mathcal{P}$. Second, we define a translation that maps $\mathcal{P}$ to a ground definite OLP $\mathcal{O}$ in $\mathcal{LO}_\mathcal{P}$. Third, we define the minimal model of $\mathcal{P}$ as the correspondent (under this translation) of $\mathcal{O}$'s minimal model in the usual OLP semantics [19].

We begin by defining $\mathcal{LO}_\mathcal{P}$. Let $N$ be the number of all principals in $\mathcal{P}$, $MaxDepth$ be the greatest integer used as a delegation depth in $\mathcal{P}$, and $MaxArity$ be the maximum arity of any predicate in $\mathcal{P}$. The language $\mathcal{LO}_\mathcal{P}$ has two predicates: *holds* and *delegates*. The predicate *holds* takes four parameters:

$$holds(subject, pred, [params], length)$$

Here, the domain of $subject$ is the set of all principals appearing in $\mathcal{P}$, which we write as $Principals$. The domain of $pred$ is the set of all the predicate symbols appearing in $\mathcal{P}$. The domain of $[params]$ is all the length-$l$ lists of constants that appear in $\mathcal{P}$, where $0 \le l \le MaxArity$. The domain of $length$ is integers $[1..N]$.

A ground atom of the predicate *holds* represents a ground direct statement

$$\textit{subject}\ \texttt{says}\ \textit{pred}(\textit{params})$$

Intuitively, *length* represents the number of delegation steps that is enough to derive the corresponding direct statement. When $length = 1$, it means this direct statement can be derived directly without the use of delegation. We need not consider cases in which this length exceeds the number $N$ of principals.

The predicate *delegates* takes six parameters:

$$delegates(subject, pred, [params], depth, delegatee, length)$$

Here, *subject*, *pred*, *params*, and *length* are as above. The domain of *depth* is $[1..MaxDepth] \cup \{*\}$. The domain of *delegatee* is the set of all principal sets, which we write as $2^{Principals}$. Recall that there are $N$ principals altogether, and thus $2^{Principals}$ is finite. Notice that only principal sets, rather than more general principal structures, are permitted as *delegatee* here. The reason this suffices to represent $\mathcal{P}$ will become clear soon.

A ground atom of the predicate *delegates* represents a delegation statement

$$\textit{subject}\ \texttt{delegates}\ \textit{pred}(\textit{params})\hat{\ }\textit{depth}\ \texttt{to}\ \textit{delegatee}$$

The *Herbrand base* of $\mathcal{LO_P}$ is the set of all ground atoms in $\mathcal{LO_P}$. Because all the domains used above are finite, the Herbrand base of $\mathcal{LO_P}$ is also finite. An *interpretation* of $\mathcal{LO_P}$ is an assignment of truth values (*true* and *false*) to the Herbrand base of $\mathcal{LO_P}$. Such an interpretation can also be viewed as a set of true ground atoms, *i.e.*, as a *conclusion set*.

Given an interpretation $I$ of $\mathcal{LO_P}$ and a principal structure $PS$, we define $PS^I$, the *normal form of a principal structure* under (*i.e.*, relative to) $I$, as follows. A principal structure $PS^I$ is in normal form when it is of the form: "$PS_1^I; PS_2^I; ...; PS_r^I$," where each $PS_i^I$ is a principal set and, for any $i \neq j$, $PS_i^I \not\subseteq PS_j^I$. One can view $PS$ as a negation-free formula in propositional logic; $PS$'s normal form $PS^I$ is then the result of converting that propositional-logic formula into its reduced disjunctive normal form (DNF). Here, the reduced DNF is logically equivalent (in propositional logic) to $PS$ given $I$. "Reduced" means that there is no subsumption: neither within a conjunct (*i.e.*, no repetitions of principals) nor between conjuncts (*i.e.*, no conjunct is a subset of another conjunct). For dynamic threshold structures like "$\texttt{threshold}(k,\ Prin\ \texttt{says}\ pred/x)$," the interpretation $I$ determines the P-W set defined by $Prin$ and $pred$. A threshold structure

$$\texttt{threshold}(k,\ \{(A_1, w_1), (A_2, w_2), \ldots, (A_n, w_n)\})$$

is converted to the disjunction of all minimal subsets of $\{A_1, \ldots, A_n\}$ whose corresponding weights sum to be greater than or equal to $k$. For example,

$$\texttt{threshold}(3, \{(A, 2), (B, 1), (C, 1), (D, 1)\})$$

is converted to

$$\{A, B\}; \{A, C\}; \{A, D\}; \{B, C, D\}.$$

After two principal structures have been transformed, their conjunction and disjunction are convertible to normal form using methods similar to the usual ones used in propositional logic.

Equipped with the definitions of $\mathcal{LO}_{\mathcal{P}}$ and $PS^I$, we are ready next to give the main definition of the translation. Given an interpretation $I$ of $\mathcal{LO}_{\mathcal{P}}$, the translation $Trans^I$ maps $\mathcal{P}$ into a definite OLP $\mathcal{O}^I$ in the language $\mathcal{LO}_{\mathcal{P}}$, *i.e.*, $\mathcal{O}^I = Trans^I(\mathcal{P})$. We define $Trans^I$ via four steps.

Semantically, we treat a rule containing variables as shorthand for the set of all its ground instances. This is standard in the logic programming literature. We write $\mathcal{P}^{instd}$ to stand for the LP that results when each rule $r$ in $\mathcal{P}$ is replaced by the set of all its possible *ground instantiations*, *i.e.*, by all of the ground clauses that can be obtained by replacing $r$'s variables with constants (or "instantiating" them).

The first step of $Trans^I$ is to replace $\mathcal{P}$ by $\mathcal{P}^{instd}$.

The second step of $Trans^I$ is to replace all the delegation statements in $\mathcal{P}^{instd}$ by those that delegate to principal sets, as follows. Let $PS^I$ be written as $PS^I_1; PS^I_2; ...; PS^I_r$; each $PS^I_i$ is a principal set.

