Analyzing First-order Role Based Access Control

Carlos Cotrini*, Thilo Weghorn*‡, David Basin*, and Manuel Clavel†
*Department of Computer Science
ETH Zurich, Switzerland
{basin, ccarlos, thilo.weghorn}@inf.ethz.ch
†IMDEA Software Institute, Spain
manuel.clavel@imdea.org

***Abstract*—We propose FORBAC, an extension of Role-Based Access Control (RBAC) based on first-order logic. FORBAC is expressive enough to formalize a wide range of access control policies. However, it is simple enough so that relevant policy analysis queries can be analyzed in NP, which we argue is a natural complexity class for this problem. To analyze queries efficiently, we reduce them to the problem of satisfiability modulo appropriate theories, and use off-the-shelf SMT solvers. We evaluate FORBAC's expressiveness and our approach to policy analysis in a case study, analyzing access control in a European bank.**

I. Introduction

RBAC [15] is a predominant access control model for centralized access-control. However, it is not the last word, and researchers have investigated numerous extensions that allow RBAC to scale better and be easier to administrate, e.g. [18], [20], [24], [25], [28], [29]. However, the expressive power of these extensions makes it difficult to understand the behavior of policies, which in turn has motivated a plethora of research on policy analysis for RBAC, e.g. [4], [8], [16], [34].

Many RBAC extensions use first-order logic in their syntax, but first-order logic is simply too expressive for policy specification languages. This is reflected in the syntax of different logic-based languages [9], [19] that have been used in practice; for instance, they exclude disjunction and limit quantifier alternation. Moreover, these languages have been defined with a focus on policy formulation rather than policy analysis. As a result, policy analysis can handle only fragments of these languages. For example, [22] defines a language for administrating user attributes, where first-order logic is used to define administrative rules that specify how users' attribute values change. Later, in [23], the authors study the complexity of the reachability problem, a common analysis problem in administrative RBAC [3], [16], [21]. It turns out that this problem is PSPACE-complete, even after restricting quantifier alternation, and allowing only unary functions and binary predicates. Further restrictions must be imposed on the language to obtain fragments where this reachability problem is solvable in polynomial time.

The use of first-order logic in RBAC extensions gives rise to new problems. In some extensions, the assignments of roles to users and permissions to roles are specified by first-order formulas [11], [18], [20], [24]. This specification

‡ The first two authors contributed equally to this work.

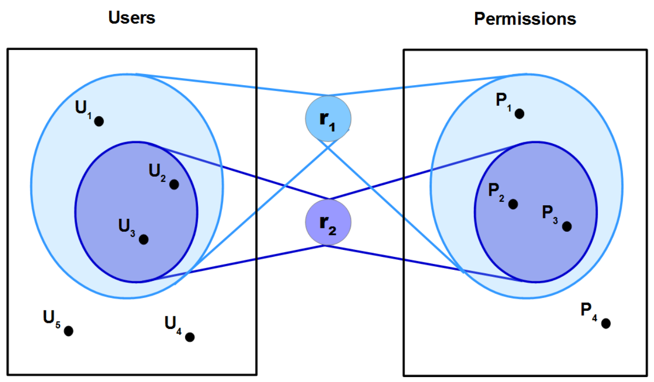


Fig. 1. The role r_2 is redundant: the permissions assigned to r_2 are contained in those assigned to r_1 and the users assigned to r_2 are also assigned to r_1.

is done by humans and is hence prone to errors. Policy administrators may fail to anticipate all the consequences of their specifications. For example, they may specify policies with redundant roles, as illustrated in Figure 1, or even worse, assign users incorrect authorizations.

The imbalance between expressiveness and efficient analysis gives rise to a new research direction: to develop frameworks strong enough to express realistic authorization policies, but simple enough to be analyzed in practice. These frameworks should provide languages for specifying policies and properties, and procedures to verify properties against policies. Other researchers have presented such frameworks [5], [31]. However there are features and problems specific to extensions of RBAC, like the one illustrated in Figure 1 [4], that were not addressed by this work. To the best of our knowledge, no prior work has attempted to establish a framework that balances expressiveness and efficient policy analysis for RBAC extensions based on first-order logic.

We propose FORBAC, an extension of RBAC that incorporates the main features of different RBAC extensions from the literature, e.g. [3], [18], [20], [24]. FORBAC strikes a balance among the variety of policies it can express, the properties that can be verified, and its complexity, which is NP. Although a polynomial complexity would be desirable, we argue that NP-hardness cannot be avoided in policy analysis. To verify properties of FORBAC policies, we reduce them to satisfiability modulo theories and use the SMT-solver Z3 [13].

To evaluate our theses that (1) FORBAC is expressive enough for substantial real-world applications and (2) realistic policies can be analyzed with reasonable overhead, we conduct a case study on the access-control infrastructure of a major European bank. The bank's PDP manages around 350 applications, each with a separate security policy. In total, it manages access for close to 50,000 users and 57,000 actions. We give an overview of the bank's rules that govern both the assignments of roles to users and the assignment of permissions to roles. We express them as FORBAC policies and conduct experiments on a variety of relevant policy analysis queries. Using SMT solvers, most of the queries are answered in seconds. For a few of the queries, the evaluation takes several minutes and we identify reasons for this and suggest improvements.

The remainder of this paper is organized as follows. In Section II we describe the features of different RBAC extensions from the literature and establish requirements for FORBAC. In Section III we define FORBAC's syntax and semantics and in Section IV we show how to specify policy analysis queries for FORBAC policies. In Section V we present experimental results. In Section VI we discuss related work and in Section VII we draw conclusions.

II. Requirements for FORBAC

FORBAC is an RBAC extension that strikes a balance among the following three factors:

- An expressive language for specifying RBAC policies.
- An expressive language for specifying properties of RBAC policies.
- A low complexity for verifying policies against properties.

In the remainder of this section, we discuss language requirements and complexity classes for policy verification.

A. Requirements for policy specification

Numerous extensions for RBAC have been proposed and the syntax of many of them (e.g. [20], [24], [29]) includes fragments of first-order logic that make policy analysis undecidable, or at best highly intractable. In the following, we review some of their features in order to elicit the central requirements for an expressive RBAC extension. Based on these requirements, we present in Section III a fragment of first-order logic that is simple, but expressive enough to formalize realistic policies.

a) Attributes: A common feature of RBAC extensions is the association of attributes to users, roles, and permissions. This stems from the need to add fine-grained access control to RBAC. For example, a user in a physician role should be authorized to access patient information, but only for those patients he supervises. Instead of defining one role for every subset of patients, an attribute is added to the role that specifies the set of patients under the physician's supervision.

Roles with attributes have been proposed in the literature in the form of *parameterized permissions* and *role templates* [1], [12], [18]. A parameterized permission represents a set of permissions that have attributes in common. For example, assigning a grade to a student could be represented as a parameterized permission $AssignGrade(s)$, where s is a variable that represents a student. The use of this parameterized permission spares administrators the burden of defining one permission for every student. Role templates are sets of parameterized permissions. For example, for a lecturer, we could define a role template $Lecturer$ that contains the parameterized permission $AssignGrade(s)$. When a user is assigned the role template $Lecturer$, he is also assigned a set of students S. The pair $(Lecturer, S)$ is called a *role instance*. Here, the user can assign a grade to every student in the set S. A role template eliminates the burden of creating a role for every lecturer. We incorporate parameterized permissions and role templates in our language and use first-order logic to define them.

b) Role and permission assignments specified in first-order logic: Another common feature of RBAC extensions is the use of rules to assign roles to users and permissions to roles. This feature is motivated by the difficulty of manually administering these relations in large environments where users' and permissions' attribute values frequently change. Many RBAC extensions, such as [18], [20], [22], use first-order logic to specify user-role and role-permission assignment relations. However, they do not limit the fragment of first-order logic used for these specifications.

We propose restrictions on the first-order fragment we use in FORBAC. For instance, we do not allow the arbitrary nesting of quantifiers. In practice, access control permissions simply require the presence or absence of values in the user's, role's, and permission's attributes. This is reflected in the syntax of logic-based policy specification languages that have been used in practice. For example, Lithium [19] forbids quantifier alternation and yet it can still express various parts of U.S. legislation, including fragments of the Privacy Rule, which governs access to electronic medical files, and Title 42, Chapter 7 of the U.S. Code, which determines who is eligible for Social Security. Another example is Cassandra [10], an earlier version of SecPAL, which does not allow quantifier alternation, but can express the policies for the national electronic health record system of the United Kingdom.

c) Numeric constraints: We incorporate this kind of constraints as they often occur in authorization policies. For example, Title 29 of the U.S. Code $\S$1181, which belongs to the HIPAA rule, says:

> A period of creditable coverage shall not be counted, with respect to enrollment of an individual under a group health plan, if, after such period and before the enrollment date, there was a 63-day period during all of which the individual was not covered under any creditable coverage.

In electronic health record systems, health organizations are authorized to request a credential asserting patient/EHR-service bindings if they can provide an RA-approved NHS health organization credential [10]. Such credentials are valid only for fixed time intervals. More generally, functions within an organization may have a limited duration. For instance, vendors may be authorized to access vendor contracts only in the second week of every quarter of every year, and vendor contracts must be submitted within two weeks of that time [11]. Such numerical constraints can usually be expressed as inequalities between two integer values.

This concludes the requirements for our language for specifying RBAC policies. Note that there are other access control features that have received attention in the literature that we have not included as requirements for our language. These include role hierarchies [33], delegation [7], and separation-of-duty constraints [2]. We leave these as future work and focus on the core features explained above.

B. A complexity class for policy analysis

Ideally, policy analysis should be efficiently computable. However, we argue that it is NP-hard for any sufficiently expressive policy specification language. To support this, we present a simple policy specification language F that can be embedded into languages like Margrave [30] and the one presented in [5] and show that checking even the simple query of whether every access request is permitted in a given policy in F is NP-hard.

Consider a first-order vocabulary consisting only of unary relation symbols. Let F be the set of first-order formulas of the form $P_1(x) \wedge \ldots \wedge P_k(x) \wedge \neg Q_1(x) \wedge \ldots \wedge \neg Q_n(x)$, where $k, n \geq 0$, x is a variable and P_i and Q_j, for $i \leq k$ and $j \leq n$, are unary relation symbols. A policy in F consists of a set of formulas in F.

