Multi-Decision Policy and Policy Combinator Specifications

by

Theophilos John Giannakopoulos

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Master of Science

in

Computer Science

February 2012

APPROVED:

Professor Daniel Dougherty, Thesis Advisor

Professor Joshua Guttman, Reader

Professor Craig Wills, Head of Department

Abstract

Margrave is a specification language and analysis tool for access control policies with semantics based in order-sorted logic. The clear logical roots of Margrave's semantics makes policies specified in the Margrave language both machine analyzable and relatively easy for users to reason about. However, the decision conflict resolution declaration and policy set features of Margrave do not have semantics that are as cleanly rooted in order-sorted logic as Margrave policies and queries are. Additionally, the current semantics of decision conflict resolution declarations and of policy sets do not permit users to take full advantage of the multi-decision capabilities of Margrave policies.

The purposes of this thesis are (i) to provide a unified extension to the semantics for policies and policy combination, (ii) to cleanly support decision conflict resolution mechanisms in a general way within those semantics and (iii) to provide insight into the properties of policy combination and decision conflict resolution for multi-decision policies in general. These goals are achieved via the realization that policy combinators may be treated as policies operating within environments extended with the results of the policies to be combined, allowing policy combinators to be defined as if they were policies. The ability to treat policy combinators as policies means that users' current understanding of policies can be easily extended to policy combinators. Additionally, the tools that Margrave has for supporting policies can be leveraged as the Margrave language and analysis tool grow to provide fuller support for policy combination and rule conflict resolution declarations.

Acknowledgements

I am indebted to many people for their help and encouragement through the process of writing this thesis, including

- my advisor Professor Daniel Dougherty for guiding me through the process of researching and writing a thesis from start to end,
- Timothy Nelson for his help with slogging through the details of my research and for his encouragement,
- my reader Professor Joshua Guttman for helping me get off the ground and for helping with the finishing touches,
- Professor Kathi Fisler for helping me find my writing voice and giving me confidence,
- the ALAS lab for all of their support,
- my parents for supporting my decision to come back to school,

and everyone else who has helped me along the way.

Contents

Chapter 1

Introduction

Margrave is a specification language and analysis tool for access control policies [Fisler et al., 2005; Nelson, 2010; Nelson et al., 2010] with semantics based in order-sorted logic. The clear logical roots of Margrave's semantics makes policies specified in the Margrave language both machine analyzable and relatively easy for users to reason about. However, the decision conflict resolution declaration and policy set features of Margrave do not have semantics that are as cleanly rooted in order-sorted logic as Margrave policies and queries are. Additionally, the current semantics of decision conflict resolution declarations and of policy sets do not permit users to take full advantage of the multi-decision capabilities of Margrave policies.

The purposes of this thesis are to provide a unified extension to the semantics for policies and policy combination, to cleanly support decision conflict resolution mechanisms in a general way within those semantics and to provide insight into the properties of policy combination and decision conflict resolution for multi-decision policies in general.

1.1 Decision conflict resolution

For many policies, it does not make sense to allow for any decisions other than one of "permit" or "deny" to be specified as the response for a request. Frequently, one wishes to specify a deny response as the absence of a permit response. This view is apparent in formalisms such as those presented by Bonatti et al. [2002] and by Wijesekera and Jajodia [2003]. More commonly, the distinction between "deny" and "does not permit" is recognized, and is present in mainstream access control languages [IBM, 2011; OASIS, 2011].

In XACML, for example, it is considered a conflict for a policy to respond with both a "permit" and a "deny" response. Resolving conflicts means to decide whether a policy says to respond with a permit decision or a deny decision when each response is indicated by some rule that applies to a request. For indicating how to resolve such a conflict XACML provides (essentially) three possible decision conflict resolution declarations:

permit-overrides indicates that the permit decision should be the response when both permit and deny decisions are indicated by the rules,

deny-overrides indicates that the deny decision should be the response when both permit and deny decisions are indicated by the rules, and

first-applicable indicates that the decision for the rule that is textually first among the applicable rules should be the response.

In Margrave, these combinators were expanded slightly. The first-applicable declaration was expanded to include all possible decisions. The permit-overrides and deny-overrides declarations were combined into a single parameterized combinator that could specify which decisions overrode which, such as "overrides d $d_1 \ldots d_n$", which would indicate that the presence of any of $d_1 \ldots d_n$ should prevent d from being a response. This allowed for specifications such as "overrides permit deny", which would have the same effect as deny-overrides but would not interfere with other decisions in the response, such as a decision indicating that the request should be logged.

It is clear that certain additions to these possibilities are desirable. For example, in a policy with many decisions, multiple overrides declarations might be desired, such as

```
overrides permit deny
overrides log-normal log-severe.
```

The goal of the declaration is to indicate that the `deny` decision overrides the `permit` decision and that the `log-severe` decision overrides the `log-normal` decision, but that the two pairs of decisions do not interact. One might also wish to alter first applicable declarations to only have effect among certain decisions, and to have multiple of those, such as

```
first-applicable permit deny
first-applicable log-normal log-severe.
```

In this declaration the goal is to indicate that a policy should determine the decision from the first applicable rule in each pair of decisions, resulting in only one of `permit` or `deny` and only one of `log-normal` or `log-severe` being the response to a request. Again, the two pairs of decisions do not interact. It may even be desirable to combine first applicable and overrides, as in

```
overrides permit deny
first-applicable log-normal log-severe.
```

This case would combine the meanings of the previous two declarations. However, while the semantics were clear with the simpler definitions, the meanings of these declarations are not so obvious.

1.2 Problems with decision conflict resolution declarations

Consider for a moment the decision conflict resolution declaration

```
overrides permit deny
overrides deny redirect-to-trap.
```

If all three decisions are indicated by the rules, one might like `deny` to prevent `permit` from being in the response, and then `redirect-to-trap` to prevent `deny` from being in the response, so that the requester is neither permitted to the system, nor explicitly denied, but instead redirected to a honeypot trap. Now consider the decision conflict resolution declaration

```
overrides permit deny
overrides deny root-access.
```

If all three decisions are indicated by the rules, one might want the conflict resolution declaration to mean that `permit` and `root-access` will be in the response, so that the presence of `deny` does not prevent a root user from accessing a system. This would be useful, for example, if a root user needs to access a system to correct a policy that was denying access to root users. However, this choice of semantics is in direct conflict with the semantics chosen for the previous example, in which only one of the three decisions was included in the response.

We could rely on the order of the declarations to resolve the ambiguity, forcing us to write

```
overrides deny root-access
overrides permit deny,
```

but that would mar the declarative nature of the conflict resolution declarations. If instead we decide to choose the second interpretation, we could conceive of the puzzling situation

```
overrides permit deny
overrides deny permit
```

where either the response containing only `permit` or the response containing only `deny` would be admitted if both were indicated by the rules. Though it would be possible to detect such ambiguous situations and forbid them, that is neither an aesthetically pleasing nor an easily maintainable solution. Alternatively, if we choose the first interpretation, then we loose the ability to easily write conflict resolution declarations that correspond to our example for the second interpretation.

The same situation arises and the same choices need to be considered with first-applicable declarations and with first-applicable declarations in combination with overrides declarations.

Even if we would like to keep the ability to write high-level decision conflict resolution declarations in Margrave, expending some effort towards precisely describing the semantic framework in which they are defined would clearly allow for a more productive conversation about the different possible meanings for the combinators. This is the approach that we will take.

1.3 Our approach and contributions

In XACML, policies are combined as if they are simply rules, using the same combinators. Margrave currently uses the same approach, meaning that the same problems that occurred with decision conflict resolution occur here. Additionally, richer policy combination options may make the modularization of policies something that users of Margrave find useful in practice. Our approach to solving the problems with decision conflict resolution and with policy combination involves precisely defining policy specifications and policy combinator specifications that have clearly unambiguous semantics, and then showing how the semantics of Margrave can be interpreted in our formalism. We improve the current mechanism by embracing and extending the notion that when combining policies, the policies being combined can be treated as rules.

The key contributions of this thesis are the realizations that policy combinator specifications can be written in the same language and with the same semantics as policies themselves, and that decision conflict resolution declarations for both Margrave policies and policy sets can be captured by those same policy combinator specifications.

1.4 Background: Order-sorted logic

The following is a very brief review of order-sorted logic without overloading of functions or predicates. The content is taken primarily from Goguen [1992] and from Nelson [2010], which have more extensive descriptions of the logic.

1.4.1 Signatures

A *signature* $\mathcal{L}$ is a triple $(\mathcal{S}, \leq, \mathcal{E})$ where $(\mathcal{S}, \leq)$ is a finite poset of sorts and $\mathcal{E}$ is a $(\mathcal{S}^* \cup \mathcal{S}^* \times \mathcal{S})$-indexed family comprised of

- $\{\mathcal{E}_w\}_{w \in \mathcal{S}^*}$ a family of sets of *predicates* or *relation symbols*, and
- $\{\mathcal{E}_{w,A}\}_{w \in \mathcal{S}^*, A \in \mathcal{S}}$ a family of sets of *function symbols*.

For a relation symbol $R \in \mathcal{E}_w$, we say w is the *arity* of R. For a function symbol $f \in \mathcal{E}_{w,A}$ we say w is the *arity* of f, A is the *result sort* of f, and (w, A) is the *rank* of f. Each relation symbol must have only one arity and each function symbol must have only one rank.

A signature is *locally filtered* if each pair of sorts in the same connected component of $(\mathcal{S}, \leq)$ has an upper bound. If $\mathcal{L} = (\mathcal{S}, \leq, \mathcal{E})$ and $\mathcal{L}^+ = (\mathcal{S}, \leq, \mathcal{E}^+)$ such that for each $w \in \mathcal{S}^*$ and each $A \in \mathcal{S}$, $\mathcal{E}_w \subseteq \mathcal{E}^+_w$ and $\mathcal{E}_{w,A} \subseteq \mathcal{E}^+_{w,A}$ then $\mathcal{L}^+$ is an expansion of $\mathcal{L}$ and $\mathcal{L}$ is a reduct of $\mathcal{L}^+$.

1.4.2 Models

Fix a signature $\mathcal{L} = (\mathcal{S}, \leq, \mathcal{E})$. An *$\mathcal{L}$-model* $\mathbb{M}$ is

- a family of sets $\{\mathbb{M}_A\}_{A \in \mathcal{S}}$, the *universe* of $\mathbb{M}$, such that $A \leq A'$ implies $\mathbb{M}_A \subseteq \mathbb{M}_{A'}$,
- for each relation symbol $R \in \mathcal{E}_w$ a relation $R^{\mathbb{M}} \subseteq \mathbb{M}_w$,
- for each function symbol $f \in \mathcal{E}_{w,A}$ a function $f^{\mathbb{M}_{w,A}} \subseteq \mathbb{M}_w \times \mathbb{M}_A$.

$\mathrm{Mod}(\mathcal{L})$ denotes the set of $\mathcal{L}$-models.

If a $\mathbb{M}$ is a model of $\mathcal{L}$ and $\mathcal{L}^+$ is an expansion of $\mathcal{L}$, then an *expansion* $\mathbb{M}^+$ of $\mathbb{M}$ to $\mathcal{L}^+$ is a model of $\mathcal{L}^+$ with the same universe as $\mathbb{M}$ and that agrees with $\mathbb{M}$ on the symbols in $\mathcal{L}$. Similarly, if a $\mathbb{M}$ is a model of $\mathcal{L}$ and $\mathcal{L}^-$ is a reduct of $\mathcal{L}$, then the *reduct* $\mathbb{M}^-$ of $\mathbb{M}$ to $\mathcal{L}^-$ is the model of $\mathcal{L}^-$ with the same universe as $\mathbb{M}$ and that agrees with $\mathbb{M}$ on the symbols in $\mathcal{L}^-$. Node that the reduct of a model is unique, but the expansion generally is not.

1.4.3 Formulas

The well-formed order-sorted $(\mathcal{S}, \leq, \mathcal{E})$-formulas are the well-formed many-sorted $(\mathcal{S}, \mathcal{E})$-formulas, with the relaxation that a term whose sort is any subsort of the expected sort of a position in a term may appear at that position.

1.4.4 Locally filtered signatures

The local filtering of signatures is a technical device that is necessary to ensure that isomorphic order-sorted logic models satisfy the same formulas. See Goguen [1992] for details on the problems that arise when using signatures that are not locally filtered. Throughout this thesis we assume that all signatures are locally filtered.

Chapter 2

Policies and policy specifications

Unlike some other approaches to policy combination (see Section 6.2), we are grounding our understanding of policy combination in an understanding of the policies themselves. Though, as we will explain later, this is not strictly necessary for defining or understanding policy combination, the resulting uniformness will be useful when showing how policies and policy combinators in our formalism are easily amenable to machine analysis à la Margrave.

The word "policy" can be used to mean either a mathematical object or a software engineering artifact. Though we had been using the word "policy" with both meanings, henceforth we will use "policy" to refer to the mathematical object and "policy specification" to refer to the artifact or document that contains the text that describes the policy, or to the abstract view of such an artifact. For example, the text of an XACML document is a policy specification, while the mathematical function from requests to responses induced by that specification is a policy. While some readers may find this unusual, it is more convenient than the alternative since there is no commonly used expression to refer to the mathematical object.

2.1 Policies

A policy is a way of getting decisions from an environment, such as user role assignments or the time of day, and a request, such "Tim wants to edit his homework submission". That is, a policy p is a function

$$p : \mathsf{Env} \to \mathsf{Req} \to 2^{\mathsf{Dec}}.$$

Recall that allowing multiple decisions to be included in the response to a request does not mean that those decisions are necessarily in conflict. Having a response include both a `deny` decision and a `log` decision would be perfectly reasonable, for example.

The type for policies presented above is not entirely accurate because the set of possible requests usually depends on the environment. For example, the possible requests for a user to read a file would require both the user and the file to exist in the system where the request is made. In order for us to see how our choices for representing the environment, requests and decisions fit into this perspective, we make the dependent type explicit:

$$p : \forall E \in \mathsf{Env}\,.\,\{E\} \to (\mathsf{Req}_E \to 2^{\mathsf{Dec}}).$$

Equivalently we could consider a function from environments to associations between decisions and the requests that would result in those decisions,

$$p : \forall E \in \mathsf{Env} \,.\, \{E\} \to (\mathsf{Dec} \to 2^{\mathsf{Req}_E}).$$

Our choice for representing the environment is an $\mathcal{L}$-model over some signature $\mathcal{L}$. The natural corresponding type for requests is the set of tuples of some sort w over the models, so that our typing is

$$p : \forall \mathbb{M} \in \mathrm{Mod}(\mathcal{L}) \,.\, \{\mathbb{M}\} \to (\mathsf{Dec} \to 2^{\mathbb{M}_w}).$$

We choose to let the decisions be predicate symbols $\mathcal{D}$ distinct from those in $\mathcal{L}$. Also, we choose not to force all of the decision predicates to have the same sort, so $\mathcal{D}$ is actually an $\mathcal{S}^*$-indexed family of predicate symbols, where $\mathcal{S}$ is the set of sorts for $\mathcal{L}$. This creates another dependent type,

$$p : \forall \mathbb{M} \in \mathrm{Mod}(\mathcal{L}) \,.\, \{\mathbb{M}\} \to (\forall w \in \mathcal{S}^* \,.\, \mathcal{D}_w \to 2^{\mathbb{M}_w}).$$

This is actually a very nice typing for policies, since given the universe for a model, the expression $\forall w \in \mathcal{S}^* \,.\, \mathcal{D}_w \to 2^{\mathbb{M}_w}$ is the type of a function assigning relations to predicates within that universe. That is, if we have a universe, which we do, then we have a model.

Later in this thesis it will become immensely useful to view policies as maps that when given a model produce an expansion of that model to $\mathcal{L}^{\mathcal{D}}$, which is the signature $\mathcal{L}$ with its predicates augmented by $\mathcal{D}$, instead of as a map from models of $\mathcal{L} = (\mathcal{S}, \leq, \mathcal{E})$ to models of $(\mathcal{S}, \leq, \mathcal{D})$. Since combining families of predicates and augmenting languages with predicates will frequently be necessary, we introduce notation for those operations.

Definition 2.1 ($\uplus$)**.** Let $\mathcal{S}$ be a set of sorts, and let $\mathcal{I}_1$ and $\mathcal{I}_2$ be $\mathcal{S}^*$-indexed families of predicates. Denote the pointwise union of $\mathcal{I}_1$ and $\mathcal{I}_2$ as $\mathcal{I}_1 \uplus \mathcal{I}_2 = \mathcal{I}$ where for each $w \in \mathcal{S}^*$, $\mathcal{I}_w = \mathcal{I}_{1w} \cup \mathcal{I}_{2w}$.

Definition 2.2 ($\mathcal{L}^{\mathcal{D}}$)**.** Let $\mathcal{L} = (\mathcal{S}, \leq, \mathcal{E})$ be a signature, and let $\mathcal{D}$ be a $\mathcal{S}^*$-indexed family of predicates. Then $\mathcal{L}^{\mathcal{D}} = (\mathcal{S}, \leq, \mathcal{E} \uplus \mathcal{D})$.

With this notation in hand, we formalize the conception of policies as model expanding functions.

