8-2009

Reasoning about the Consequences of Authorization Policies in a Linear Epistemic Logic

Henry DeYoung
Carnegie Mellon University

Frank Pfenning
Carnegie Mellon University

Published In
Workshop on Foundations of Computer Security (FCS'09), Los Angeles, California.

Reasoning about the Consequences of Authorization Policies in a Linear Epistemic Logic

Henry DeYoung and Frank Pfenning*

Carnegie Mellon University, Pittsburgh PA 15213, USA
{hdeyoung, fp}@cs.cmu.edu

Abstract. Authorization policies are not stand-alone objects: they are used to selectively permit actions that change the state of a system. Thus, it is desirable to have a framework for reasoning about the semantic consequences of policies. To this end, we extend a rewriting interpretation of linear logic with connectives for modeling affirmation, knowledge, and possession. To cleanly confine semantic effects to the rewrite sequence, we introduce a monad. The result is a richly expressive logic that elegantly integrates policies and their effects. After presenting this logic and its metatheory, we demonstrate its utility by proving properties that relate a simple file system's policies to their semantic consequences.

1 Introduction

Security-sensitive systems typically specify, if only informally, both *policies* and their *semantic effects*. There is a close interplay between the two: the policies are intended to permit the execution of only those operations that have semantic consequences deemed to be safe. For example, a file system policy might be "A principal may read a file if the file's owner permits it," while the semantic effect might be "If a read request is permitted by the policies, then the requesting principal may learn the file's contents."

Despite its critical importance, for most systems, the interface between policies and their semantics is rarely analyzed in a formal, rigorous manner. We contend that only such an analysis can provide the high-level assurance of correctness that is so crucial for security-sensitive systems.

Our goal in this work is therefore to develop a general method for formally representing a system and reasoning about the interface between its policies and their semantic effects.

Cervesato and Scedrov's rewriting view of linear logic, embodied in their ω system [1], is an attractive launching pad toward our goal for two reasons. First,

* This material is based upon work supported by the National Science Foundation under Grant No. NSF-0716469, by the Army Research Office under Grant No. DAAD19-02-1-0389 to Carnegie Mellon University's CyLab, and by a National Science Foundation Graduate Research Fellowship for the first author.

being based on linear logic [2], a logic of consumable resources, it inherently supports mutable state. This will be central to modeling changes in the system of interest. Second, most systems are concurrent, and ω has been shown to encompass a wide variety of state- and process-based concurrency formalisms.

To construct ω, Cervesato and Scedrov identify a fragment of a Gentzen-style sequent calculus for linear logic that consists primarily of left rules. They then view the assumptions in a sequent as representing a state, with the sequence of assumptions along a derivation corresponding to a rewrite sequence. By abandoning an ingrained emphasis on the finiteness of a derivation, such rewrite sequences are potentially unbounded, making it possible to model the infinite executions often assumed for concurrent systems. To simplify their system, Cervesato and Scedrov inline the minor premise of the left rule for implication as part of the main rewrite sequence; thus all derivations are unbranched.

Despite its appeal, ω is not entirely satisfactory for our goal of modeling and reasoning about policies and their semantic effects. One shortcoming is the lack of modal operators to capture the knowledge and intent of principals. This means that these concepts would need to be represented as ordinary atomic predicates, which lack logical force. Therefore, we require the addition of special purpose connectives, including a `says`-like connective [3] to model principals' policies and intent, and epistemic connectives to model the semantic concepts of knowledge and possession.

Conceptually, we would like to place policies and their semantic effects in separate layers: simply checking if access would be granted under some policy should not induce an effect on a system's state. (Proof-carrying authorization [4, 5] also makes this distinction implicitly by separating proof construction by the user from proof verification and subsequent granting of access by the reference monitor.) Moreover, in proving properties that relate policies to their semantic consequences, it will be helpful if the rewrite sequence corresponds to the changes in the system's state and not the policy manipulations.

For these reasons, we do not want to inline the policy portions of the derivation into the rewrite sequence. Thus, Cervesato and Scedrov's simplification to unbranched derivations is no longer applicable. However, by permitting side branches, we introduce a problem: semantic effects can now occur in branches. They may therefore be invisible in the rewrite sequence, which focuses exclusively on the main branch of the derivation.

To resolve this problem, we borrow the idea of a monad [6] as an isolating device for derivations from the linear logic programming language LolliMon [7]. By marking the main branch of a derivation with the monad, we can confine the semantic effects, and only the effects, to the main branch, ensuring that the rewrite sequence corresponds exactly to the sequence of effects.

We also borrow the use of Andreoli's focusing methodology [8] from LolliMon to eliminate don't-care and reduce don't-know nondeterminism in derivations. This limits the number of derivations of a given sequent, making it easier to enumerate the possible rewrite derivations for a system and thereby prove system properties. Focusing also ensures that semantic effects are applied atomically.

The contributions of this paper are two-fold. First, we develop a novel rewriting interpretation of a linear logic having lax modalities for modeling policies and having S4 modalities for modeling knowledge and possession, a substantial extension of both ω and LolliMon. The S4 modalities also serve to provide each principal with local state. To the best of our knowledge, this integration of policies, their semantics, and local state in a single framework makes the logic more expressive than existing rewriting systems.

Second, we demonstrate the utility of our rewriting interpretation by proving some properties about the interface between policies and semantics for a hypothetical file system. Though this system is small and simple, we believe it strongly suggests that our approach will generalize to more realistic examples.

Organization of the Paper. The focused logic is presented in Sect. 2. We formalize our notions of state and rewrite step in Sect. 3. In Sect. 4, we leverage these ideas to prove a few properties of a hypothetical file system. Finally, Sect. 5 gives a brief overview of related work.

2 A Weakly Focused Sequent Calculus

In this section, we present a focused linear logic of affirmation, knowledge, and possession. Andreoli's focusing methodology [8] for sequent calculi reduces the number of nondeterministic choices made in a derivation in three ways: eagerly applying invertible rules, which are rules whose conclusion imply the premises; chaining non-invertible rules together; and controlling the use of atomic propositions in initial sequents. For technical reasons, we choose to present a *weakly* focused logic in which eager application of invertible rules is not mandatory.