- Rewrite head delegation statements.
  Replace every clause of the form
  $A$ `delegates` $s\hat{}d$ `to` $PS$ `if` $F$.
  by the $r$ clauses:
  $\{ A$ `delegates` $s\hat{}d$ `to` $PS^I_i$ `if` $F. \mid i = 1..r \}$.

- Rewrite body delegation statements.
  Replace every delegation statement
  $A$ `delegates` $s\hat{}d$ `to` $PS$
  that occurs in the body of a clause, by the conjunction of the $r$ delegation statements:
  $A$ `delegates` $s\hat{}d$ `to` $PS^I_1$,
  $A$ `delegates` $s\hat{}d$ `to` $PS^I_2$,
  $\ldots$,
  $A$ `delegates` $s\hat{}d$ `to` $PS^I_r$.

Let $T^I_2$ denote the program after the above transformations.

The third step of $Trans^I$ takes $T^I_2$ as input and translates it to an OLP $T^I_3$ in the language $\mathcal{LO}_{\mathcal{P}}$, as follows.

- For any direct statement: $A$ `says` $pred(params)$
  in the body of a clause, change it to: $holds(A, pred, [params], N)$.

- For any direct statement: $A$ `says` $pred(params)$
  in the head of a clause, change it to: $holds(A, pred, [params], 1)$.

- For any delegation statement: $A$ `delegates` $pred(params)\hat{}d$ `to` $PS$
  in the body of a clause, change it to: $delegates(A, pred, [params], d, PS, N)$

- For any delegation statement: $A$ `delegates` $pred(params)\hat{}d$ `to` $PS$
  in the head of a clause, change it to: $delegates(A, pred, [params], d, PS, 1)$.

Here, as before, $N$ is the number of principals in $\mathcal{P}$. Intuitively, if a statement in the head is deduced, then it is deduced directly (length 1); if a statement can be deduced within any length, it is true.

The result of these changes is $T_3^I$.

The fourth step of $Trans^I$ is to add a further collection of clauses to $T_3^I$; the resulting OLP program is $\mathcal{O}^I$ in the language $\mathcal{LO}_\mathcal{P}$. Intuitively, these additional clauses represent all instances of several *meta-rules of deduction involving delegation.* Because the relevant domains are finite, there are a finite number of such instances.

Let $A$ be a principal; $BS$ be a principal set $\{B_1, \ldots, B_n\}$; $CS$ be another principal set; $d$, $d_0$, and $d_1$ be delegation depths (*i.e.,* elements of $[1..MaxDepth] \cup \{*\}$); and, $l$, $l_0$, and $l_1$ be lengths (*i.e.,* elements of $[1..N]$). Let $pred$ and $[params]$ be as above (recall the definition of $holds$). Below, "length" means delegation-path length. The relations $=, \leq, \geq, <, >$, and the functions $+$, $-$, and min, are defined on the domain $[1..max(N, MaxDepth)] \cup \{*\}$. The predicates $\subseteq$ and $\supseteq$ are defined on the domain of principal sets $2^{Principals}$. The behavior of "*" is similar to $\infty$, *e.g.,* for any integer $l$ in $[1, ..., max(N, MaxDepth)]$: $l < *$, $* - l = *$, $* = *$, and $\min(*, l) = l$.

The additional clauses are as follows.[6]

1. Every ground clause that has the form:
   $holds(A, pred, [params], l)$
   `if` $holds(A, pred, [params], l_0)$.
   and satisfies $l \geq l_0$.
   Intuitively, this represents the deduction meta-rule that: If a direct statement is deducible within length $l_0$, then the direct statement is deducible within any longer length $l \geq l_0$.

2. Every ground clause that has the form:
   $delegates(A, pred, [params], d, CS, l)$
   `if` $delegates(A, pred, [params], d_0, BS, l_0)$.
   and satisfies $d \leq d_0$, $l \geq l_0$, and $CS \supseteq BS$.
   Intuitively, this represents the deduction meta-rule that: If a delegation statement is deducible, then any weaker delegation statement is deducible within any longer length. Smaller depth and larger conjunctive delegatee set each weaken a delegation statement.

3. Every ground clause that has the form:
   $holds(A, pred, [params], l)$
   `if` $delegates(A, pred, [params], 1, BS, l_0)$,
   $holds(B_1, pred, [params], 1)$,
   $holds(B_2, pred, [params], 1)$,
   $\ldots,$
   $holds(B_n, pred, [params], 1)$.
   and satisfies $l = l_0 + 1$ and $l_0 < N$.
   Intuitively, this represents the deduction meta-rule that enforces the effect of depth-1 delegations.

4. Every ground clause that has the form:
   $delegates(A, pred, [params], d, CS, l)$

[6] These clauses are slightly different from the ones in the extended abstract version that appeared in CSFW-12. However, they are virtually equivalent.

$$\begin{array}{ll}
\texttt{if} & delegates(A, pred, [params], d_0, BS, l_0), \\
& delegates(B_1, pred, [params], d_1, CS, l_1), \\
& delegates(B_2, pred, [params], d_1, CS, l_1), \\
& \ldots, \\
& delegates(B_n, pred, [params], d_1, CS, l_1),
\end{array}$$

and satisfies $d = \min(d_1, d_0 - l_1)$, $l = l_0 + l1$, $l_1 < d_0$, and $l_1 \leq (N - l_0)$.

Intuitively, this represents the deduction meta-rule that enforces the effect of chaining of delegations. The depth of the deduced (head) delegation of $A$ is bounded both by the depth of the delegation from $A$ to $BS$ and by the depth of the delegations from $Bi$'s to $CS$. The depth of the delegation from $A$ to $BS$ has to cover the path lengths that have already been used in deriving delegations from $Bi$'s to $CS$.

The $\mathcal{O}^I$ that results (from adding these further clauses to $T_3^I$) is a ground definite OLP. It thus has one or more *OLP-models*, *i.e.*, models in the usual OLP semantics [19]. In particular, it has a model $MinOLPModel(\mathcal{O}^I)$ that is minimal in the sense of the usual OLP semantics.

