SecPAL: Formalization and Extensions

Moritz Y. Becker
Microsoft Research, Cambridge CB3 0FB, United Kingdom
moritzb@microsoft.com

September 2009

Technical Report
MSR–TR–2009–127

Microsoft Research
Roger Needham Building
7 J.J. Thomson Avenue
Cambridge, CB3 0FB
United Kingdom

SecPAL: Formalization and Extensions

Moritz Y. Becker
Microsoft Research, Cambridge CB3 0FB, United Kingdom
moritzb@microsoft.com

September 2009

1 Introduction

This report presents an updated formalization of standard SecPAL as well as possible extensions to the language. For background, examples, general discussion and comparison with related work, please refer to previous publications on SecPAL [1, 2]. (Abductive evaluation of SecPAL is also discussed elsewhere [4, 5, 3].)

- Section 2 provides a specification of the core language that is slightly more formal than the one presented in [2, 1]. It provides a model-theoretic FOL-based semantics for compound queries, and a more compact proof system for ground atomic assertions that more closely reflects the intuition behind cansay_0.
- Section 3 adds guarded universal quantification to the query language, enabling queries such as $\forall f\ (A\ \mathsf{says}\ B\ \mathsf{can\ read}\ f\ \Rightarrow\ \neg A\ \mathsf{says}\ f\ \mathsf{is\ secret})$.
- Section 4 presents a relaxed safety condition based on input/output-modes of parameters to guarantee groundness of constraints during evaluation. Under this new safety condition, previously unsafe, but useful assertions such as $A\ \mathsf{says}\ B\ \mathsf{can\ read}\ f$ are IN/OUT-safe if the position after $\mathsf{can\ read}$ is declared as an input position.

  IN/OUT-safety can also be used to reduce evaluation complexity. For example, if it is known that the file system contains a large number of files, it may be advisable to declare the predicate positions in which file resources are placed as input positions. This ensures that these parameters do not contribute to excessive backtracking and the computation of large joins.
- Section 5 considers policies consisting of possibly infinitely many assertions that are retrieved dynamically at evaluation time by a so-called assertion provider. Such assertion providers are used for instance as an interface

between SecPAL and large relational databases. A safety condition is provided guaranteeing termination even if the assertion provider represents an infinite set of assertions.

- Section 6 extends SecPAL with hierarchical parameters such as file directory structures, URIs, strings with prefix ordering etc. Under the hierarchical semantics, a fact about one object automatically implies that the fact also holds for all descendants of that object.

2 Standard SecPAL

This section formalises the core language of SecPAL. The presentation differs from the one in [2] in a few details:

1. The treatment here is slightly more formal. Facts and constraints are treated as atoms from two separate signatures.
2. The canactas construct is omitted, as it has been of limited use in practice, and it can easily be encoded in a more generalized form using hierarchical parameters (see Section 6).
3. The proof system is presented in a simpler form. In particular, it reflects more directly the meaning of the construct for specifying delegation of authority with limited redelegation, cansay_0.
4. The semantics for compound queries is not defined as a proof system, but more directly as first order formulas on atomic assertions.

2.1 Preliminaries

The set of *free variables* of a syntactic phrase φ is given by $\mathbf{FV}(\varphi)$, defined in the usual manner (essentially as all variables occurring in the phrase apart from those that are in the scope of a quantifier). φ is *closed* iff it does not contain any free variables, and *ground* iff it does not contain any variables at all.

We choose a (generally domain-specific) first-order signature $\Sigma = (\mathbf{Const}, \mathbf{Pred})$, where $\mathbf{Const}$ only contains nullary function symbols (i.e., constants). We write $\mathbf{ar}(\varphi)$ for the arity of a predicate or function symbol. As usual, an *expression* e is a variable or a constant in $\mathbf{Const}$, and an *atom* is of the form $p(e_1, .., e_{\mathbf{ar}(p)})$, where $p \in \mathbf{Pred}$. Atoms are often written in infix notation (e.g. "x canread y").