2.1 Syntax

We assume the following polarized syntax of propositions. Positive (negative) propositions are those with invertible left (resp., right) rules and non-invertible right (resp., left) rules.

$$A^+ ::= p^+ \mid A^+ \otimes B^+ \mid \mathbf{1} \mid \exists x{:}\tau.A^+ \mid !A^- \mid [K]A^- \mid [\![K]\!]A^- \mid A^-$$
$$A^- ::= p^- \mid A^+ \multimap B^- \mid \forall x{:}\tau.A^- \mid \langle K\rangle A^+ \mid \{A^+\}$$

Special note should be taken of the existence of an implicit coercion from negative propositions, A^-, to positive propositions.

Standard Propositions from Linear Logic. Atomic propositions of both polarities, p^+ and p^-, are included. $A^+ \otimes B^+$ is multiplicative conjunction; it represents the simultaneous existence of resources A^+ and B^+. Its unit is $\mathbf{1}$, which represents the empty set of resources. $A^+ \multimap B^-$ is linear logic's form of implication, in which resource A^+ is consumed to produce resource B^-. $!A^-$ is the exponential, and stands for an unrestricted number of copies (including zero) of resource A^-. $\exists x{:}\tau.A^+$ and $\forall x{:}\tau.A^-$ are first-order existential and universal quantification over terms of type τ, respectively.

Affirmation: $\langle K\rangle A^+$. To model a principal's policies, or affirmations, we follow Garg *et al.* [9] and adopt a family of lax modalities [10], $\langle\cdot\rangle$, indexed by principals, which we polarize for our purposes. That is, $\langle K\rangle A^+$ is read "principal K says A^+." Each of these (unpolarized) lax modalities, $\langle K\rangle$, satisfies the following axioms:

$$\begin{array}{ll} \vdash A \multimap \langle K\rangle A & \text{(T)} \\ \vdash \langle K\rangle(A \multimap B) \multimap \langle K\rangle A \multimap \langle K\rangle B & \text{(K)} \\ \vdash \langle K\rangle(\langle K\rangle A) \multimap \langle K\rangle A & \text{(4)} \end{array}$$

We assume that every principal K is rational: it is willing to affirm any true statement (T). (The converse, in general, does not hold because malicious or ignorant principals may affirm false statements.) K's affirmations are closed under logical consequence (K). Finally, K should trust its own affirmations (4).

The choice of these axioms is by no means the only possible one, and indeed there is much discussion in the literature on alternative axioms for affirmation (see, e.g., [11]). However, the lax axioms represent a simple, and, as argued in [9, 12], reasonable, choice, permitting us to focus on issues relevant to the interface between the policy and semantic layers. Moreover, we believe that the results of this paper will easily extend to systems with other forms of affirmation.

Knowledge: $[\![K]\!]A^-$. Because we need to model the semantic concept of a principal's potential knowledge, we also borrow a family of S4 epistemic modalities indexed by principals, $[\![\cdot]\!]$, from Garg *et al.* [9]. (We again polarize the modalities.) That is, $[\![K]\!]A^-$ is read "principal K is capable of knowing A^-." Being an S4 modality, each (unpolarized) $[\![K]\!]$ satisfies the following rule and axioms:

$$\begin{array}{ll} \dfrac{\vdash A}{\vdash [\![K]\!]A} & \text{(nec)} \\ \vdash [\![K]\!](A \multimap B) \multimap [\![K]\!]A \multimap [\![K]\!]B & \text{(K)} \\ \vdash [\![K]\!]A \multimap A & \text{(T)} \\ \vdash [\![K]\!]A \multimap [\![K]\!]([\![K]\!]A) & \text{(4)} \end{array}$$

Because we interpret $[\![K]\!]A$ as potential knowledge of A, and not immediate knowledge of A, we are justified in assuming that each principal K is capable of knowing all theorems (nec) and of reasoning logically from his potential knowledge (K). If a principal K can know (contrast with belief) a fact, then that fact will be provably true (T). Finally, K should be able to reflect upon his own knowledge (4).

Possession: $[K]A^-$. We also borrow the modality $[K]A^-$ (appropriately polarized) from Garg *et al.* [9] to model principals' possessions. $[K]A^-$ is read "principal K can possess resource A^-." Possession, like knowledge, is private. But, unlike knowledge, possessions are expendable. For this reason, $[K]A^-$ is the linear form of $[\![K]\!]A^-$, and also an S4 modality.

Monad: $\{A^+\}$. As alluded to in the introduction, we include a monad (lax modality) [6, 10] to mark the main branch of a derivation, following a related use in LolliMon [7].

2.2 Judgments

Following Martin-Löf's philosophy [13], we now present the judgments that compose our sequent calculus. There are three forms of sequent:

$$\begin{array}{ll} \text{Neutral} & \Gamma; \Delta \vdash J \\ \text{Right Focusing} & \Gamma; \Delta \vdash [A^+] \\ \text{Left Focusing} & \Gamma; \Delta, [A^-] \vdash J \end{array}$$

Neutral sequents permit invertible left and right rules to be applied, and allow transitions to right and left focusing sequents. Right and left focusing sequents chain non-invertible right and left rules together, respectively. The polarized syntax was chosen so that propositions with non-invertible right and left rules are positive and negative, respectively. Thus, we need only consider right focusing for positive propositions and left focusing for negative propositions.

Both unrestricted assumptions, Γ, and linear assumptions, Δ, are permitted:

$$\begin{array}{ll} \text{Unrestricted Assumptions} & \Gamma ::= \cdot \mid \Gamma, A^- \ \mathsf{valid} \mid \Gamma, K \ \mathsf{knows} \ A^- \\ \text{Linear Assumptions} & \Delta ::= \cdot \mid \Delta, A^+ \ \mathsf{hyp} \mid \Delta, K \ \mathsf{has} \ A^- \end{array}$$

The judgments A^- valid and K knows A^- represent assumptions about the existence of resource A^- and K's knowledge of A^-, respectively. They are unrestricted in the sense that they may be used any number of times (including zero) in the sequent's derivation. The single-use counterparts to validity and knowledge are the judgments A^+ hyp and K has A^-: they must be used exactly once in the sequent's derivation. We overload '$\cdot$' to stand for empty contexts of unrestricted and linear assumptions.