The semantics of this language is as follows. Let T be a policy in F and S a first-order structure. For an element a in the domain of S, we say that a is *permitted in* T if a satisfies at least one formula in T. Basically, S represents a set of access requests and every relation symbol Q represents an attribute. For an access request $a \in S$, a has the attribute Q, if and only if a is in Q^S, the interpretation of Q under S. A policy defines whether an access request is permitted depending on the request's attributes.

For a policy $T = \{\varphi_1(x), \ldots, \varphi_\ell(x)\}$ in F, suppose that we want to verify if every access request is permitted. This can be done by checking validity of $\forall x\,.\,(\varphi_1(x) \vee \ldots \vee \varphi_\ell(x))\,.$ However, we prove in Section A in the Appendix that checking this for an arbitrary T is NP-hard.

F is extremely simple. It uses only unary relation symbols and it can be embedded into state-of-the-art analysis frameworks used for analyzing realistic XACML policies like Continue [27] (see Section A in the Appendix for details). Nevertheless, despite its low expressiveness, even basic policy analysis queries are NP-hard. For this reason, we believe P is too restrictive (unless P = NP) and we therefore set our sights on performing policy analysis in NP.

III. Syntax and semantics of FORBAC

Given the requirements for our framework, we start by defining the vocabulary for writing policies.

Definition 1: A *FORBAC signature* is a triple $\Sigma = \langle \mathcal{S}, \mathcal{A}_1, \mathcal{A}_2 \rangle$ where $\mathcal{S}$ is a set of sorts $\mathcal{S} = \mathcal{S}_{RBAC} \cup \{\textbf{Integer}, \textbf{String}\}$ with,

$$\mathcal{S}_{RBAC} = \{\mathit{Users}, \mathit{Roles}_1, \mathit{Roles}_2, \ldots, \mathit{Roles}_T, \mathit{Perms}\},$$

for $T \in \mathbb{N}$. $\mathcal{A}_1$ and $\mathcal{A}_2$ are sets of unary function symbols. Every $f \in \mathcal{A}_1 \cup \mathcal{A}_2$ has a type $W_f \to V_f$ with $W_f \in \mathcal{S}_{RBAC}$. For $f \in \mathcal{A}_1$, $V_f \in \{\textbf{Integer}, \textbf{String}\}$. For $f \in \mathcal{A}_2$, $V_f \in \{2^{\textbf{Integer}}, 2^{\textbf{String}}\}$, where $2^{\textbf{Integer}}$ and $2^{\textbf{String}}$ denote the sets of finite sets of integers and strings, respectively. ■

The symbols in $\mathcal{A}_1$ denote *single-valued attributes* and those in $\mathcal{A}_2$ denote *set-valued attributes*. We use the term *attribute* to refer to any single or set-valued attribute. We use $RT(\Sigma)$ as shorthand for the set $\{\mathit{Roles}_1, \mathit{Roles}_2, \ldots, \mathit{Roles}_T\}$ of *role templates* of Σ.

Example 2: We present a simple FORBAC-signature $\Sigma_B = \langle \mathcal{S}, \mathcal{A}_1, \mathcal{A}_2 \rangle$ for specifying the access control policies of a bank's account administration tool. The sorts of $\mathcal{S}$ are Users, $R_{\texttt{Student}}$, $R_{\texttt{Employee}}$, *Perms*, **Integer**, and **String**. Here, $RT(\Sigma_B) = \{R_{\texttt{Student}}, R_{\texttt{Employee}}\}$. $R_{\texttt{Student}}$ and $R_{\texttt{Employee}}$ represent two kinds of customer accounts: "student accounts" and "employee accounts".

We define the following single-valued attributes for the sort Users:

- $\mathit{name} : \mathit{Users} \to \textbf{String}$.
- $\mathit{age} : \mathit{Users} \to \textbf{Integer}$.
- $\mathit{nationality} : \mathit{Users} \to \textbf{String}$.
- $\mathit{salary} : \mathit{Users} \to \textbf{Integer}$.

The role template $R_{\texttt{Employee}}$ has one single-valued attribute:

- $\mathit{limit} : R_{\texttt{Employee}} \to \textbf{Integer}$.

This attribute specifies the maximal amount that an employee may use in one transaction on her bank account.

The role template $R_{\texttt{Student}}$ has one set-valued attribute:

- $\mathit{country} : R_{\text{Student}} \to 2^{\textbf{String}}$.

This attribute specifies in which countries a student is authorized to carry out transactions.

The sort *Perms* has three single-valued attributes:

- $\mathit{action} : \mathit{Perms} \to \textbf{String}$.
- $\mathit{amount} : \mathit{Perms} \to \textbf{Integer}$.
- $\mathit{location} : \mathit{Perms} \to \textbf{String}$.

The attribute action denotes the kind of transaction (e.g., withdrawing or transferring money from a bank account), amount denotes how much money is involved, and $\mathit{location}$ denotes the country where the transaction occurs. ■

Definition 3: Let $\Sigma = \langle \mathcal{S}, \mathcal{A}_1, \mathcal{A}_2 \rangle$ be a FORBAC signature. A Σ*-structure* is an entity S consisting of the following:

- A finite non-empty set W^S, for each sort $W \in \mathcal{S}$, where $\textbf{Integer}^S$ and $\textbf{String}^S$ are the sets of integers and strings, respectively and $2^{\textbf{Integer}^S}$ and $2^{\textbf{String}^S}$ are the sets of finite sets of integers and strings, respectively.
- A function $f^S : W_f^S \to V_f^S$, for every $f \in \mathcal{A}_1 \cup \mathcal{A}_2$ with type $W_f \to V_f$.

■

We call an element of Users^S a *user* in S. For a role template $R \in RT(\Sigma)$, we call an element of R^S a *role instance of* R. We call an element of Perms^S a *permission* of S.

Example 4: Let Σ_B be the FORBAC-signature from Example 2. Figure 2 shows a Σ_B-structure S with three users, two role instances of $R_{\texttt{Student}}$, one role instance of $R_{\texttt{Employee}}$, and three permissions. ■

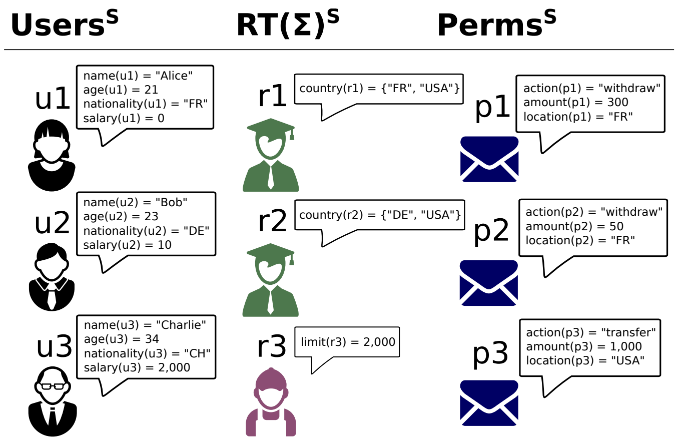


Fig. 2. An example of a Σ_B-structure

Definition 5: An *atomic FORBAC formula* is any expression of the following form:

- $t_1 \sim t_2$, where t_1 and t_2 are *single-valued terms*. These are constants of type **Integer** or **String**, or expressions of the form $f(x)$, where f is a single-valued attribute and x is a variable. The symbol $\sim$ can be $=$, $\leq$, or $<$.
- $T_1 \propto T_2$, where T_1 and T_2 are *set-valued terms*. These are constant symbols denoting finite sets of strings, constant symbols denoting finite sets of finite intervals of integers, or expressions of the form $F(x)$, where F is a set-valued attribute and x is a variable. The symbol $\propto$ is either $=$ or $\subseteq$.
- $t \in T$, where t and T are a single-valued and a set-valued term, respectively.

The following BNF grammar summarizes the syntax of atomic FORBAC-formulas:

$$\begin{array}{rl} \psi & ::= t \sim t \mid T \propto T \mid t \in T \\ t & ::= c \mid f(x) \\ T & ::= C \mid F(x) \\ \sim & ::= \leq \mid = \mid < \\ \propto & ::= \subseteq \mid = \end{array}$$

Here, c ranges over integer and string constants, f ranges over single-valued attributes, F ranges over set-valued attributes, C is any finite set of strings or any finite set of integer intervals, and x is a variable of an appropriate type. Finally, a *FORBAC-formula* is a Boolean combination of atomic FORBAC formulas. ■

Definition 6: The *size* of a FORBAC-formula ϕ is the number of occurrences of ϕ's atomic FORBAC-formulas and is recursively defined as follows:

$$|\phi| = \begin{cases} 1 & \text{if } \phi \text{ is atomic} \\ |\psi| & \text{if } \phi \equiv \neg\psi \\ |\psi_1| + |\psi_2| & \text{if } \phi \equiv \psi_1 \bowtie \psi_2, \end{cases}$$

where $\bowtie \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$. ■

Remark 7: Every FORBAC-formula can be translated into a formula in many-sorted first-order logic as follows. For every set-valued attribute F, define a binary relation symbol R_F. Then rewrite every atomic FORBAC subformula containing a set-valued term. We illustrate this with three cases, where F and F' range over set-valued attributes and t and t' range over single-valued terms. The remaining cases are analogous.

$$\begin{array}{rcl} t' \in F(t) & \rightsquigarrow & R_F(t, t'). \\ F(t) \subseteq F'(t') & \rightsquigarrow & \forall y\,.\,(R_F(t,y) \rightarrow R_{F'}(t',y))\,. \\ F(t) = F'(t') & \rightsquigarrow & \forall y\,.\,(R_F(t,y) \leftrightarrow R_{F'}(t',y))\,. \end{array}$$

■

Definition 8: A *FORBAC-policy* is a triple $(\Sigma, \mathcal{UA}, \mathcal{PA})$, where Σ is a FORBAC-signature. The *user-assignment specification*

$$\mathcal{UA} = \{\mathcal{UA}_R(u, r) : R \in RT(\Sigma)\}$$

and the *permission-assignment specification*

$$\mathcal{PA} = \{\mathcal{PA}_R(r, p) : R \in RT(\Sigma)\}$$

are sets of FORBAC-formulas over Σ. The *user-assignment formulas* $\mathcal{UA}_R(u, r)$ have (just) the two free variables u and r of sorts Users and R, respectively, and the *permission-assignment formulas* $\mathcal{PA}_R(r, p)$ have (just) the two free variables r and p of sorts R and Perms, respectively. ■