**Proposition 1** *Given two interpretations $I \subseteq J$ of $\mathcal{LO}_{\mathcal{P}}$, if an interpretation $K$ of $\mathcal{LO}_{\mathcal{P}}$ is an OLP-model of $\mathcal{O}^J$, then $K$ is also an OLP-model of $\mathcal{O}^I$.*

**Proof**. The dependence of $\mathcal{O}^I$ upon $I$ is due solely to principal structures that depend upon $I$. Furthermore, such affected principal structures depend upon $I$ solely via the dynamic threshold structures that those principal structures contain. The differences that may exist between $\mathcal{O}^I$ and $\mathcal{O}^J$ can thus only be caused by dynamic threshold structures. Note also that such dynamic threshold structures are only allowed to appear in heads of clauses, not their bodies, and moreover only in heads that are delegation statements.

More precisely, for a given principal structure $PS$, such a difference can be viewed as the difference between the normal form $PS^I$ and the normal form $PS^J$. We say that $PS^I$ is *dominated by* $PS^J$ when for each $i$, there exists a $j$ such that $PS_j^J \subseteq PS_i^I$. Next, we will show that such domination holds.

Let $PS^I$ and $PS^J$ be viewed as propositional formulas in reduced DNF form. Let $\rightarrow$ be material implication. Then, tautologically, $PS^I \rightarrow PS^J$ is true if and only if $PS^I$ is dominated by $PS^J$.

Consider a dynamic threshold structure

$$TS = \texttt{threshold}(k, Prin\ \texttt{says}\ pred/2).$$

Let $W^I$ and $W^J$ be the P-W sets defined in $TS$ under $I$ and $J$, respectively. Since $I \subseteq J$, any direct statement "$Prin$ `says` $pred(P, w)$" that is true in $I$ is also true in $J$. Thus the weight of each principal in $W^J$ is greater than or equal to its weight in $W^I$. Now consider the normal forms of $TS$ under $I$ and $J$, *i.e.*, $TS^I$ and $TS^J$. Let $TS^I = TS_1^I; \ldots; TS_m^I$ and $TS^J = TS_1^J; \ldots; TS_n^J$. Each $TS_i^I$ is a minimal subset of principals in $W^I$ such that the sum of these principals' weights is greater than or equal to the threshold $k$.

Since the weight of each principal in $W^J$ is greater than or equal to its weight in $W^I$, for each $TS_i^I$ (i.e., for each $i$, given $I$) there must exist a $TS_j^J$ such that $TS_j^J \subseteq TS_i^I$. In short, $TS^I$ is dominated by $TS^J$. When principal structures are viewed as DNF propositional formulas, they are nonnegative. Since domination is equivalent to material implication, therefore, for any principal structure $PS$ that contains dynamic threshold structures, $PS^I$ is dominated by $PS^J$.

Next, we compare the rules in $\mathcal{O}^I$ to the rules in $\mathcal{O}^J$. The essence of the transform $Trans^I$'s dependence on $I$ is in the step two (of the four steps in the definition of $Trans^I$), where a principal structure $PS$ is replaced by its normal form $PS^I$. Consider a rule $r$ in $\mathcal{O}^I$. If $r$ is added in step four of $Trans^I$, it also appears in $\mathcal{O}^J$, since step four actually does not depend upon $I$. Otherwise, $r$ must result from step three of $Trans^I$. If $r$ results from step three of $Trans^I$, we can view it in terms of the corresponding rule $r'$ that resulted from step two. Let $ddr$ be a rule in $P^{instd}$. We call $ddr$ a *dynamic delegation rule* when $ddr$'s head is a delegation statement that contains dynamic threshold structures. If $r'$ is not a rule that results from translating in step two from a dynamic delegation rule, then $r'$ is not dependent upon $I$ and $r$ also appears in $\mathcal{O}^J$.

Thus, the rules in $\mathcal{O}^I$ and $\mathcal{O}^J$ differ only in regard to the results of applying step two of $Trans^I$ to dynamic delegation rules. Consider such a dynamic delegation rule $ddr$. It has the form

$$A \texttt{ delegates } p\hat{\ }d \texttt{ to } PS \texttt{ if } F.$$

where $A$ is a principal, $F$ is an instantiated statement formula in which no dynamic threshold structures appear, and $PS$ is a principal structure that contains one or more dynamic threshold structures. Observe that every principal structure (in $\mathcal{P}$) does not contain logical variables, and thus is unaffected by step one of $Trans^I$. Step two transforms $ddr$ into a set of one or more rules in each of which $PS$ is replaced by a disjunct $PS_i^I$ of its normal form $PS^I$. Consider the $i^{th}$ such resulting rule:

$$A \texttt{ delegates } p\hat{\ }d \texttt{ to } PS_i^I \texttt{ if } F.$$

If $r'$ results from translating in step two from a dynamic delegation rule, then it must have this form. Next comes a crucial point of our argument. By the domination property shown above, there exists in $\mathcal{O}^J$ a corresponding rule

$$A \texttt{ delegates } p\hat{\ }d \texttt{ to } PS_j^J \texttt{ if } F.$$

such that $PS_j^J \subseteq PS_i^I$.

We say that delegating to a smaller (in the sense of subset) principal set is more *undemanding*. Therefore, for every rule $r$ in $\mathcal{O}^I$, there is a corresponding rule in $\mathcal{O}^J$ that is either identical or is at least as undemanding as $r$.

Intuitively, more undemanding delegations result in more (in the sense of superset) conclusions inferred via delegation. Formally, this property is implied by the rules added in step four of the transform. We say that a ruleset is *stronger in deduction power* when it entails more (in the sense of superset) conclusions.

$\mathcal{O}^J$ is thus at least as strong (in deduction power) as $\mathcal{O}^I$. So any model of $\mathcal{O}^J$ is also a model of $\mathcal{O}^I$. ∎

**Definition 2** *An interpretation $I$ of $\mathcal{LO}_\mathcal{P}$ is an* **O-model** *of a D1LP program $\mathcal{P}$ if and only if $I$ is an OLP-model of $\mathcal{O}^I = Trans^I(P)$.*

We observe this definition's flavor in that we use the interpretation itself in reducing principal structures This is similar to It turns out that every D1LP program $\mathcal{P}$ has at least one O-model, as we will show below in Theorem 7.