Definition 2.3 (Policy)**.** Let $\mathcal{L} = (\mathcal{S}, \leq, \mathcal{E})$ be a signature, and let $\mathcal{D}$ be a $\mathcal{S}^*$-indexed family of predicates, distinct from the predicates in $\mathcal{E}$. A $(\mathcal{L}, \mathcal{D})$-policy is a function

$$p : \mathrm{Mod}(\mathcal{L}) \to \mathrm{Mod}(\mathcal{L}^{\mathcal{D}})$$

such that for each $\mathbb{M} \in \mathrm{Mod}(\mathcal{L})$, $p(\mathbb{M})$ is an expansion of $\mathbb{M}$ to $\mathcal{L}^{\mathcal{D}}$.

The set of $(\mathcal{L}, \mathcal{D})$-policies is denoted $\mathrm{Pol}(\mathcal{L}, \mathcal{D})$. The set of all policies whose signatures are reductions of $\mathcal{L}$ and whose decision sets contain $\mathcal{D}$ but do not share any symbols with $\mathcal{L}$ is denoted $\mathrm{Pol}^*(\mathcal{L}, \mathcal{D})$.

2.2 Policy specifications

Since in our definition policies are intimately tied up with order-sorted logic models, our policy specifications will be based on order-sorted logic theories. Theories will be used to define to which

expansion a policy maps a model. The theories that we will use are sets of rules, which are formulas with certain syntactic constraints.

Definition 2.4 (Rule)**.** Let $\mathcal{L} = (\mathcal{S}, \leq, \mathcal{E})$ be a signature, and let $\mathcal{I}$ be a $\mathcal{S}^*$-indexed family of predicate symbols, distinct from the predicates in $\mathcal{E}$. A $(\mathcal{L}, \mathcal{I})$-rule is a sentence of $\mathcal{L}^{\mathcal{I}}$

$$\forall^{A_1} x_1 \cdots \forall^{A_k} x_k \,.\, d(x_1, \ldots, x_k) \Leftarrow \alpha$$

where the arity of d is some $B_1 \cdots B_k$ such that $A_i \leq B_i$ for $1 \leq i \leq k$. The formula $d(x_1, \ldots, x_k)$ is the head of the rule, and α is the body of the rule. The body α may be any formula of $\mathcal{L}$ so long as the whole rule is a sentence. When it is clear that the predicate symbol is being referred to, we may also call d the head of the rule.

We will also use an alternative syntax for rules

$$d(x_1 : A_1, \ldots, x_k : A_k) \text{ :- } \alpha$$

which is just syntactic sugar for the original form. The alternative syntax was intentionally chosen to resemble Datalog rules, since rules in policy specifications will be similarly interpreted.

Policy specifications are collections of rules along with information about the language and decisions of the policy.

Definition 2.5 (Policy specification)**.** A policy specification is a 4-tuple

$$(\mathcal{L}, \mathcal{D}, \mathcal{I}, \Gamma)$$

where

- $\mathcal{L} = (\mathcal{S}, \leq, \mathcal{E})$ is a signature, where the predicates of $\mathcal{E}$ are called the extensional predicates of the policy specification,
- $\mathcal{D}$ is a $\mathcal{S}^*$-indexed family of predicate symbols, the decision predicates of the policy specification, which are distinct from the symbols of $\mathcal{E}$,
- $\mathcal{I}$ is a $\mathcal{S}^*$-indexed family of predicate symbols, called the intensional predicates of the policy specification, which are distinct from the symbols in $\mathcal{E}$ and which include all of $\mathcal{D}$, and
- Γ is a set of $(\mathcal{L}, \mathcal{I})$-rules.

The intensional predicates $\mathcal{I}$ that are not decision predicates are the "helper" predicates of a policy specification. By separating the two, policy specifications can include predicates that are useful writing the policy but that do not represent decisions.

We will frequently refer to the set of rules Γ as a $(\mathcal{L}, \mathcal{D}, \mathcal{I})$-policy specification, and use the notation for the set of rules Γ as shorthand for $(\mathcal{L}, \mathcal{D}, \mathcal{I}, \Gamma)$. When every intensional predicate is a decision predicate, we may omit $\mathcal{I}$ and refer to Γ as a $(\mathcal{L}, \mathcal{D})$-policy specification.

The policy corresponding to a specification takes a model to the least expansion of the model that satisfies the rules, and then trims off the intensional relations that are not decision relations. Policy specifications have not been sufficiently restricted in order to prevent such a semantics from

being unambiguous. Consider the $(\mathcal{L}, \mathcal{D})$-policy specification Γ, where $\mathcal{L}$ contains no predicate or function symbols and $\mathcal{D}_A = \{q, r\}$, consisting of the two rules

$$q(x : A) \text{ :- } \neg r(x)$$
$$r(x : A) \text{ :- } \neg q(x).$$

The model $\mathbb{M}$ of the signature consisting of the universe with one element $a \in \mathbb{M}_A$ has two minimal extensions to $\mathcal{L}^{\mathcal{D}}$ that satisfy the rules: one $\mathbb{M}_1$ in which $a \in q^{\mathbb{M}_1}$ but $a \notin r^{\mathbb{M}_1}$, and the other $\mathbb{M}_2$ in which $a \notin q^{\mathbb{M}_2}$ but $a \in r^{\mathbb{M}_2}$. As a result of the combination of recursion and negation, the policy resulting from Γ is not well-defined.

In order to have a policy be well defined for a policy specification, we require that the policy specification satisfy the following property.

Definition 2.6 (Unique minimal expansion property)**.** Let $(\mathcal{L}, \mathcal{D}, \mathcal{I}, \Gamma)$ be a policy specification. If for every model of $\mathcal{L}$ there is a unique minimal expansion to $\mathcal{L}^{\mathcal{I}}$ that satisfies Γ, then $(\mathcal{L}, \mathcal{D}, \mathcal{I}, \Gamma)$ has the unique minimal expansion property.

The definition of the semantics for a policy specification require notation for taking the reduct of a model to a signature.

Definition 2.7 ($\mathbb{M}\downarrow_{\mathcal{L}^-}$)**.** Let $\mathcal{L}$ and $\mathcal{L}^-$ be signatures, where $\mathcal{L}^-$ is a reduct of $\mathcal{L}$. Denote the reduct of a model $\mathbb{M}$ of $\mathcal{L}$ to the signature $\mathcal{L}^-$ as $\mathbb{M}\downarrow_{\mathcal{L}^-}$. Note that $\mathbb{M}\downarrow_{\mathcal{L}^-}$ exists and is unique.

We formally define the semantics for policy specifications that have the unique minimal expansion property.

Definition 2.8 ($[\![\cdot]\!]$)**.** Let $(\mathcal{L}, \mathcal{D}, \mathcal{I}, \Gamma)$ be a policy specification with the unique minimal expansion property. Then $[\![(\mathcal{L}, \mathcal{D}, \mathcal{I}, \Gamma)]\!]$ is a $(\mathcal{L}, \mathcal{D})$-policy, where for every model $\mathbb{M}$ of $\mathcal{L}$, the model $[\![(\mathcal{L}, \mathcal{D}, \mathcal{I}, \Gamma)]\!](\mathbb{M}) = \mathbb{M}_\Gamma\downarrow_{\mathcal{L}^{\mathcal{D}}}$, where $\mathbb{M}_\Gamma$ is the minimal expansion of $\mathbb{M}$ to $\mathcal{L}^{\mathcal{I}}$ that satisfies Γ.

When referring to Γ as an $(\mathcal{L}, \mathcal{D}, \mathcal{I})$-policy specification, we write $[\![\Gamma]\!]$ instead of $[\![(\mathcal{L}, \mathcal{D}, \mathcal{I}, \Gamma)]\!]$.

Given the restriction of the unique minimal expansion property, $[\![\cdot]\!]$ is clearly well defined. However, it is useful to have a syntactic condition on policy specifications that ensures the semantic condition of the unique minimal expansion property. Such a syntactic condition on policy specifications can be achieved by insisting that the rules be stratifiable, as one restricts rules in recursive Datalog with negation. Actually, we are more restrictive than recursive Datalog with negation, because we disallow an intensional predicate from being defined in terms of itself even when negation is not involved. Though the elimination of recursion is not necessary to satisfy the unique minimal expansion property, it greatly simplifies several proofs and the compilation of specifications that we define later for the purpose of analysis. The simplicity is important for what is intended to be a core language for user-facing semantics.

Definition 2.9 (Strict stratification of policy specifications)**.** A strict stratification of a $(\mathcal{L}, \mathcal{D}, \mathcal{I})$-policy Γ is a function

$$\sigma : \left(\bigcup_{w \in \mathcal{S}^*} \mathcal{I}_w \right) \longrightarrow \mathbb{N}$$

such that for each rule $d(x_1 : A_1, \ldots, x_k : A_k)$:- α in Γ, for each intensional predicate q occurring in α

$$\sigma(d) > \sigma(q).$$

If a strict stratification exists for a policy specification, we say that the policy specification is strictly stratifiable.

We call such a σ a *strict* stratification because it requires that the stratum for the head of a rule be *strictly* greater than the stratum for each of the intensional predicates appearing in the body, which eliminates recursion from our specifications. Since the extensional predicates occur only on the right-hand side of a rule, they are not included in the assignment to strata.

Strictly satisfiable policies have the unique minimal expansion property. At a high level, this is because each the relations assigned to intensional predicates at each stratum are completely defined by the relations assigned to intensional predicates at lower strata and the extensional predicates. The relations assigned to the intensional predicates at the lowest stratum are completely defined by the relations assigned to the extensional predicates, which are fixed by the original model and the constraint that a policy produces an extension of the model to which it is applied.

Theorem 2.10. *Let Γ be a strictly stratifiable $(\mathcal{L}, \mathcal{D}, \mathcal{I})$-policy specification. Then Γ has the unique minimal expansion property.*

Proof. We proceed by a recursively constructing a model $\mathbb{M}_\Gamma$ of $\mathcal{L}^{\mathcal{I}}$, given $\mathbb{M}$.

Let σ be a strict stratification of Γ. For each intensional predicate d assigned to stratum 0 by σ, let w be the arity of d. For each tuple $\langle a_1, \ldots, a_k \rangle \in \mathbb{M}_w$, and each rule $d(x_1 : A_1, \ldots, x_k : A_k)$:- α, we can determine if $\mathbb{M}_\Gamma$ satisfies the body α of the rule with an assignment $\{x_1 \mapsto a_1, \ldots, x_k \mapsto a_k\}$ without knowing all of $\mathbb{M}_\Gamma$, since only extensional predicates, whose instances in $\mathbb{M}_\Gamma$ we do know, appear in the body of the rule. If it does satisfy the body of the rule, then we add the tuple to $d^{\mathbb{M}_\Gamma}$, since it is required in order for $\mathbb{M}_\Gamma$ to satisfy the rule.

For each intensional predicate d assigned to stratum $\sigma(d) > 0$, we repeat the same process, noting that we have already defined $q^{\mathbb{M}_\Gamma}$ for each q appearing in the body of each rule for d because the stratum of q is strictly less than the stratum of d.

Every tuple added to the instances of intensional predicates by $\mathbb{M}_\Gamma$ is required in order for $\mathbb{M}_\Gamma$ to satisfy Γ. Additionally, we have constructed $\mathbb{M}_\Gamma$ so that it satisfies each rule in Γ. Therefore $\mathbb{M}_\Gamma$ is a minimal extension of $\mathbb{M}$ that satisfies Γ. Since there was no choice in which tuples to add to instances of the intensional predicates, $\mathbb{M}_\Gamma$ is the unique minimal model that satisfies Γ. □

For the rest of this thesis we will only be considering strictly stratifiable policy specifications. Note that Theorem 2.10 also gives us a procedure for computing $[\![\Gamma]\!](\mathbb{M})$ when the policy specification Γ is strictly stratifiable. Now that we have defined policies, as well as the syntax and semantics for policy specifications, we will look at an example of a policy defined using this formalism.

2.3 Policy specification example

We define here a small example specification for a policy about access control for an assignment submission and grading system. In the example, `typewriter` font indicates syntactic objects, while **bold serif** font indicates semantic elements of a model.

The signature for our specification is $\mathcal{L}_{\mathsf{G}} = (\mathcal{S}_{\mathsf{G}}, \leq_{\mathsf{G}}, \mathcal{E}_{\mathsf{G}})$. The sorts $\mathcal{S}_{\mathsf{G}}$ consist of

$$\{\mathtt{S}, \mathtt{A}, \mathtt{R}, \mathtt{Student}, \mathtt{Grader}, \mathtt{Professor}, \mathtt{Quiz}, \mathtt{Homework}\}.$$

The sort $\mathtt{S}$ represents subjects, $\mathtt{A}$ represents actions and $\mathtt{R}$ represents resources. The relation $\leq_{\mathsf{G}}$ is equal to the reflexive closure of

$$\begin{aligned}\mathtt{Student} &\leq_{\mathsf{G}} \mathtt{S}\\ \mathtt{Grader} &\leq_{\mathsf{G}} \mathtt{S}\\ \mathtt{Professor} &\leq_{\mathsf{G}} \mathtt{S}\\ \mathtt{Homework} &\leq_{\mathsf{G}} \mathtt{R}\\ \mathtt{Quiz} &\leq_{\mathsf{G}} \mathtt{R}.\end{aligned}$$

The vocabulary $\mathcal{E}_{\mathsf{G}}$ has one predicate $\mathtt{authorOf}$ with arity $\mathtt{Student} \times \mathtt{R}$, and two constants $\mathtt{edit}$ and $\mathtt{assignGrade}$ of sort $\mathtt{A}$. The decisions $\mathcal{D}_{\mathsf{G}}$ consists of $\mathtt{permit}$, $\mathtt{deny}$ and $\mathtt{log}$, all with arity $\mathtt{S} \times \mathtt{A} \times \mathtt{R}$. Every intensional predicate is a decision predicate.

An example policy specification Γ_{G} over $\mathcal{L}_{\mathsf{G}}$ is

$$\mathtt{permit}(s : \mathtt{Student}, a : \mathtt{A}, r : \mathtt{R}) \text{ :- } a = \mathtt{edit} \wedge \mathtt{authorOf}(s, r) \tag{R2.1}$$

$$\mathtt{permit}(s : \mathtt{Grader}, a : \mathtt{A}, r : \mathtt{Homework}) \text{ :- } a = \mathtt{assignGrade} \tag{R2.2}$$

$$\mathtt{permit}(s : \mathtt{Professor}, a : \mathtt{A}, r : \mathtt{R}) \text{ :- } a = \mathtt{assignGrade} \tag{R2.3}$$

$$\mathtt{deny}(s : \mathtt{Student}, a : \mathtt{A}, r : \mathtt{R}) \text{ :- } a = \mathtt{assignGrade} \wedge \mathtt{authorOf}(s, r) \tag{R2.4}$$

$$\mathtt{log}(s : \mathtt{Grader}, a : \mathtt{A}, r : \mathtt{R}) \text{ :- } a = \mathtt{assignGrade} \wedge \mathtt{deny}(s, a, r). \tag{R2.5}$$

Rule (R2.1) says that students are permitted to edit assignments which they authored. Rule (R2.2) says that graders are permitted to assign grades to homework assignments. Rule (R2.3) says that professors are permitted to assign grades to any resource. Rule (R2.4) says that students are denied from assigning their own grades. Rule (R2.5) says to log the request when a grader attempts to assign a grade and is denied. Clearly this is not a complete policy, but it is enough to demonstrate the semantics of a policy without getting bogged down in irrelevant details. The policy is also strictly stratifiable, since we can assign $\mathtt{permit}$ and $\mathtt{deny}$ to stratum 0 and $\mathtt{log}$ to stratum 1.

Since $\mathtt{permit}$, $\mathtt{deny}$ and $\mathtt{log}$ are just predicate symbols, a model that satisfies this theory may assign the same tuple to its relation for $\mathtt{deny}$ and to its relation for $\mathtt{permit}$ or assign a tuple to neither relation with no compunction. We construct a model $\mathbb{M}$ and examine the result of applying $[\![\Gamma_{\mathsf{G}}]\!]$ to $\mathbb{M}$. Define $\mathbb{M}$ as

$$\begin{aligned}\mathbb{M}_{\mathtt{Student}} &= \{\mathbf{tim}, \mathbf{sarah}\}\\ \mathbb{M}_{\mathtt{Grader}} &= \{\mathbf{sarah}\}\\ \mathbb{M}_{\mathtt{Professor}} &= \{\mathbf{dan}\}\\ \mathbb{M}_{\mathtt{A}} &= \{\mathbf{assignGrade}, \mathbf{edit}\}\\ \mathbb{M}_{\mathtt{Homework}} &= \{\mathbf{hw1}\}\\ \mathbb{M}_{\mathtt{Quiz}} &= \{\mathbf{quiz1}\}\\ \mathtt{authorOf}^{\mathbb{M}} &= \{(\mathbf{tim}, \mathbf{quiz1}), (\mathbf{sarah}, \mathbf{hw1})\}\\ \mathtt{edit}^{\mathbb{M}} &= \mathbf{edit}\\ \mathtt{assignGrade}^{\mathbb{M}} &= \mathbf{assignGrade}.\end{aligned}$$

The relations $\mathbb{M}_{\mathtt{S}}$, $\mathbb{M}_{\mathtt{A}}$, and $\mathbb{M}_{\mathtt{R}}$ are the unions of their subsorts' relations. Using the construction defined in Theorem 2.10 we construct $\mathbf{permit} = \mathtt{permit}^{[\![\Gamma_{\mathsf{G}}]\!](\mathbb{M})}$, $\mathbf{deny} = \mathtt{deny}^{[\![\Gamma_{\mathsf{G}}]\!](\mathbb{M})}$ and $\mathbf{log} =$

$\texttt{log}^{[\![\Gamma_G]\!](\mathbb{M})}$. We start with **permit** since `permit` is in the lowest stratum of the stratification we identified earlier. The relation **permit** contains the following tuples, due to the corresponding rules:

(**tim**, **edit**, **quiz1**) (**sarah**, **edit**, **hw1**)	R2.1
(**sarah**, **assignGrade**, **hw1**)	R2.2
(**dan**, **assignGrade**, **hw1**) (**dan**, **assignGrade**, **quiz1**)	R2.3

The relation **deny** contains:

(**tim**, **assignGrade**, **quiz1**) (**sarah**, **assignGrade**, **hw1**)	R2.4

The relation **log** contains:

(**sarah**, **assignGrade**, **hw1**)	R2.5

The tuple (**sarah**, **assignGrade**, **hw1**) is in both **permit** and **deny**. We could correct this by modifying rule R2.2 to be

$$\texttt{permit}(s : \texttt{Grader}, a : \texttt{A}, r : \texttt{Homework}) \texttt{ :- } a = \texttt{assignGrade} \wedge \neg\texttt{deny}(s, a, r), \tag{R2.2a}$$

but this way of handling conflicts among decisions would be difficult and inelegant for larger policies. Additionally, we have many tuples which are in neither **permit** nor **deny**, for example (**sarah**, **edit**, **quiz1**). To handle this situation, we might like our policy to deny anything that is not explicitly permitted. Both of these situations will be rectified using policy combinators, which are described in the next section.