Example 9: Consider the FORBAC-signature Σ_B from Example 2 and suppose that we have the following policy. Users no older than 25 are assigned an instance of $R_{\texttt{Student}}$, which entitles them to withdraw up to \$1,000 in the user's home country or in the USA. Users whose salary exceeds \$1,500 are assigned an instance of $R_{\texttt{Employee}}$, which entitles them to withdraw and transfer money in any country provided the sum does not exceed the user's salary. We present a FORBAC-policy $(\Sigma_B, \mathcal{UA}, \mathcal{PA})$ that models this. Since Σ_B was already specified in Example 2, we just present the formulas in $\mathcal{UA}$ and $\mathcal{PA}$:

$$\mathcal{UA}_{R_{\texttt{Student}}}(u, r) \equiv \left(\begin{array}{l} \mathit{age}(u) \leq 25 \wedge \\ \mathit{country}(r) = \{\mathit{nationality}(u), \text{“USA”}\} \end{array} \right)$$

$$\mathcal{PA}_{R_{\texttt{Student}}}(r, p) \equiv \left(\begin{array}{l} \mathit{action}(p) \in \{\text{“withdraw”}\} \wedge \\ \mathit{amount}(p) \leq 1{,}000 \wedge \\ \mathit{location}(p) \in \mathit{country}(r) \end{array} \right)$$

$$\mathcal{UA}_{R_{\texttt{Employee}}}(u,r) \equiv \left(\begin{array}{l} \mathit{salary}(u) > 1{,}500 \wedge \\ \mathit{limit}(r) = \mathit{salary}(u) \end{array} \right)$$

$$\mathcal{PA}_{R_{\texttt{Employee}}}(r,p) \equiv \left(\begin{array}{l} \mathit{action}(p) \in \{\text{“withdraw”}, \text{“transfer”}\} \wedge \\ \mathit{amount}(p) \leq \mathit{limit}(r) \end{array} \right) .$$

■

Let Σ be a FORBAC-signature and S be a Σ-structure. Let $\overline{u}$, $\overline{r}$, and $\overline{p}$ be a user, a role instance of R, and a permission of S, respectively. We say that $\overline{u}$ *is assigned* $\overline{r}$ if $\overline{u}$ and $\overline{r}$ satisfy $\mathcal{UA}_R(u,r)$ in S. We say that $\overline{r}$ *is assigned* $\overline{p}$ if $\overline{r}$ and $\overline{p}$ satisfy $\mathcal{PA}_R(r,p)$ in S.

Example 10: Figure 3 illustrates, in the context of the Σ_B-structure of Example 4, which role instances are assigned to which users and which permissions are assigned to which role instances. ■

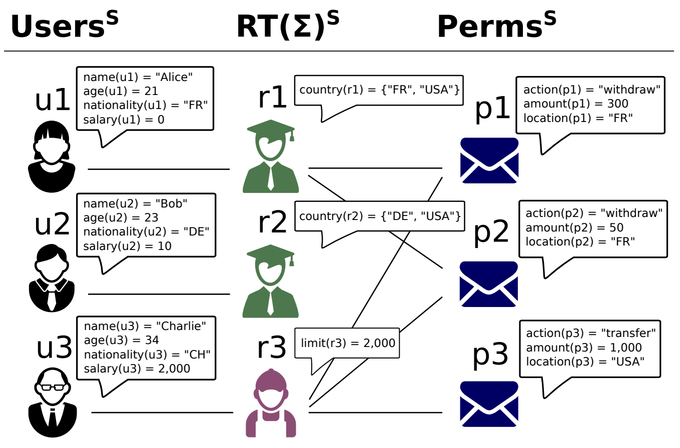


Fig. 3. User and permission-assignments in the Σ_B-structure S

Definition 11: For a FORBAC-policy $(\Sigma, \mathcal{UA}, \mathcal{PA})$ and a role template $R \in RT(\Sigma)$, let $\mathit{Auth}_R(u,p)$ denote the formula

$$\exists r : R \,.\, \mathcal{UA}_R(u,r) \;\wedge\; \mathcal{PA}_R(r,p).$$

Let $\mathit{Auth}(u,p)$ denote the formula $\bigvee_{R \in RT(\Sigma)} \mathit{Auth}_R(u,p)$. For a Σ-structure S, we say that *a user* $\overline{u}$ *of* S *is authorized for a permission* $\overline{p}$ *of* S if $\overline{u}$ and $\overline{p}$ satisfy $\mathit{Auth}(u,p)$ in S. ■

When quantifying over variables, we do not specify the sorts *Users* and *Perms*, as these should be clear from the context. For example, instead of writing

$$\forall u : \mathit{Users}\, \exists r_1 : R_1, r_2 : R_2 \,.\, \mathcal{UA}_{R_1}(u,r_1) \vee \mathcal{UA}_{R_2}(u,r_2),$$

we write

$$\forall u\, \exists r_1 : R_1, r_2 : R_2 \,.\, \mathcal{UA}_{R_1}(u,r_1) \;\vee\; \mathcal{UA}_{R_2}(u,r_2).$$

Example 12: Consider the FORBAC-signature presented in Example 2, the Σ_B-structure S presented in Example 4, and the FORBAC-policy presented in Example 9. User u_1 is authorized for permissions p_1 and p_2 and user u_3 is authorized for permissions p_1, p_2, and p_3. ■

Role templates are not essential and one could use instead just one sort per role and functions to distinguish different role templates. However, always using just one role template could create an overhead when specifying FORBAC-policies. In Example 9, if we had used just one sort R' for roles, we would have to define the functions $\mathit{country}' : R' \to 2^{\textbf{String}}$, $\mathit{limit}' : R' \to \textbf{Integer}$, and a special function $\mathit{type}' : R' \to \textbf{String}$ to distinguish between “students” and “employees”. Also, when specifying $\mathcal{UA}_{R'}(u,r)$, we would have to specify limit' when we assign a role instance to a student. Similarly, we would have to specify $\mathit{country}'$ when we assign a role instance to an employee.

We conclude our presentation of FORBAC by observing that authorization can be decided in polynomial time. The proof is given in Section B in the appendix.

Theorem 13: Given a FORBAC policy $(\Sigma, \mathcal{UA}, \mathcal{PA})$, a Σ-structure S, a user $\overline{u} \in \mathit{Users}^S$, and a permission $\overline{p} \in \mathit{Perms}^S$, deciding whether $\overline{u}$ is authorized for $\overline{p}$ takes at most polynomial time.

IV. Policy analysis in FORBAC

We now define the language for posing analysis queries for FORBAC-policies. Since FORBAC-formulas can be expressed in first-order logic, this language is also a natural choice for reasoning about FORBAC-policies. However, first-order logic is undecidable in general and its restriction to fragments must be done with care. Halpern and Weissman [19] studied several fragments of first-order logic for specifying access control policies. They showed that even after limiting the number of quantifier alternations and removing function symbols, one can end up with a fragment where merely deciding authorization is intractable.

To strike a balance between expressiveness in property specification and efficiency in policy analysis, we propose the set of existential FORBAC-formulas as the language for specifying analysis queries.

Definition 14: An *existential FORBAC-formula* is a first-order formula of the form $\exists x_1, x_2 \ldots, x_n \,.\, \varphi(x_1, x_2, \ldots, x_n)$, where $\varphi(x_1, x_2, \ldots, x_n)$ is a Boolean combination of FORBAC-formulas over a FORBAC-signature. ■

To verify if a property holds for a FORBAC-policy, we build an existential FORBAC-formula that describes a countermodel that violates the property. The Boolean combination of FORBAC-formulas describes the negation of the property and the existential quantifiers specify the elements that should appear in a countermodel. The formula can then be input into an SMT solver, which attempts to find such a countermodel. The syntax of existential FORBAC-formulas limits quantifier alternation and the behavior of relations and functions so that deciding satisfiability is NP-complete. We prove this in Section B in the appendix:

Theorem 15: Deciding the satisfiability of an existential FORBAC-formula is NP-complete.

The low complexity, NP, is not for free. There are relevant policy analysis queries like observational equivalence and conflict [5] that cannot be expressed as existential FORBAC-formulas. However, they can be expressed in first-order logic and can be passed as input to an SMT-solver.

We present now four kinds of policy analysis queries and explain how to reduce them to satisfiability of existential FORBAC-formulas.

A *Authorization inspection* can be used to verify that a FORBAC-policy does not grant undesired access.
B *Assignment simplification* can be used to identify redundancies in FORBAC formulas.
C *Role subsumption* can be used to identify redundant role templates.
D *Redundant assignments* can be used to identify redundancies in the user-assignment relation.

These queries illustrate the expressive power of existential FORBAC formulas as a language for policy analysis for FORBAC. Moreover, they are all natural queries, that arise and require answers, when administrating policies specified in rich policy languages, e.g., where role templates and first-order user and permission-assignments interact.

A. Authorization inspection

Suppose we are given a FORBAC-policy $(\Sigma, \mathcal{UA}, \mathcal{PA})$, a FORBAC formula $\psi_{user}(u)$ with a free variable u of sort *Users*, and FORBAC formulas $\psi_1(p_1)$, $\psi_2(p_2), \ldots, \psi_k(p_k)$, with free variable $p_1, p_2, \ldots, p_k$ of sort *Perms*. Authorization inspection can be cast as the question of whether a formula of the following form is satisfiable:

$$\exists u, p_1, \ldots, p_k \,.\, \psi_{user}(u) \;\wedge\; \bigwedge_{i \leq k} \left(\psi_i(p_i) \;\wedge\; Auth(u, p_i)\right). \quad (1)$$

Checking this formula's satisfiability amounts to searching for a Σ-structure S with a user u who matches the criteria of ψ_{user} and who is authorized for some permissions $p_1, p_2, \ldots$, p_k that match the criteria of $\psi_1, \psi_2, \ldots, \psi_k$, respectively.