**Theorem 3** *The intersection of any two O-models of $\mathcal{P}$ is also an O-model of $\mathcal{P}$.*

**Proof**. Given two O-models $I$ and $J$ of $\mathcal{P}$, one can conclude by the definition of O-models that $I$ is an OLP-model of $\mathcal{O}^I$ and that $J$ is an OLP-model of $\mathcal{O}^J$. Let $K = I \cap J$. Because $K \subseteq I$ and $K \subseteq J$, Proposition 1 implies that $I$ and $J$ are both OLP-models of $\mathcal{O}^K$. Because definite ordinary logic programs have the property that two models' intersection is still a model [19], $K$ is an OLP-model of $\mathcal{O}^K$. By the definition of O-models, $K$ is thus an O-model of P. ∎

**Definition 4** *The* **minimal O-model** *of $\mathcal{P}$ is the intersection of all of its O-models. We write this as $MinOModel(\mathcal{P})$.*

It turns out that every D1LP $\mathcal{P}$ has a minimal O-model; below, in Theorem 7, we show how to construct it.

Ultimately, we are interested in models expressed in $\mathcal{LI}_\mathcal{P}$, the original D1LP language of $\mathcal{P}$. We define a simple reverse translation $ReverseTrans$ that maps each O-model $I$ of $\mathcal{P}$ to its corresponding *D1LP-model* in $\mathcal{LI}_\mathcal{P}$, as follows.

- For each O-conclusion of the form: $holds(A, pred, [params], length)$,
  include the D1LP-conclusion: $A$ `says` $pred(params)$.

- For each O-conclusion of the form:
  $delegates(A, pred, [params], depth, Delegatee, length)$,
  include the D1LP-conclusion: $A$ `delegates` $pred(params)$`^`$depth$`to` $Delegatee$.

Notice here that (delegation path) length is ignored after the OLP conclusions are drawn.

We define the *Herbrand base* of $\mathcal{P}$, in $\mathcal{LI}_\mathcal{P}$, as the set of all ground statements of $\mathcal{P}$, restricted to require that every principal structure be a principal set. We define an *interpretation* of $\mathcal{P}$ to be an assignment of truth values ($true$ and $false$) to the Herbrand base of $\mathcal{P}$. Such an interpretation can also be viewed as a set of true ground statements, *i.e.*, as a *conclusion set*.

**Definition 5** *An interpretation $M$ of $\mathcal{LI}_\mathcal{P}$ is a* **model** *of a D1LP program $\mathcal{P}$ if and only if $M = ReverseTrans(I)$ and $I$ is an O-model of $\mathcal{P}$. The* **minimal model** *of $\mathcal{P}$ is $ReverseTrans(MinOModel(P))$. We write this as $MinModel(\mathcal{P})$.*

Sometimes, for the sake of explicitness, we will also speak of these as (minimal) *D1LP-models*.

Next, we show that $MinOModel(\mathcal{P})$ and thus its corresponding $MinModel(\mathcal{P})$ actually *exist*, by showing how to *construct* $MinOModel(\mathcal{P})$.

**Definition 6** *Given $\mathcal{P}$, we define an* **operator** *$\Psi_\mathcal{P}$ that takes an interpretation $I$ of $\mathcal{LO}_\mathcal{P}$ and returns another one: $\Psi_\mathcal{P}(I) = MinOLPModel(\mathcal{O}^I)$, where $MinOLPModel$ is the standard minimal model operator for OLP.*

**Theorem 7 (Construct Minimal Model)** *$MinOModel(\mathcal{P})$ is the* **least fixpoint of** *$\Psi_{\mathcal{P}}$. This fixpoint is obtained by iterating $\Psi_{\mathcal{P}}$ a* **finite** *number of times, starting from $\emptyset$. $MinModel(\mathcal{P})$ thus* **exists**, *for every D1LP $\mathcal{P}$.*

**Proof**. Let $I_0 = \emptyset$, and, for $i > 0$, let $I_i = \Psi_{\mathcal{P}}(I_{i-1}) = MinOLPModel(\mathcal{O}^{I_{i-1}})$. We first show that the sequence $I_0, I_1, \ldots$ is increasing. Clearly, $I_0 \subseteq I_1$. Suppose that $I_{k-1} \subseteq I_k$. Now consider $I_{k+1}$. It is a model of $\mathcal{O}^{I_k}$. From Proposition 1, $I_{k+1}$ is also a model of $\mathcal{O}^{I_{k-1}}$. By definition of the operator $\Psi_{\mathcal{P}}$, $I_k$ is the *minimal* OLP-model of $\mathcal{O}^{I_{k-1}}$. Thus $I_k \subseteq I_{k+1}$. So the sequence $I_0, I_1, \ldots$ is increasing.

Since the Herbrand base of $\mathcal{LO}_{\mathcal{P}}$ is finite, there are only finite number of interpretations of $\mathcal{LO}_{\mathcal{P}}$. Therefore, there must exist a first ordinal $\lambda$ such that $I_\lambda = I_{\lambda+1}$. And $I_\lambda$ is the least fixpoint of $\Psi_{\mathcal{P}}$.

We now prove that $I_\lambda = MinOModel(\mathcal{P})$. By definition of the operator $\Psi_{\mathcal{P}}$, $I_\lambda$ is an O-model of $\mathcal{P}$. Let $J = MinOModel(\mathcal{P})$. Then $J \subseteq I_\lambda$. Suppose, by contradiction, that $I_\lambda \not\subseteq J$. Since $I_0 \subseteq J$, there must exist an ordinal $0 < \theta < \lambda$ such that $I_\theta \subseteq J$ but $I_{\theta+1} \not\subseteq J$. Because $J$ is an OLP-model of $\mathcal{O}^J$, Proposition 1 implies that $J$ is also an OLP-model of $\mathcal{O}^{I_\theta}$. However, $I_{\theta+1}$ is the *minimal* OLP-model of $\mathcal{O}^{I_\theta}$. Thus $I_{\theta+1} \subseteq J$, which is a contradiction. ∎

**Inferencing**: Because of the finiteness properties mentioned above, computing $MinModel(\mathcal{P})$ is decidable. Given the minimal model $MinModel(\mathcal{P})$, queries in D1LP can be translated as we did for the body of a clause, then answered using the model. We have a current implementation of restricted D1LP. It is written in Prolog and uses a top-down query-answering method.