Chapter 3

Policy combinators and policy combinator specifications

3.1 Policy combinators

As with policies, we begin with a general notion of policy combinators. A policy combinator is simply a function that takes several policies and produces a single policy.

Definition 3.1 (Policy combinator)**.** Let $\mathcal{L} = (\mathcal{S}, \subseteq, \mathcal{E})$ be a signature, $\mathcal{D}, \mathcal{D}_1, \ldots, \mathcal{D}_k$ be $\mathcal{S}^*$-indexed families of predicates, all of which are distinct from those in $\mathcal{E}$. Then a $(\mathcal{L}, \mathcal{D}, \langle \mathcal{D}_i \rangle_k)$-policy combinator is a function

$$P : \mathrm{Pol}^*(\mathcal{L}, \mathcal{D}_1) \times \cdots \times \mathrm{Pol}^*(\mathcal{L}, \mathcal{D}_k) \to \mathrm{Pol}(\mathcal{L}, \mathcal{D}).$$

The use of Pol^* instead of Pol is because we want to allow P to be applied to policies that do not depend on all of the relations provided by models of $\mathcal{L}$ and that provide information about decisions on which the resulting policy does not depend. Pol^* captures this property, as described in Definition 2.3. This is similar to how, in a typed programming language with subtyping, one may pass to a function a value of a subtype of the function's parameter type, and may handle the result of the function call as if it were a supertype of the actual return type.

3.2 Auxiliary definitions

In order to define policy combinator specifications, some auxiliary definitions are required. One necessity is a mechanism for combining signatures and models that are sufficiently similar.

Definition 3.2 ($\oplus$)**.** Let $\mathcal{L}_1 = (\mathcal{S}_1, \leq_1, \mathcal{E}_1)$ and $\mathcal{L}_2 = (\mathcal{S}_2, \leq_2, \mathcal{E}_2)$ be signatures. If $\mathcal{L}_1$ and $\mathcal{L}_2$ have the same sorts and subsort relations and all of their common functions and predicates have the same ranks and arities, then $\mathcal{L}_1$ and $\mathcal{L}_2$ are *compatible*.

For compatible signatures $\mathcal{L}_1$ and $\mathcal{L}_2$, denote $\mathcal{L}_1 \oplus \mathcal{L}_2 = (\mathcal{S}_1, \leq_1, \mathcal{E}_1 \uplus \mathcal{E}_2)$. Recall that $\uplus$ denotes the pointwise union, as defined in Definition 2.1.

Let $\mathbb{M}_1$ and $\mathbb{M}_2$ be models of compatible signatures $\mathcal{L}_1$ and $\mathcal{L}_2$ respectively. If $\mathbb{M}_1$ and $\mathbb{M}_2$ agree on the interpretations for their common predicate and function symbols, we call them *compatible* and denote the expansion of both $\mathbb{M}_1$ and $\mathbb{M}_2$ to $\mathcal{L}_1 \oplus \mathcal{L}_2$ as $\mathbb{M}_1 \oplus \mathbb{M}_2$.

The following fact about combining signatures and models will be necessary defining the semantics of policies.

Lemma 3.3. *$\mathbb{M}_1 \oplus \mathbb{M}_2$ is the unique model which is an expansion of both of $\mathbb{M}_1$ and $\mathbb{M}_2$ to $\mathcal{L}_1 \oplus \mathcal{L}_2$.*

Proof. The relations and functions assigned to the common predicate and function symbols clearly must be preserved. For any predicate or function symbol that is not part of both signatures, the associated relation or function is completely defined in the expansion by the model whose signature does define it. The model exists and since there is no choice about construction the combined model, it must be unique. □

In order to refer unambiguously to the decision predicates of the policies to be combined in policy combinator specifications, the decision predicates must be unique to each policy that is to be combined, and must be distinct from the decisions of and the symbols in the signature of the resulting policy. This requirement can be achieved by renaming the decisions. To this end, two additional notations are necessary. The first notation indicates the renaming of the symbols in a family of predicates by prepending a symbol to each symbol in the family.

Definition 3.4 ($n.\mathcal{I}$)**.** Let $\mathcal{S}$ be a set of sorts, let $\mathcal{I}$ be a $\mathcal{S}^*$-indexed family of predicate symbols, and let n be a symbol. We denote $n.\mathcal{I} = \mathcal{I}'$ where for each $w \in \mathcal{S}^*$, $\mathcal{I}'_w = \{n.d \mid d \in \mathcal{I}_w\}$. That is, $n.\mathcal{I}$ denotes the predicate family $\mathcal{I}$ with each predicate prepended by the symbol n.

The second notation is for mapping models to equivalent models in signatures with renamed predicates. The notation is overloaded for use with signatures, models and theories.

Definition 3.5 ($\rho_{D,n}$)**.** Let $\mathcal{L} = (\mathcal{S}, \leq, \mathcal{E})$ be a signature, D be a set of predicates in $\mathcal{E}$, and n be a symbol. Denote $\rho_{D,n}(\mathcal{L}) = (\mathcal{S}, \leq, \mathcal{E}')$ where $\mathcal{E}'$ is $\mathcal{E}$ with each predicate d in D replaced by $n.d$.

Let $\mathbb{M}$ be a model of $\mathcal{L}$. Then $\rho_{D,n}(\mathbb{M}) = \mathbb{M}'$ is a model of $\rho_{D,n}(\mathcal{L})$ with the same universe as $\mathbb{M}$, where $n.d^{\mathbb{M}'} = d^{\mathbb{M}}$ for each $d \in D$, and $q^{\mathbb{M}'} = q^{\mathbb{M}}$ for each predicate and function symbol q of $\rho_{D,n}(\mathcal{L})$ not in D.

Let Γ be a theory of $\mathcal{L}$. Then $\rho_{D,n}(\Gamma)$ is a theory of $\rho_{D,n}(\mathcal{L})$ where every predicate symbol d from D appearing in Γ is replaced with $n.d$. It is clear that for every $\mathbb{M}$ of $\mathcal{L}$, $\mathbb{M}$ satisfies Γ if and only if $\rho_{D,n}(\mathbb{M})$ satisfies $\rho_{D,n}(\Gamma)$.

When an indexed family of predicates is used in the place of D in $\rho_{D,n}$, treat it as if it were the set of predicates in the family.

All of the groundwork necessary for considering policy combinator specifications has been defined.

3.3 Policy combinator specifications

We are only going to consider a particular class of policy combinators. In particular, let P be a $(\mathcal{L}, \mathcal{D}, \langle \mathcal{D}_i \rangle_k)$-policy combinator and $p_1, \ldots, p_k$ be policies of the appropriate types for P. Denote $p = P(p_1, \ldots, p_k)$. We consider policy combinators where $p(\mathbb{M})$ is dependent only on $p_1(\mathbb{M}), \ldots, p_k(\mathbb{M})$ and on $\mathbb{M}$ itself. Interestingly, if we insist that the decisions predicates for all p_i are distinct, it is equivalent for p to be dependent on the combination $\mathbb{M} \oplus p_1(\mathbb{M}) \oplus \cdots \oplus p_k(\mathbb{M})$. It would even make sense to define p in terms of p', by

$$p(\mathbb{M}) = p'\left((\mathbb{M} \oplus p_1(\mathbb{M}) \oplus \cdots \oplus p_k(\mathbb{M}))\downarrow_{\mathcal{L}^{\mathcal{D}_1 \uplus \ldots \uplus \mathcal{D}_k}}\right)\downarrow_{\mathcal{L}^{\mathcal{D}}}$$

where the proper reduct of $\mathbb{M}$ is taken for each p_i. What we have done with this is define a $(\mathcal{L}, \mathcal{D}, \langle \mathcal{D}_i \rangle)$-policy combinator P in terms of a $\left(\mathcal{L}^{\mathcal{D}_1 \uplus \cdots \uplus \mathcal{D}_k}, \mathcal{D}\right)$-policy! This is exactly the approach that we will take to define policy combinator specifications. The only difference is that instead of insisting that the arguments to a policy combinator be distinct, we will rename them.

Definition 3.6 (Policy combinator specification)**.** A policy combinator specification is a 5-tuple

$$(\mathcal{L}, \mathcal{D}, \mathcal{I}, \langle n_i, \mathcal{D}_i \rangle_k, \Gamma)$$

where

- $\mathcal{L} = (\mathcal{S}, \leq, \mathcal{E})$ is a signature, where the predicates of $\mathcal{E}$ are called the extensional predicates of the policy combinator specification,
- $\mathcal{D}$ is a $\mathcal{S}^*$-indexed family of predicate symbols, the decision predicates of the policy combinator specification, which are distinct from the symbols of $\mathcal{E}$,
- $\mathcal{I}$ is a $\mathcal{S}^*$-indexed family of predicate symbols, called the intensional predicates of the policy combinator specification, which are distinct from the symbols in $\mathcal{E}$ and which include all of $\mathcal{D}$,
- $\langle n_i, \mathcal{D}_i \rangle_k$ is a k-length sequence of pairs each comprised of a symbol n_i and a $\mathcal{S}^*$-indexed family of predicates $\mathcal{D}_i$, with each n_i distinct and no intensional or extensional predicates having any n_i as a prefix, and
- Γ is a set of $\left(\mathcal{L}^{n_1.\mathcal{D}_1 \uplus \cdots \uplus n_k.\mathcal{D}_k}, \mathcal{I}\right)$-rules.

The symbols $n_1, \ldots, n_k$ are the parameters of the policy combinator specification.

As with policy specifications, we will refer to the set of rules Γ as a $(\mathcal{L}, \mathcal{D}, \mathcal{I}, \langle n_i, \mathcal{D}_i \rangle_k)$-policy combinator specification and use the notation for the set of rules Γ as shorthand for the full policy specification $(\mathcal{L}, \mathcal{D}, \mathcal{I}, \langle n_i, \mathcal{D}_i \rangle_k, \Gamma)$. When every intensional predicate is a decision predicate, we may omit $\mathcal{I}$ and refer to Γ as a $(\mathcal{L}, \mathcal{D}, \langle n_i, \mathcal{D}_i \rangle_k)$-policy combinator specification.

These requirements intentionally ensure that $\left(\mathcal{L}^{n_1.\mathcal{D}_1 \uplus \cdots \uplus n_k.\mathcal{D}_k}, \mathcal{D}, \mathcal{I}, \Gamma\right)$ is a policy specification. As with policy specifications, there is an additional property that the policy combinator specifications for which we define semantics must satisfy.

Definition 3.7 (Unique minimal expansion property)**.** Let $(\mathcal{L}, \mathcal{D}, \mathcal{I}, \langle n_i, \mathcal{D}_i \rangle_k, \Gamma)$ be a policy combinator specification. If $\left(\mathcal{L}^{n_1.\mathcal{D}_1 \uplus \cdots \uplus n_k.\mathcal{D}_k}, \mathcal{D}, \mathcal{I}, \Gamma\right)$ has the unique minimal expansion property (for policy specifications) then $(\mathcal{L}, \mathcal{D}, \mathcal{I}, \langle n_i, \mathcal{D}_i \rangle_k, \Gamma)$ has the unique minimal expansion property (for policy combinator specifications).

The semantics of policy combinator specifications are derived from our original observation about the connection between policy combinators and policies.

Definition 3.8 ($\underline{[\![\cdot]\!]}$)**.** Let $(\mathcal{L}, \mathcal{D}, \mathcal{I}, \langle n_i, \mathcal{D}_i \rangle_k, \Gamma)$ be a policy combinator specification with the unique minimal expansion property. Then $\underline{[\![}\, (\mathcal{L}, \mathcal{D}, \mathcal{I}, \langle n_i, \mathcal{D}_i \rangle_k, \Gamma) \,\underline{]\!]}$ is a $(\mathcal{L}, \mathcal{D}, \langle \mathcal{D}_i \rangle_k)$-policy combinator, defined as follows.

Let each p_i be a $(\mathcal{L}_i, \mathcal{D}'_i)$-policy, where $\mathcal{L}_i$ is a reduct of $\mathcal{L}$ and for each $w \in \mathcal{S}^*$, $\mathcal{D}_{i,w} \subseteq \mathcal{D}'_{i,w}$. This makes each p_i an element of $\mathrm{Pol}^*(\mathcal{L}, \mathcal{D})$. For each p_i define

$$p'_i(\mathbb{M}) = \rho_{\mathcal{D}_i, n_i}(p_i(\mathbb{M}\downarrow_{\mathcal{L}_i})\downarrow_{\mathcal{L}_i^{\mathcal{D}_i}}).$$

Then $\underline{[\![}\,(\mathcal{L},\mathcal{D},\mathcal{I},\langle n_i,\mathcal{D}_i\rangle_k,\Gamma)\,\underline{]\!]}$ is defined by

$$\underline{[\![}\,(\mathcal{L},\mathcal{D},\mathcal{I},\langle n_i,\mathcal{D}_i\rangle_k,\Gamma)\,\underline{]\!]}(p_1,\ldots,p_k) = \lambda\mathbb{M}\,.\,[\![(\mathcal{L}^{n_1.\mathcal{D}_1\uplus\cdots\uplus n_k.\mathcal{D}_k},\mathcal{D},\mathcal{I},\Gamma)]\!]\,(p_1'(\mathbb{M})\oplus\cdots\oplus p_k'(\mathbb{M})\oplus\mathbb{M})\,.$$

That $\underline{[\![}\cdot\,\underline{]\!]}$ is well-defined is not as clear as it is that $[\![\cdot]\!]$ is well defined.

Theorem 3.9 ($\underline{[\![}\cdot\,\underline{]\!]}$ is well-defined)**.** *Let Γ be a $(\mathcal{L},\mathcal{D},\mathcal{I},\langle n_i,\mathcal{D}_i\rangle_k)$-policy combinator specification with the unique minimal expansion property, let $p_i\in \mathrm{Pol}^*(\mathcal{L},\mathcal{D}_i)$ be a policy for $0\le i\le k$. Then there exists a unique $\mathbb{M}'$ such that $\underline{[\![}\Gamma\underline{]\!]}(\mathbb{M})=\mathbb{M}'$.*

Proof. We have that

$$\underline{[\![}\Gamma\underline{]\!]}(p_1,\ldots,p_k) = \lambda\mathbb{M}\,.\,[\![\Gamma]\!]\,(p_1'(\mathbb{M})\oplus\cdots\oplus p_k'(\mathbb{M})\oplus\mathbb{M})\restriction_{\mathcal{L}^{\mathcal{D}}}$$

where for each p_i'

$$p_i'(\mathbb{M}) = \rho_{\mathcal{D}_i,n_i}(p_i(\mathbb{M}\restriction_{\mathcal{L}_i})\restriction_{\mathcal{L}_i^{\mathcal{D}_i}}).$$

The models $p_i'(\mathbb{M})$ and $\mathbb{M}$ are reducts of $\mathbb{M}$ (a model is trivially a reduct of itself), with relations for distinct predicates added. The added predicates are distinct because of renaming. Since the added predicate symbols are distinct, their signatures are compatible. Since the common relations are all the result of taking the reduct of one model, the models are compatible as well, and so their combination exists and is unique, by Lemma 3.3. Because of the unique minimal expansion property, $[\![\Gamma]\!]$ is well-defined. The reduct of a model also exists and is unique. Therefore $\underline{[\![}\Gamma\underline{]\!]}$ is well-defined. □

As with policy specifications, there is a syntactic property of policy combinator specifications that ensures that policy combinator specifications have the unique minimal expansion property.

Definition 3.10 (Strict stratification of policy combinator specifications)**.** A strict stratification of a $(\mathcal{L},\mathcal{D},\mathcal{I},\langle n_i,\mathcal{D}_i\rangle)$-policy combinator specification Γ is a strict stratification of the rules Γ as a $(\mathcal{L}^{n_1.\mathcal{D}_1,\ldots,n_k.\mathcal{D}_k},\mathcal{D},\mathcal{I})$-policy specification. If a strict stratification exists for a policy combinator specification, we say that the policy combinator specification is strictly stratifiable.