Example 16: Consider again the FORBAC-policy from Example 9. According to $\mathcal{UA}_{R_{\texttt{Student}}}(u, r)$, users can be assigned instances of $R_{\texttt{Student}}$ if they are at most 25 years old. Also, according to $\mathcal{PA}_{R_{\texttt{Student}}}(r, p)$, instances of $R_{\texttt{Student}}$ can never be granted permission to withdraw amounts larger than \$1,000. One may conjecture that users who are at most 25 years old can never withdraw large amounts of money; they cannot, at least, for the Σ_B-structure in Figure 3. To determine whether this property holds for any Σ_B-structure, we instantiate Formula (1) as follows.

$$\psi_{user}(u) \quad \equiv \quad age(u) \leq 25,$$

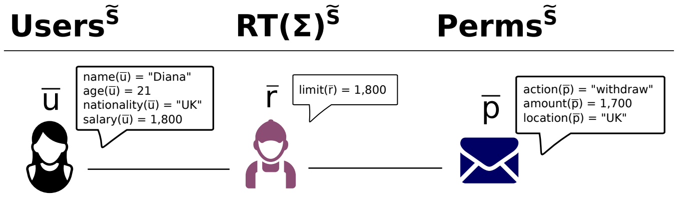
$$\psi_1(p_1) \quad \equiv \quad \left(\begin{array}{l} action(p_1) = \text{“withdraw”} \wedge \\ amount(p_1) > 1{,}000 \end{array}\right).$$

The resulting instance of Formula (1) is

$$\exists u, p_1 \,.\, \left(\begin{array}{l} age(u) \leq 25 \wedge \\ action(p_1) = \text{“withdraw”} \wedge \\ amount(p_1) > 1{,}000 \wedge \\ Auth(u, p_1) \end{array}\right). \quad (2)$$

If this formula is unsatisfiable, then we have confirmed our conjecture. However, if we input this formula to the SMT-solver Z3, then Z3 outputs that it is satisfiable and provides a model satisfying the formula. This model can be used to build a Σ_B-structure that refutes our conjecture.

The following is a Σ_B-structure $\tilde{S}$ that satisfies Formula (2). Let Σ_B be the FORBAC-signature from Example 2. Figure 4 shows a Σ_B-structure $\tilde{S}$ with one user $\overline{u}$, one role instance $\overline{r}$ of type $R_{\texttt{Employee}}$, none of type $R_{\texttt{Student}}$, and one permission $\overline{p}$. It is easy to see that $\overline{u}$ and $\overline{r}$ satisfy $\mathcal{UA}_{R_{\texttt{Employee}}}(u, r)$ and that



$\mathbf{Users}^{\tilde{S}}$
$\mathbf{RT(\Sigma)}^{\tilde{S}}$
$\mathbf{Perms}^{\tilde{S}}$
$\overline{u}$
name($\overline{u}$) = "Diana"
age($\overline{u}$) = 21
nationality($\overline{u}$) = "UK"
salary($\overline{u}$) = 1,800
$\overline{r}$
limit($\overline{r}$) = 1,800
$\overline{p}$
action($\overline{p}$) = "withdraw"
amount($\overline{p}$) = 1,700
location($\overline{p}$) = "UK"

Fig. 4. User and permission-assignments in the Σ_B-structure $\tilde{S}$

$\overline{r}$ and $\overline{p}$ satisfy $\mathcal{PA}_{R_{\texttt{Employee}}}(r, p)$. Therefore $\overline{u}$ is authorized for $\overline{p}$. This means that it is possible for users who are at most 25 years old to withdraw amounts greater than \$1,000. What they should do is to have a salary greater than \$1,500, so they obtain an instance $\overline{r}$ of $R_{\texttt{Employee}}$ with a limit higher than \$1,500. This would allow them to withdraw more than \$1,000. ■

Note that, as given, Formula (1) is not an existential FORBAC-formula because $Auth(u, p_i)$, for $i \leq k$, contains existential quantifiers. However, it can be rewritten into an existential FORBAC-formula by moving the existential quantifiers in $Auth(u, p_i)$, for $i \leq k$, to the front of the formula, using standard first-order equivalences.

B. Assignment simplification

Poor design or changes in policy specifications may lead to redundancies, which humans have difficulty detecting. We explain how we can identify redundancies using existential FORBAC formulas.

Example 17: Consider the following FORBAC formula that specifies $\mathcal{UA}$ for some policy:

$$\mathcal{UA}_R(u, r) \equiv \psi_1(u, r) \;\vee\; \psi_2(u, r),$$

where

$$\begin{array}{ll} \psi_1(u, r) & = unit(r) = 45 \;\wedge\; level(u) = 23 \;\text{ and} \\ \psi_2(u, r) & = unit(r) = 45 \;\wedge\; level(u) > 20. \end{array}$$

$\mathcal{UA}_R(u, r)$ consists of a disjunction of two formulas, where the satisfaction of the first formula implies the satisfaction of the second one. This means that $\psi_1(u, r)$ is redundant. To confirm this, we can show that the following formula is valid: $\forall u \forall r : R \,.\, (\psi_1(u, r) \;\vee\; \psi_2(u, r)) \leftrightarrow \psi_2(u, r)$. This is equivalent to showing that the following existential FORBAC formula $\exists u, r : R \,.\, \neg\left((\psi_1(u, r) \;\vee\; \psi_2(u, r)) \leftrightarrow \psi_2(u, r)\right)$ is not satisfiable. ■

The same technique can be used to detect redundancies in $\mathcal{PA}_R(r, p)$. In general, whenever one conjectures that a

FORBAC formula $\psi(x_1, x_2, \ldots, x_k)$ is equivalent to another formula $\psi'(x_1, x_2, \ldots, x_k)$, one can check this by determining whether the following formula is valid:

$$\forall x_1, x_2, \ldots, x_k \,.\, \psi(x_1, x_2, \ldots, x_k) \leftrightarrow \psi'(x_1, x_2, \ldots, x_k)\,.$$

This is equivalent to determining whether the following existential FORBAC formula is unsatisfiable:

$$\exists x_1, x_2, \ldots, x_k \,.\, \neg\psi(x_1, x_2, \ldots, x_k) \leftrightarrow \psi'(x_1, x_2, \ldots, x_k)\,.$$

C. Role subsumption

RBAC systems used in large enterprises with multiple administrators may end up with equivalent redundant roles, especially, when the administrators are unaware of roles previously created by other administrators. Identifying these roles helps simplify RBAC policies. We explain how this situation can occur in FORBAC.

Example 18: Consider a FORBAC-policy $(\Sigma, \mathcal{UA}, \mathcal{PA})$ with $RT(\Sigma) = \{R_1, R_2\}$ and

$$\mathcal{PA}_{R_1}(r,p) \equiv \left(\begin{array}{l} \mathit{action}(p) \in \{\text{“read”}, \text{“write”}\} \ \wedge \\ (\mathit{level}(r) = \mathit{level}(p) \ \vee \ \mathit{level}(r) > \mathit{level}(p)) \end{array}\right)$$

$$\mathcal{PA}_{R_2}(r,p) \equiv \left(\begin{array}{l} (\mathit{action}(p) = \text{“read”} \ \vee \ \mathit{action}(p) = \text{“write”}) \ \wedge \\ \mathit{level}(r) \geq \mathit{level}(p) \end{array}\right).$$

Now, consider a Σ-structure S with two role instances $\overline{r_1}$ and $\overline{r_2}$ of R_1 and R_2, respectively. Suppose that $\mathit{level}^S(\overline{r_1}) = \mathit{level}^S(\overline{r_2})$. Observe that both instances are assigned the same set of permissions. As a result, whenever a user is assigned an instance $\overline{r}$ of R_2, she can be assigned instead an instance $\overline{r}'$ of R_1 with $\mathit{level}^S(\overline{r}') = \mathit{level}^S(\overline{r})$. The user would be authorized for the same set of permissions. Hence R_2 is redundant. ■

We now formally define the ideas from the previous example. For simplicity, we ignore set-valued attributes, but the presentation is analogous for the general case.

Definition 19: Let Σ be a FORBAC-signature and let $R_1, R_2 \in RT(\Sigma)$. We say that R_1 *expands* R_2 if for every attribute f of type $R_2 \to W$, with $W \in \{\textbf{String}, \textbf{Integer}\}$, there is the same symbol f, but of type $R_1 \to W$. ■

Definition 20: Let R_1 and R_2 be two role templates in some FORBAC-signature. We say that R_1 *subsumes* R_2 if R_1 expands R_2 and the following formula is valid:

$$\forall r_1 : R_1, r_2 : R_2 \,.\, \textstyle\bigwedge_{f:R_2\to W} f(r_1) = f(r_2) \to$$
$$\forall p \,.\, \mathcal{PA}_{R_2}(r_2, p) \to \mathcal{PA}_{R_1}(r_1, p)\,.$$

■

Here, f ranges over attributes of type $R_2 \to W$, with $W \in \{\textbf{String}, \textbf{Integer}\}$. This formula says the following. Let $\overline{r}_1$ and $\overline{r}_2$ be two role instances of R_1 and R_2, respectively. If $f^S(\overline{r}_1) = f^S(\overline{r}_2)$, for every attribute f of type $R_2 \to W$, then any permission assigned to $\overline{r}_2$ is also assigned to $\overline{r}_1$.

Using first-order logic equivalencies, it is easy to prove that R_1 subsumes R_2 iff R_1 expands R_2 and the following existential FORBAC-formula is unsatisfiable:

$$\exists r_1 : R_1, r_2 : R_2 \exists p \,.\, \textstyle\bigwedge_{f:R_2\to W} f(r_1) = f(r_2) \ \wedge$$
$$\mathcal{PA}_{R_2}(r_2, p) \ \wedge \ \neg\mathcal{PA}_{R_1}(r_1, p)\,.$$

Finally, we call two roles *equivalent* if they subsume each other. Equivalent roles point to potential redundancies in the policy. However, we note that two equivalent roles are not necessarily redundant. It may happen that such roles have different functions from an organizational perspective. For example, the role of a programmer may have exactly the same types of permissions as the role of a tester, but they need to be distinguished in an organization [4].

D. Redundant assignments

In classical RBAC, the assignment of roles to users and the assignment of permissions to roles are two tasks performed by different people who do not necessarily communicate with each other. The assignment of roles may be performed, for example, by people in human resources; whereas the assignment of permissions may be performed by the application owners. This might lead to a situation where for two roles r_1 and r_2 the permissions assigned to r_2 are contained in those assigned to r_1 and the users who are assigned r_2 are also assigned r_1. This is illustrated in Figure 1 in the introduction. In this case, role r_2 might be redundant.

This situation, presented in [4], occurs in a kind of FORBAC policies that we call *functional FORBAC-policies*. In a functional FORBAC-policy, for every role template R, any two role instances assigned to a same user have exactly the same attribute values. The policy presented in Example 9 is a functional FORBAC policy. For any two role instances $\overline{r}$ and $\overline{r}'$ of $R_{\texttt{Student}}$ assigned to a user $\overline{u}$, we have that $\mathit{country}(\overline{r}) = \mathit{country}(\overline{r}') = \{\mathit{nationality}(\overline{u}), \text{“US”}\}$. Similarly, for any two instances $\overline{r}$ and $\overline{r}'$ of $R_{\texttt{Employee}}$ assigned to a user $\overline{u}$, we have that $\mathit{limit}(\overline{r}) = \mathit{limit}(\overline{r}') = \mathit{salary}(\overline{u})$. Contrast this with a FORBAC-policy with a role template R such that

$$\mathcal{UA}_R(u,r) \equiv \mathit{age}(u) \geq 18 \ \wedge \ \mathit{level}(r) \leq 50.$$

This is not a functional FORBAC-policy. A user over 18 can be assigned several role instances, each with a different value for *level*. We now formally define functional FORBAC-policies.