There are three forms of consequents J: A^- true represents the existence of resource A^-; K affirms A^+ is the judgmental form of $\langle K \rangle A^+$; and A^+ lax is the judgmental form of $\{A^+\}$.

$$\text{Consequents} \quad J ::= A^- \ \mathsf{true} \mid K \ \mathsf{affirms} \ A^+ \mid A^+ \ \mathsf{lax}$$

Note that we will typically omit the judgment labels valid, hyp, and true for the sake of brevity; the intended labels can always be inferred from the context.

2.3 Inference Rules

The full set of inference rules for the weakly focused sequent calculus is given in Fig. 1. However, we now describe a few rules in detail.

`atom`$^-$ *Rule.* When p^- is under left focus, the consequent must be p^-: there is no other left focal rule for negative atoms.

$\multimap_L$ *Rule.* When $A^+ \multimap B^-$ is under left focus, the available resources Δ_1, Δ_2 must be split, with Δ_1 used to prove the precondition A^+ under right focus. We are then justified in assuming B^-, and must prove the consequent J from Δ_2 and B^-; left focus is retained on B^-.

Initial Sequents

$$\frac{}{\Gamma; p^+ \vdash [p^+]}\ \mathtt{atom}^+ \qquad \frac{}{\Gamma; [p^-] \vdash p^-}\ \mathtt{atom}^-$$

Positive Connectives

$$\frac{\Gamma; \Delta_1 \vdash [A^+] \quad \Gamma; \Delta_2 \vdash [B^+]}{\Gamma; \Delta_1, \Delta_2 \vdash [A^+ \otimes B^+]}\ \otimes_R \qquad \frac{\Gamma; \Delta, A^+, B^+ \vdash J}{\Gamma; \Delta, A^+ \otimes B^+ \vdash J}\ \otimes_L$$

$$\frac{}{\Gamma; \cdot \vdash [\mathbf{1}]}\ \mathbf{1}_R \qquad \frac{\Gamma; \Delta \vdash J}{\Gamma; \Delta, \mathbf{1} \vdash J}\ \mathbf{1}_L$$

$$\frac{\Gamma; \Delta \vdash [[t/x]A^+]}{\Gamma; \Delta \vdash [\exists x{:}\tau.A^+]}\ \exists_R \qquad \frac{\Gamma; \Delta, [a/x]A^+ \vdash J}{\Gamma; \Delta, \exists x{:}\tau.A^+ \vdash J}\ \exists^a_L$$

$$\frac{\Gamma, A^-; \Delta, [A^-] \vdash J}{\Gamma, A^-; \Delta \vdash J}\ \mathtt{copy} \qquad \frac{\Gamma; \cdot \vdash A^-}{\Gamma; \cdot \vdash [!A^-]}\ !_R \qquad \frac{\Gamma, A^-; \Delta \vdash J}{\Gamma; \Delta, !A^- \vdash J}\ !_L$$

$$\frac{\Gamma; \Delta, [A^-] \vdash J}{\Gamma; \Delta, K\ \mathsf{has}\ A^- \vdash J}\ \mathsf{has}_L \qquad \frac{\Gamma|_K; \Delta|_K \vdash A^-}{\Gamma; \Delta|_K \vdash [[K]A^-]}\ []_R \qquad \frac{\Gamma; \Delta, K\ \mathsf{has}\ A^- \vdash J}{\Gamma; \Delta, [K]A^- \vdash J}\ []_L$$

$$\frac{\Gamma, K\ \mathsf{knows}\ A^-; \Delta, [A^-] \vdash J}{\Gamma, K\ \mathsf{knows}\ A^-; \Delta \vdash J}\ \mathsf{knows}_L \qquad \frac{\Gamma|_K; \cdot \vdash A^-}{\Gamma; \cdot \vdash [[\![K]\!]A^-]}\ [\![]\!]_R$$

$$\frac{\Gamma, K\ \mathsf{knows}\ A^-; \Delta \vdash J}{\Gamma; \Delta, [\![K]\!]A^- \vdash J}\ [\![]\!]_L$$

$$\frac{\Gamma; \Delta \vdash A^-}{\Gamma; \Delta \vdash [A^-]}\ \mathtt{blur} \qquad \frac{\Gamma; \Delta, [A^-] \vdash J}{\Gamma; \Delta, A^- \vdash J}\ \mathtt{lfoc}$$

Negative Connectives

$$\frac{\Gamma; \Delta, A^+ \vdash B^-}{\Gamma; \Delta \vdash A^+ \multimap B^-}\ \multimap_R \qquad \frac{\Gamma; \Delta_1 \vdash [A^+] \quad \Gamma; \Delta_2, [B^-] \vdash J}{\Gamma; \Delta_1, \Delta_2, [A^+ \multimap B^-] \vdash J}\ \multimap_L$$

$$\frac{\Gamma; \Delta \vdash [a/x]A^-}{\Gamma; \Delta \vdash \forall x{:}\tau.A^-}\ \forall^a_R \qquad \frac{\Gamma; \Delta, [[t/x]A^-] \vdash J}{\Gamma; \Delta, [\forall x{:}\tau.A^-] \vdash J}\ \forall_L$$

$$\frac{\Gamma; \Delta \vdash [A^+]}{\Gamma; \Delta \vdash K\ \mathsf{affirms}\ A^+}\ \mathsf{affirms}_R \qquad \frac{\Gamma; \Delta \vdash K\ \mathsf{affirms}\ A^+}{\Gamma; \Delta \vdash \langle K\rangle A^+}\ \langle\rangle_R$$

$$\frac{\Gamma; \Delta, A^+ \vdash K\ \mathsf{affirms}\ C^+}{\Gamma; \Delta, [\langle K\rangle A^+] \vdash K\ \mathsf{affirms}\ C^+}\ \langle\rangle_L$$

$$\frac{\Gamma; \Delta \vdash [A^+]}{\Gamma; \Delta \vdash A^+\ \mathsf{lax}}\ \mathsf{lax}_R \qquad \frac{\Gamma; \Delta \vdash A^+\ \mathsf{lax}}{\Gamma; \Delta \vdash \{A^+\}}\ \{\}_R \qquad \frac{\Gamma; \Delta, A^+ \vdash C^+\ \mathsf{lax}}{\Gamma; \Delta, [\{A^+\}] \vdash C^+\ \mathsf{lax}}\ \{\}_L$$