# 4 Use of D1LP

In this section, we use the public-key infrastructure problem to demonstrate the use of D1LP. The trust-management approach views the PKI problem from one user's point of view. The user has trust beliefs and certificates from other principals, and it needs to decide whether a particular binding is valid according to its information. All of these beliefs, certificates, and decisions can be represented uniformly in D1LP.

D1LP can also be used to represent authorizations. Authorizing a principal to do something can be represented as a delegation to that principal. Whether to allow this principal to further grant this authorization to other principals can be controlled by delegation depth. An authorization request can be answered by deciding whether a delegation statement is true or false. Moreover, separation of duty [8] can be achieved by delegations to threshold structures.

We first show how certificates from different PKI proposals can be represented in D1LP. Pretty Good Privacy (PGP)'s certificates only establish key bindings; they have no delegation semantics. The delegations in PGP are expressed by trust degrees that are stored in local key rings. They all have depth 1. In PGP, a user can also specify threshold values for accepting a key binding. In D1LP, this can be achieved through dynamic threshold structures. One way is to use several predicates to denote different trust levels, for example, `fully_trusted` and `partly_trusted`. A user Alice may have the following policies:

```
Alice says fully_trusted(Bob).
Alice says fully_trusted(Sue).
Alice says partly_trusted(Carl).
Alice says partly_trusted(Joe).
Alice says partly_trusted(Peg).
Alice delegates is_key(_Key,_User)^1 to threshold(1,fully_trusted/1).
Alice delegates is_key(_Key,_User)^1 to threshold(2,partly_trusted/1).
```

Of course, one can also use *weighted* dynamic threshold structures.

In X.509 [6], certification authorities (CAs)' certificates are chained together to establish a key binding. For such delegation chains to be really meaningful, the certificates on such chains must also imply delegations. Since there is no limit on the length of delegation chains, all such delegations have depth '$*$'. Privacy Enhanced Mail (PEM) uses X.509's certificates but limits the CA hierarchy to three levels: IPRA, PCAs, and CAs. Thus PEM's trust model requires that every user give a depth-three delegation to IPRA.

A SPKI certificate does not establish key binding; it is a delegation from the issuer to the subject of the certificate. It has one field that controls whether a delegation can be further delegated or not; this means that every delegation has depth 1 or '$*$'.

There are other proposals to use lengths of delegation paths as a metric for public-key bindings. The difference between these path lengths and our delegation depths is that, in the former, there is only one length, and it is specified by the trust root. However, every DL delegation statement can have a delegation depth limit. This has the effect of allowing every principal on the delegation path to specify how much further this path can go. Together, the set of delegation depths along the path determine whether the path is invalidly deep.

Next, we give an extended example of public key authorization with delegation. Consider a user Alice who wants to decide whether a public key `M_Key` is web site `M_Site`'s public key. There are many certification systems that may be relevant. In particular, Alice trusts three of them: systems $X$, $Y$, and $Z$. System $X$ has three levels: `XRCA`, `XPCA`'s, and `XCA`'s, where `XRCA` is the root, `XCA`'s are CA's that certify users' public keys directly, and `XPCA`'s certify `XCA`'s public keys and are in turn certified by `XRCA`.[7] System $Y$ has two levels: `YRCA` and `YCA`'s. System $Z$ has only one level: `ZRCA`, which certifies users' keys directly. Alice first translates (*i.e.*, represents) certificates of these systems into statements using the predicates `Xcertificate(`*signature_key*, *subject*, `...)`, `Ycertificate(...)`, `Zcertificate(...)`.[8] Then Alice asserts some rules that translate these into statements of a common certificate predicate, say, `is_site_key(_Key,_Site)`. For example:

```
_Issuer says is_site_key(_Key,_Site)
   if  Xcertificate(_Issuer,_Key,_Site).
```

Next, Alice specifies the sense in which she trusts the three systems, by asserting the rule:

[7] In Privacy Enhanced Mail (PEM)[18], the three levels of CA's are Internet Policy Registration Authority (IPRA), Policy Certification Authority (PCA), and Certification Authority (CA).

[8] The exact fields of these predicates are determined by Alice. They should be whichever elements of the certificates are relevant to Alice's policies.

```
Alice delegates is_site_key(_K,_S)^3 to {XRCA,{YRCA;ZRCA}}.
```

This means that Alice requires system $X$ and one of system $Y$ and system $Z$ to certify a website public key. She does this with delegation depth 3 because she knows that 3 is the maximum number of levels in those certification systems. (Note that, for other purposes, Alice can use predicates other than `is_site_key` and trust these systems differently.) Suppose that `M_Key` is certified by both system $Y$ and system $Z$, *i.e.*, there are certificates that translate into:

```
YRCA delegates is_site_key(_K,_S)^1 to YCA1.
YCA1 says is_site_key(M_Key,M_Site).
ZRCA says is_site_key(M_Key,M_Site).
```

Then the given information is not enough to deduce (*i.e.*, entail) that "`Alice says is_key(M_Key,M_Site).`", because there is no certification of that key from `XRCA`.