Definition 21: A FORBAC-policy $(\Sigma, \mathcal{UA}, \mathcal{PA})$ is *functional* if every role template $R \in RT(\Sigma)$ satisfies the following two requirements:

1) $\mathcal{UA}_R(u,r)$ can be written as a conjunction $\mathcal{UA}^u_R(u) \ \wedge \ \mathcal{UA}^r_R(u,r)$. This means that the conditions for assigning a role instance to a user can be split in two: requirements the user must fulfill and requirements that the role instance must fulfill based on the user attributes.
2) The following formula is valid:
$$\forall u \forall r : R, r' : R \,.\, \mathcal{UA}^r_R(u,r) \ \wedge \ \mathcal{UA}^r_R(u,r') \to$$
$$\textstyle\bigwedge_{f:R\to W} f(r) = f(r'),$$
where f ranges over attributes of type $R \to W$, with $W \in \{\textbf{Integer}, \textbf{String}, 2^{\textbf{Integer}}, 2^{\textbf{String}}\}$. This means that any two role instances of R assigned to a same user have the same attribute values.

■

It is easy to automatically check if a FORBAC-policy $(\Sigma, \mathcal{UA}, \mathcal{PA})$ is functional. To check the second requirement, one checks, for every $R \in RT(\Sigma)$, whether the following existential FORBAC-formula is unsatisfiable.

$$\exists u \exists r : R, r' : R \,. \\ \mathcal{UA}^r_R(u,r) \ \wedge\ \mathcal{UA}^r_R(u,r') \ \wedge\ \bigvee_{f : R \to W} f(r) \neq f(r') \,.$$

Having defined what a functional FORBAC-policy is, we now introduce a policy analysis query for identifying redundant formulas in $\mathcal{UA}$. We start with an example.

Example 22: Consider a FORBAC-policy $(\Sigma, \mathcal{UA}, \mathcal{PA})$ with $RT(\Sigma) = \{R_1, R_2\}$ and

$$\mathcal{UA}_{R_1}(u,r) \equiv \mathit{age}(u) \geq 18 \ \wedge\ \mathit{level}(r) = 6$$

$$\mathcal{UA}_{R_2}(u,r) \equiv \mathit{age}(u) \geq 21 \ \wedge\ \mathit{level}(r) = 5$$

$$\mathcal{PA}_{R_1}(r,p) \equiv \\ \left(\begin{array}{l} \mathit{action}(p) \in \{\text{“read”}, \text{“write”}\} \ \wedge \\ \mathit{level}(r) \geq \mathit{level}(p) \end{array} \right)$$

$$\mathcal{PA}_{R_2}(r,p) \equiv \\ \left(\begin{array}{l} \mathit{action}(p) \in \{\text{“read”}\} \ \wedge \\ \mathit{level}(r) \geq \mathit{level}(p) \end{array} \right) .$$

Note that this is a functional FORBAC-policy. Now, observe that whenever a user is assigned an instance $\overline{r_2}$ of R_2, he is also assigned an instance $\overline{r_1}$ of R_1. Moreover, $\overline{r_1}$ would get more permissions than $\overline{r_2}$. This implies that $\mathcal{UA}_{R_2}(u,r)$ is redundant. ■

We now formally define this policy analysis query. Given a functional FORBAC-policy $(\Sigma, \mathcal{UA}, \mathcal{PA})$ and two role templates $R_1, R_2 \in RT(\Sigma)$, we want to check if the following formula is valid:

$$\left(\forall u \,. \mathcal{UA}^u_{R_2}(u) \to \mathcal{UA}^u_{R_1}(u)\right) \ \wedge \\ \forall u \forall r_1 : R_1, r_2 : R_2 \,. \\ \mathcal{UA}^r_{R_1}(u, r_1) \ \wedge\ \mathcal{UA}^r_{R_2}(u, r_2) \to \\ \forall p \,. \mathcal{PA}_{R_2}(r_2, p) \to \mathcal{PA}_{R_1}(r_1, p) \,.$$

If the previous formula is valid, then $\mathcal{UA}_{R_2}(u,r)$ is redundant in the FORBAC-policy. Checking the validity of the previous formula is equivalent to checking whether the following existential FORBAC-formula is unsatisfiable:

$$\left(\exists u \,. \mathcal{UA}^u_{R_2}(u) \ \wedge\ \neg \mathcal{UA}^u_{R_1}(u)\right) \vee \\ \exists u \exists r_1 : R_1, r_2 : R_2 \exists p \,. \\ \mathcal{UA}^r_{R_1}(u, r_1) \ \wedge\ \mathcal{UA}^r_{R_2}(u, r_2) \ \wedge \\ \mathcal{PA}_{R_2}(r_2, p) \ \wedge\ \neg \mathcal{PA}_{R_1}(r_1, p) \,.$$

V. EXPERIMENTAL RESULTS

We present here the evaluations of our two theses: the FORBAC language is suitable for specifying realistic access control policies and these policies can be analyzed with reasonable overhead. For this, we conducted a case study on the access-control infrastructure of a European bank. We had access to the access control policies of 350 applications, in particular the rules defining the assignments of roles to users and the assignments of permissions to roles. We chose 10 of those policies and translated their rules into FORBAC-policies. In our translation, we omitted those parts dealing with delegation of role instances and separation-of-duty constraints, which are out of the scope of FORBAC, as explained in Section II. For each of the 10 resulting FORBAC-policies, we randomly generated 10 different instances of the problems from Section IV and checked them against their respective policies using Z3.

A. Policy structure

We now describe the structure of the 10 translated FORBAC-policies $(\Sigma, \mathcal{UA}, \mathcal{PA})$.

a) User-assignment relation: In the FORBAC-policies, users are assigned role instances in two different ways. First, depending on a user's attribute values, like job or $\mathit{country}$, the user is automatically assigned the role instances that allow him to perform his duties. Second, users may require for some tasks more role instances than what the bank's policy automatically assigns to them. They therefore request additional role instances from the policy administrator, who assigns them individually.

These two ways are called *provisioned* and *individual* assignments. Both can be expressed as FORBAC-formulas of the following form:

$$\left(\bigwedge_i \mathit{conditions}_i(u) \right) \ \wedge\ \mathit{instanceAssigned}(u,r) . \qquad (3)$$

Whenever a user's attribute values satisfy $\bigwedge_i \mathit{conditions}_i(u)$, then the user can be assigned a role instance whose attribute values are defined by $\mathit{instanceAssigned}(u,r)$.

An example of this formula is

$$(\mathit{job}(u) = \text{“trader”} \ \wedge\ \mathit{country}(u) \in \{\text{“FR”}, \text{“USA”}\}) \\ \wedge\ \mathit{location}(r) = \mathit{country}(u) \ \wedge\ \mathit{value}(r) = 10{,}000.$$

This expresses a *provisioned* assignment, which assigns users, who are traders working in France or the USA, to a role instance with their own country as location and a value of $10,000$.

Individual assignments are a special case of *provisioned* assignments, where $\bigwedge_i \mathit{conditions}_i(u)$ contains only one conjunct of the form $\mathit{userID}(u) = c$, with c a constant, and $\mathit{instanceAssigned}(u,r)$ does not contain any attribute of the sort User. An example of an *individual* assignment is

$$\mathit{userID}(u) = 73{,}134 \\ \wedge\ \mathit{location}(u) = \text{“FR”} \ \wedge\ \text{value}(r) = 10{,}000.$$

Here, userID is an attribute of type $\mathit{Users} \to \textbf{Integer}$ used to identify the application's users.

The provisioned and the individual assignments for each role template $R \in RT(\Sigma)$ are expressed in $\mathcal{UA}_R(u,r)$ as a large disjunction of FORBAC-formulas of the form (3). To differentiate between provisioned and individual assignments we partition the disjunctions

$$\mathcal{UA}_R(u,r) = \mathcal{UA}^1_R(u,r) \ \vee\ \mathcal{UA}^2_R(u,r)$$

in two parts, where $\mathcal{UA}^1_R(u,r)$ contains the provisioned assignments and $\mathcal{UA}^2_R(u,r)$ contains the individual assignments.

b) Permission-assignment relation: For a role template R, the formula $\mathcal{PA}_R(r,p)$ has the form $\bigvee_i \bigwedge_j f_{ij}(p) \sim g_{ij}(r)$, where f_{ij} and g_{ij} are attributes and $\sim$ is one of the following: $=, \neq, \in,$ or $\notin$.

c) Size of the policies: In Table I, we report on the size of the 10 translated FORBAC-policies. The label $\sharp\mathcal{A}$ denotes the number of (single- and set-valued) attributes in the signature . $|\mathcal{UA}^1|$ denotes $\sum_R |\mathcal{UA}^1_R(u,r)|$, $|\mathcal{UA}^2|$ denotes $\sum_R |\mathcal{UA}^2_R(u,r)|$ and $|\mathcal{PA}|$ denotes $\sum_R |\mathcal{PA}_R(u,r)|$. $\sharp \mathit{Users}$ is the number of users. Finally, $\sharp RT(\Sigma)$ is the number of role templates.

Policy	$\sharp\mathcal{A}$	$\lvert\mathcal{UA}^1\rvert$	$\lvert\mathcal{UA}^2\rvert$	$\lvert\mathcal{PA}\rvert$	$\sharp \mathit{Users}$	$\sharp RT(\Sigma)$
App1	19	33	238,052	126	3,490	6
App2	24	1,646	174,655	1668	9,330	96
App3	56	694	256,439	232	34,782	51
App4	20	78	135,089	262	17,554	11
App5	20	16	3,262	156	85,949	8
App6	9	56	4,451	200	23,368	17
App7	15	363	1,911	237	44,276	14
App8	36	318	13,144	661	20,438	14
App9	15	249	9,427	160	8,152	11
App10	34	46	1,734	120	24,199	12

TABLE I
SIZE OF THE FORBAC POLICIES FOR 10 BANK APPLICATIONS

B. Generating instances of queries

For each of the selected policies and for each query presented in Section IV, we generated 10 instances and verified them against the selected policy with Z3. We present next, for each query, how we generated these instances.

a) Authorization inspection: We generate each instance as follows.