Fig. 1. Inference rules for the weakly focused sequent calculus

$$
\begin{array}{rl}
(\cdot)|_K = \cdot & \qquad (\cdot)|_K = \cdot \\
(\Delta, A^+\ \mathsf{hyp})|_K = \Delta|_K & \qquad (\Gamma, A^-\ \mathsf{valid})|_K = \Gamma|_K \\
(\Delta, K\ \mathsf{has}\ A^-)|_K = \Delta|_K, K\ \mathsf{has}\ A^- & \qquad (\Gamma, K\ \mathsf{knows}\ A^-)|_K = \Gamma|_K, K\ \mathsf{knows}\ A^- \\
(\Delta, L\ \mathsf{has}\ A^-)|_K = \Delta|_K \text{ if } L \neq K & \qquad (\Gamma, L\ \mathsf{knows}\ A^-)|_K = \Gamma|_K \text{ if } L \neq K
\end{array}
$$

Fig. 2. Restriction operator $|_K$ used in Fig. 1

has_L, $[]_R$, *and* $[]_L$ *Rules.* We can use any possession $K\ \mathsf{has}\ A^-$ by left focusing on its component A^- because the possession represents ownership of an existing resource A^- (has_L rule). To establish $[K]A^-$, i.e., that K possesses A^-, we must show A^- using only K's knowledge and possessions, ensured by the restriction operator $|_K$ on the assumptions ($[]_R$ rule). Finally, $[K]A^-$ is the propositional form of the judgment $K\ \mathsf{has}\ A^-$ ($[]_L$ rule).

$\{\}_L$ *Rule.* When the monad $\{A^+\}$ is under left focus, we are justified in assuming A^+ and blurring the focus, provided that the consequent is a lax judgment. This proviso (and the existence of no other left focal rule for $\{A^+\}$) implies that assumptions with monadic heads may only be used under lax consequents. Moreover, there is no other rule that terminates left focusing with a lax consequent.

These properties will be crucial to our approach. By careful construction of the formal policies and semantic actions to ensure that the lax judgment appears only on the main branch of the derivation and that only semantic actions have monadic heads, we can ensure that semantic effects only occur along the main branch. This will keep the resulting rewrite sequence sensible and further reduce the space of available derivations, making it easier to prove properties of systems.

2.4 Metatheory

In our sequent calculus, the meanings of the connectives are given entirely by their left and right rules, along with the necessary judgmental rules. We must ensure that these rules respect the meaning of a sequent, consisting of two properties: the admissibility of cut and identity principles. The statements and proofs of these properties are left to a companion technical report [14].

3 Notions of State and Rewrite Step

Having presented the inference rules of our sequent calculus, we can now make formal the notions of state and rewrite step alluded to in the introduction.

Like Cervesato and Scedrov [1], we define a state as a pair of contexts $\Gamma; \Delta$. However, we make the additional restriction that all possible invertible left rules have been applied:

Definition 1 (State). *A pair of contexts $\Gamma; \Delta$ is a* state *if and only if Δ fits the following grammar:*

$$\Delta ::= \cdot \mid \Delta, A^-\ \mathsf{hyp} \mid \Delta, p^+\ \mathsf{hyp} \mid \Delta, K\ \mathsf{has}\ A^-$$

A rewrite step occurs by using an assumption to transform the state:

Definition 2 (Rewrite Step). *The rewrite step $\Gamma; \Delta \longrightarrow \Gamma'; \Delta'$ holds if and only if there exists a derivation of the form*

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma'; \Delta' \vdash C^+ \ \mathsf{lax}}}{\vdots}}{\Gamma; \Delta_2, A_2^+ \ \mathsf{hyp} \vdash C^+ \ \mathsf{lax}} \ \{\}_L}{\Gamma; \Delta_2, [\{A_2^+\}] \vdash C^+ \ \mathsf{lax}}$$

$$\vdots$$

$$\dfrac{\Gamma; \Delta_1, [A_1^-] \vdash C^+ \ \mathsf{lax}}{\Gamma; \Delta \vdash C^+ \ \mathsf{lax}} \ *$$

parametric in C^+, where the rule marked $*$ is a rule transitioning from a neutral sequent to a left focused sequent ($\texttt{lfoc}$, $\texttt{copy}$, has_L *or* knows_L*), and the subderivation above $\{\}_L$ uses only invertible left rules ($\otimes_L$, $\mathbf{1}_L$, $\exists^a_L$, $!_L$, $[]_L$, or $[\![]\!]_L$). Furthermore, we require both $\Gamma; \Delta$ and $\Gamma'; \Delta'$ to be states.*

Because the rules above $\{\}_L$ are invertible and represent a don't-care nondeterminism, the resulting rewrite sequence is still sound and nondeterministically complete. We could make this precise with formal proof, but we elide these details to focus on other aspects more germane to the main thread of this paper.

Finally, we would like to reiterate a salient point about this definition. The main branch of this derivation is parametric in $C^+ \ \mathsf{lax}$. This monadic judgment identifies the main branch, and forces the assumption, A_1^-, under focus to have a monadic head, $\{A_2^+\}$. If we ensure that only semantic actions have monadic heads, then rewrite steps will contain only the semantic effects, as desired.

4 File System Example

In this section, we present a formal description of a simple file system and prove two properties about its actions. The file system supports four operations: create, read, write, and delete. Files are tagged with versions. Only the current version of a file is readable, but the file system maintains records of all previous versions' contents. For simplicity, we assume that there is no directory structure.

4.1 Policies and Semantic Actions

The formal policies and semantic actions are given in Fig. 3. For clarity, we omit type annotations in the quantifiers, and omit polarity annotations on atoms since all atoms in this example have negative polarity.