It is not hard to see that strictly stratifiable policy combinator specifications have the unique minimal expansion property.

Theorem 3.11. *Let Γ be a $(\mathcal{L},\mathcal{D},\mathcal{I},\langle n_i,\mathcal{D}_i\rangle_k)$-policy combinator specification. Then Γ has the unique minimal expansion property.*

Proof. Since Γ is a strictly stratifiable $(\mathcal{L}^{n_1.\mathcal{D}_1,\ldots,n_k.\mathcal{D}_k},\mathcal{D},\mathcal{I})$-policy specification, by Theorem 2.10 it has the unique minimal expansion property. Therefore Γ has the unique minimal expansion property. □

Since strictly stratifiable policy combinator specifications have the unique model expansion property, their semantics are defined by $\underline{[\![}\cdot\,\underline{]\!]}$. We will only be working with policy combinator specifications that are strictly stratifiable in this thesis.

Remark. Policy combinators defined by policy combinator specifications do not require that the policies they operate on be defined by policy specifications from Definition 2.5. They only require that the policies to be combined be able to derive their decision relations from order-sorted logic models. This generality means that the semantics of policy combinators support the combination of heterogeneous policies.

3.4 Policy combinator specification examples

We present examples of policy combinator specifications in order to demonstrate the definitions. The first addresses the need to add conflict resolution and a default decision to the example from Section 2.3. Letting $\mathcal{L}_{\mathsf{G}}$ and $\mathcal{D}_{\mathsf{G}}$ be as they were in that earlier example, the following is a $(\mathcal{L}_{\mathsf{G}}, \mathcal{D}_{\mathsf{G}}, \langle \texttt{pol}, \mathcal{D}_{\mathsf{G}} \rangle)$-policy combinator specification.

$$\texttt{permit}(s:\texttt{S}, a:\texttt{A}, r:\texttt{R}) \texttt{ :- } \texttt{pol.permit}(s,a,r) \wedge \neg \texttt{pol.deny}(s,a,r) \tag{R3.1}$$

$$\texttt{deny}(s:\texttt{S}, a:\texttt{A}, r:\texttt{R}) \texttt{ :- } \texttt{pol.deny}(s,a,r)$$

$$\texttt{log}(s:\texttt{S}, a:\texttt{A}, r:\texttt{R}) \texttt{ :- } \texttt{pol.log}(s,a,r)$$

$$\texttt{deny}(s:\texttt{S}, a:\texttt{A}, r:\texttt{R}) \texttt{ :- } \neg\texttt{permit}(s,a,r) \tag{R3.2}$$

Rule (R3.1) states that the `permit` decision should hold whenever the underlying `pol.permit` decision holds and the underlying `pol.deny` decision does not hold. This allows `pol.deny` to override `pol.permit` in a very straightforward way. In the orignal example from Section 2.3, the tuple $(\textbf{sarah}, \textbf{assignGrade}, \textbf{hw1})$ was in both **pol.permit** and **pol.deny**. Since the tuple is in both relations, it will not be in **permit** becaue $\neg\texttt{pol.deny}(s,a,r)$ in rule (R3.1) prevents it. However, it will still be in **deny**, because **deny** is specified to contain all of **pol.deny**.

Rule (R3.2) states that whenever the final `permit` decision does not hold, the `deny` decision should, which has the effect of making `deny` the default decision. In the original example, **tim** and **sarah** are permitted to edit their own quiz and homework resources, but they are not denied from editing each other's. With the application of this policy combinator, because the tuples $(\textbf{tim}, \textbf{assignGrade}, \textbf{hw1})$ and $(\textbf{sarah}, \textbf{assignGrade}, \textbf{quiz1})$ are not in **pol.permit**, they will be in **deny**. In addition to these tuples, anything that is not in **permit** once it is constructed, that is anything not explicitly permitted, will be in **deny**.

Since the specification does not actually depend on $\mathcal{L}_{\mathsf{G}}$, it is more general than originally indicated. It is not difficult to see how the specification could be generalized to a template specification over different decision sets for combinations of decision overriding and establishing default decisions, which is a point that will appear again in the next example and in Section 5.2.

In Section 1.2 we raised the problem of determining what semantics to assign to the decision conflict resolution declaration

```
overrides permit deny
overrides deny rootAccess.
```

The question was: if all three decisions hold for a particular tuple before rule conflict resolution, what decisions should result from the resolution, `permit` and `rootAccess` or just `rootAccess`?[1] Our formalism allows us to encode either possibility in a straightforward way. The first interpretation is written

$$\begin{aligned}
\texttt{permit}(x:A) &\texttt{ :- } \texttt{pol.permit}(x) \wedge \neg\texttt{pol.deny}(x) \\
\texttt{permit}(x:A) &\texttt{ :- } \texttt{pol.permit}(x) \wedge \texttt{pol.rootAccess}(x) \\
\texttt{deny}(x:A) &\texttt{ :- } \texttt{pol.deny}(x) \wedge \neg\texttt{pol.rootAccess}(x) \\
\texttt{rootAccess}(x:A) &\texttt{ :- } \texttt{pol.rootAccess}(x),
\end{aligned}$$

[1] We had a different example for showing why we would want just the one decision in the introduction, but we will use the same example twice here for conciseness and so that the two interpretations are easier to compare.

and the second is written

$$\begin{aligned}
\texttt{permit}(x : A) \texttt{ :- } & \texttt{pol.permit}(x) \wedge \neg \texttt{pol.deny}(x) \\
\texttt{deny}(x : A) \texttt{ :- } & \texttt{pol.deny}(x) \wedge \neg \texttt{pol.rootAccess}(x) \\
\texttt{rootAccess}(x : A) \texttt{ :- } & \texttt{pol.rootAccess}(x).
\end{aligned}$$

Whichever is selected, the semantics of the declaration would then be clear to the user of the policy specification language. If a policy specification language author was so inclined, a declaration for each choice could be offered, and the the semantics of each would be perfectly understandable.

We can also specify policy combinators to accomplish what is commonly achieved by algebraic operators on policies. For example, we could define the union of two policies, by saying that whenever a decision holds in either policy it holds in the result. To do so, let $\mathcal{D}$ be an $\mathcal{S}^*$-indexed family of decision predicates. For each sort $A_1 \cdots A_k \in \mathcal{S}^*$ and each decision $d \in \mathcal{D}_{A_1 \cdots A_k}$ include in $\Gamma_\cup$ the rules

$$\begin{aligned}
d(x_1 : A_1, \ldots, x_k : A_k) \texttt{ :- } & n_1.d(x_1, \ldots, x_k) \\
d(x_1 : A_1, \ldots, x_k : A_k) \texttt{ :- } & n_2.d(x_1, \ldots, x_k).
\end{aligned}$$

A concrete example, borrowing from earlier $\mathcal{L}_\mathsf{G}$ as the signature and $\mathcal{D}_\mathsf{G}$ as the family of decision symbols, is

$$\begin{aligned}
\texttt{permit}(s : \texttt{S}, a : \texttt{A}, r : \texttt{R}) \texttt{ :- } & \texttt{pol}_1\texttt{.permit}(s, a, r) \\
\texttt{deny}(s : \texttt{S}, a : \texttt{A}, r : \texttt{R}) \texttt{ :- } & \texttt{pol}_1\texttt{.deny}(s, a, r) \\
\texttt{log}(s : \texttt{S}, a : \texttt{A}, r : \texttt{R}) \texttt{ :- } & \texttt{pol}_1\texttt{.log}(s, a, r) \\
\texttt{permit}(s : \texttt{S}, a : \texttt{A}, r : \texttt{R}) \texttt{ :- } & \texttt{pol}_2\texttt{.permit}(s, a, r) \\
\texttt{deny}(s : \texttt{S}, a : \texttt{A}, r : \texttt{R}) \texttt{ :- } & \texttt{pol}_2\texttt{.deny}(s, a, r) \\
\texttt{log}(s : \texttt{S}, a : \texttt{A}, r : \texttt{R}) \texttt{ :- } & \texttt{pol}_2\texttt{.log}(s, a, r)
\end{aligned}$$

This yields a general pattern for producing policy combinators that take the union of policies.

3.5 Policy combinator application

It is important to understand the policies that result from applying a policy combinator to specific policies. In order to examine that resulting policy, one can make use of Theorem 3.19 and Theorem 3.20, which are given in this chapter, to construct the specification to which the policy corresponds.

The proofs of the theorems proceed by demonstrating small equivalencies between policy specifications. The main proofs rely on several equivalencies whose proofs are broken out into lemmas for ease of reading.

3.5.1 Lemmas

The first lemma shows how a reduction of a model can be moved outside of a renaming of predicates in the model.

Lemma 3.12. *(Moving $\restriction_{\mathcal{L}}$ outside of $\rho_{\mathcal{I},n}$) Let $\mathcal{L} = (\mathcal{S}, \leq, \mathcal{E})$ be a signature, let $\mathcal{I}$ be a $\mathcal{S}^*$-indexed family of predicates, all of which are in $\mathcal{E}$, let $\mathbb{M}$ be a model of some expansion of $\mathcal{L}$, and let n be a symbol such that no symbol of $\mathcal{E}$ has n as a prefix. Then*

$$\rho_{\mathcal{I},n}(\mathbb{M}\restriction_{\mathcal{L}}) = \rho_{\mathcal{I},n}(\mathbb{M})\restriction_{\rho_{\mathcal{I},n}(\mathcal{L})}.$$

In particular, if $\mathcal{L}$ is of the form $\mathcal{L}_1^{\mathcal{I}}$, then

$$\rho_{\mathcal{I},n}(\mathbb{M}\restriction_{\mathcal{L}_1^{\mathcal{I}}}) = \rho_{\mathcal{I},n}(\mathbb{M})\restriction_{\mathcal{L}_1^{n.\mathcal{I}}}.$$

Proof. The relations that are removed by the reduct on the left-hand side of the equation are still removed by the reduct on the right-hand side, just possibly under a different name. No additional relations are removed, and none that remain have their contents changed.

The renaming on the left-hand side are renaming the same predicates without modifying any others. On the right-hand side the predicates renamed are preserved by the reduction since the signature that the model is reduced to is also renamed. □

The second involves renaming a policy instead of renaming the model that the policy produces.

Lemma 3.13 (Rename policy instead of model)**.** *Let $(\mathcal{L}, \mathcal{D}, \mathcal{I}, \Gamma)$ be a strictly stratifiable policy specification, let $\mathbb{M}$ be a model of $\mathcal{L}$, and let n be a symbol. Then*

$$\rho_{\mathcal{D},n}([\![(\mathcal{L}, \mathcal{D}, \mathcal{I}, \Gamma)]\!](\mathbb{M})) = [\![(\mathcal{L}, n.\mathcal{D}, n.\mathcal{I}, \rho_{\mathcal{I},n}(\Gamma))]\!](\mathbb{M})$$

Proof. Since for any $\mathbb{M}^+$ of $\mathcal{L}^{\mathcal{I}}$ that satisfies Γ, we have that $\rho_{\mathcal{I},n}(\mathbb{M}^+)$ satisfies $\rho_{\mathcal{I},n}(\Gamma)$, the equivalence is clear. The only oddity is that on the left-hand side we rename only $\mathcal{D}$. This is because applying the policy corresponding to the specification involves taking the reduct to $\mathcal{L}^{\mathcal{D}}$, so the rest of $\mathcal{I}$ is not present in the model to be renamed. □

The third shows how policies can be promoted to handle models of expanded signatures without affecting the result of applying the policy.

Lemma 3.14 (Promote policy to expanded signature)**.** *Let $(\mathcal{L}, \mathcal{D}, \mathcal{I}, \Gamma)$ be a strictly stratifiable policy specification, let $\mathbb{M}$ be a model of $\mathcal{L}^+$ which is an expansion of $\mathcal{L}$, let $\mathcal{D}'$ be a family of predicates all from $\mathcal{D}$, and let n be a symbol. Assume that $\mathcal{L}^+$ and $\mathcal{I}$ share no predicate symbols. Then*

$$[\![(\mathcal{L}, \mathcal{D}, \mathcal{I}, \Gamma)]\!](\mathbb{M}\restriction_{\mathcal{L}})\restriction_{\mathcal{L}^{\mathcal{D}'}} = [\![(\mathcal{L}^+, \mathcal{D}, \mathcal{I}, \Gamma)]\!](\mathbb{M})\restriction_{\mathcal{L}^{\mathcal{D}'}}$$

Proof. Since $\mathcal{I}$ and $\mathcal{L}^+$ share no predicate symbols, there is no way for the extra tuples in $\mathbb{M}$ to have an effect on what instances of $\mathcal{I}$ are required for the expansion of $\mathbb{M}$ to satisfy Γ. Also, since we are taking the reduct to $\mathcal{L}^{\mathcal{D}'}$ after applying the policy, instances of predicates of $\mathcal{L}^+$ that are not in $\mathcal{L}$ will be removed from the model, giving the same result as the left-hand side. □

Lemma 3.15 shows that instead of taking a reduct to a language that includes only some of the decision predicates, we can simply designate the predicates as non-decision predicates.

Lemma 3.15 (Remove decisions instead of reduct)**.** *Let $(\mathcal{L}, \mathcal{D}, \mathcal{I}, \Gamma)$ be a strictly stratifiable policy specification, let $\mathbb{M}$ be a model of $\mathcal{L}$, and let $\mathcal{D}'$ be a family of predicates all from $\mathcal{D}$. Then*

$$[\![(\mathcal{L}, \mathcal{D}, \mathcal{I}, \Gamma)]\!](\mathbb{M})\restriction_{\mathcal{L}^{\mathcal{D}'}} = [\![(\mathcal{L}, \mathcal{D}', \mathcal{I}, \Gamma)]\!](\mathbb{M})$$

Proof. The definition of policy application already includes taking a reduct to the original signature with the set of decision predicates added. Rather than taking a second reduct using a smaller set of predicates after applying the policy, the original set of decision predicates can be reduced, essentially treating the removed decision predicates as if they were just non-decision predicates in $\mathcal{I}$. □

Lemma 3.16 is one of the three key pieces of the proof of the theorem: the ability to turn model combination into the union of the policy specifications of the policies that created the models.

Lemma 3.16. *(Policy specification combination to union) Let* $(\mathcal{L}, \mathcal{D}_1, \mathcal{I}_1, \Gamma_1)$ *and* $(\mathcal{L}, \mathcal{D}_2, \mathcal{I}_2, \Gamma_2)$ *be strictly stratifiable policy specifications where* $\mathcal{I}_1$ *and* $\mathcal{I}_2$ *share no predicates. Then, for every model* $\mathbb{M}$ *of* $\mathcal{L}$*,*

$$[\![(\mathcal{L}, \mathcal{D}_1, \mathcal{I}_1, \Gamma_1)]\!](\mathbb{M}) \oplus [\![(\mathcal{L}, \mathcal{D}_2, \mathcal{I}_2, \Gamma_2)]\!](\mathbb{M}) = [\![(\mathcal{L}, \mathcal{D}_1 \uplus \mathcal{D}_2, \mathcal{I}_1 \uplus \mathcal{I}_2, \Gamma_1 \cup \Gamma_2)]\!](\mathbb{M}).$$

Proof. Γ_1 defines the instances of predicates of $\mathcal{I}_1$ and Γ_2 defines the instances of the predicates of $\mathcal{I}_2$ independently, and because $\mathcal{I}_1$ and $\mathcal{I}_2$ share no predicates, Γ_1 does not refer to any predicate of $\mathcal{I}_2$ and Γ_2 does not refer to any predicate of $\mathcal{I}_1$. Thus, the right-hand side of the equation defines the same model that combining two separately generated models defines, which is the left-hand side of the equation. □

The second key piece of the proof of the theorem is that the composition of two policies with specifications is equivalent to another policy with a specification.

Lemma 3.17 (Composition to union of policy specifications)**.** *Let* $\left(\mathcal{L}^{\mathcal{D}'}, \mathcal{D}, \mathcal{I}, \Gamma\right)$ *and* $(\mathcal{L}, \mathcal{D}', \mathcal{I}', \Gamma')$ *be strictly stratifiable policy specifications such that* $\mathcal{I}$ *and* $\mathcal{I}'$ *share no symbols. Then, for each model* $\mathbb{M}$ *of* $\mathcal{L}$*,*

$$\left[\!\left[\left(\mathcal{L}^{\mathcal{D}'}, \mathcal{D}, \mathcal{I}, \Gamma\right)\right]\!\right]([\![(\mathcal{L}, \mathcal{D}', \mathcal{I}', \Gamma')]\!](\mathbb{M})) = [\![(\mathcal{L}, \mathcal{D}, \mathcal{I} \uplus \mathcal{I}', \Gamma \cup \Gamma')]\!](\mathbb{M})$$

Proof. First apply Lemma 3.15 and Lemma 3.14 to the left-hand side of the equation.

$$\begin{aligned}
&\left[\!\left[\left(\mathcal{L}^{\mathcal{D}'}, \mathcal{D}, \mathcal{I}, \Gamma\right)\right]\!\right]([\![(\mathcal{L}, \mathcal{D}', \mathcal{I}', \Gamma')]\!](\mathbb{M})) \\
=&\left[\!\left[\left(\mathcal{L}^{\mathcal{D}'}, \mathcal{D}, \mathcal{I}, \Gamma\right)\right]\!\right]([\![(\mathcal{L}, \mathcal{I}', \mathcal{I}', \Gamma')]\!](\mathbb{M})\restriction_{\mathcal{L}^{\mathcal{D}'}}) \\
=&\left[\!\left[\left(\mathcal{L}^{\mathcal{I}'}, \mathcal{D}, \mathcal{I}, \Gamma\right)\right]\!\right]([\![(\mathcal{L}, \mathcal{I}', \mathcal{I}', \Gamma')]\!](\mathbb{M}))
\end{aligned}$$

The application of $[\![(\mathcal{L}, \mathcal{I}', \mathcal{I}', \Gamma')]\!]$ to $\mathbb{M}$ is just populating the instances of predicates in $\mathcal{I}'$ before the application of $\left[\!\left[\left(\mathcal{L}^{\mathcal{I}'}, \mathcal{D}, \mathcal{I}, \Gamma\right)\right]\!\right]$ populates the instances of predicates in $\mathcal{I}$, just as would happen according to a stratification of $\Gamma \cup \Gamma'$ where all of the predicates in $\mathcal{I}'$ were assigned to lower strata than any of $\mathcal{I}$. Since no predicate in $\mathcal{I}$ appears in Γ' and the predicates of $\mathcal{I}'$ only appear in the bodies of rules in Γ, such a stratification exists. Thus instead of performing an application of each policy in order, we can achieve the same result with one application of

$$[\![(\mathcal{L}, \mathcal{D}, \mathcal{I} \uplus \mathcal{I}', \Gamma \cup \Gamma')]\!].$$

□

Theorem 3.19 requires one more lemma, showing that combining the application of a policy to a model with another model can be treated as applying the policy to the combination of the models.