1) Build $\psi_{user}(u)$. Randomly choose an attribute f of type $\mathit{Users} \to W$, with $W \in \{\textbf{Integer}, \textbf{String}\}$ and a constant value c of type W. Let $\psi_{user}(u)$ be $f(u) = c$.
2) Randomly choose a value $k \leq 10$.
3) for $i = 1$ to k,
   a) Build $\psi_i(p_i)$. Randomly choose up to 5 attributes $f_1, f_2, \ldots, f_5$ and constant values $c_1, c_2, \ldots, c_5$. Let $\psi_i(p_i)$ be $\bigwedge_{i \leq 5} f_i(p_i) = c_i$.
   b) Build $\mathit{Auth}(u, p_i)$ as
   $$\bigvee_R \exists r : R \,.\, \mathcal{UA}_R(u,r) \ \wedge\ \mathcal{PA}_R(r,p_i),$$
4) The instance is the formula
$$\exists u \exists p_1, \ldots, p_k \,.\, \psi_{user(u)} \wedge \left(\bigwedge_{i \leq k} \psi_i(p_i) \wedge \mathit{Auth}(u,p_i) \right).$$

b) Assignment simplification: Recall that Σ is the FORBAC-signature used to specify the application's FORBAC-policy. Recall too that, for a role template R in Σ, $\mathcal{UA}^1_R(u,r)$ is a disjunction of formulas of the form
$$\left(\bigwedge_{i \leq K} \mathit{conditions}_i(u) \right) \ \wedge\ \mathit{instanceAssigned}(u,r).$$

To generate an instance of the assignment simplification query, we randomly choose $j \leq K$. The generated instance is then
$$\exists u \,.\, \neg \left(\bigwedge_{i \leq K} \mathit{conditions}_i(u) \leftrightarrow \bigwedge_{\substack{i \leq K \\ i \neq j}} \mathit{conditions}_i(u) \right).$$

Intuitively, we want to know if $\mathit{conditions}_j(u)$ is redundant. Note that $\mathit{instanceAssigned}(u,r)$ is not part of the formula because we want to simplify the conditions that assign the role to a user and $\mathit{instanceAssigned}(u,r)$ just describes the role instance.

c) Role subsumption: To generate an instance of the role subsumption query, we randomly choose two role templates R_1 and R_2. We then take $\mathcal{PA}_{R_1}(r,p)$ and $\mathcal{PA}_{R_2}(r,p)$ and define the instance as
$$\exists r_1 : R_1, r_2 : R_2 \ \exists p \,. \\ \textstyle\bigwedge_f f(r_1) = f(r_2) \ \wedge\ \mathcal{PA}_{R_2}(r_2,p) \ \wedge\ \neg\mathcal{PA}_{R_1}(r_1,p) \,.$$

Intuitively, we ask if R_1 subsumes R_2. The bank's policies, once translated in FORBAC, have the following property: for any two role templates R_1 and R_2, if there is defined a function $f : R_1 \to W$, then there is also defined a function $f : R_2 \to W$. In other words, every attribute is defined for all role templates, so it follows immediately that any two role templates expand each other.

d) Redundant assignments: Recall that, for every role template R, $\mathcal{UA}^1_R(u,r)$ is a disjunction of formulas of the form $(\bigwedge_i \mathit{conditions}_i(u)) \ \wedge\ \mathit{instanceAssigned}(u,r)$. These disjuncts are always functional FORBAC-formulas . To generate an instance of the redundant assignments query, we randomly choose two role templates R_1 and R_2 and one disjunct from each of $\mathcal{UA}^1_{R_1}(u,r)$ and $\mathcal{UA}^1_{R_2}(u,r)$. Let $(\bigwedge_i \mathit{conditions}_i(u)) \ \wedge\ \mathit{instanceAssigned}(u,r)$ and $(\bigwedge_j \mathit{conditions}'_j(u)) \ \wedge\ \mathit{instanceAssigned}'(u,r)$ be the selected disjuncts. The instance is
$$\left(\exists u \,.\, \textstyle\bigwedge_i \mathit{conditions}_i(u) \wedge \neg \bigwedge_j \mathit{conditions}'_j(u) \right) \ \vee \\ \exists u \ \exists r_1 : R_1, r_2 : R_2 \ \exists p \,. \\ \mathit{instanceAssigned}(u,r_1) \ \wedge \\ \mathit{instanceAssigned}'(u,r_2) \ \wedge \\ \mathcal{PA}_{R_2}(r_2,p) \ \wedge\ \neg\mathcal{PA}_{R_1}(r_1,p) \,.$$

C. Results and conclusions

We used the SMT solver Z3 to verify each of these instances against the corresponding FORBAC-policy. We ran the checks on a 2.50 GHz Intel Core i5 CPU, with 8GB of RAM. Table II shows how much time Z3 took to verify, for each application, 10 instances of the queries of Authorization Inspection (AI), Assignment Simplification (AS), Role Subsumption (RS), and Redundant Assignments (RA). A cell with NA indicates that there were not enough provisioned assignments to create 10 instances for the corresponding policy. A cell with >360 indicates that Z3 required more than 60 minutes for 10 instances, i.e. more than 360 seconds on average.

Policy	App1	App2	App3	App4	App5	App6	App7	App8	App9	App10
AI	357.87	>360	>360	>360	2.98	1.85	3.52	32.69	38.02	0.30
AS	0.61	0.63	0.57	0.54	*NA*	0.75	0.87	0.5	0.49	*NA*
RS	0.53	0.55	0.43	0.43	0.45	0.47	0.46	0.47	0.47	0.44
RA	0.73	0.47	0.46	0.49	*NA*	0.58	0.53	0.59	0.49	*NA*

TABLE II
TIME (IN SECONDS) NEEDED BY Z3 FOR AN INSTANCE ON AVERAGE

Regarding our first thesis, expressiveness, we could express the main parts of the policies in FORBAC, except for delegation of role instances and separation of duty constraints. Regarding our second thesis, that policies can be analyzed with reasonable overhead, the instances we generated for AS, RS, and RA can be analyzed by Z3 within one second for realistic policies. The only exceptions are the first four applications in AI, which include many individual assignments.

For policies where many individual assignments must be analyzed, we see two ways of proceeding in practice:

1) Check the AI queries only on the provisioned assignments $\mathcal{UA}^1_R(u,r)$. We note that in practice $\mathcal{UA}^2_R(u,r)$ should not be too large and can often be replaced by provisioned assignments to improve policy maintenance.
2) In case $\mathcal{UA}^2_R(u,r)$ is large, evaluate AI on each individual assignment separately. Based on our experience it is possible to restrict the query to a proper subset of all individual assignments before creating the SMT-files.

If we evaluate the 10 instances on $\mathcal{UA}^1_R(u,r)$ only, as described in 1), we obtain the following average times:

Policy	App1	App2	App3	App4	App5	App6	App7	App8	App9	App10
AI^1	0.11	4.97	1.56	0.15	0.13	0.12	0.40	0.45	0.31	0.12

Alternatively if we evaluate as in 2) on 10 randomly chosen users with individual assignments, we obtain the following average times:

Policy	App1	App2	App3	App4	App5	App6	App7	App8	App9	App10
AI^2	0.97	6.73	1.75	0.96	0.89	0.78	0.87	1.21	0.95	2.02

Finally note that the evaluation of the individual assignments can be executed in parallel on a cluster since they can be analyzed independently. If parallelization is not supported, one can instead proceed iteratively. In each of the 10 FORBAC-policies, there were at most 10,000 users with individual assignments. Therefore, an upper bound on the time required for 2) is one day in the worst case. From the bank's point of view this is still reasonable since AI queries must be executed only rarely for audits and can be run offline over night.

VI. RELATED WORK

In the early days of RBAC, it was sufficient to propose an RBAC model that overcame the limitations that were observed in previous RBAC models. Here, limitations were understood in terms of expressiveness, not policy analysis. Later, some authors (such as [22] and [17]) noted that the expressive power of policy specification languages hindered policy analysis. Hence, our perspective is that new RBAC models should be frameworks that provide three components: a language for policy specification, a language for specifying policy analysis queries, and procedures for efficiently evaluating these queries. To the best of our knowledge, there are two such frameworks that have gained acceptance in the literature: [31] and [5]. We discuss them as well as other work related to RBAC policy analysis.

A. Margrave

Margrave [31] is a framework for policy specification and analysis. With Margrave, users can specify policies and query their properties. Margrave then searches for representative scenarios that satisfy the property.

In our framework, we reason about RBAC policies and compute one satisfying scenario rather than a set of scenarios. In contrast to Margrave, our framework can reason about integer constraints and, therefore, express policies like "Alice can read a file if her clearance level is greater than the file's clearance level".

FORBAC's interaction between set-valued terms and roles gives rise to policies that cannot be modeled in Margrave. Consider, for example, a FORBAC signature with a role template $Technician$ and the following functions:

- OU that assigns a string to each user which represents the user's organizational unit.
- $Sectors$ that assigns a set of integers to each role instance of $Technician$.
- $sector$ that assigns an integer to each permission.

Suppose that only those users u with $OU(u) = A$ are assigned an instance r of $Technician$ with $Sectors(r) = [100, 110]$, and that an instance r of $Technician$ is authorized to any permission p with $sector(p) \in Sectors(r)$.