Operations. The four operations are represented by the grammar of terms in Fig. 3. We separate the *create* operation from those operations O that act on existing files and factor out the filename F with *onfile* to facilitate a clean formulation of the `owner` and `delegate` policies. The *write* operation takes an argument S that is the string to be written.

Operations

$$O^* ::= \mathit{create} \mid \mathit{onfile}(F, O) \qquad O ::= \mathit{read} \mid \mathit{write}(S) \mid \mathit{delete}$$

Policies

$$\texttt{maycreate} : \langle \mathsf{fs} \rangle (\forall K.\ \mathit{user}(K) \multimap \mathit{may}(K, \mathit{create}))$$

$$\texttt{owner} : \langle \mathsf{fs} \rangle (\forall K. \forall F. \forall O.\ \mathit{owns}(K, F) \multimap \mathit{may}(K, \mathit{onfile}(F, O)))$$

$$\begin{aligned}\texttt{delegate} : \langle \mathsf{fs} \rangle (\forall K. \forall L. \forall F. \forall O.\ & \langle K \rangle \mathit{may}(L, \mathit{onfile}(F, O)) \otimes \mathit{owns}(K, F) \multimap \\ & \mathit{may}(L, \mathit{onfile}(F, O)))\end{aligned}$$

Semantic Actions

$$\begin{aligned}\texttt{create} : \forall K.\ & \langle K \rangle \mathit{do}(K, \mathit{create}) \otimes \langle \mathsf{fs} \rangle \mathit{may}(K, \mathit{create}) \multimap \\ & \{\exists f. \exists t.\ !\langle \mathsf{fs} \rangle \mathit{owns}(K, f) \otimes \\ & \quad [\mathsf{fs}] \mathit{current}(f, t) \otimes \\ & \quad [\![\mathsf{fs}]\!] \mathit{contents}(f, t, \epsilon) \otimes \\ & \quad [\![K]\!] \mathit{contents}(f, t, \epsilon)\}\end{aligned}$$

$$\begin{aligned}\texttt{read} : \forall K. \forall F. \forall T. \forall S.\ & \langle K \rangle \mathit{do}(K, \mathit{onfile}(F, \mathit{read})) \otimes \langle \mathsf{fs} \rangle \mathit{may}(K, \mathit{onfile}(F, \mathit{read})) \otimes \\ & [\mathsf{fs}] \mathit{current}(F, T) \otimes [\![\mathsf{fs}]\!] \mathit{contents}(F, T, S) \multimap \\ & \quad \{[\![K]\!] \mathit{contents}(F, T, S) \otimes \\ & \quad\ [\mathsf{fs}] \mathit{current}(F, T)\}\end{aligned}$$

$$\begin{aligned}\texttt{write} : \forall K. \forall F. \forall S. \forall T.\ & \langle K \rangle \mathit{do}(K, \mathit{onfile}(F, \mathit{write}(S))) \otimes \\ & \langle \mathsf{fs} \rangle \mathit{may}(K, \mathit{onfile}(F, \mathit{write}(S))) \otimes [\mathsf{fs}] \mathit{current}(F, T) \multimap \\ & \quad \{\exists t.\ [\mathsf{fs}] \mathit{current}(F, t) \otimes \\ & \quad\quad [\![\mathsf{fs}]\!] \mathit{contents}(F, t, S) \otimes \\ & \quad\quad [\![K]\!] \mathit{contents}(F, t, S)\}\end{aligned}$$

$$\begin{aligned}\texttt{delete} : \forall K. \forall F. \forall T.\ & \langle K \rangle \mathit{do}(K, \mathit{onfile}(F, \mathit{delete})) \otimes \langle \mathsf{fs} \rangle \mathit{may}(K, \mathit{onfile}(F, \mathit{delete})) \otimes \\ & [\mathsf{fs}] \mathit{current}(F, T) \multimap \\ & \quad \{\mathbf{1}\}\end{aligned}$$

$$\texttt{environment} : \forall K. \forall O^*.\ \{\langle K \rangle \mathit{do}(K, O^*)\}$$

Fig. 3. File system policies and semantic actions

Policies. The first policy, `maycreate`, states the file system fs's affirmation that any principal K who is a valid user may create files. The second, `owner`, states fs's affirmation that any principal K that owns file F may perform operation O on F. The third policy, `delegate`, states fs's affirmation that any principal L may perform operation O on file F provided F's owner affirms this permission.

Semantic Actions. There are five semantic actions, one for each of the four operations and an additional action to simulate the environment.

`create` *Action.* `create` states that if: a principal K issues a command to create a file ($\langle K\rangle do(K, create)$); and the file system affirms that K may create a file ($\langle \mathsf{fs}\rangle may(K, create)$); then a fresh filename f and fresh version tag t are generated ($\exists f.\exists t.$), and: the file system affirms that K is a permanent owner of f ($!\langle \mathsf{fs}\rangle owns(K, f)$); the file system records t as the current version tag for f ($[\mathsf{fs}]current(f, t)$); and the file system and K learn that the contents of version t of file f are the empty string ϵ ($[\![\mathsf{fs}]\!]contents(f, t, \epsilon)$ and $[\![K]\!]contents(f, t, \epsilon)$).

Observe the use of a monad, indicated by $\{\}$, around the postcondition of this action. This is typical of semantic actions in our framework: the monad marks them so that they may only be used along the main branch of a rewriting derivation and must therefore appear in the rewrite sequence.

`read` *Action.* `read` states that if: principal K issues a read request for file F; the file system permits K to read F; the current version tag of F is T; and the file system knows that the contents of version T of file F are string S; then K learns that the contents of version T of file F are S. Moreover, because $[\mathsf{fs}]current(F, T)$ is linear and therefore consumed by the use of $\multimap$, this information must be regenerated.

Note that, because the $\forall F.$ quantifier will be instantiated at runtime, this action can be used with filenames freshly generated by previous `create` actions. This observation also holds for the remaining actions.