Additionally, we choose a *constraint signature* $\Gamma = (\mathbf{Const}, \mathbf{Pred}_\Gamma)$ such that $\mathbf{Pred}_\Gamma \cap \mathbf{Pred} = \emptyset$. A *constraint* is a Γ-atom.[1] We further assume a unary

[1] In practice, constraint signatures may contain function symbols, and constraints may be constructed using logical connectives and quantifiers. However, for our theoretical treatment the above definition is sufficient, since any complex constraint formula can always be mapped to a single function-less atom.

relation $\models_\Gamma$ on ground[2] constraints c, and we say that c *is true* whenever $\models_\Gamma c$.

2.2 Syntax

The syntax for *facts* and *assertions* is defined below. Henceforth, we keep to the following conventions for symbols: x, y denote variables, E denotes a constant, e an expression, c a Γ-constraint, p a predicate, a an atom, f a fact, F a ground fact, α an assertion, and $\mathcal{A}$ a set of assertions.

$$
\begin{array}{lrcl}
\text{Fact} & f & ::= & a \\
 & & | & e \ \mathsf{cansay}\ f \\
 & & | & e \ \mathsf{cansay}_0\ f \\
\text{Assertion} & \alpha & ::= & e \ \mathsf{says}\ f \ \mathsf{if}\ f_1, ..., f_n \ \mathsf{where}\ c
\end{array}
$$

Let f be a fact. If it is of the form $p(\vec{e})$ is called *flat* and has *arity* $\mathbf{ar}(f) = \mathbf{ar}(p)$. Otherwise it is of the form $e \ \mathsf{cansay}\ f'$ or $e \ \mathsf{cansay}_0\ f'$, in which case it is called *nested* and has *arity* $\mathbf{ar}(f) = \mathbf{ar}(f') + 1$. We write $f[e_1, ...e, n]$ to denote an n-ary fact with parameters $e_1, ..., e_n$ (from left to right within the fact).

In an assertion $\alpha = \langle e \ \mathsf{says}\ f \ \mathsf{if}\ f_1, ..., f_n \ \mathsf{where}\ c\rangle$, e is the *issuer*, f the *head*, $f_1, ..., f_n$ the *body*, and c the *constraint* of α. The keyword if is omitted when $n = 0$; likewise, $\mathsf{where}\ c$ is omitted when $c = \mathsf{true}$. An assertion of the form $e \ \mathsf{says}\ f$ is called an *atomic assertion*.

The syntax of *queries* q is defined as follows.

$$
\begin{array}{lrcl}
\text{Query} & q & ::= & e \ \mathsf{says}\ f \\
 & & | & c \\
 & & | & \neg q \\
 & & | & q_1 \wedge q_2 \\
 & & | & q_1 \vee q_2 \\
 & & | & \exists x(q) \\
 & & | & \forall x(q_1 \Rightarrow q_2)
\end{array}
$$

Syntactic sugar. Let $\vec{x} = \langle x_1, ..., x_n\rangle$. We write $\exists\vec{x}(q)$ to denote $\exists x_1(...\exists x_n(q)...)$.

2.3 Proof semantics

The proof-theoretic semantics of SecPAL is given by an inference relation $\vdash$ defining judgements of the form $\mathcal{A} \vdash \alpha$ where α is a ground atomic assertion (Fig. 1). The proof system presented here is equivalent to the one in [2], but it is simpler as it does not have the depth flat in the premises. Also, the rule for cansay_0 expresses more directly the intention of the 0-subscript.