Lemma 3.18 (Combining to application)**.** *Let $(\mathcal{L}, \mathcal{D}, \mathcal{I}, \Gamma)$ be a strictly stratifiable policy specification and $\mathcal{L}'$ be a signature compatible with $\mathcal{L}^{\mathcal{I}}$ that shares no symbols with $\mathcal{I}$. Then for any compatible models $\mathbb{M}$ of $\mathcal{L}$ and $\mathbb{M}'$ of $\mathcal{L}'$,*

$$[\![(\mathcal{L}, \mathcal{D}, \mathcal{I}, \Gamma)]\!](\mathbb{M}) \oplus \mathbb{M}' = [\![(\mathcal{L} \oplus \mathcal{L}', \mathcal{D}, \mathcal{I}, \Gamma)]\!](\mathbb{M} \oplus \mathbb{M}').$$

Proof.

$$[\![(\mathcal{L}, \mathcal{D}, \mathcal{I}, \Gamma)]\!](\mathbb{M}) \oplus \mathbb{M}'$$

Expanding and then reducting back to the original language is the identity, and reducting a model to the language of the model is the identity.

$$= [\![(\mathcal{L}, \mathcal{D}, \mathcal{I}, \Gamma)]\!]((\mathbb{M} \oplus \mathbb{M}')|_{\mathcal{L}})|_{\mathcal{L}^{\mathcal{D}}} \oplus \mathbb{M}'$$

By Lemma 3.14, the policy can be promoted to handle the expanded model.

$$= [\![(\mathcal{L} \oplus \mathcal{L}', \mathcal{D}, \mathcal{I}, \Gamma)]\!](\mathbb{M} \oplus \mathbb{M}')|_{\mathcal{L}^{\mathcal{D}}} \oplus \mathbb{M}'$$

Since the relations in $\mathbb{M}'$ that we are removing with the reduct are being recombined,

$$= [\![(\mathcal{L} \oplus \mathcal{L}', \mathcal{D}, \mathcal{I}, \Gamma)]\!](\mathbb{M} \oplus \mathbb{M}').$$

□

3.5.2 Policy combinator application theorems

Theorem 3.19 states that *partially* applying a policy combinator to a sequence of policies, where the policy combinator and policies all have strictly stratifiable specifications (as defined in Definition 3.10 and Definition 2.9, respectively), yields a *policy combinator* that has a strictly stratifiable specification.

Theorem 3.19. *Let $(\mathcal{L}, \mathcal{D}, \mathcal{I}, \langle n_i, \mathcal{D}_i \rangle_k, \Gamma)$ be a strictly stratifiable policy combinator specification. Fix j where $1 \le j \le k$. Let $(\mathcal{L}_i, \mathcal{D}'_i, \mathcal{I}_i, \Gamma_i)$ be a strictly stratifiable policy specification where $[\![(\mathcal{L}_i, \mathcal{D}'_i, \mathcal{I}_i, \Gamma_i)]\!] \in \mathrm{Pol}^*(\mathcal{L}, \mathcal{D}_i)$ for $1 \le i \le j$. Then,*

$$\begin{aligned}
&\underline{[\![}\,(\mathcal{L}, \mathcal{D}, \mathcal{I}, \langle n_i, \mathcal{D}_i \rangle_k, \Gamma)\,\underline{]\!]}([\![(\mathcal{L}_1, \mathcal{D}'_1, \mathcal{I}_1, \Gamma_1)]\!], \ldots, [\![(\mathcal{L}_j, \mathcal{D}'_j, \mathcal{I}_j, \Gamma_j)]\!]) \\
&= \underline{[\![}\,(\mathcal{L}, \mathcal{D}, \mathcal{I} \uplus n_1.\mathcal{I}_1 \uplus \cdots \uplus n_j.\mathcal{I}_j, \langle (n_{j+1}, \mathcal{D}_{j+1}), \ldots, (n_k, \mathcal{D}_k) \rangle, \Gamma \cup \rho_{\mathcal{I}_1, n_1}(\Gamma_1) \cup \cdots \cup \rho_{\mathcal{I}_j, n_j}(\Gamma_j))\,\underline{]\!]}.
\end{aligned}$$

Proof. Let $p_i \in \mathrm{Pol}^*(\mathcal{L}, \mathcal{D}_i)$ be a $(\mathcal{L}_i, \mathcal{D}'_i)$-policy for $j+1 \le i \le k$. Note that each signature $\mathcal{L}_i$ is a reduct of $\mathcal{L}$ and that each $\mathcal{D}'_i$ includes all of $\mathcal{D}_i$ for $1 \le i \le k$. Define $\mathbb{M}'$ as

$$\mathbb{M}' = \bigoplus_{j+1 \le i \le k} \rho_{\mathcal{D}_i, n_i}(p_i(\mathbb{M}|_{\mathcal{L}_i})|_{\mathcal{L}_i^{\mathcal{D}_i}}).$$

We show how to simplify the application of policies in the context of the policy combinator semantics definition.

$$
\begin{aligned}
&\rho_{\mathcal{D}_i,n_i}(\llbracket(\mathcal{L}_i,\mathcal{D}'_i,\mathcal{I}_i,\Gamma_i)\rrbracket(\mathbb{M}\restriction_{\mathcal{L}_i})\restriction_{\mathcal{L}_i^{\mathcal{D}_i}}) \\
&= (\rho_{\mathcal{D}_i,n_i}(\llbracket(\mathcal{L}_i,\mathcal{D}'_i,\mathcal{I}_i,\Gamma_i)\rrbracket(\mathbb{M}\restriction_{\mathcal{L}_i})))\restriction_{\mathcal{L}_i^{\mathcal{D}_i}} && \text{by Lemma 3.12} \\
&= \llbracket(\mathcal{L},n_i.\mathcal{D}'_i,n_i.\mathcal{I}_i,\rho_{\mathcal{I}_i,n_i}(\Gamma_i))\rrbracket(\mathbb{M}\restriction_{\mathcal{L}_i})\restriction_{\mathcal{L}_i^{n_i.\mathcal{D}_i}} && \text{by Lemma 3.13} \\
&= \llbracket(\mathcal{L},n_i.\mathcal{D}'_i,n_i.\mathcal{I}_i,\rho_{\mathcal{I}_i,n_i}(\Gamma_i))\rrbracket(\mathbb{M})\restriction_{\mathcal{L}_i^{\mathcal{D}_i}} && \text{by Lemma 3.14}
\end{aligned}
$$

Then we take the definition of the policy combinator semantics and manipulate the expression until we get the desired result.

$$\underline{\llbracket}(\mathcal{L},\mathcal{D},\mathcal{I},\langle n_i,\mathcal{D}_i\rangle_k,\Gamma)\underline{\rrbracket}(\llbracket(\mathcal{L}_1,\mathcal{D}'_1,\mathcal{I}_1,\Gamma_1)\rrbracket,\ldots,\llbracket(\mathcal{L}_j,\mathcal{D}'_j,\mathcal{I}_j,\Gamma_j)\rrbracket,p_{j+1},\ldots,p_k)(\mathbb{M})$$

Expand via the Definition 3.8, using $\mathbb{M}'$ in place of the combination of the applications of the p_i.

$$= \llbracket(\mathcal{L}^{n_1.\mathcal{D}_1,\ldots,n_k.\mathcal{D}_k},\mathcal{D},\mathcal{I},\Gamma)\rrbracket\left(\bigoplus_{1\le i\le j}\rho_{\mathcal{D}_i,n}\left(\llbracket(\mathcal{L}_i,\mathcal{D}'_i,\mathcal{I}_i,\Gamma_i)\rrbracket(\mathbb{M}\restriction_{\mathcal{L}_i})\restriction_{\mathcal{L}_i^{\mathcal{D}_i}}\right)\oplus\mathbb{M}'\oplus\mathbb{M}\right)$$

Apply the simplification described earlier.

$$= \llbracket(\mathcal{L}^{n_1.\mathcal{D}_1,\ldots,n_k.\mathcal{D}_k},\mathcal{D},\mathcal{I},\Gamma)\rrbracket\left(\bigoplus_{1\le i\le j}\llbracket(\mathcal{L},n_i.\mathcal{D}'_i,n_i.\mathcal{I}_i,\rho_{\mathcal{I}_i,n_i}(\Gamma_i))\rrbracket(\mathbb{M})\restriction_{\mathcal{L}_i^{n_i.\mathcal{D}_i}}\oplus\mathbb{M}'\oplus\mathbb{M}\right)$$

By Lemma 3.15 and since the relations in $\mathbb{M}$ that we are removing with the reduct are being recombined, remove the reduct.

$$= \llbracket(\mathcal{L}^{n_1.\mathcal{D}_1,\ldots,n_k.\mathcal{D}_k},\mathcal{D},\mathcal{I},\Gamma)\rrbracket\left(\bigoplus_{1\le i\le j}\llbracket(\mathcal{L},n_i.\mathcal{D}_i,n_i.\mathcal{I}_i,\rho_{\mathcal{I}_i,n_i}(\Gamma_i))\rrbracket(\mathbb{M})\oplus\mathbb{M}'\oplus\mathbb{M}\right)$$

By Lemma 3.16, turn the combinations into unions on the policy specifications, denoting $\bar{\mathcal{D}} = n_1.\mathcal{D}_1 \uplus \cdots \uplus n_j.\mathcal{D}_j$, $\bar{\mathcal{I}} = n_1.\mathcal{I}_1 \uplus \cdots \uplus n_j.\mathcal{I}_j$ and $\bar{\Gamma} = \rho_{\mathcal{I}_1,n_1}(\Gamma_1)\cup\cdots\cup\rho_{\mathcal{I}_j,n_j}(\Gamma_j)$.

$$= \llbracket(\mathcal{L}^{n_1.\mathcal{D}_1,\ldots,n_k.\mathcal{D}_k},\mathcal{D},\mathcal{I},\Gamma)\rrbracket\left(\llbracket(\mathcal{L},\bar{\mathcal{D}},\bar{\mathcal{I}},\bar{\Gamma})\rrbracket(\mathbb{M})\oplus(\mathbb{M}'\oplus\mathbb{M})\right)$$

By Lemma 3.18, turn the combination into an application of the poilcy.

$$= \llbracket(\mathcal{L}^{n_1.\mathcal{D}_1,\ldots,n_k.\mathcal{D}_k},\mathcal{D},\mathcal{I},\Gamma)\rrbracket\left(\llbracket(\mathcal{L},\bar{\mathcal{D}},\bar{\mathcal{I}},\bar{\Gamma})\rrbracket(\mathbb{M}'\oplus\mathbb{M})\right)$$

By Lemma 3.17, turn the sequenced application into a union of policy specifications.

$$= \llbracket(\mathcal{L},\mathcal{D},\mathcal{I}\uplus\bar{\mathcal{I}},\Gamma\cup\bar{\Gamma})\rrbracket(\mathbb{M}'\oplus\mathbb{M})$$

Which is the result we want, by Definition 3.8.

$$= \underline{\llbracket}(\mathcal{L},\mathcal{D},\mathcal{I}\uplus\bar{\mathcal{I}},\langle(n_{j+1},\mathcal{D}_{j+1}),\ldots,(n_k,\mathcal{D}_k)\rangle,\Gamma\cup\bar{\Gamma})\underline{\rrbracket}(p_{j+1},\ldots,p_k)(\mathbb{M})$$

□

Theorem 3.20 is a corollary to Theorem 3.19 stating that a policy combinator *fully* applied to policies, where the policy combinator and policies all have strictly stratifiable specifications, yields a *policy* that has a strictly stratifiable policy specification.

Theorem 3.20. *Let* $(\mathcal{L}, \mathcal{D}, \mathcal{I}, \langle n_i, \mathcal{D}_i \rangle_k, \Gamma)$ *be a strictly stratifiable policy combinator specification. Let* $(\mathcal{L}_i, \mathcal{D}'_i, \mathcal{I}_i, \Gamma_i)$ *be a strictly stratifiable policy specification where* $[\![(\mathcal{L}_i, \mathcal{D}'_i, \mathcal{I}_i, \Gamma_i)]\!] \in \mathrm{Pol}^*(\mathcal{L}, \mathcal{D}_i)$ *for* $1 \le i \le k$*. Then,*

$$\underline{[\![}(\mathcal{L}, \mathcal{D}, \mathcal{I}, \langle n_i, \mathcal{D}_i \rangle_k, \Gamma)\underline{]\!]}([\![(\mathcal{L}_1, \mathcal{D}'_1, \mathcal{I}_1, \Gamma_1)]\!], \ldots, [\![(\mathcal{L}_k, \mathcal{D}'_k, \mathcal{I}_k, \Gamma_k)]\!]) = [\![(\mathcal{L}, \mathcal{D}, \mathcal{I} \uplus n_1.\mathcal{I}_1 \uplus \cdots \uplus n_k.\mathcal{I}_k, \Gamma \cup \rho_{\mathcal{I}_1, n_1}(\Gamma_1) \cup \cdots \cup \rho_{\mathcal{I}_k, n_k}(\Gamma_k))]\!].$$

Proof. Start by applying Theorem 3.19.

$$\begin{aligned}&\underline{[\![}(\mathcal{L}, \mathcal{D}, \mathcal{I}, \langle n_i, \mathcal{D}_i \rangle_k, \Gamma)\underline{]\!]}([\![(\mathcal{L}_1, \mathcal{D}'_1, \mathcal{I}_1, \Gamma_1)]\!], \ldots, [\![(\mathcal{L}_k, \mathcal{D}'_k, \mathcal{I}_k, \Gamma_k)]\!])(\mathbb{M})\\ =&\underline{[\![}\left(\mathcal{L}, \mathcal{D}, \mathcal{I} \uplus n_1.\mathcal{I}_1 \uplus \cdots \uplus n_k.\mathcal{I}_k, \langle\rangle, \Gamma \cup \rho_{\mathcal{I}_1, n_1}(\Gamma_1) \cup \cdots \cup \rho_{\mathcal{I}_j, n_k}(\Gamma_k)\right)\underline{]\!]}()(\mathbb{M})\end{aligned}$$

Then use Definition 3.8.

$$= [\![(\mathcal{L}, \mathcal{D}, \mathcal{I} \uplus n_1.\mathcal{I}_1 \uplus \cdots \uplus n_k.\mathcal{I}_k, \Gamma \cup \rho_{\mathcal{I}_1, n_1}(\Gamma_1) \cup \cdots \cup \rho_{\mathcal{I}_k, n_k}(\Gamma_k))]\!](\mathbb{M})$$

□

Flattening the application of policy combinators allows for the techniques in Chapter 4 to be used to analyze hierarchies of policies and policy combinators.