In Margrave, we can use two policies to model the user and permission-assignments, respectively. The first policy indicates when a user u is assigned a role instance r and can be expressed in Margrave as the following first-order formula:

$$\Phi_{UA} := \forall u, r \, . \left(\begin{pmatrix} OU(u) = A \wedge \\ \bigwedge_{100 \leq i \leq 110} Sectors(r,i) \end{pmatrix} \rightarrow mUA(u,r) \right) .$$

The second policy indicates when a role instance r is authorized for permission p:

$$\Phi_{PA} := \forall r, p \, . \, (Sectors(r, sector(p)) \rightarrow mPA(r,p)) \, .$$

To decide authorization, we use the Margrave policy analysis framework. A user u is authorized for permission p if the following query is satisfiable:

$$\exists r \, . \, (\Phi_{UA} \ \wedge \ \Phi_{PA} \ \wedge \ mUA(u,r) \ \wedge \ mPA(r,p)) \, .$$

Now, consider the following property: There is a user who is authorized for a permission in sector 300. Such a user does not exist since instances of role $Technician$ can access only the sectors between 100 and 110. However, if we query Margrave with

$$\exists u, r, p \, . \, (mUA(u,r) \ \wedge \ mPA(r,p) \ \wedge \ sector(p) = 300) \, ,$$

then Margrave responds with a scenario consisting of a user $\tilde{u}$ with $OU(\tilde{u}) = A$, a permission $\tilde{p}$ with $sector(\tilde{p}) = 300$, and a role instance $\tilde{r}$ with $\mathit{Sectors}(\tilde{r}) = \{100, \ldots, 110, 300\}$. This is because the policy that assigns instances of *Technician* allows one to assign to a user any instance that contains at least the sectors from 100 through 110. It is not possible to assign an instance of *Technician* that contains only the sectors from 100 through 110 unless we explicitly say that all other sectors must not be assigned:

$$\left(\begin{array}{l} OU(u) = A \wedge \\ \bigwedge_{100 \leq i \leq 110} \mathit{Sectors}(r,i) \wedge \\ \bigwedge_{i \geq 111} \neg \mathit{Sectors}(r,i) \end{array} \right) \rightarrow mUA(u,r). \qquad (4)$$

Such specifications, however, are difficult to maintain. If new sectors are created in the environment, then the policy must be adapted to prevent the new sectors from being authorized for technicians. In addition, Margrave cannot efficiently handle policies with large attribute domains [5]. In contrast, this policy can be expressed in FORBAC as follows:

$$\begin{array}{l} \mathcal{UA}_{\mathit{Technician}}(u,r) \equiv \left(\begin{array}{l} OU(u) = A \wedge \\ \mathit{Sectors}(r) = [100, 110] \end{array} \right) \\ \mathcal{PA}_{\mathit{Technician}}(r,p) \equiv \ \ \mathit{sector}(p) \in \mathit{Sectors}(r). \end{array} \qquad (5)$$

The fact that no user is authorized for permission in sector 300 follows from the unsatisfiability of the existential FORBAC formula: $\exists u, p \,.\, (\mathit{Auth}(u,p) \ \wedge\ \mathit{sector}(p) = 300)\,.$

B. Athena+Yices

Another framework related to our work is Athena+Yices [5], which merges functional programming with first-order logic for both policy specification and property verification. For verifying properties, the Yices SMT-solver is used. They do experiments on the CONTINUE [27] policy and achieve results better than Margrave [17] and the framework used in [26]. In contrast to FORBAC, their language can express arithmetic constraints and they can reason about XACML policies.

Athena+Yices faces the same problem as Margrave when dealing with the administration of set-valued attributes. Although Athena+Yices can reason about arithmetic constraints, they do not allow quantification when specifying policies. This makes it impossible to write Policy (5), unless users explicitly list all the sectors that should and should not be allowed.

The authors [5] do not provide complexity bounds for the policy analysis problems they consider. The policy properties they propose are undecidable in general because their syntax allows addition, subtraction, and multiplication of integer variables, which allows Diophantine equations to be expressed as requirements for authorization. This shows again how one ends up in undecidable fragments of policy analysis if the language is not restricted. In contrast, FORBAC ensures that all the given policy analysis queries are evaluated in NP. However, the low complexity of policy analysis for FORBAC does not come for free. Existential FORBAC formulas cannot express relevant queries presented in [5] like observational equivalence, conflict detection, and change-impact analysis.

C. Other related work

Other researchers examine policy analysis for RBAC, but they neither propose a language for specifying properties of RBAC policies nor procedures for verifying them. They only propose procedures for verifying specific properties.

For example, [16], [21], [34] focus on the reachability problem. In this problem, a set of users is given together with a set of administrative rules. These administrative rules specify who can assign and remove role assignments according to the roles assigned to the users. The objective of the reachability problem is to decide if it is possible for the given set of users to reach a goal set of roles using the administrative rules. We did not include the reachability problem in our framework because there are many efficient frameworks proposed for this problem. Additionally, the reachability problem is defined only for classical roles. To the best of our knowledge, no work has investigated administrative rules for role templates.

[6] uses SMT-solvers to identify induced role hierarchies. They model rule-based user-role assignments and decide which role assignments are entailed by others within an RBAC model. Their framework can reason about negative authorization via negated roles. However, they only work with classical roles and single-valued attributes. In contrast, we analyze this problem with role templates and set-valued attributes.

Another language that combines policy specification and policy analysis is presented in [14]. It is a Datalog-based language that can express policies in dynamic environments and performs verification on goal reachability and contextual policy containment. Goal reachability consists of deciding if the system can reach a state where a given property holds, expressed as an existential formula. Contextual policy containment consists of deciding, for two given policies and in every state the system can reach, whether those permissions authorized by one policy are authorized by the other. However, their framework neither expresses nor reasons about numeric constraints and set-valued attributes.

Lithium [19] is a policy specification language based on a fragment of first-order logic. Lithium policies are sets of formulas consisting of either conjunctions of ground literals or universally quantified formulas of the form $\forall x_1 \ldots \forall x_n \,.\, (\ell_1 \ \wedge\ \ldots\ \wedge\ \ell_k \rightarrow \ell_{k+1})$, where $\ell_1, \ldots, \ell_{k+1}$ are literals. It can express a variety of authorization policies used in practice and also decide authorization in polynomial time. In contrast to our framework, Lithium allows predicates and function symbols of any arity as long as they respect a set of conditions (see [19] for details). For example, in a Lithium policy, there must not be two formulas, which contain two literals ℓ and ℓ' such that ℓ and $\neg\ell'$ are unifiable. However, they cannot handle numeric constraints and their policy analysis framework is limited. They only show how to efficiently decide whether, for a given policy, a permission is permitted and denied at the same time.

VII. Conclusion and future work

Many policy specification languages have been proposed and new languages are often more expressive than their predecessors. However, policy analysis can only handle a fragment of the languages for which they are intended [17], [22], [26], [32]. This is also the case for RBAC where new extensions have richer expressiveness, but policy analysis becomes more difficult.

In this work, we have presented FORBAC as a framework that strikes a balance between expressiveness and efficient policy analysis for RBAC. We have provided strong evidence that FORBAC can specify and reason about relevant policies. As future work, we propose to extend FORBAC with two RBAC idioms that have received attention in the literature. The first is role hierarchies, which define a partial order on a set of roles. Roles are assigned those permissions granted to that role and those granted to subroles in the hierarchy. There is no standard that specifies how to define role hierarchies with role templates. The second idiom consists of constraints. The NIST standard for RBAC [15] offers two types of constraints: static and dynamic separation-of-duty constraints and cardinality constraints. It remains open how to define these constraints in RBAC models where users, roles, and permissions have attributes

References

[1] Ali E Abdallah and Etienne J Khayat. A formal model for parameterized role-based access control. In *Formal Aspects in Security and Trust*, pages 233–246. Springer, 2005.

[2] Gail-Joon Ahn and Ravi Sandhu. The RSL99 language for role-based separation of duty constraints. In *Proceedings of the fourth ACM workshop on Role-based access control*, pages 43–54. ACM, 1999.

[3] Mohammad A Al-Kahtani and Ravi Sandhu. A model for attribute-based user-role assignment. In *Computer Security Applications Conference, 2002. Proceedings. 18th Annual*, pages 353–362. IEEE, 2002.

[4] Mohammad A Al-Kahtani and Ravi Sandhu. Induced role hierarchies with attribute-based RBAC. In *Proceedings of the eighth ACM symposium on access control models and technologies*, pages 142–148. ACM, 2003.

[5] Konstantine Arkoudas, Ritu Chadha, and Jason Chiang. Sophisticated access control via SMT and logical frameworks. *ACM Transactions on Information and System Security (TISSEC)*, 16(4):17, 2014.

[6] Alessandro Armando and Silvio Ranise. Automated and efficient analysis of role-based access control with attributes. In *Data and Applications Security and Privacy XXVI*, pages 25–40. Springer, 2012.

[7] Ezedin Barka and Ravi Sandhu. A role-based delegation model and some extensions. In *23rd National Information Systems Security Conference*, pages 396–404. Citeseer, 2000.

[8] David Basin, Manuel Clavel, Jürgen Doser, and Marina Egea. Automated analysis of security-design models. *Information and Software Technology*, 51(5):815–831, 2009.

[9] Moritz Y Becker, Cédric Fournet, and Andrew D Gordon. SecPAL: Design and semantics of a decentralized authorization language. *Journal of Computer Security*, 18(4):619–665, 2010.

[10] Moritz Y Becker and Peter Sewell. Cassandra: Flexible trust management, applied to electronic health records. In *Computer Security Foundations Workshop, 2004. Proceedings. 17th IEEE*, pages 139–154. IEEE, 2004.

[11] Rafae Bhatti, Arif Ghafoor, Elisa Bertino, and James BD Joshi. X-GTRBAC: an XML-based policy specification framework and architecture for enterprise-wide access control. *ACM Transactions on Information and System Security (TISSEC)*, 8(2):187–227, 2005.

[12] Piero Bonatti, Clemente Galdi, and Davide Torres. ERBAC: event-driven RBAC. In *Proceedings of the 18th ACM symposium on Access control models and technologies*, pages 125–136. ACM, 2013.

[13] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[14] Daniel J Dougherty, Kathi Fisler, and Shriram Krishnamurthi. Specifying and reasoning about dynamic access-control policies. In *Automated Reasoning*, pages 632–646. Springer, 2006.

[15] David F Ferraiolo, Ravi Sandhu, Serban Gavrila, D Richard Kuhn, and Ramaswamy Chandramouli. Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security (TISSEC)*, 4(3):224–274, 2001.

[16] Anna Lisa Ferrara, P Madhusudan, and Gennaro Parlato. Policy analysis for self-administrated role-based access control. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 432–447. Springer, 2013.

[17] Kathi Fisler, Shriram Krishnamurthi, Leo A. Meyerovich, and Michael Carl Tschantz. Verification and change-impact analysis of access-control policies. In *Proceedings of the 27th International Conference on Software Engineering*, ICSE '05, pages 196–205, New York, NY, USA, 2005. ACM.

[18] Luigi Giuri and Pietro Iglio. Role templates for content-based access control. In *Proceedings of the second ACM workshop on Role-based access control*, pages 153–159. ACM, 1997.

[19] Joseph Y Halpern and Vicky Weissman. Using first-order logic to reason about policies. *ACM Transactions on Information and System Security (TISSEC)*, 11(4):21, 2008.