The inference relation is extended to general closed queries by interpreting them as formulas in first-order logic. Formally, let q be a closed query, $\mathcal{M}_A =$

[2]If we allowed explicit connectives and quantifiers, $\models_\Gamma$ would be a relation on *closed* constraints.

$$\text{(mdp)}\ \frac{\langle e \text{ says } f \text{ if } f_1, ..., f_n \text{ where } c\rangle \in \mathcal{A} \quad \models_\Gamma \gamma(c) \quad \forall i \in \{1, ..., n\} : \ \mathcal{A} \vdash \gamma(e \text{ says } f_i)}{\mathcal{A} \vdash \gamma(e \text{ says } f)}$$

$$\text{(cs)}\ \frac{\mathcal{A} \vdash E_1 \text{ says } E_2 \text{ cansay } F \quad \mathcal{A} \vdash E_2 \text{ says } F}{\mathcal{A} \vdash E_1 \text{ says } F} \qquad \text{(cs0)}\ \frac{\mathcal{A} \vdash E_1 \text{ says } E_2 \text{ cansay}_0 F \quad \mathcal{A}|_{E_2} \vdash E_2 \text{ says } F}{\mathcal{A} \vdash E_1 \text{ says } F}$$

Figure 1: Proof system for ground atomic assertions. Above, γ is a ground substitution (i.e., one that maps every variable to a constant), and $\mathcal{A}|_E$ denotes the set of all assertions in $\mathcal{A}$ with issuer E.

$\{\alpha \mid \mathcal{A} \vdash \alpha\}$ and $\mathcal{M}_\Gamma = \{c \mid \models_\Gamma c\}$. Then $\mathcal{A} \vdash q$ iff $\mathcal{M}_A \cup \mathcal{M}_\Gamma \models q$ in first order logic. If Q is a set of closed queries, we write $\mathcal{A} \vdash Q$ iff for all $q \in Q : \mathcal{A} \vdash q$.

3 Guarded Universal Quantification

This section adds universal quantification to the query language. The construct requires the quantified formula to have a *guard* that restricts the values of the quantified variables.

$$\begin{array}{lll} \text{Query} \quad q & ::= & ... \\ & | & \forall x(q_1 \Rightarrow q_2) \end{array}$$

Syntactic sugar We write $\forall \vec{x}(q_1 \Rightarrow q_2)$ to denote $\forall x_1(\mathsf{true} \Rightarrow ... \Rightarrow \forall x_n(q_1 \Rightarrow q_2)...)$.

4 IN/OUT-safety

This version of safety using In/Out-modes is a relaxed version of the simpler safety condition in [2], where all flat facts had implicit Out-parameters, and all nested facts In-parameters.

Given signature Σ, we fix a *mode map* $\mathbf{Mode}$ which is a function mapping each $p \in \mathbf{Pred}$ to a tuple in $\{\mathsf{In}, \mathsf{Out}\}^{\mathbf{ar}(p)}$.

We extend $\mathbf{Mode}$ to facts as follows. If f is an atom $p(e_1, ..., e_n)$, then $\mathbf{Mode}(f) = \mathbf{Mode}(p)$. If f is of the form $\langle e \text{ cansay } f'\rangle$ or $\langle e \text{ cansay}_0 \ f'\rangle$, then $\mathbf{Mode}(f) = \langle \mathsf{In}, ..., \mathsf{In}\rangle$ (where the tuple has the same arity as f). Let $f_0 = f[e_1, ..., e_n]$ be a fact with $\mathbf{Mode}(f_0) = \langle \mu_1, ..., \mu_n\rangle$. Then $\mathsf{Out}(f)$ ($\mathsf{In}(f)$) denotes the set of e_i such that e_i is a variable and $\mu_i = \mathsf{Out}$ ($\mu_i = \mathsf{In}$).

Definition 4.1. An assertion $\alpha = \langle e \text{ says } f \text{ if } f_1, ..., f_n \text{ where } c\rangle$ is In/Out-*safe* iff all of the following conditions hold:

$$\frac{\mathsf{In}(f) \subseteq I \quad O = I \cup \mathbf{FV}(e) \cup \mathsf{Out}(f)}{I \Vdash e \text{ says } f : O} \qquad \frac{\mathbf{FV}(c) \subseteq I}{I \Vdash c : I} \qquad \frac{\mathbf{FV}(q) \subseteq I}{I \Vdash \neg q : I}$$