Now suppose that, in addition to these systems above, Alice has a friend Bob. For whatever reasons, Alice trusts Bob unconditionally to certify websites' public keys, *i.e.*,

```
Alice delegates is_site_key(_K,_S)^* to Bob.
```

Bob thinks that certification by system $Z$ is itself enough for those sites that belong to association `assoc`, and he trusts `ASSOC` in deciding which sites belong to `assoc`, *i.e.*,

```
Bob delegates is_site_key(_K,_S)^1 to ZRCA
  if I says belongs_to(_S,assoc).
Bob delegates belongs_to(_S,assoc)^1 to ASSOC.
```

Alice stores these policies that she got from Bob earlier. Suppose that site `M_Site` also sent the following certificate issued by `ASSOC`:

```
ASSOC says belongs_to(M_Site,assoc).
```

With the above additional information, Alice can deduce the following:

```
Bob says belongs_to(M_Site,assoc).
Bob delegates is_site_key(M_Key,M_Site)^1 to ZRCA.
Alice delegates is_site_key(M_Key,M_Site)^1 to ZRCA.
```

Finally, Alice can deduce:

```
Alice says is_site_key(M_Key,M_Site).
```

# 5 Extension: Negations and Priorities

D1LP is (logically) monotonic. It cannot express negation or negative knowledge.

However, many security policies are non-monotonic or more easily specified as non-monotonic ones, *e.g.*, certificate revocation. In many applications, a natural policy is to make a decision in one direction, *e.g.*, in favor of authorizing $H$, if there is no information/evidence to the contrary, *e.g.*, no known revocation. Using *negation-as-failure* (a.k.a. *default negation* or *weak negation*) is often an easy and intuitive way to do this. Also useful in representation of many policies is *classical negation* (a.k.a. *explicit negation* or *strong negation*), which allows policies that explicitly forbid something. Classical negation in rules, especially in the consequents (heads) of rules, enables one to specify both the positive and negative *sides* of a policy, (*i.e.*, both permission and prohibition) using the expressive power of rules, *e.g.*, using inferential chaining. As argued in [16, 17], this allows more flexible security policies. Classical negation is particularly desirable for authorization in Internet scenarios, where the number of potential requesters is huge. For low-value transactions, users sometimes have security policies that give access to all except a few requesters. Without negations, it would be effectively impossible to do this.

Introducing classical negation leads to the potential for *conflict*: Two rules for opposing sides may both apply to a given situation. Care must be taken to avoid producing inconsistency. *Priorities*, which specify that one rule overrides another, are an important tool for specifying how to handle such conflict. *E.g.*, a known revocation overrides a general rule to presume trustworthiness. *E.g.*, one principal overrides another's decision/recommendation. Some form of prioritization is generally present in many rule-based systems; prioritization has also received a great deal of attention in the non-monotonic reasoning literature (see, *e.g.*, [12] for some literature review and pointers).

Prioritization information is naturally available. One common basis for prioritization is *specificity*: Often it is desirable to specify that more specific-case rules should override more general-case rules. Another basis is *recency*, in which more recent rules override less recent rules. A third common basis is relative *authority*, in which rules from a more authoritative source override rules from a less authoritative one. For example, a superior legal/bureaucratic/organizational jurisdiction or a more knowledgeable/expert source may be given higher priority. It is often useful to reason about prioritization, *e.g.*, to reason from organization roles or timestamps to deduce priorities. Reasoning about prioritization may itself involve conflict, *e.g.*, a less recent rule may be more specific or more authoritative.

To allow users to express non-monotonic policies in a natural and powerful fashion, we define **D2LP**, which stands for version 2 of Delegation Logic Programs. D2LP is (logically) non-monotonic. D2LP expressively extends D1LP to include negation-as-failure, classical negation, and partially-ordered priorities. Just as D1LP bases its syntax and semantics on definite ordinary LP's, D2LP bases its syntax and semantics on Courteous LP's [14, 15]. The version of Courteous LP's we use is expressively generalized as compared to the previous version in [12, 13].

In the rest of this section, we give an overview of D2LP. Full details will be given in a forthcoming companion paper.

Syntactically, each D2LP rule (clause) is generalized to permit each statement to be negated in two ways: by classical negation $\neg$ and/or by negation-as-failure (NAF) $\sim$. Each rule also is generalized to permit an optional *rule label*, which is a handle for specifying priority. Prioritization is specified via the predicate *overrides*. $overrides(label_1, label_2)$ means that every rule having rule label $label_1$ has strictly higher priority than every rule having rule label $label_2$. *overrides* is treated specially in the semantics to generate the prioritization used by all rules. Otherwise, however, *overrides* is treated as an ordinary predicate.

A D2LP direct statement has the form:

$$A \;\; \texttt{says} \;\; [\sim]\,[\neg]\,p$$

A D2LP delegation statement has the form:

$$A \;\; [\sim]\,[\neg] \;\; \texttt{delegates}\; p\hat{\ }d \;\texttt{to}\; PS$$

Here, as when we described D1LP, $A$ is a principal, $p$ is a base atom, $d$ is a depth, and $PS$ is a principal structure. Square brackets ("[...]") indicate the optionality of what they enclose. $\neg$ stands for classical negation and is read in English as "not". $\sim$ stands for negation-as-failure and is read in English as "fail".

When a statement does not contain $\sim$, we say it is *classical*; when it contains neither $\sim$ nor $\neg$, we say it is *atomic*.

Semantically, the negations' scope can be viewed as applying to the whole statement. Intuitively, $\neg s$ means that $s$ is believed to be false. By contrast, $\sim cs$ means that $cs$ is not believed to be true, *i.e.*, either $cs$ is believed to be false, or $cs$ is *unknown*. "Unknown" here means in the sense that there is no belief one way or the other about whether $cs$ is true versus false.

A D2LP statement formula is defined as the result of conjunctions and disjunctions applied to D2LP statements, similarly to the way in which a D1LP statement formula is defined as the result of conjunctions and disjunctions applied to D1LP statements (*i.e.*, atomic statements). A D2LP rule (clause) is defined as:

$$\langle lab \rangle \;\; S \;\texttt{if}\; F.$$

Here, $S$ is a *classical* statement. $F$ is a statement formula. The rule label $lab$ is an ordinary logical term (e.g., the constant $frau$ below)

in the D2LP language. The rule label may optionally be omitted. Note that D2LP relaxes the D1LP prohibition on dynamic threshold structures in the body. Syntactically, a D1LP rule is a special case of a D2LP rule.