`write` *Action.* `write` is roughly analogous to `create`: instead of writing the empty contents, K chooses a string S for the new contents. However, because writing is only sensible for existing files, we require the file to have some version tag T. Linearity of $[\mathsf{fs}]current(F, T)$ causes it to be consumed by the use of $\multimap$, critically ensuring that T is no longer considered the current version tag. After the write operation, the current version is the new tag t, since new contents have been written.

`delete` *Action.* The `delete` action consumes fs's record of the current version tag of F. As $\mathbf{1}$ is the empty set of resources, there are no outputs to this action. Intuitively, because a $[\mathsf{fs}]current(F, T)$ assumption is required in the `read` and `write` actions and we maintain the invariant of keeping only one $[\mathsf{fs}]current(F, *)$ assumption, consuming the record of F's current version prevents further reads and writes to F. This effectively deletes F. This intuition is made precise in the following Theorem 2.

`environment` *Action.* To simulate the environment outside the file system, we use an action which, when taken, introduces a $\langle K\rangle do(K, O^*)$. This corresponds to users' ability to issue an arbitrary request at any time during the system's execution. Since the act of issuing a request is a semantic effect, this action is appropriately confined to the monad.

If we wished to reason about the effects of a *particular* sequence of requests, we would need a more detailed model of the environment. But, for the kind of properties that interest us, this action suffices. In fact, being agnostic about the

specifics of the environment's behavior strengthens our theorems: these security properties will hold regardless of the sequence of requests made.

4.2 Proving Properties of the File System

Before proving a few properties of the file system, we must have a definition of state specific to the file system. A file system state is a generic state (Definition 1), but is refined by enumerating the exact forms of permissible assumptions:

Definition 3 (File System State). *A state of the file system is a pair of contexts $\Gamma; \Delta$, satisfying:*

1. *Each assumption in Γ has one of the forms: (i) a policy from Fig. 3; (ii) a semantic action from Fig. 3; (iii)* fs knows $\mathit{contents}(F, T, S)$ *or* K knows $\mathit{contents}(F, T, S)$*; (iv)* $\langle \mathsf{fs} \rangle \mathit{user}(K)$*; or (v)* $\langle \mathsf{fs} \rangle \mathit{owns}(K, F)$*.*
2. *Each assumption in Δ has one of the forms: (i)* fs has $\mathit{current}(F, T)$*; (ii)* $\langle K \rangle \mathit{do}(K, O^*)$*; or (iii)* $\langle K \rangle \mathit{may}(L, \mathit{onfile}(F, O))$*.*
3. *For each file F, there exists at most one tag T such that Δ contains* fs has $\mathit{current}(F, T)$*.*

Using this kind of specific definition of state and the definition of a rewrite step (Definition 2), we can construct derived inference rules [15] for each left focusing phase that might begin a rewrite derivation:

Theorem 1 (Rewrite Step Schemata). *Each rewrite step from a file system state has exactly one of the following forms:*

1. $\Gamma; \langle K \rangle \mathit{do}(K, \mathit{create}), \Delta_1, \Delta_2$
   $\longrightarrow \Gamma, \langle \mathsf{fs} \rangle \mathit{owns}(K, f), \mathsf{fs}$ knows $\mathit{contents}(f, t, \epsilon), K$ knows $\mathit{contents}(f, t, \epsilon)$;
   Δ_2, fs has $\mathit{current}(f, t)$
   such that $\Gamma; \Delta_1 \vdash \langle \mathsf{fs} \rangle \mathit{may}(K, \mathit{create})$*, for some fresh filename f and tag t.*
2. Γ, fs knows $\mathit{contents}(F, T, S)$;
   $\langle K \rangle \mathit{do}(K, \mathit{onfile}(F, \mathit{read})), \Delta_1, \mathsf{fs}$ has $\mathit{current}(F, T), \Delta_2$
   $\longrightarrow \Gamma, \mathsf{fs}$ knows $\mathit{contents}(F, T, S), K$ knows $\mathit{contents}(F, T, S)$;
   Δ_2, fs has $\mathit{current}(F, T)$
   such that Γ, fs knows $\mathit{contents}(F, T, S); \Delta_1 \vdash \langle \mathsf{fs} \rangle \mathit{may}(K, \mathit{onfile}(F, \mathit{read}))$.
3. $\Gamma; \langle K \rangle \mathit{do}(K, \mathit{onfile}(F, \mathit{write}(S))), \Delta_1, \mathsf{fs}$ has $\mathit{current}(F, T), \Delta_2$
   $\longrightarrow \Gamma, \mathsf{fs}$ knows $\mathit{contents}(F, t, S), K$ knows $\mathit{contents}(F, t, S)$;
   Δ_2, fs has $\mathit{current}(F, t)$
   such that $\Gamma; \Delta_1 \vdash \langle \mathsf{fs} \rangle \mathit{may}(K, \mathit{onfile}(F, \mathit{write}(S)))$*, for some fresh tag t.*
4. $\Gamma; \langle K \rangle \mathit{do}(K, \mathit{onfile}(F, \mathit{delete})), \Delta_1, \mathsf{fs}$ has $\mathit{current}(F, T), \Delta_2 \longrightarrow \Gamma; \Delta_2$
   such that $\Gamma; \Delta_1 \vdash \langle \mathsf{fs} \rangle \mathit{may}(K, \mathit{onfile}(F, \mathit{delete}))$.
5. $\Gamma; \Delta \longrightarrow \Gamma; \Delta, \langle K \rangle \mathit{do}(K, O^*)$.

Moreover, the conclusions of these rewrite steps are file system states.

Proof. By Definition 2, a rewrite derivation must begin by left focusing on an assumption with a monadic head. By construction, the only file system state assumptions with monadic heads are the five semantic actions. We show the case for `delete`; the others are similar.