$$\frac{I \Vdash q_1 : O_1 \quad I \Vdash q_2 : O_2}{I \Vdash q_1 \ \vee \ q_2 : O_1 \cap O_2} \qquad \frac{I \Vdash q_1 : O_1 \quad O_1 \Vdash q_2 : O_2}{I \Vdash q_1 \wedge q_2 : O_2}$$

$$\frac{I \Vdash q : O \quad x \notin I}{I \Vdash \exists x(q) : O - \{x\}} \qquad \frac{I \Vdash q_1 : O_1 \quad O_1 \Vdash q_2 : O_2 \quad \vec{x} \cap I = \emptyset \quad \vec{x} \subseteq O_1}{I \Vdash \forall \vec{x}(q_1 \Rightarrow q_2) : O_2 \setminus \vec{x}}$$

Figure 2: Query In/Out-safety rules.

1. $n \geq 1$ or $\mathbf{FV}(e) = \emptyset$,
2. $\mathsf{Out}(f) \subseteq \bigcup_{j=1}^{n} \mathsf{Out}(f_j)$,
3. for all body facts f_i: $\mathsf{In}(f_i) \subseteq \mathsf{In}(f) \cup \bigcup_{j=1}^{i-1} \mathsf{Out}(f_j)$,
4. $\mathbf{FV}(c) \subseteq \mathbf{FV}(e, f, f_1, ..., f_n)$.

A set of assertions $\mathcal{A}$ is In/Out-*safe* iff $\mathcal{A}$ is finite and all assertions in $\mathcal{A}$ are In/Out-safe.

Fig. 2 defines an inference relation $\Vdash$ that is used for defining query In/Out-safety.

Definition 4.2. A query q is In/Out-*safe* iff there exists a set of variables O such that $\emptyset \Vdash q : O$.

Theorem 4.3. All theorems about soundness, completeness, termination and complexity of the constrained Datalog translation and evaluation algorithm in [2] are still valid with the relaxed version of safety using In/Out-parameters.

Proof. The Datalog translation produces a In/Out-safe Datalog program as defined in [1]. The theorems in [1] are proven for general In/Out-safe Datalog programs. □

5 Infinite Policies and Dynamic Assertion Providers

We now consider the case where the policy is a possibly infinite set of assertions that are fetched on demand during Datalog evaluation. The set of assertions is represented by an *assertion provider* which maps a Datalog subgoal G to a set of assertions, each of which have the property that at least one of the clauses resulting from its Datalog translation has a head that is unifiable with G. The Datalog proof engine is then modified to fetch and dynamically translate

assertions from the assertion provider during the clause resolution step. It is easy to see that the proof procedure remains sound and complete with this modification.

This mechanism is useful in systems where the policy contains a huge number of assertions that could not practically be loaded into memory all at once. In fact, the assertions may be stored as tuples in a large relational database, and converted to SecPAL assertions by the assertion provider on the fly. Assertion providers are also useful if the policy consists of an infinite number of assertions, but we know that only a finite portion will ever be used for any given query (see for instance Section 6).

In the remainder of this section we present a safety condition that guarantees termination even if the assertion set may be infinite. Note that query evaluation may not terminate if the assertion set is infinite, even if the query and all assertions are In/Out-safe. For example, consider the query $\langle A \text{ says } p(0)\rangle$ and the assertion set $\{\langle A \text{ says } p(i) \text{ if } p(i')\rangle \mid i \in \mathbb{N}, i' = i+1\}$. On the other hand, if $\mathbf{Mode}(p) = \langle \mathsf{In}\rangle$, then any In/Out-safe query will terminate in the context of $\{A \text{ says } p(i) \mid i \text{ is prime}\}$.

Definition 5.1. Let $\alpha = \langle e \text{ says } f \text{ if } f_1, ..., f_n \text{ where } c\rangle$ be an assertion. We define $\mathbf{hd}(\alpha) = e \text{ says } f$ and $\mathbf{bd}(\alpha) = \{\langle e \text{ says } f_1\rangle, ..., \langle e \text{ says } f_n\rangle\}$.