We say that $c$ is a *base classical literal* when $c$ has the form $a$ or $\neg a$, where $a$ is a base atom. Here, $\neg$ stands for classical negation, and is read in English as "not". We say that $f$ is a *base literal* when $f$ has the form $c$ or $\sim c$, where $c$ is a classical literal. $\sim$ stands for negation-as-failure and is read in English as "fail". Intuitively, $\neg a$ means that $a$ is believed to be false. By contrast, $\sim a$ means that $a$ is not believed to be true, *i.e.*: $a$ is either believed to be false, or $a$ is unknown. Unknown here means in the sense that there is no belief one way or the other about whether $a$ is true versus false.

Next, we give a simple example that illustrates the use of classical negation and priorities. Let D2LP program $\mathcal{R}_1$ be the following set of rules:

$$\begin{array}{l}
\langle cred \rangle \; A \;\; \texttt{says} \;\; honest(_X) \;\; \texttt{if} \;\; B \;\; \texttt{says} \;\; creditRating(_X, good). \\
\langle frau \rangle \; A \;\; \texttt{says} \;\; \neg honest(_X) \\
\qquad\qquad \texttt{if} \;\; C \;\; \texttt{says} \;\; fraudulent(_X) \;,\; D \;\; \texttt{says} \;\; fraudExpert(C). \\
\qquad\quad A \;\; \texttt{says} \;\; overrides(frau, cred). \\
\qquad\quad B \;\; \texttt{says} \;\; creditRating(Joe, good). \\
\qquad\quad D \;\; \texttt{says} \;\; fraudExpert(C).
\end{array}$$

$\mathcal{R}_1$ entails the conclusion: $\quad A \;\; \texttt{says} \;\; honest(Joe)$.

Continuing the example, suppose the following statement is added to $\mathcal{R}_1$ to form $\mathcal{R}_2$.

$$C \texttt{ says } fraudulent(Joe).$$

$\mathcal{R}_2$ instead entails the conclusion:

$$A \texttt{ says } \neg honest(Joe).$$

The semantics of a D2LP are defined via a translation $Trans2$ to a ground courteous LP that is roughly similar to D1LP's translation $Trans$ to a ground OLP. Courteous LP's expressively extend OLP's to include $\neg$ and $\sim$, as well as prioritization represented via an $overrides$ predicate on rule labels.

We impose some further expressive restrictions in D2LP, related especially to cyclicity of dependency between predicates, to ensure D2LP's well-behavior semantically and computationally. The generalized version of Courteous LP's relies on the well-founded semantics [10], which is computationally tractable (worst-case polynomial-time) for the ground case.

Courteous LP's semantic well-behavior includes having a unique (minimal) model that is *consistent* ($s$ and $\neg s$ are never both sanctioned as conclusions). D2LP inherits this same well-behavior. D2LP inferencing remains decidable; its finiteness properties are similar to those of D1LP.

In a forthcoming paper, we cover DL's treatment of negation in detail, as we have covered delegation in detail in this paper.

# 6 Discussion and Future Work

As explained in previous sections, our design of DL was primarily influenced by earlier work on trust management and on logic programming and knowledge representation. There is other, more tenuously related work in the computer security literature, which we now review.

In [1, 21, 27], Abadi, Burrows, Lampson, Wobber, and Plotkin developed a logic for authentication and access control in distributed systems. Their logic is similar to DL in one respect: It focuses on delegation, which it expresses via the "*speaks for*" relation. In all other respects, their logic is quite different from DL in its ends and means. Its approach to delegation does not include delegation-depth or threshold structures; in this respect, DL's notion of delegation is more powerful. Their treatment of authorization is considerably less general than DL's; all of their "policies" are expressed as access-control lists, whereas DL (which takes the "trust-management approach") is a general authorization language. Their approach also differs from DL in that it is not based on logic programming, and it does not have a model-theoretic semantics. Finally, their logic does not incorporate negation.

Maurer [23] modeled public-key infrastructure via recommendations with levels and confidence values. "Delegation" in DL is very similar to "recommendation" in [23], but there are several differences. One is that Maurer's model supports direct statements and delegation statements but not clauses (rules). A second is that Maurer's model supports reasoning about the delegations and beliefs of what DL calls "Local," but it does not allow, say, Local A to reason about the beliefs of B (*e.g.*, about whether, in A's view, B delegates to C or B believes a particular statement). Thus, one cannot express, in Maurer's model, that "A says $X$ if B says $Y$." This second restriction eliminates the need to maintain "lengths," and permits delegation chaining to be much simpler in [23] than it is in DL. Thirdly, Maurer's model does not support delegation to lists, threshold structures, *etc.*. Finally, DL does not use numerical confidence values, because it is our view that users would have little if any factual basis for choosing specific numbers; in many scenarios, threshold structures can provide similar functionality. Like Abadi *et al.* [1, 21, 27], Maurer [23] does not treat negation and does not provide a model-theoretic semantics.

Previous work on authorization languages that incorporate negation includes that of Woo and Lam [28] and Jajodia *et al.* [16, 17]. Compared to DL, those previous approaches differ in that they do not deal with delegation; in particular, they have no complex principal structures such as thresholds. Furthermore, the way in which [28, 16, 17] handles negation is different from the way it is handled in D2LP. Although we have not fully specified D2LP in this paper, the discussion in Section 5 above suffices to allow us to explain how our approach to negation differs from that of [28, 16, 17].

The language of Woo and Lam [28], which is based on Default Logic [24], does not guarantee that every program (or, in their terms, every "policy base") has a model; furthermore, when a model exists, it might not have a meaningful interpretation, because that model might be inconsistent. Well-founded semantics and prioritized conflict handling enable D2LP to support a more expressive set of non-monotonic policies and to give a unique and meaningful model to every program.

Jajodia *et al.* [16, 17] propose a logical authorization language that is based on Datalog OLP extended with two forms of negation (classical negation and negation as failure). Their support for non-monotonic policies is limited, however, because they impose syntactic restrictions that ensure that policies are conflict-free. Moreover, they predefine all predicates and require stratifiability. By contrast, D2LP is more expressive, via well-founded semantics and conflict handling.