Case. Left focusing on `delete` yields the derived rule

$$\frac{\begin{array}{cc}\Gamma;\Delta_1' \vdash \langle K\rangle do(K, onfile(F, delete)) & \Gamma|_{\mathsf{fs}};\Delta_2'|_{\mathsf{fs}} \vdash current(F,T) \\ \Gamma;\Delta_1 \vdash \langle \mathsf{fs}\rangle may(K, onfile(F, delete)) & \Gamma;\Delta_2, \mathbf{1} \vdash C^+\ \mathsf{lax}\end{array}}{\Gamma;\Delta_1', \Delta_1, \Delta_2'|_{\mathsf{fs}}, \Delta_2 \vdash C^+\ \mathsf{lax}}$$

We appeal to Lemma 1 (below) to show that $\Delta_1' = \langle K\rangle do(K, onfile(F, delete))$. Also, given the permissible file system state assumptions, the third premise could be derived only if $\Delta_2'|_{\mathsf{fs}} = \mathsf{fs}\ \mathsf{has}\ current(F,T)$. Finally, because the $\mathbf{1}_L$ rule is invertible, we may drop $\mathbf{1}$ from the fourth premise without loss of generality. We conclude that rewrite step 4 arises from focusing on `delete`. □

Lemma 1. *If $\Gamma;\Delta$ is a file system state such that $\Gamma;\Delta \vdash \langle K\rangle do(K, O^*)$, then $\Delta = \langle K\rangle do(K, O^*)$.*

Proof. By structural induction on the given derivation, where we generalize the lemma so that Δ may contain state affirmation assumptions with the outer $\langle\cdot\rangle$ removed (e.g., $\forall K.\ user(K) \multimap may(K, create)$ from `maycreate`). We also generalize the given derivation's consequent to $K\ \mathsf{affirms}\ do(K, O^*)$ and only require either $\Delta = \langle K\rangle do(K, O^*)$ or $\Delta = do(K, O^*)$. This generalization implies the original lemma by the invertibility of the $\langle\rangle_R$ rule. □

Note that the proof of Theorem 1 is made considerably easier by focusing. Its chaining of non-invertible rules allows us to go from an assumption under focus to a derived rule. Also note that, from now on, we will implicitly assume, for brevity, that all pairs $\Gamma;\Delta$ that begin a rewrite step are file system states.

Because Theorem 1 gives the schemata for file system rewrite steps, we can perform an easy case analysis to show that principals do not learn file contents unless allowed by the policy to do so:

Theorem 2 (Knowledge Safety). *If $\Gamma;\Delta \longrightarrow \Gamma', K\,\mathsf{knows}\,contents(F,T,S);\Delta'$, then either $K\ \mathsf{knows}\ contents(F,T,S) \in \Gamma$ or the step was a `create`, `read`, or `write` step (1–3 in Theorem 1) triggered by K and permitted by the policy (as evidenced by the $\Gamma;\Delta_1 \vdash \langle \mathsf{fs}\rangle may(K, O^*)$ derivation from the step).*

By a simple induction, the previous theorem yields the following corollary that states that principals may not learn the contents of deleted files, a nontrivial result enabled by our rewriting system.

Corollary 1. *Let $\longrightarrow^*$ be the reflexive, transitive closure of $\longrightarrow$. If*

$$\begin{array}{ll} & \Gamma;\langle K\rangle do(K, onfile(F, delete)), \Delta_1, \mathsf{fs}\ \mathsf{has}\ current(F,T), \Delta_2 \\ \longrightarrow & \Gamma;\Delta_2 \\ \longrightarrow^* & \Gamma', L\ \mathsf{knows}\ contents(F,T',S);\Delta' \end{array}$$

and $\Gamma;\Delta_1 \vdash \langle \mathsf{fs}\rangle may(K, onfile(F, delete))$, then $L\ \mathsf{knows}\ contents(F,T',S) \in \Gamma$.

Finally, we can also prove that if the record of a file's current version is consumed during a rewrite step, the step was a delete triggered by some principal permitted to do so: deletes cannot happen spontaneously.

Theorem 3 (Delete Safety). *If* $\Gamma; \Delta, \mathsf{fs}\,\mathsf{has}\,\mathit{current}(F,T) \longrightarrow \Gamma'; \Delta'$ *and there exists no version tag* T' *such that* $\mathsf{fs}\,\mathsf{has}\,\mathit{current}(F,T') \in \Delta'$*, then the rewrite step was a* `delete` *step (4 in Theorem 1).*

Proof. By Theorem 1, we know all possible forms of the given step. We case analyze these forms (with appeals to Lemma 2) to obtain the result. □

In principle, $\mathsf{fs}\ \mathsf{has}\ \mathit{current}(F,T)$ could be used in the policy derivation, $\Gamma; \Delta_1 \vdash \langle \mathsf{fs} \rangle \mathit{may}(K, O^*)$, of the given rewrite step. However, we can prove a form of strengthening in the following lemma, which shows such uses to be impossible.

Lemma 2 (Possession Strengthening). *If* $\Gamma; \Delta$ *is a file system state and* $\Gamma; \Delta \vdash \langle \mathsf{fs} \rangle \mathit{may}(K, O^*)$*, then there exists no file* F *and version tag* T *such that* $\mathsf{fs}\ \mathsf{has}\ \mathit{current}(F,T) \in \Delta$*.*

Proof. By structural induction on the given derivation. To allow the induction to go through, we use a generalization similar to the one in Lemma 1. □

Note that this lemma only holds because the semantic concepts, including possession, are stratified from the policies. This stratification is evidenced and assisted by the controlled use of the monad to isolate such semantic effects.

5 Related Work

Our work is heavily inspired by a somewhat informal treatment of reasoning about course registration and banking systems in prior work on a linear logic of affirmation and knowledge by Garg *et al.* [9]. Indeed, our method affirmatively resolves their conjecture that the informal approach can be generalized via focusing and forward chaining. However, our work differs in that it stratifies policies from their semantic effects.

As previously discussed, other closely related works are the rewriting system ω [1] and the linear logic programming language LolliMon [7], from which we borrow the idea of rewriting and the use of a monad, respectively.

MultiSet Rewriting (MSR) [16–18] is a predecessor of ω that has been used to analyze security protocols, notably to prove a correctness result for Kerberos 5 [19] and prove, using data access specification rules, that the Dolev-Yao intruder can indeed emulate an arbitrary symbolic adversary [20]. MSR differs from our logic in that it uses a lower level of abstraction, reasoning about atomic certificates rather than policies, and does not support principals' knowledge or local state.