Definition 5.2. We define $\preceq$ to be the smallest reflexive-transitive relation on atomic assertions such that $\langle e \text{ says } f\rangle \preceq \langle e \text{ says } e' \text{ cansay } f\rangle$ and $\langle e \text{ says } f\rangle \preceq \langle e \text{ says } e' \text{ cansay}_0 \ f\rangle$, for all facts f and all expressions e, e'.

In the following, we write β to denote an atomic assertion (or equivalently, an atomic query).

Definition 5.3. Let α_1, α_2 be assertions. We write $\alpha_1 \rightarrow \alpha_2$ iff

- there exist $\beta_1 \in \mathbf{bd}(\alpha_1)$ and $\beta_2 \preceq \mathbf{hd}(\alpha_2)$ such that β_1 and β_2 are unifiable; or
- $\mathbf{hd}(\alpha_1)$ is of the form $\langle e \text{ says } e_0 \text{ cansay } f\rangle$ and there exists $\beta \preceq \mathbf{hd}(\alpha_2)$ such that $\langle e_0 \text{ says } f\rangle$ and β are unifiable.

Let $\beta = \langle e \text{ says } f\rangle$. We write $\beta \rightsquigarrow \alpha$ iff $\langle e \text{ says } \_ \text{ if } f\rangle \rightarrow^* \alpha$, where $\rightarrow^*$ is the reflexive-transitive closure of $\rightarrow$, and $\_$ is any fact (it does not matter which one).

Definition 5.4. A set of assertions $\mathcal{A}$ is *assertion-provider-safe* (or short: *AP-safe*) iff

- all assertions in $\mathcal{A}$ are In/Out-safe, and
- for all In/Out-safe atomic queries β, $\{\alpha \in \mathcal{A} \mid \beta \rightsquigarrow \alpha\}$ is finite.

Theorem 5.5. The termination result in [2, 1] is still valid with AP-safety.

Proof. (Sketch.) Given a In/Out-safe atomic query β, $\mathcal{A}' = \{\alpha \in \mathcal{A} \mid \beta \rightsquigarrow \alpha\}$ is an upper bound on the assertions needed for proving or disproving β. More precisely, $\mathcal{A}' \vdash \beta$ iff $\mathcal{A} \vdash \beta$. Furthermore, the modified proof procedure is constructed in such a way that it will only ever access assertions in $\mathcal{A}'$ when proving β. □

6 Adding Hierarchical Parameters

We now associate each parameter position i of each predicate symbol $p \in \mathbf{Pred}$ (i.e. $i \in \{1, ..., \mathbf{ar}(p)\}$) with a *hierarchy relation*, i.e. a (possibly empty) non-reflexive[3] binary relation $\lhd_i^p \subseteq \mathbf{Const} \times \mathbf{Const}$.

This induces the following non-reflexive relation $\lhd$ on (possibly non-ground) facts:

$$
\begin{aligned}
&p(e_1, ..., e_n) \lhd p(e'_1, ..., e'_n) \;\text{ if}\\
&\quad \exists i \in \{1, ..., n\} : e_i \neq e'_i \text{ and}\\
&\quad \forall i \in \{1, ..., n\} : e_i = e'_i \text{ or}\\
&\qquad\qquad e_i, e'_i \in \mathbf{Const} \text{ and } e_i \lhd_i^p e'_i\\
&\langle e \;\mathsf{cansay}\; f\rangle \lhd \langle e \;\mathsf{cansay}\; f'\rangle \;\text{ if } f \lhd f'\\
&\langle e \;\mathsf{cansay}_0\; f\rangle \lhd \langle e \;\mathsf{cansay}_0\; f'\rangle \;\text{ if } f \lhd f'
\end{aligned}
$$

We write $\lhd^+$ to denote the (non-reflexive) transitive closure of $\lhd$.