Future work will address the computational complexity of compliance checking in DL, syntactically restricted classes of DL programs for which compliance can be checked very efficiently, implementation of the DL interpreter (for which we now have only a preliminary version for restricted D1LP), and deployment of DL in an e-commerce platform.

# References


[1] M. Abadi, M. Burrows, B. Lampson, and G. Plotkin, "A Calculus for Access Control in Distributed Systems," *ACM Transactions on Programming Languages and Systems*, 15 (1993), pp. 706–734.

[2] C. Baral and M. Gelfond, "Logic Programming and Knowledge Representation", *Journal of Logic Programming*, 19,20(1994), pp. 73–148. Includes extensive review of literature.

[3] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis, "The KeyNote Trust-Management System," submitted for publication as an Internet RFC, March 1999. `http://www.cis.upenn.edu/~angelos/Papers/draft-keynote.txt`.

[4] M. Blaze, J. Feigenbaum, and J. Lacy, "Decentralized Trust Management," in *Proceedings of the Symposium on Security and Privacy*, IEEE Computer Society Press, Los Alamitos, 1996, pp. 164–173.

[5] M. Blaze, J. Feigenbaum, and M. Strauss, "Compliance-Checking in the PolicyMaker Trust Management System," in *Proceedings of Financial Crypto '98*, Lecture Notes in Computer Science, vol. 1465, Springer, Berlin, 1998, pp. 254–274.

[6] ITU-T Rec. X.509 (revised), *The Directory - Authentication Framework*, International Telecommunication Union, 1993.

[7] Y.-H. Chu, J. Feigenbaum, B. LaMacchia, P. Resnick, and M. Strauss, "REFEREE: Trust Management for Web Applications," *World Wide Web Journal*, 2 (1997), pp. 706–734.

[8] D. Clark and D. Wilson, "A Comparison of Commercial and Military Computer Security Policies," In *Proceedings of the IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, Los Alamitos, 1987.

[9] C. Ellison, "SPKI Certificate Documentation," `http://www.pobox.com/~cme/html/spki.html`.

[10] A. Van Gelder, K. A. Ross, and J. S. Schlipf, "The Well-founded Semantics for Logic Programming," *Journal of the ACM*, 38 (1991), pp. 620-650.

[11] J. Gosling and H. McGilton, *The Java Language Environment, A White Paper*, Sun Microsystems, Inc., Mountain View, 1995.

[12] B. Grosof, "Courteous Logic Programs: Prioritized Conflict Handling for Rules," IBM Research Report RC20836, May 1997. This is an extended version of [13].

[13] B. Grosof, "Prioritized Conflict Handling for Logic Programs," in *Proceedings of the International Symposium on Logic Programming*, MIT Press, Cambridge, 1997, pp. 197–212.

[14] B. Grosof, "Compiling Prioritized Default Rules Into Ordinary Logic Programs," IBM Research Report RC 21472, May 1999.

[15] B. Grosof, "DIPLOMAT: Compiling Prioritized Default Rules Into Ordinary Logic Programs, for E-Commerce Applications (extended abstract of Intelligent Systems Demonstration)," in *Proceedings of AAAI-99*, Morgan Kaufmann, 1999. Extended version is IBM Research Report RC 21473, May 1999.

[16] S. Jajodia, P. Samarati, and V. S. Subrahmanian, "A Logical Language for Expressing Authorizations," in *Proceedings of the Symposium on Security and Privacy*, IEEE Computer Society Press, Los Alamitos, 1997, pp. 31–42.

[17] S. Jajodia, P. Samarati, V. S. Subrahmanian, and E. Bertino, "A Unified Framework for Enforcing Multiple Access Control Policies," in *Proceedings ACM SIGMOD Conference on Management of Data*, 1997.

[18] S. T. Kent, "Internet Privacy Enhanced Mail," *Communications of the ACM*, 8 (1993), pp. 48–60.

[19] J. W. Lloyd, *Foundations of Logic Programming*, second edition, Springer, Berlin, 1987.

[20] M. Longhair (editor), "A P3P Preference Exchange Language (APPEL) Working Draft," W3C Working Draft 9, October 1998,
`http://www.w3.org/P3P/Group/Preferences/Drafts/WD-P3P-preferences-19981009`.

[21] B. Lampson, M. Abadi, M. Burrows, and E. Wobber, "Authentication in Distributed Systems: Theory and Practice," *ACM Transactions on Computer Systems*, 10 (1992), pp. 265–310.

[22] M. Marchiori, J. Reagle, and D. Jaye (editors), "Platform for Privacy Preferences (P3P1.0) Specification," W3C Working Draft 9 November 1998, `http://www.w3.org/TR/WD-P3P/`.

[23] U. Maurer, "Modelling a Public-Key Infrastructure," in *Proceedings of the 1996 European Symposium on Research in Computer Security*, Lecture Notes in Computer Science, vol. 1146, Springer, Berlin, 1997, pp. 325–350.

[24] R. Reiter, "A Logic for Default Reasoning," *Artificial Intelligence*, 13 (1980), pp. 81–132.

[25] P. Resnick and J. Miller, "PICS: Internet access controls without censorship," *Communications of the ACM*, 39 (1996), pp. 87–93.

[26] R. Rivest and B. Lampson, "Cryptography and Information Security Group Research Project: A Simple Distributed Security Infrastructure," `http://theory.lcs.mit.edu/~cis/sdsi.html`.

[27] E. Wobber, M. Abadi, M. Burrows, and B. Lampson, "Authentication in the TAOS Operating System," *ACM Transactions on Computer Systems*, 12 (1994), pp. 3–32.

[28] T. Woo and S. Lam, "Authorization in Distributed Systems: A New Approach," *Journal of Computer Security*, 2 (1993), pp. 107–136.

[29] P. Zimmermann, *The Official PGP User's Guide*, MIT Press, Cambridge, 1995.