3.6 Policy combinator application example

We have already constructed a policy specification in Section 2.3 and a policy combinator specification in Section 3.4 with the intent that the latter be applied to the former. To review, the policy specification $(\mathcal{L}_\mathsf{G}, \mathcal{D}_\mathsf{G}, \mathcal{D}_\mathsf{G}, \Gamma_\mathsf{G})$ had decision set $\mathcal{D}_\mathsf{G}$ composed of `permit`, `deny`, and `log`, all with arity $\mathtt{S} \times \mathtt{A} \times \mathtt{R}$. The rules Γ_G are

$$\begin{aligned}\mathtt{permit}(s : \mathtt{Student}, a : \mathtt{A}, r : \mathtt{R}) &\text{ :- } a = \mathtt{edit} \wedge \mathtt{authorOf}(s, r)\\ \mathtt{permit}(s : \mathtt{Grader}, a : \mathtt{A}, r : \mathtt{Homework}) &\text{ :- } a = \mathtt{assignGrade}\\ \mathtt{permit}(s : \mathtt{Professor}, a : \mathtt{A}, r : \mathtt{R}) &\text{ :- } a = \mathtt{assignGrade}\\ \mathtt{deny}(s : \mathtt{Student}, a : \mathtt{A}, r : \mathtt{R}) &\text{ :- } a = \mathtt{assignGrade} \wedge \mathtt{authorOf}(s, r)\\ \mathtt{log}(s : \mathtt{Grader}, a : \mathtt{A}, r : \mathtt{R}) &\text{ :- } a = \mathtt{assignGrade} \wedge \mathtt{deny}(s, a, r).\end{aligned}$$

The policy combinator specification $(\mathcal{L}_\mathsf{G}, \mathcal{D}_\mathsf{G}, \mathcal{D}_\mathsf{G}, \langle \mathtt{pol}, \mathcal{D}_\mathsf{G} \rangle, \Gamma'_\mathsf{G})$ has the rules

$$\begin{aligned}\mathtt{permit}(s : \mathtt{S}, a : \mathtt{A}, r : \mathtt{R}) &\text{ :- } \mathtt{pol.permit}(s, a, r) \wedge \neg \mathtt{pol.deny}(s, a, r)\\ \mathtt{deny}(s : \mathtt{S}, a : \mathtt{A}, r : \mathtt{R}) &\text{ :- } \mathtt{pol.deny}(s, a, r)\\ \mathtt{log}(s : \mathtt{S}, a : \mathtt{A}, r : \mathtt{R}) &\text{ :- } \mathtt{pol.log}(s, a, r)\\ \mathtt{deny}(s : \mathtt{S}, a : \mathtt{A}, r : \mathtt{R}) &\text{ :- } \neg \mathtt{permit}(s, a, r).\end{aligned}$$

To find the rules for the policy specification for the policy

$$[\![(\mathcal{L}_{\mathsf{G}}, \mathcal{D}_{\mathsf{G}}, \mathcal{D}_{\mathsf{G}}, \langle \mathtt{pol}, \mathcal{D}_{\mathsf{G}} \rangle)]\!]([\![(\mathcal{L}_{\mathsf{G}}, \mathcal{D}_{\mathsf{G}}, \mathcal{D}_{\mathsf{G}}, \Gamma_{\mathsf{G}})]\!]),$$

rename Γ_{G} and take the union of the result with Γ'_{G} to get

$$\begin{aligned}
\mathtt{pol.permit}(s : \mathtt{Student}, a : \mathtt{A}, r : \mathtt{R}) &\text{ :- } a = \mathtt{edit} \wedge \mathtt{authorOf}(s, r) \\
\mathtt{pol.permit}(s : \mathtt{Grader}, a : \mathtt{A}, r : \mathtt{Homework}) &\text{ :- } a = \mathtt{assignGrade} \\
\mathtt{pol.permit}(s : \mathtt{Professor}, a : \mathtt{A}, r : \mathtt{R}) &\text{ :- } a = \mathtt{assignGrade} \\
\mathtt{pol.deny}(s : \mathtt{Student}, a : \mathtt{A}, r : \mathtt{R}) &\text{ :- } a = \mathtt{assignGrade} \wedge \mathtt{authorOf}(s, r) \\
\mathtt{pol.log}(s : \mathtt{Grader}, a : \mathtt{A}, r : \mathtt{R}) &\text{ :- } a = \mathtt{assignGrade} \wedge \mathtt{pol.deny}(s, a, r). \\
\mathtt{permit}(s : \mathtt{S}, a : \mathtt{A}, r : \mathtt{R}) &\text{ :- } \mathtt{pol.permit}(s, a, r) \wedge \neg \mathtt{pol.deny}(s, a, r) \\
\mathtt{deny}(s : \mathtt{S}, a : \mathtt{A}, r : \mathtt{R}) &\text{ :- } \mathtt{pol.deny}(s, a, r) \\
\mathtt{log}(s : \mathtt{S}, a : \mathtt{A}, r : \mathtt{R}) &\text{ :- } \mathtt{pol.log}(s, a, r) \\
\mathtt{deny}(s : \mathtt{S}, a : \mathtt{A}, r : \mathtt{R}) &\text{ :- } \neg \mathtt{permit}(s, a, r).
\end{aligned}$$

Starting with the model from the example in Section 2.3, the relation **pol.permit** here ends up being the same as **permit** in the original example. The relation **permit** will be the same as **permit** in the example from Section 3.4. Despite the complexity of the proofs of the theorems involved, the resulting specification for a policy combinator composed with a policy is simple and understandable.

Chapter 4

Analysis

4.1 Compiling a policy

Here we change focus from defining policy and policy combinator specifications to the techniques necessary for analyzing those specifications. This chapter discusses how to convert the rules of a policy into an order-sorted logic theory that can be used in conjunction with a tool like KodKod [Torlak and Jackson, 2007], which is how Margrave currently operates [Nelson, 2010; Nelson et al., 2010]. The goal is to take the specification for a policy p and produce a theory such that the set of models that satisfy the theory is the image of p Recall that p is a model expanding function, so the image of p is a set of models that include the instances of the decision predicates.

Definition 4.1 (compile)**.** Let Γ be a $(\mathcal{L}, \mathcal{D}, \mathcal{I})$-policy specification, with $\mathcal{L} = (\mathcal{S}, \leq, \mathcal{E})$. Define

$$\mathsf{compile}(\Gamma) = \left\{ \forall^{A_1} v_1 \cdots \forall^{A_k} v_k \,.\, d(v_1, \ldots, v_k) \iff \bigvee \mathsf{bodies}(\Gamma, d, \langle v_1, \ldots, v_k \rangle) \mid A_i \in \mathcal{S} \wedge d \in \mathcal{I}_{A_1 \cdots A_k} \right\}$$

where

$$\mathsf{bodies}(\Gamma, d, \langle v_1, \ldots, v_k \rangle) = \left\{ \exists^{B_1} x_1 \cdots \exists^{B_k} x_k \,.\, v_1 = x_1 \wedge \cdots \wedge x_k = v_k \wedge \alpha \mid [\forall^{B_1} x_1 \cdots \forall^{B_k} x_k \,.\, d(x_1, \ldots, x_k) \Leftarrow \alpha] \in \Gamma \right\}$$

such that $v_1, \ldots, v_k$ are fresh variables.

For each decision, the resulting formula states that for every tuple, the decision holds if and only if that tuple matches the types declared in a rule and the body of that rule is true for the tuple. Only tuples that are necessarily true by virtue of the rules are included in models that satisfy the compiled theory. Note that the sorts of the variables bound by the existential quantifiers are different from the sorts of the variables bound by the universal quantifiers. This is because while each decision is declared over one particular sort $A_1 \cdots A_k$, rules may be written with respect to any sort $B_1 \cdots B_k$ where $B_i \leq A_i$ for $1 \leq i \leq k$. In order to have rules only apply to the types on which they are declared, the combination of universal and existential quantifiers is required.

It is not correct to ask for the image of a policy $[\![(\mathcal{L}, \mathcal{D}, \mathcal{I}, \Gamma)]\!]$ to be the set of models that satisfies $\mathsf{compile}(\mathcal{L}, \mathcal{D}, \mathcal{I}, \Gamma)$, since the theory is over $\mathcal{L}^{\mathcal{I}}$ not over $\mathcal{L}^{\mathcal{D}}$. Instead we want the set of models in the image before the reducts are taken.

Theorem 4.2. *Let Γ be a $(\mathcal{L}, \mathcal{D}, \mathcal{I})$-policy specification and $\mathbb{M}$ a model of $\mathcal{L}^{\mathcal{I}}$. Then $\mathbb{M}$ satisfies* $\mathsf{compile}(\Gamma)$ *if and only if there is some model $\mathbb{M}'$ of $\mathcal{L}$ such that the minimal expansion of $\mathbb{M}'$ to $\mathcal{L}^{\mathcal{I}}$ that satisfies Γ is $\mathbb{M}$.*

Proof. ($\Rightarrow$) Assume that $\mathbb{M}$ satisfies $\mathsf{compile}(\Gamma)$. Let $\mathbb{M}^- = \mathbb{M}{\downarrow_{\mathcal{L}}}$. We are going to show that $\mathbb{M}$ is the model arrived at by applying the procedure in Theorem 2.10 to $\mathbb{M}^-$. Since $\mathbb{M}$ is an expansion of $\mathbb{M}^-$ we know that the instance of each extensional predicate is the same in both models. Let σ be a stratification of Γ. Pick some d in the lowest stratum of σ. The tuples in $d^{\mathbb{M}}$ are exactly the tuples added to the institution of d by the procedure. In particular, for every assignment that satisfies the body of some rule of d in Γ, the tuple is a witness for some disjunct on the right-hand side of the bi-implication in the formula for d in $\mathsf{compile}(\Gamma)$, and satisfaction only depends on the part of $\mathbb{M}$ that we know matches the model we are building, which so far is $\mathbb{M}^-$. This means that it satisfies the left-hand side of the bi-implication and so the tuples added by the procedure are in $d^{\mathbb{M}}$. Those that do not satisfy any rule with d as the head are not added by the procedure and do not appear in $d^{\mathbb{M}}$ because the sentences in $\mathsf{compile}(\Gamma)$ are bi-implications. The same argument applies for higher strata, with the instances of predicates in the lower strata having been confirmed to match the results of the procedure from Theorem 2.10.

($\Leftarrow$) Assume $\mathbb{M}$ is the least expansion of $\mathbb{M}{\downarrow_{\mathcal{L}}}$ that satisfies Γ. The same argument as above works in the other direction to show that the procedure of Theorem 2.10 provides a model that satisfies $\mathsf{compile}(\Gamma)$. □

An unapplied policy combinator, or one that was partially applied and converted into an unapplied combinator using Theorem 3.19, can be analyzed in the same way by treating the decisions of the missing policies as part of the input model. This allows instances of those decision predicates to range freely, essentially making no assumptions about what behavior the policies that are arguments might take.

4.2 Example of compilation

Continuing with the example from Section 3.6, we demonstrate here how find $\mathsf{compile}(\Gamma)$, where Γ is the result from the applying Theorem 3.20 to the original policy and combinator,

$$
\begin{aligned}
\mathtt{pol.permit}(s:\mathtt{Student}, a:\mathtt{A}, r:\mathtt{R}) &\text{ :- } a = \mathtt{edit} \wedge \mathtt{authorOf}(s, r)\\
\mathtt{pol.permit}(s:\mathtt{Grader}, a:\mathtt{A}, r:\mathtt{Homework}) &\text{ :- } a = \mathtt{assignGrade}\\
\mathtt{pol.permit}(s:\mathtt{Professor}, a:\mathtt{A}, r:\mathtt{R}) &\text{ :- } a = \mathtt{assignGrade}\\
\mathtt{pol.deny}(s:\mathtt{Student}, a:\mathtt{A}, r:\mathtt{R}) &\text{ :- } a = \mathtt{assignGrade} \wedge \mathtt{authorOf}(s, r)\\
\mathtt{pol.log}(s:\mathtt{Grader}, a:\mathtt{A}, r:\mathtt{R}) &\text{ :- } a = \mathtt{assignGrade} \wedge \mathtt{pol.deny}(s, a, r).
\end{aligned}
$$

$$
\begin{aligned}
\mathtt{permit}(s:\mathtt{S}, a:\mathtt{A}, r:\mathtt{R}) &\text{ :- } \mathtt{pol.permit}(s, a, r) \wedge \neg\mathtt{pol.deny}(s, a, r)\\
\mathtt{deny}(s:\mathtt{S}, a:\mathtt{A}, r:\mathtt{R}) &\text{ :- } \mathtt{pol.deny}(s, a, r)\\
\mathtt{log}(s:\mathtt{S}, a:\mathtt{A}, r:\mathtt{R}) &\text{ :- } \mathtt{pol.log}(s, a, r)\\
\mathtt{deny}(s:\mathtt{S}, a:\mathtt{A}, r:\mathtt{R}) &\text{ :- } \neg\mathtt{permit}(s, a, r).
\end{aligned}
$$

Starting with the lowest stratum, we first find $\mathsf{bodies}(\Gamma, \texttt{pol.permit}, \langle v_s, v_a, v_r\rangle)$

$$\begin{aligned}
&\exists^{\texttt{Student}} s \,.\, \exists^{\texttt{A}} a \,.\, \exists^{\texttt{R}} r \,.\, s = v_s \wedge a = v_a \wedge r = v_r \wedge (a = \texttt{edit} \wedge \texttt{authorOf}(s, r)) \\
&\exists^{\texttt{Grader}} s \,.\, \exists^{\texttt{A}} a \,.\, \exists^{\texttt{Homework}} r \,.\, s = v_s \wedge a = v_a \wedge r = v_r \wedge (a = \texttt{assignGrade}) \\
&\exists^{\texttt{Professor}} s \,.\, \exists^{\texttt{A}} a \,.\, \exists^{\texttt{R}} r \,.\, s = v_s \wedge a = v_a \wedge r = v_r \wedge (a = \texttt{assignGrade}).
\end{aligned}$$

The other predicate in the lowest stratum is $\texttt{pol.deny}$, so $\mathsf{bodies}(\Gamma, \texttt{pol.deny}, \langle v_r, v_a, v_r\rangle)$ is

$$\exists^{\texttt{Student}} s \,.\, \exists^{\texttt{A}} a \,.\, \exists^{\texttt{R}} r \,.\, s = v_s \wedge a = v_a \wedge r = v_r \wedge (a = \texttt{assignGrade} \wedge \texttt{authorOf}(s, r)).$$

The set $\mathsf{bodies}(\Gamma, \texttt{pol.log}, \langle v_r, v_a, v_r\rangle)$ consists of the one formula

$$\exists^{\texttt{Grader}} s \,.\, \exists^{\texttt{A}} a \,.\, \exists^{\texttt{R}} r \,.\, s = v_s \wedge a = v_a \wedge r = v_r \wedge (a = \texttt{assignGrade} \wedge \texttt{pol.deny}(s, a, r)).$$

For the higher strata predicates $\texttt{permit}$, $\texttt{log}$ and $\texttt{deny}$, $\mathsf{bodies}(\Gamma, \texttt{permit}, \langle v_r, v_a, v_r\rangle)$ is

$$\exists^{\texttt{S}} s \,.\, \exists^{\texttt{A}} a \,.\, \exists^{\texttt{R}} r \,.\, s = v_s \wedge a = v_a \wedge r = v_r \wedge (\texttt{pol.permit}(s, a, r) \wedge \neg \texttt{pol.deny}(s, a, r)),$$

$\mathsf{bodies}(\Gamma, \texttt{deny}, \langle v_r, v_a, v_r\rangle)$ is

$$\begin{aligned}
&\exists^{\texttt{S}} s \,.\, \exists^{\texttt{A}} a \,.\, \exists^{\texttt{R}} r \,.\, s = v_s \wedge a = v_a \wedge r = v_r \wedge (\texttt{pol.deny}(s, a, r)) \\
&\exists^{\texttt{S}} s \,.\, \exists^{\texttt{A}} a \,.\, \exists^{\texttt{R}} r \,.\, s = v_s \wedge a = v_a \wedge r = v_r \wedge (\neg \texttt{permit}(s, a, r))
\end{aligned}$$

and $\mathsf{bodies}(\Gamma, \texttt{log}, \langle v_r, v_a, v_r\rangle)$

$$\exists^{\texttt{S}} s \,.\, \exists^{\texttt{A}} a \,.\, \exists^{\texttt{R}} r \,.\, s = v_s \wedge a = v_a \wedge r = v_r \wedge (\texttt{pol.log}(s, a, r)).$$

All together, this results in $\mathsf{compile}(\Gamma)$ being

$$\begin{aligned}
&\forall^{\texttt{S}} v_s \,.\, \forall^{\texttt{A}} v_a \,.\, \forall^{\texttt{R}} v_r \,.\, \texttt{pol.permit}(v_s, v_a, v_r) \iff \\
&\qquad [(\exists^{\texttt{Student}} s \,.\, \exists^{\texttt{A}} a \,.\, \exists^{\texttt{R}} r \,.\, s = v_s \wedge a = v_a \wedge r = v_r \wedge (a = \texttt{edit} \wedge \texttt{authorOf}(s, r))) \\
&\qquad\quad \vee (\exists^{\texttt{Grader}} s \,.\, \exists^{\texttt{A}} a \,.\, \exists^{\texttt{Homework}} r \,.\, s = v_s \wedge a = v_a \wedge r = v_r \wedge (a = \texttt{assignGrade})) \\
&\qquad\quad \vee (\exists^{\texttt{Professor}} s \,.\, \exists^{\texttt{A}} a \,.\, \exists^{\texttt{R}} r \,.\, s = v_s \wedge a = v_a \wedge r = v_r \wedge (a = \texttt{assignGrade}))] \\
&\forall^{\texttt{S}} v_s \,.\, \forall^{\texttt{A}} v_a \,.\, \forall^{\texttt{R}} v_r \,.\, \texttt{pol.deny}(v_s, v_a, v_r) \iff \\
&\qquad [\exists^{\texttt{Student}} s \,.\, \exists^{\texttt{A}} a \,.\, \exists^{\texttt{R}} r \,.\, s = v_s \wedge a = v_a \wedge r = v_r \wedge (a = \texttt{assignGrade} \wedge \texttt{authorOf}(s, r))] \\
&\forall^{\texttt{S}} v_s \,.\, \forall^{\texttt{A}} v_a \,.\, \forall^{\texttt{R}} v_r \,.\, \texttt{pol.log}(v_s, v_a, v_r) \iff \\
&\qquad [\exists^{\texttt{Grader}} s \,.\, \exists^{\texttt{A}} a \,.\, \exists^{\texttt{R}} r \,.\, s = v_s \wedge a = v_a \wedge r = v_r \wedge (a = \texttt{assignGrade} \wedge \texttt{pol.deny}(s, a, r))] \\
&\forall^{\texttt{S}} v_s \,.\, \forall^{\texttt{A}} v_a \,.\, \forall^{\texttt{R}} v_r \,.\, \texttt{permit}(v_s, v_a, v_r) \iff \\
&\qquad [\exists^{\texttt{S}} s \,.\, \exists^{\texttt{A}} a \,.\, \exists^{\texttt{R}} r \,.\, s = v_s \wedge a = v_a \wedge r = v_r \wedge (\texttt{pol.permit}(s, a, r) \wedge \neg \texttt{pol.deny}(s, a, r))] \\
&\forall^{\texttt{S}} v_s \,.\, \forall^{\texttt{A}} v_a \,.\, \forall^{\texttt{R}} v_r \,.\, \texttt{deny}(v_s, v_a, v_r) \iff \\
&\qquad [(\exists^{\texttt{S}} s \,.\, \exists^{\texttt{A}} a \,.\, \exists^{\texttt{R}} r \,.\, s = v_s \wedge a = v_a \wedge r = v_r \wedge (\texttt{pol.deny}(s, a, r))) \\
&\qquad\quad \vee (\exists^{\texttt{S}} s \,.\, \exists^{\texttt{A}} a \,.\, \exists^{\texttt{R}} r \,.\, s = v_s \wedge a = v_a \wedge r = v_r \wedge (\neg \texttt{permit}(s, a, r)))] \\
&\forall^{\texttt{S}} v_s \,.\, \forall^{\texttt{A}} v_a \,.\, \forall^{\texttt{R}} v_r \,.\, \texttt{log}(v_s, v_a, v_r) \iff \\
&\qquad [\exists^{\texttt{S}} s \,.\, \exists^{\texttt{A}} a \,.\, \exists^{\texttt{R}} r \,.\, s = v_s \wedge a = v_a \wedge r = v_r \wedge (\texttt{pol.log}(s, a, r))].
\end{aligned}$$