Hierarchy semantics. The hierarchical proof semantics $\vdash^H$ consists of the rules from Fig. 1, and the following (ground) rule:

$$\text{(hry)}\;\frac{\mathcal{A} \vdash^H E \;\mathsf{says}\; F \qquad F \lhd^+ F'}{\mathcal{A} \vdash^H E \;\mathsf{says}\; F'}$$

This proof system captures the intuitive meaning of hierarchical parameters, but is not so easy to implement directly. This is why we now present an alternative semantics that is equivalent and easier to implement, although less intuitive.

Implementing hierarchies. We can implement the hierarchy semantics by transforming the assertion set $\mathcal{A}$ into a possibly infinite set $\mathcal{A}^H$ such that $\mathcal{A} \vdash^H q$ iff $\mathcal{A}^H \vdash q$.

Definition 6.1. The *quotation depth* $\mathbf{depth}(f)$ of a fact f is 0 if f is flat, and $1 + \mathbf{depth}(f')$ if f is of the form $\langle e \;\mathsf{says}\; e' \;\mathsf{cansay}\; f'\rangle$ or $\langle e \;\mathsf{says}\; e' \;\mathsf{cansay}_0\; f'\rangle$. We write $\mathbf{maxDepth}(\mathcal{A})$ to denote the maximum quotation depth of any fact occurring in any assertion in $\mathcal{A}$.

We define $\mathcal{A}^H = \mathcal{A} \cup \{\langle x \;\mathsf{says}\; f' \;\mathsf{if}\; f\rangle \mid f \lhd f',\ \mathbf{depth}(f) \leq \mathbf{maxDepth}(\mathcal{A})\}$, where x is a variable occurring neither in f nor in f', and the assertions in $\mathcal{A}^H$ are identified up to invertible variable renaming.

[3] A binary relation $\lhd$ is non-reflexive iff $e_1 \lhd e_2$ implies $e_1 \neq e_2$.

Theorem 6.3 states the correctness property of the hierarchy implementation.

Lemma 6.2. For all $\mathcal{A}$, e, f', and β the following holds: $\mathcal{A} \cup \{e \text{ says } f' \text{ if } f \mid f \lhd^+ f'\} \vdash^H \beta$ iff $\mathcal{A} \vdash^H \beta$.

Proof. We assume $\mathcal{A} \cup \{e \text{ says } f' \text{ if } f \mid f \lhd^+ f'\} \vdash^H \beta$ and prove $\mathcal{A} \vdash^H \beta$ by rule induction on $\vdash^H$. (The other direction follows from monotonicity of $\vdash^H$.)

Consider the following application of rule (mdp), where $\beta = \gamma(e \text{ says } f')$ for some ground substitution γ (the other cases follow directly from the induction hypothesis):

$$\text{(mdp)}\frac{\mathcal{A} \cup \{e \text{ says } f' \text{ if } f \mid f \lhd^+ f'\} \vdash^H \gamma(e \text{ says } f_0) \qquad \text{for some } f_0 \lhd^+ f'}{\mathcal{A} \cup \{e \text{ says } f' \text{ if } f \mid f \lhd^+ f'\} \vdash^H \gamma(e \text{ says } f')}$$

By the induction hypothesis, $\mathcal{A} \vdash^H \gamma(e \text{ says } f_0)$. Then by (hry), $\mathcal{A} \vdash^H \gamma(e \text{ says } f')$, since $\gamma(f_0) \lhd^+ \gamma(f')$. □

Theorem 6.3. For all $\mathcal{A}$ and q: $\mathcal{A} \vdash^H q$ iff $\mathcal{A}^H \vdash q$.

Proof. It is sufficient to consider atomic queries β. First assume $\mathcal{A} \vdash^H \beta$. We prove $\mathcal{A}^H \vdash \beta$ by rule induction on $\vdash^H$. The only interesting case is (hry). From the induction hypothesis, we have $\mathcal{A}^H \vdash E \text{ says } F$ for some chain $F = F_0 \lhd F_1 ... \lhd F_n = F'$. By construction of $\mathcal{A}^H$, we can then apply (mdp) n times to get $\mathcal{A}^H \vdash E \text{ says } F_n$.