Other approaches to the problem of proving properties of protocols include finite model checking and automated theorem proving. For example, Clarke *et al.* [21] describe a logic of knowledge suitable for model checking commerce protocols, and Paulson [22] uses inductive definitions of traces and the Isabelle prover to verify protocols. Neither work, as presented, fully unifies principals' policies and their knowledge: both lack a means of specifying principals' policies,

and Paulson's treats principals' knowledge only implicitly. It would be interesting to consider whether these lines of work could be extended to cover policies.

With its emphasis on knowledge and targeted communication, the authorization language DKAL [23] is also related to our work. However, in DKAL, knowledge is fundamentally linked to communication, and possession (linear knowledge) is not expressible. Also, it is not clear to us if properties of systems, such as those in our example, can be proven using DKAL.

6 Conclusion

In this paper, we have extended the rewriting system ω and the linear logic programming language LolliMon with modalities for modeling principals' policies, knowledge, and possessions. By using a monad, we were able to confine all semantic effects, and only semantic effects, to the rewrite sequence. We showed our logic to be well defined by proving the admissibility of cut and identity. Finally, we established knowledge and delete safety theorems for a hypothetical file system, demonstrating the utility of our approach. Having a focused logic simplified these proofs by eliminating nondeterminism in the derivations.

This line of work provides rich opportunities for future investigation. We would like to extend the logic to support obligations at the semantic level. It would also be instructive to test our logic against several real-world case studies. As a long-term goal, we intend to explore dynamic logic [24] as a possible means for mechanically verifying the properties proved by our method (e.g., Theorems 2 and 3). Another long-term goal is to investigate compilation of semantic actions to executable code, similar in spirit to work on compiling session abstractions [25].

References

1. Cervesato, I., Scedrov, A.: Relating state-based and process-based concurrency through linear logic. Information and Computation (2009) To appear.
2. Girard, J.Y.: Linear logic. Theoretical Computer Science **50** (1987) 1–102
3. Abadi, M., Burrows, M., Lampson, B., Plotkin, G.: A calculus for access control in distributed systems. ACM Transactions on Programming Languages and Systems **15** (1993) 706–734
4. Appel, A.W., Felten, E.W.: Proof-carrying authentication. In: Proceedings of the 6th Conference on Computer and Communications Security. (1999) 52–62
5. Bauer, L.: Access Control for the Web via Proof-Carrying Authorization. PhD thesis, Princeton University (November 2003)
6. Moggi, E.: Notions of computation and monads. Information and Computation **93**(1) (1991) 55–92
7. López, P., Pfenning, F., Polakow, J., Watkins, K.: Monadic concurrent linear logic programming. In: Proceedings of the 7th International Symposium on Principles and Practice of Declarative Programming. (2005) 35–46
8. Andreoli, J.M.: Logic programming with focusing proofs in linear logic. Journal of Logic and Computation **2**(3) (1992) 297–347

9. Garg, D., Bauer, L., Bowers, K., Pfenning, F., Reiter, M.: A linear logic of affirmation and knowledge. In: Proceedings of the 11th European Symposium on Research in Computer Security. (2006) 297–312
10. Fairtlough, M., Mendler, M.: Propositional lax logic. Information and Computation **137**(1) (August 1997) 1–33
11. Abadi, M.: Variations in access control logic. In: Ninth International Conference on Deontic Logic in Computer Science. (2008) 96–109
12. Abadi, M.: Access control in a core calculus of dependency. In: Proceedings of the 11th ACM SIGPLAN International Conference on Functional Programming. (2006) 263–273
13. Martin-Löf, P.: On the meanings of the logical constants and the justifications of the logical laws. Nordic Journal of Philosophical Logic **1**(1) (1996) 11–60
14. DeYoung, H., Pfenning, F.: Reasoning about the consequences of authorization policies in a linear epistemic logic. Technical Report CMU-CS-09-140, Carnegie Mellon University (July 2009)
15. Andreoli, J.M.: Focussing and proof construction. Annals of Pure and Applied Logic **107**(1) (2001) 131–163
16. Cervesato, I., Durgin, N., Lincoln, P., Mitchell, J., Scedrov, A.: A meta-notation for protocol analysis. In: Proceedings of the 12th IEEE Computer Security Foundations Workshop. (1999) 55–69
17. Cervesato, I.: Typed MSR: Syntax and examples. In: 1st International Workshop on Mathematical Methods, Models, and Architectures for Computer Networks Security. Volume 2052 of Springer-Verlag LNCS. (2001) 159–177
18. Bistarelli, S., Cervesato, I., Lenzini, G., Martinelli, F.: Relating multiset rewriting and process algebras for security protocol analysis. Journal of Computer Security **13**(1) (2005) 3–47
19. Cervesato, I., Jaggard, A.D., Scedrov, A., Tsay, J.K., Walstad, C.: Breaking and fixing public-key kerberos. Information and Computation **206** (2008) 402–424
20. Cervesato, I.: Data access specification and the most powerful symbolic attacker in MSR. In: Software Security – Theories and Systems, Mext-NSF-JSPS International Symposium. (2002) 384–416
21. Clarke, E.M., Jha, S., Marrero, W.: A machine checkable logic of knowledge for specifying security properties of electronic commerce protocols. In: 13th IEEE Symposium on Logic in Computer Science Workshop on Formal Methods and Security Protocols. (1998)
22. Paulson, L.C.: Proving properties of security protocols by induction. In: Proceedings of the 10th IEEE Computer Security Foundations Workshop. (1997) 70–83
23. Gurevich, Y., Neeman, I.: DKAL: Distributed-knowledge authorization language. In: Proceedings of the 21st IEEE Computer Security Foundations Symposium. (2008) 149–162
24. Pratt, V.: Semantical considerations on Floyd-Hoare logic. In: Proceedings of the 17th Annual IEEE Symposium on Foundations of Computer Science. (1976) 109–121
25. Corin, R., Deniélou, P.M., Fournet, C., Bhargavan, K., Leifer, J.: A secure compiler for session abstractions. Journal of Computer Security **16**(5) (2008) 573–636