One benefit of this definition for $\mathsf{compile}$ is that the results of compilation are human readable and closely match what was written in the original policy, giving end users confidence that the results provided by an analysis engine really are results about the policy they wrote. The resulting theory becomes even more readable if a shorthand is used for $\exists^{B} b \,.\, b = v \wedge \alpha$, such as $\mathrm{isA}^{B} b \,.\, \alpha$, which makes the formulas look like

$$\begin{aligned}\forall^{\mathtt{S}} s \,.\, \forall^{\mathtt{A}} a \,.\, \forall^{\mathtt{R}} r \,.\, \mathtt{pol.permit}(s,a,r) \iff \\ [(\mathrm{isA}^{\mathtt{Student}} s \,.\, \mathrm{isA}^{\mathtt{A}} a \,.\, \mathrm{isA}^{\mathtt{R}} r \,.\, (a = \mathtt{edit} \wedge \mathtt{authorOf}(s,r))) \\ \vee (\mathrm{isA}^{\mathtt{Grader}} s \,.\, \mathrm{isA}^{\mathtt{A}} a \,.\, \mathrm{isA}^{\mathtt{Homework}} r \,.\, (a = \mathtt{assignGrade})) \\ \vee (\mathrm{isA}^{\mathtt{Professor}} s \,.\, \mathrm{isA}^{\mathtt{A}} a \,.\, \mathrm{isA}^{\mathtt{R}} r \,.\, (a = \mathtt{assignGrade}))],\end{aligned}$$

which can even be read aloud as "permit s, a, r if and only if s is a student, a is an action and r is a resource, where…".

4.3 Queries

The theory $\mathsf{compile}(\Gamma)$ becomes useful when considering how users would examine properties of their policy. For example, a user might wish to ask if it is ever permitted for a student to assign a grade to her own assignment. The question being asked is whether there is any model $\mathbb{M}$ such that $[\![\Gamma]\!](\mathbb{M})$ has the specific property, which could be expressed as a sentence of order-sorted logic α

$$\alpha \equiv \exists^{\mathtt{Student}} s \,.\, \exists^{\mathtt{R}} r \,.\, \mathtt{permit}(s, \mathtt{assignGrade}, r) \wedge \mathtt{authorOf}(s,r).$$

Since Theorem 4.2 provides a way for us to have a theory that only admits what essentially is the image of $[\![\Gamma]\!]$, checking that property amounts to asking about the satisfiability of

$$\mathsf{compile}(\Gamma) \cup \{\alpha\}\,.$$

On the other hand, asking if a student can always edit her own assignments is asking about the validity of

$$\mathsf{compile}(\Gamma) \cup \left\{\forall^{\mathtt{Student}} s \,.\, \forall^{\mathtt{R}} r \,.\, \mathtt{authorOf}(s,r) \implies \mathtt{permit}(s, \mathtt{edit}, r)\right\}.$$

Used in this fashion, Theorem 4.2 is the key to being able to perform analysis on policies. The theorem also demonstrates why it was important to be able to represent fully applied policy combinators as policy specifications as Theorem 3.20 states. In order to perform queries over fully applied policy combinators, we turn the policy combinators into policies, and then compile those policies into theories over which queries can be made.

As we stated earlier, partially applied policy combinators can be treated similarly. Instead of requiring application to more policies so that the combinator can be treated as a policy itself, moving the decisions from the parameters to the language of the policy allows them to range freely over possible sets of tuples, which allows for meaningful queries to be made. Specifically, a policy combinator specification $(\mathcal{L}, \mathcal{D}, \mathcal{I}, \langle n_i, \mathcal{D}_i \rangle_k, \Gamma)$ can be viewed as a combinator that has no parameters $\left(\mathcal{L}^{n_1 \cdot \mathcal{D}_1 \uplus \cdots \uplus n_k \cdot \mathcal{D}_k}, \mathcal{D}, \mathcal{I}, \langle\rangle, \Gamma\right)$, which can then be converted into a policy using Theorem 3.20. Queries should then be interpreted to include ranging over every possible sequence of policies to which the combinator could be applied. This means that if a particular query is satisfiable by some

of the models that the compiled theory admits, then there exists some sequence of policies where $[\![\Gamma]\!]$ applied to those policies will result in a policy whose image contains a model that satisfies the query. If a particular query is valid within the models that the compiled theory admits, then for every sequence of policies, $[\![\Gamma]\!]$ applied to those policies will result in a policy in which every model in the image of the policy satisfies the query.

Chapter 5

Margrave

5.1 Margrave policies

Since one purpose of this thesis was to address the lack of a clear way to extend semantics for Margrave policies and policy sets, here we present a way to encode current Margrave policies and policy sets as policy specifications and policy combinator specifications. The intention is that these definitions may serve as a starting point for embracing the possible generality of Margrave both mathematically and in the Margrave analyzer tool. The definitions of the abstract syntax and semantics of Margrave policies and Margrave policy sets presented in this chapter correspond to the Margrave language as of January 2012, which includes some changes to the language as presented by Nelson [2010].

There are a few significant differences between the formalism presented in this thesis and Margrave policies. The most obvious is that all of the decision predicates in a Margrave policy have the same arity. Additionally, Margrave policies include a target construct, which filters the tuples that may be included in the instance of a decision predicate. Policy specifications do not explicitly include such a construct, but an equivalent effect may be achieved by directly encoding the target formula into the appropriate rules. Rules in policy specifications do not have names, but because non-decision intensional predicates can be constructed, what would be the names of the rules can be encoded as non-decision intensional predicates, which can then be used in the bodies of decision predicates. Finally, instead of having a finite set of decision conflict resolution declarations, the formalism presented in this thesis permits the specification of arbitrary combinations of decisions via policy combinator specifications.

Definition 5.1 (Margrave policy)**.** A Margrave policy is a 6-tuple

$$(\mathcal{L}, \langle v_i : A_i \rangle_k, \texttt{target}, \{d_1, \ldots, d_j\}, R, c)$$

where

- $\mathcal{L}$ is a signature,
- $\langle v_i : A_i \rangle_k$ are the request variables of the policy, with each A_i being a sort in $\mathcal{L}$,
- `target`, the target of the policy, is a formula over $\mathcal{L}$ with free variables in $\{v_1, \ldots, v_k\}$,

- $\{d_1, \ldots, d_j\}$ are the decision predicates of the policy, each with arity $A_1 \cdots A_k$,
- R is a list of triples (r, d_r, α_r) where r is a rule name, d_r is a decision predicate with arity $A_1 \cdots A_k$ and α_r is a formula over $\mathcal{L}$ with free variables in $\{v_1, \ldots, v_k\}$, where the rule names are all distinct, and
- c is a decision conflict resolution declaration, consisting of either a set of decisions, the first-applicable set, or the edges of a directed acyclic graph on the set of all decisions, the overrides graph, or nothing.

A Margrave policy corresponds to a policy specification and a policy combinator specification, as defined in Definition 2.5 and in Definition 3.6, respectively. The policy specification encodes the rules, and the policy combinator specification encodes the decisions that result from the rules and the decision conflict resolution declarations.

Definition 5.2 (MARGRAVE)**.** Let $m = (\mathcal{L}, \langle v_i : A_i \rangle_k, \texttt{target}, \{d_1, \ldots, d_j\}, R, c)$ be a Margrave policy. Then MARGRAVE(m) is the policy specification found by applying Theorem 3.20 to policy combinator specification P with the policy specification p as its argument, where p is the policy specification $(\mathcal{L}, \mathcal{D}_p, \mathcal{D}_p, \Gamma_p)$ and P is the policy combinator specification $(\mathcal{L}, \mathcal{D}_P, \mathcal{D}_P, \langle \texttt{pol}, \mathcal{D}_p \rangle \Gamma_P)$. The decision set $\mathcal{D}_p$ consists of the rule names in R, all with the arity $A_1 \cdots A_k$. The rules Γ_p are

$$r(v_1 : A_1, \ldots, v_k : A_k) \text{ :- } \alpha_r$$

for each (r, d_r, α_r) in R. The decisions in the policy are the rule names instead of the original decisions in order to allow the policy combinator to handle the first-applicable decision conflict resolution declaration, which is dependent on the ordering of the rules themselves, rather than just the decisions. Since Γ_p is a set of rules, we distinguish between the results of each by giving each one a unique predicate.

The decisions of the policy combinator consist of the decisions $\{d_1, \ldots, d_j\}$ of the Margrave policy, all with arity $A_1 \cdots A_k$. The rules of the policy combinator are more complicated than the rules of the policy since they have to take the decision conflict resolution declarations into account, and they make the semantics of those declarations explicit. The rules Γ_P are

$$d_r(v_1 : A_1, \ldots, v_k : A_k) \text{ :- } \texttt{pol}.r(v_1, \ldots, v_k) \wedge \texttt{target} \wedge \neg(\bigvee \text{Rcomb}(r, \langle v_i \rangle_k)).$$

The target is included to achieve the effect of filtering tuples from the decision. Since it was written to have only the free variables $v_1, \ldots, v_k$, it can be included in the formula as it is. The set $\text{Rcomb}(r, \langle v_i \rangle_k)$ has a different definition for each possible decision conflict resolution declaration. If there is no declaration, then $\text{Rcomb}(r, \langle v_i \rangle_k)$ is the empty set, and the related portion of the formula reduces to vacuous truth. If the declaration is overrides, then the set consists of the formulas $\{\texttt{pol}.r'(v_1, \ldots, v_k)\}$ where there exists a non-empty path from d_r to $d_{r'}$ in the overrides graph. The acyclicity of the graph ensures that the policy remains stratifiable. If the declaration is first-applicable, then the set consists of the formulas $\{\texttt{pol}.r'(v_, \ldots, v_k)\}$ where both d_r and d'_r appear in the first-applicable set and r' precedes r in R.

The way in which Margrave constrains the arities of all of the decisions to be the same also forces all of the rules to be universally quantified over the same sorts. Because of this, in Margrave policies, one frequently finds statements constraining the sorts of tuples at the front of the bodies of Margrave policy rules. To produce a more readable translation, one could take this pattern into account, using the implicit constraints the sorts of tuples in the Margrave policy to determine more specific sort specifications on the rules produced, so that the extra constraints could be removed.

5.2 Margrave policy sets

Margrave policy sets are a way to specify policies depending on other policies and are semantically similar to fully applied policy combinators.

Definition 5.3 (Margrave policy set)**.** A Margrave policy set is a 5-tuple

$$\left(\mathcal{L}, \langle v_i : A_i \rangle_k, \texttt{target}, \langle n_i, m_i \rangle_j, c\right)$$

where

- $\mathcal{L}$ is a signature,
- $\langle v_i : A_i \rangle_k$ are the request variables of the policy set, with each A_i being a sort in $\mathcal{L}$,
- `target`, the target of the policy set, is a formula over $\mathcal{L}$ with free variables in $\{v_1, \ldots, v_k\}$,
- $\langle n_i, m_i \rangle_j$ are the Margrave policies or policy sets to which this combinator is applied, each with signature $\mathcal{L}$ and decision predicate sort $A_1 \cdots A_k$, and with each name n_i distinct,
- c is a decision conflict resolution declaration, consisting of either a set of decisions, the first-applicable set, or the edges of a directed acyclic graph on the set of all decisions, the overrides graph, or nothing.

The m_i may be Margrave policy sets because, as stated before, they are Margrave policies. The decision conflict resolution declarations are the same as they are in Margrave policies because policies are treated almost as if they were nothing more than compound rules in Margrave policy sets. The primary difference between the two is in how the first-applicable declaration interacts with the targets of the policies m_i, which can be seen from the encoding.

Definition 5.4 (MargraveSet)**.** For each Margrave policy m_i, $0 \leq i \leq j$, let the policy specification $(\mathcal{L}, \mathcal{D}_i, \mathcal{I}_i, \Gamma_i) = \textsc{Margrave}(m_i)$. Also let $m = \left(\mathcal{L}, \langle v_i : A_i \rangle_k, \texttt{target}, \langle n_i, m_i \rangle_j, c\right)$ be a Margrave policy set. Then $\textsc{MargraveSet}(m)$ is the policy specification corresponding to the application of Theorem 3.20 to the policy combinator specification $\left(\mathcal{L}, \mathcal{D}, \mathcal{D}, \langle n_i, \mathcal{D}_i \rangle_j, \Gamma\right)$ with arguments $(\mathcal{L}, \mathcal{D}_i, \mathcal{I}_i, \Gamma_i)$. The set of decisions $\mathcal{D}$ is the union of the argument policy decisions $\biguplus_{1 \leq i \leq j} \mathcal{D}_i$. The rules are

$$d(v_1 : A_1, \ldots, v_k : A_k) \text{ :- } n.d(v_1, \ldots, v_k) \wedge \texttt{target} \wedge \neg \text{SetRcomb}(d, n, \langle v_i \rangle_k)$$

for each decision d and policy name n, where d is a decision of the policy corresponding to n. As with Margrave, the set $\text{SetRcomb}(d, n, \langle v_i \rangle_k)$ is empty when there is no decision conflict resolution declaration. When there is an overrides declaration, the set consists of the formulas $\{n.d'(v_1, \ldots, v_k)\}$ where d' is a decision of the Margrave policy with name n and there exists a non-empty path form d to d' in the overrides graph. Once again the acyclicity of the graph ensures that the resulting policy combinator specification is stratifiable. If the declaration is first-applicable, then $\text{SetRcomb}(d, n, \langle v_i \rangle_k)$ consists of the formulas $\{\texttt{target}_{n'}\}$ where n' is the name of a Margrave policy with decision d' such that both d and d' appear in the first-applicable set, n' precedes n in the sequence $\langle n_1, \ldots, n_j \rangle$, and $\texttt{target}_{n'}$ is the target of the Margrave policy with name n'.

MargraveSet clearly extends to the case where a Margrave policy set is composed of other Margrave policy sets as well as Margrave policies.

Chapter 6

Related work

6.1 Margrave

Margrave was first presented by Fisler et al. [2005] as a tool for performing analysis of XACML policies, with an emphasis on change-impact analysis. Nelson [2010] and Nelson et al. [2010] expanded the semantics of Margrave to handle policies written in a slightly simplified version of the language discussed in Chapter 5. The Margrave policy language as discussed in Chapter 5 matches the semantics of the tool as of January 2012.

6.2 Policies and policy combination

There are a menagerie of policy languages and other policy specification techniques in existence [Abad-Piero et al., 1999; Bai and Varadharajan, 1997; Dougherty et al., 2006; Halpern and Weissman, 2008; Koch et al., 2001; Kolaczek, 2003; Li and Wang, 2008; McLean, 1988; Samarati and de Vimercati, 2001; Woo and Lam, 1992, 1993; Zhang et al., 2005], but most of them include mechanisms only for decision conflict resolution and not for policy combination, which is the primary focus of this thesis. Additionally, the focus of many of them is on dynamic policies which is outside of the scope of this thesis. Margrave is the only policy language of which we know that allows the user to define arbitrary decisions, and the policy specifications presented here appear to be the first to allow requests of varying sorts.

Bonatti et al. [2000, 2002] define an algebra for combining policies and provide a translation from policies expressed by the algebra into logic programming, which clearly can be captured by policy specifications. However, the policies that can be combined using their algebra only can expresses positive permit decisions. We initially considered an extension of their approach for expressing policy combination, but found that it became too cumbersome when working with policies that had varying decisions and request vector sorts. Wijesekera and Jajodia [2001, 2002, 2003] extend Bonatti's work to dynamic policies, which are outside the scope of this thesis.