The other direction follows from Lemma 6.2. □

The following is a sufficient condition on the hierarchy relations to ensure AP-safety of the corresponding assertion provider. In practice, AP-safety is often obviously achieved because $\lhd$ is finite.

Definition 6.4. Let $\lhd_p^i$ be a hierarchy relation and $\mathbf{Mode}(p) = \langle \mu_1, ..., \mu_{\mathbf{ar}(p)} \rangle$. Then $\lhd_p^i$ is *safe* iff

1. $\lhd_p^i$ is well-founded[4];
2. for all $E \in \mathbf{Const}$, the set $\{E' \mid E' \lhd_p^i E\}$ is finite; and
3. $\mu_i = \mathsf{Out}$ implies $\lhd_p^i$ is finite.

Theorem 6.5. If $\mathcal{A}$ is In/Out-safe and for all $p \in \mathbf{Pred}$ and $i \in \{1, ..., \mathbf{ar}(p)\}$, $\lhd_p^i$ is safe, then $\mathcal{A}^H$ is AP-safe.

Proof. (Sketch.) Firstly, $\mathcal{A}^H$ is In/Out-safe because $\mathcal{A}$ is In/Out-safe, and every assertion in the set comprehension is In/Out-safe. Secondly, given any In/Out-safe query β, the set $\{\alpha \in \mathcal{A}^H \mid \beta \rightsquigarrow \alpha\}$ is finite, because $\mathcal{A}^H$ is finite, and because of the three conditions in Definition 6.4 as well as the fact that the quotation depth in $\mathcal{A}^H$ is bounded. □

[4] A relation is *well-founded* iff it does not contain any infinite descending chains.

6.1 Examples

Suppose we have two user-defined ternary fact types e_1 can e_2 e_3 and e_1 $\mathsf{possesses}$ e_2 $= e_3$.

To simulate the $\mathsf{canactas}$ construct from [2], suppose we have a hierarchy relation $\lhd^{\mathsf{act}}$ on principals where each edge is expressed as $E_1 \lhd^{\mathsf{act}} E_2$ (corresponding to E_1 $\mathsf{canactas}$ E_2).

Furthermore, suppose there is a hierarchy relation $\lhd^{\mathsf{res}}$ on resources, such that $E_1 \lhd^{\mathsf{res}} E_2$ holds if E_1 is the immediate parent resource of E_2 (e.g. `file://foo/` $\lhd^{\mathsf{res}}$ `file://foo/bar`).

We then choose $\lhd_1^{\mathsf{can}} = \lhd_1^{\mathsf{possesses}} = \lhd^{\mathsf{act}}$, and $\lhd_3^{\mathsf{can}} = \lhd^{\mathsf{res}}$. All other hierarchy relations are empty, i.e. $\lhd_2^{\mathsf{can}} = \lhd_2^{\mathsf{possesses}} = \lhd_3^{\mathsf{possesses}} = \emptyset$.

The hierarchy mechanism is very flexible. For example, there may be other predicates that also take resources as parameters, but do not apply the hierarchy relation $\lhd^{\mathsf{res}}$ on that parameter, or a different relation.

References

[1] M. Y. Becker, C. Fournet, and A. D. Gordon. SecPAL: Design and semantics of a decentralized authorization language. Technical Report MSR-TR-2006-120, Microsoft Research, 2006.

[2] M. Y. Becker, C. Fournet, and A. D. Gordon. Design and semantics of a decentralized authorization language. In *IEEE Computer Security Foundations Symposium*, pages 3–15, 2007.

[3] M. Y. Becker, J. F. Mackay, and B. Dillaway. Abductive authorization credential gathering. In *IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY'09)*, pages 1–8, 2009.

[4] M. Y. Becker and S. Nanz. The role of abduction in declarative authorization policies. Technical Report MSR-TR-2007-105, Microsoft Research, 2007.

[5] M. Y. Becker and S. Nanz. The role of abduction in declarative authorization policies. In *10th International Symposium on Practical Aspects of Declarative Languages (PADL)*, 2008.