Jajodia et al. [2001] are concerned with being able to capture the different access control policies expressible by the common policy languages at the time. They do not address the combination of policies, though they do have a Datalog-like language for expressing policies, including sort hierarchies for users, roles and objects; however, request vectors are fixed to be user, action, object

triples.

Backes et al. [2003, 2004] define an algebra for combining policies with the semantics of IBM's Enterprise Privacy Authorization Language [IBM, 2011]. Backes et al. [2003] provide only one mechanism for policy combination, with the intent to capture the combination of mandatory and discretionary policies. The mechanism amounts to a first applicable rule conflict resolution declaration for the decision portion of the policies (obligations are outside of the scope of this thesis). They expand their policy combination mechanism to an algebra over policy decisions [Backes et al., 2004], introducing two binary operators AND and OR, and a unary scoping operator. The semantics of the AND and OR operators amounts to a three-way overrides of "deny" overriding "don't care" overriding "permit" for AND and "permit" overriding "don't care" overriding "deny" for OR. Both of these are encodable as policy combinator specifications using the same mechanism as was used in encoding Margrave's overrides decision conflict resolution declaration. The scoping operator restricts policies based on the sorts of the components of the request vector. The effect can be achieved in a policy combinator specification using the same technique as is used with Margrave's targets, including in them a formula that checks the sort of the components of the requests, such as $\exists^{\texttt{Student}} s \,.\, s = v_s \wedge \cdots$.

Hughes and Bultan [2004, 2007, 2008] define operators for combining rules and policies for the purpose of decision conflict resolution. Their work captures the decision conflict resolution declarations available in XACML in a composable way, but does not go beyond that, since the focus of their work is on the analysis of, not the expression of, policies.

Mazzoleni et al. [2006, 2008] discuss the integration of policies at points of enforcement, where the policies must be evaluated to determine decisions. This is distinct from policy combination: policy integration refers to the decision of a point of enforcement of whether and how to evaluate a request with respect to a third-party policy in situations where policies may be provided by different entities.

Bruns and Huth [2008] and Bruns et al. [2007] use Belnap logic to capture the combination of policies. Belnap logic is a four-valued logic, where the decisions "permit", "deny", "not applicable" and "conflict" can be mapped to the values. Though the policy language PBel that they define based on Belnap logic is functionally complete, PBel does not support more decisions. Additionally, PBel is parameterized over the language for basic policies and request vectors, but it does not appear that PBel itself is useful for defining basic policies, and so it does not provide the consistency of semantics that policy specifications and policy combinator specifications do.

Li et al. [2009] define a language called PCL for declaring policy combination algorithms. PCL allows a policy combination algorithm to be defined either in terms of a 4-by-4 matrix on the possible XACML [OASIS, 2011] decisions "permit", "deny", "not applicable" and "indeterminate" which is expanded automatically to handle the results from an arbitrary number of policies, or in terms of constraints on the number of each decision returned from the composed policies. For any fixed number of policies, the algorithms can be encoded in a policy combinator specification, though in the case where the resulting decision is determined by constraints on the number of each decision returned from composed policies, the policy combinator specification is much more verbose. PCL does not support more possible decisions, dispatching to a combined policy based on a property of the environment or weighting the results of different policies being combined, all of which are supported by policy combinator specifications, although with varying levels of verbosity.

Ni et al. [2009] define a type of multi-valued algebra called $\mathfrak{D}$-algebra for composing policies. Though $\mathfrak{D}$-algebra appears sufficiently expressive for a core policy combinator language, we did not find it suitable for defining policies. Since Margrave already uses order-sorted logic as the

underlying semantics for defining policies, it is more natural and more convenient to view policy combinators as nothing more than policies operating on an expanded environment because that requires little to be added to the existing semantics in order to either understand or to provide tool support for policy combinators.

Rao et al. [2009, 2011] define the Fine-grained Integration Algebra (FIA) for combining policies. FIA bears strong resemblance to the algebra defined by Bonatti et al. [2000, 2002], with the main difference being that FIA handles explicit "permit", "deny" and "not applicable" decisions instead of explicit "permit" decisions only. Though they do not provide a compilation to logic programming as Bonatti et al. do, it is easy to see that a similar encoding would suffice to capture FIA as policy combinator specifications.

6.3 Order-sorted logic

Goguen [1992] did seminal work on order-sorted algebra; Goguen and Diaconescu [1994] present a good survey of the field through 1994. Order sorted predicate logic was first considered by Oberschelp [1990]. We make use of the presentation in Nelson [2010], since it represents the understanding of order-sorted logic as used in the Margrave language and analysis tool.

Chapter 7

Future work

There are a variety of small modifications that could be made to the syntax and semantics of policies, policy combinators and their specifications. The modifications were not included in this thesis because they caused an increase in the complexity of the notation without adding to the understanding of policy specifications and policy combinator specifications. However, they do increase the flexibility of the formalism in a way that would be useful since our specification languages are intended to be used as an intermediate language for an analysis tool. For example, the types of policies could be changed to allow for models of any expansion of the declared language not conflicting with the decision predicates, and policy combinators could be changed to allow parameter policies of any (non-conflicting) language, with the type of the resulting policy being dependent on the arguments. Another small addition that might be beneficial to an implementer is that there appears to be a theorem about separate compilation that could be proved, where compile could be applied to policies separately, which could then be renamed and combined.

A larger addition would be a lambda-calculus-like language that could serve as a high level language for "gluing together" policy specifications and policy combinator specifications. Such a language would provide a formal basis for language features already existing in the Margrave analysis tool, dealing with the naming and combining of policies and policy sets. Additionally, a template language for describing a more general form policy combinators, in which things like overrides declarations could be specified, so that rule conflict resolution declarations could be treated as part of a library and could be user-definable.

A rudimentary form of rule blaming is already available in the encoding of Margrave since rules are available as predicates distinct form the decisions that they induce. This could be taken further by either breaking down the bodies of the decision predicates into the bodies of non-decision intensional predicates which could then be referenced in queries or by augmenting the formalism itself with rule names and a mechanism for rule blaming. The second approach seems more appropriate, especially if Margrave's user language evolves towards the flexibility of the core language presented. Foundational work on rule blaming could provide a basis for a feature in the analysis tool that assisted users in determining the causes of properties of their policies. The ease of creating such a feature would be dependent on the logical semantics, core language and user facing language all being connected in a way that allows a straightforward propagation of the foundational rule blaming semantics upwards to the user language. The formalism we have presented is connected to the Margrave policy and policy set language in such a straightforward way.

It would also be useful to apply the work by Nelson [2010] on extending Herbrand's Theorem to our notion of policy specifications and policy combinator specifications, and especially to the results of encoding Margrave policies, to see for which policies and queries determining satisfiability is decidable. Since the classification already exists for Margrave policies, it would be beneficial if the encoding in Chapter 5 were to produce theories that preserve the decidability classification.

One of the main reasons for creating this formalism was to provide a basis for the expansion of the Margrave policy language. It should be more straightforward to work on adding support to policy specifications and to policy combinator specifications for reasoning about policies temporally and for logical extensions like arithmetic before attempting to raise the features to the level of Margrave, since the semantics are simpler at the level of this formalism.

Bibliography

J. L. Abad-Piero, H. Debar, T. Schweinberger, and P. Trommler. PLAS—policy langauge for authorizations. Technical report, IBM Research Division, May 1999.

M. Backes, B. Pfitzmann, and M. Schunter. A toolkit for managing enterprise privacy policies. In E. Snekkenes and D. Gollmann, editors, *Computer Security—ESORICS 2003*, volume 2808 of *Lecture Notes in Computer Science*, pages 162–180. Springer Berlin / Heidelberg, 2003. ISBN 978-3-540-20300-1. doi: 10.1007/978-3-540-39650-5_10.

M. Backes, M. Dürmuth, and R. Steinwandt. An algebra for composing enterprise privacy policies. In P. Samarati, P. Ryan, D. Gollmann, and R. Molva, editors, *Computer Security—ESORICS 2004*, volume 3193 of *Lecture Notes in Computer Science*, pages 33–52. Springer Berlin / Heidelberg, 2004. ISBN 978-3-540-22987-2. doi: 10.1007/978-3-540-30108-0_3.

Y. Bai and V. Varadharajan. A logic for state transformations in authorization policies. In *Proceedings of the 10th IEEE Workshop on Computer Security Foundations*, CSFW '97, pages 173–182, Washington, DC, USA, June 1997. IEEE Computer Society. ISBN 0-8186-7990-5. doi: 10.1109/CSFW.1997.596810.

P. Bonatti, S. de Capitani di Vimercati, and P. Samarati. A modular approach to composing access control policies. In *Proceedings of the 7th ACM Conference on Computer and Communications Security*, CCS '00, pages 164–173, New York, NY, USA, 2000. ACM. ISBN 1-58113-203-4. doi: 10.1145/352600.352623.

P. Bonatti, S. De Capitani di Vimercati, and P. Samarati. An algebra for composing access control policies. *ACM Trans. Inf. Syst. Secur.*, 5:1–35, February 2002. ISSN 1094-9224. doi: 10.1145/504909.504910.

G. Bruns and M. Huth. Access-control policies via Belnap logic: Effective and efficient composition and analysis. In *Proceedings of the 2008 21st IEEE Computer Security Foundations Symposium*, pages 163–176, Washington, DC, USA, June 2008. IEEE Computer Society. ISBN 978-0-7695-3182-3. doi: 10.1109/CSF.2008.10.

G. Bruns, D. S. Dantas, and M. Huth. A simple and expressive semantic framework for policy composition in access control. In *Proceedings of the 2007 ACM Workshop on Formal Methods in Security Engineering*, FMSE '07, pages 12–21, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-887-9. doi: 10.1145/1314436.1314439.

D. Dougherty, K. Fisler, and S. Krishnamurthi. Specifying and reasoning about dynamic access-control policies. In U. Furbach and N. Shankar, editors, *Automated Reasoning*, volume 4130 of *Lecture Notes in Computer Science*, pages 632–646. Springer Berlin / Heidelberg, 2006. ISBN 978-3-540-37187-8. doi: 10.1007/11814771_51.

K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz. Verification and change-impact analysis of access-control policies. In *Proceedings of the 27th International Conference on Software E*, ICSE '05, pages 196–205, New York, NY, USA, 2005. ACM. ISBN 1-58113-963-2. doi: 10.1145/1062455.1062502.

J. Goguen and R. Diaconescu. An Oxford survey of order sorted algebra. *Mathematical Structures in Computer Science*, 4(03):363–392, 1994. doi: 10.1017/S0960129500000517.

J. A. Goguen. Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Computer Science*, 105:217–273, 1992.

J. Y. Halpern and V. Weissman. Using first-order logic to reason about policies. *ACM Trans. Inf. Syst. Secur.*, 11:21:1–21:41, July 2008. ISSN 1094-9224. doi: 10.1145/1380564.1380569.

G. Hughes and T. Bultan. Automated verification of access control policies. Technical Report 2004-22, Department of Computer Science, University of California, Santa Barbara, September 2004.

G. Hughes and T. Bultan. Automated verification of XACML policies using a SAT solver. In *Workshop Proceedings of the 7th International Conference on Web Engineering*, pages 378–392. Workshop on Web Quality, Verification and Validation, July 2007.

G. Hughes and T. Bultan. Automated verification of access control policies using a SAT solver. *International Journal on Software Tools for Technology Transfer*, 10(6):473–534, December 2008.

IBM. Enterprise privacy authorization language (EPAL), 2011. URL http://www.zurich.ibm.com/pri/projects/epal.html.

S. Jajodia, P. Samarati, M. L. Sapino, and V. S. Subrahmanian. Flexible support for multiple access control policies. *ACM Trans. Database Syst.*, 26:214–260, June 2001. ISSN 0362-5915. doi: 10.1145/383891.383894.

M. Koch, L. V. Mancini, and F. Parisi-Presicce. On the specification and evolution of access control policies. In *Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies*, SACMAT '01, pages 121–130, New York, NY, USA, 2001. ACM. ISBN 1-58113-350-2. doi: 10.1145/373256.373280.

G. Kolaczek. Specification and verification of constraints in role based access control. In *Proceedings of the Twelfth International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 190–195, June 2003. doi: 10.1109/ENABL.2003.1231406.

N. Li and Q. Wang. Beyond separation of duty: An algebra for specifying high-level security policies. *J. ACM*, 55:12:1–12:46, August 2008. ISSN 0004-5411. doi: 10.1145/1379759.1379760.

N. Li, Q. Wang, W. Qardaji, E. Bertino, P. Rao, J. Lobo, and D. Lin. Access control policy combining: theory meets practice. In *Proceedings of the 14th ACM Symposium on Access Control Models and Technologies*, SACMAT '09, pages 135–144, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-537-6. doi: 10.1145/1542207.1542229.

P. Mazzoleni, E. Bertino, B. Crispo, and S. Sivasubramanian. XACML policy integration algorithms: not to be confused with XACML policy combination algorithms! In *Proceedings of the Eleventh ACM Symposium on Access Control Models and Technologies*, SACMAT '06, pages 219–227, New York, NY, USA, 2006. ACM. ISBN 1-59593-353-0. doi: 10.1145/1133058.1133089.

P. Mazzoleni, B. Crispo, S. Sivasubramanian, and E. Bertino. XACML policy integration algorithms. *ACM Trans. Inf. Syst. Secur.*, 11:4:1–4:29, February 2008. ISSN 1094-9224. doi: 10.1145/1330295.1330299.

J. McLean. The algebra of security. In *Proceedings of the 1988 IEEE Conference on Security and Privacy*, SP '88, pages 2–7, April 1988. ISBN 0-8186-0850-1. doi: 10.1109/SECPRI.1988.8092.

T. Nelson. Margrave: An improved analyzer for access-control and conguration policies. Master's thesis, Worcester Polytechnic Institute, April 2010.

T. Nelson, C. Barratt, D. J. Dougherty, K. Fisler, and S. Krishnamurthi. The Margrave tool for firewall analysis. In *Proceedings of the 24th International Conference on Large Installation System Administration*, LISA '10, pages 1–8, Berkeley, CA, USA, 2010. USENIX Association.

Q. Ni, E. Bertino, and J. Lobo. D-algebra for composing access control policy decisions. In *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, ASIACCS '09, pages 298–309, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-394-5. doi: 10.1145/1533057.1533097.

OASIS. OASIS eXtensible Access Control Markup Language (XACML) TC, 2011. URL www.oasis-open.org/committees/xacml.

A. Oberschelp. Order sorted predicate logic. In K. Bläsius, U. Hedtstück, and C.-R. Rollinger, editors, *Sorts and Types in Artificial Intelligence*, volume 418 of *Lecture Notes in Computer Science*, pages 7–17. Springer Berlin / Heidelberg, 1990. ISBN 978-3-540-52337-6. doi: 10.1007/3-540-52337-6_16.

P. Rao, D. Lin, E. Bertino, N. Li, and J. Lobo. An algebra for fine-grained integration of XACML policies. In *Proceedings of the 14th ACM Symposium on Access Control Models and Technologies*, SACMAT '09, pages 63–72, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-537-6. doi: 10.1145/1542207.1542218.

P. Rao, D. Lin, E. Bertino, N. Li, and J. Lobo. Fine-grained integration of access control policies. *Computers & Security*, 30(2–3):91–107, 2011. ISSN 0167-4048. doi: 10.1016/j.cose.2010.10.006.

P. Samarati and S. de Vimercati. Access control: Policies, models, and mechanisms. In R. Focardi and R. Gorrieri, editors, *Foundations of Security Analysis and Design*, volume 2171 of *Lecture Notes in Computer Science*, pages 137–196. Springer Berlin / Heidelberg, 2001. ISBN 978-3-540-42896-1. doi: 10.1007/3-540-45608-2_3.

E. Torlak and D. Jackson. Kodkod: A relational model finder. In O. Grumberg and M. Huth, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4424 of *Lecture Notes in Computer Science*, pages 632–647. Springer Berlin / Heidelberg, 2007. ISBN 978-3-540-71208-4. doi: 10.1007/978-3-540-71209-1_49.

D. Wijesekera and S. Jajodia. Policy algebras for access control: the propositional case. In *Proceedings of the 8th ACM Conference on Computer and Communications Security*, CCS '01, pages 38–47, New York, NY, USA, 2001. ACM. ISBN 1-58113-385-5. doi: 10.1145/501983.501990.

D. Wijesekera and S. Jajodia. Policy algebras for access control the predicate case. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, pages 171–180, New York, NY, USA, 2002. ACM. ISBN 1-58113-612-9. doi: 10.1145/586110.586134.

D. Wijesekera and S. Jajodia. A propositional policy algebra for access control. *ACM Trans. Inf. Syst. Secur.*, 6:286–325, May 2003. ISSN 1094-9224. doi: 10.1145/762476.762481.

T. Y. C. Woo and S. S. Lam. Authorization in distributed systems: a formal approach. In *Proceedings of the 1992 IEEE Symposium on Security and Privacy*, pages 33–50, May 1992. ISBN 0-8186-2825-1. doi: 10.1109/RISP.1992.213272.

T. Y. C. Woo and S. S. Lam. Authorizations in distributed systems: A new approach. *Journal of Computer Security*, 2(2–3):107–136, 1993.

N. Zhang, M. Ryan, and D. Guelev. Evaluating access control policies through model checking. In J. Zhou, J. Lopez, R. Deng, and F. Bao, editors, *Information Security*, volume 3650 of *Lecture Notes in Computer Science*, pages 446–460. Springer Berlin / Heidelberg, 2005. ISBN 978-3-540-29001-8. doi: 10.1007/11556992_32.