[20] Jingwei Huang, David M Nicol, Rakesh Bobba, and Jun Ho Huh. A framework integrating attribute-based policies into role-based access control. In *Proceedings of the 17th ACM symposium on Access Control Models and Technologies*, pages 187–196. ACM, 2012.

[21] Somesh Jha, Ninghui Li, Mahesh Tripunitara, Qihua Wang, and William H Winsborough. Towards formal verification of role-based access control policies. *Dependable and Secure Computing, IEEE Transactions on*, 5(4):242–255, 2008.

[22] Xin Jin, Ram Krishnan, and Ravi Sandhu. A role-based administration model for attributes. In *Proceedings of the First International Workshop on Secure and Resilient Architectures and Systems*, pages 7–12. ACM, 2012.

[23] Xin Jin, Ram Krishnan, and Ravi Sandhu. Reachability analysis for role-based administration of attributes. In *Proceedings of the 2013 ACM workshop on Digital identity management*, pages 73–84. ACM, 2013.

[24] Xin Jin, Ravi Sandhu, and Ram Krishnan. RABAC: role-centric attribute-based access control. In *Computer Network Security*, pages 84–96. Springer, 2012.

[25] James BD Joshi. Access-control language for multidomain environments. *Internet Computing, IEEE*, 8(6):40–50, 2004.

[26] Vladimir Kolovski, James Hendler, and Bijan Parsia. Analyzing web access control policies. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 677–686, New York, NY, USA, 2007. ACM.

[27] Shriram Krishnamurthi. The CONTINUE server (or, How I administered PADL 2002 and 2003). In *Practical aspects of declarative languages*, pages 2–16. Springer, 2003.

[28] D Richard Kuhn, Edward J Coyne, and Timothy R Weil. Adding attributes to role-based access control. *Computer*, 43(6):79–81, 2010.

[29] Torsten Lodderstedt, David Basin, and Jürgen Doser. SecureUML: A UML-based modeling language for model-driven security. In *UML 2002 The Unified Modeling Language*, pages 426–441. Springer, 2002.

[30] Tim Nelson, Christopher Barratt, Daniel J. Dougherty, Kathi Fisler, Shriram Krishnamurthi, and Varun Singh. The Margrave tool. http://www.margrave-tool.org/v3/. Accessed: 2015-01-12.

[31] Timothy Nelson, Christopher Barratt, Daniel J Dougherty, Kathi Fisler, and Shriram Krishnamurthi. The Margrave tool for firewall analysis. In *LISA*, 2010.

[32] Salman Saghafi, Tim Nelson, and Daniel J Dougherty. Geometric logic for policy analysis. In *International Workshop on Automated Reasoning in Security and Software Verification (ARSEC 2013)*, pages 12–20, 2013.

[33] Ravi Sandhu. Role hierarchies and constraints for lattice-based access controls. In *Computer Security ESORICS 96*, pages 65–79. Springer, 1996.

[34] Scott D Stoller, Ping Yang, Mikhail I Gofman, and CR Ramakrishnan. Symbolic reachability analysis for parameterized administrative role-based access control. *Computers & Security*, 30(2):148–164, 2011.

Appendix

A. Complexity results for F

Theorem 23: For a policy $T = \{\varphi_1(x), \dots, \varphi_\ell(x)\}$ in F, deciding whether the formula $\forall x\,.\,(\varphi_1(x) \vee \dots \vee \varphi_\ell(x))$ is valid is NP-hard.

Proof. By reduction to the validity problem for propositional Boolean formulas in disjunctive normal form. Let $\psi = \bigvee_{i \leq M} \bigwedge_{j \leq N} \ell_{ij}$ be a propositional Boolean formula in disjunctive normal form. For every propositional variable p occurring in ψ, pick a unary relation symbol Q_p. Let x be a first-order variable. For $i \leq M$ and $j \leq N$, let $L_{ij}(x)$ be the following formula:

$$L_{ij}(x) = \begin{cases} Q_p(x) & \text{if } \ell_{ij} = p \\ \neg Q_p(x), & \text{if } \ell_{ij} = \neg p. \end{cases}$$

Finally, let T_ψ be the following set of formulas in F:

$$\left\{ \bigwedge_{j \leq N} L_{ij}(x) \,:\, i \leq M \right\}.$$

It is easy to check that ψ is valid iff the following formula is valid. $\forall x\,.\,\left(\bigvee_{i \leq M} \bigwedge_{j \leq N} L_{ij}(x)\right)$. ■

We now explain how to translate a policy T in F to a Margrave [30] policy T'. Define a Margrave predicate $Permit(x)$ with a free variable x and one Margrave predicate $Q'(x)$ for every first-order predicate $Q(x)$ occurring in T. For every formula $\bigwedge_{i \leq M} Q_i(x) \wedge \bigwedge_{j \leq N} \neg R_j(x)$ in T, define the following Margrave rule in T':

$$\begin{array}{rl} Permit(x) :- & Q'_1(x), Q'_2(x), \dots, Q'_M(x), \\ & \neg R'_1(x), \neg R'_2(x), \dots, \neg R'_N(x). \end{array}$$

The rule says that those access requests that satisfy Q'_1, $Q'_2, \dots, Q'_M$ and not R'_1, $R'_2, \dots, R'_N$ are permitted. It is easy to check that an access request is permitted in T iff the access request is also permitted in T'. The same technique can be used to translate policies in F into policies in the language presented in [5].

B. Complexity results for FORBAC

In this section we prove Theorems 13 and 15. Theorem 13 gives a complexity bound on deciding, for a given FORBAC-policy $(\Sigma, \mathcal{UA}, \mathcal{PA})$, a Σ-structure S, a user $\overline{u}$ in S, and a permission $\overline{p}$ in S, whether $\overline{u}$ is authorized for $\overline{p}$ in S. For this, we define first how to store S in memory. S is stored as a tuple

$$\left(n_{Users}, n_{Perms}, (n_R : R \in RT(\Sigma)), \left([\![f^S]\!] : f \in \mathcal{A}_1 \cup \mathcal{A}_2\right)\right),$$

where n_{Users} denotes the number of users in S; n_{Perms} denotes the number of permissions in S; n_R, for $R \in RT(\Sigma)$, denotes the number of role instances of R in S; and $[\![f^S]\!]$, for f of type $V \to W$, is an array of elements of W^S of length n_V. We assume that the users of S are enumerated from 1 to n_{Users}. For every f in $\mathcal{A}_1 \cup \mathcal{A}_2$ of type $Users \to W$ and for $i \leq n_{Users}$, $[\![f^S]\!][i]$ contains the value of f^S for the i-th user. Similar assumptions hold for the permissions and the role instances of S. If f is a set-valued attribute, then $[\![f^S]\!][i]$ stores the elements of the set as a list. Finally, let $Roles^S := \bigcup_{R \in RT(\Sigma)} R^S$ denote all role instances of S.

After these preparations, we prove Theorem 13.

Theorem 13: Given a FORBAC-policy $(\Sigma, \mathcal{UA}, \mathcal{PA})$, a Σ-structure S, a user $\overline{u} \in Users^S$ and a permission $\overline{p} \in Perms^S$, deciding whether $\overline{u}$ is authorized for $\overline{p}$ takes time

$$O\left(\left|Roles^S\right| \cdot |S|^2 \cdot (|\mathcal{UA}| + |\mathcal{PA}|)\right),$$

where $|S|$ is the size of S in memory and $|\mathcal{UA}| + |\mathcal{PA}| = \sum_{R \in RT(\Sigma)} (|\mathcal{UA}_R(u,r)| + |\mathcal{PA}_R(r,p)|)$.

Proof. We propose the following algorithm to check if $\overline{u}$ is authorized for $\overline{p}$:

1) For every $R \in RT(\Sigma)$ and every role instance $\overline{r}$ of R in S, do the following.
   a) Compute $[\![\mathcal{UA}_R(u,r) \wedge \mathcal{PA}_R(r,p)]\!]$, a formula obtained from $\mathcal{UA}_R(u,r) \wedge \mathcal{PA}_R(r,p)$ by replacing every atomic formula φ with $\top$ or $\bot$, depending on whether S satisfies φ or not.
   b) If $[\![\mathcal{UA}_R(u,r) \wedge \mathcal{PA}_R(u,r)]\!]$ evaluates to true, then output that $\overline{u}$ is authorized for $\overline{p}$.
2) If at this point it has not been output that $\overline{u}$ is authorized for $\overline{p}$, then output that $\overline{u}$ is not authorized for $\overline{p}$.

Note that Steps 1b and 2 take $O(|\mathcal{UA}| + |\mathcal{PA}|)$ and constant time, respectively. It suffices to show then that Step 1a takes $O\left(|S|^2 \cdot (|\mathcal{UA}| + |\mathcal{PA}|)\right)$ time. To show this, observe that, by the way we store S in memory, we can check in $O(|S|^2)$-time whether S satisfies an atomic FORBAC-formula. ■

Theorem 15: Deciding satisfiability of an existential-FORBAC formula is NP-complete.

We prove NP-hardness by reduction to the satisfiability problem for propositional Boolean formulas. Let ψ be a Boolean propositional formula. Define a FORBAC-signature that contains a unary symbol f_p of type $Users \to$ **Integer** for each proposition p in ψ. Let $\exists u\, \psi'(u)$ be the existential FORBAC formula where $\psi'(u)$ is obtained from ψ by replacing every proposition p in ψ with $f_p(u) = 1$. Note that $\exists u\, \psi'(u)$ is satisfiable iff ψ is satisfiable.

We now prove that satisfiability of existential FORBAC formulas is in NP. Let Σ be a FORBAC-signature and $\Phi = \exists x_1, x_2, \dots, x_k\,.\,\varphi(x_1, x_2, \dots, x_k)$ be an existential FORBAC formula over Σ. We assume that all unary functions and all binary relations map to integers. The proof for the general case is analogous. A *certificate for* Φ is a function $\mathcal{C}$ mapping every single-valued term in Φ to an integer and every set-valued term in Φ to a set of integers. A *term* is any single or set-valued term. For a term t occurring in $\mathcal{C}$, we denote with $[\![t]\!]_{\mathcal{C}}$ the interpretation of t by $\mathcal{C}$. $\mathcal{C}$ *satisfies a formula* ψ if the Boolean expression that results from replacing every term t in ψ for $[\![t]\!]_{\mathcal{C}}$ evaluates to true. Note that Φ is satisfiable iff there is a certificate that satisfies Φ.

We define the *size* of a certificate $\mathcal{C}$ for Φ. The *size* of an integer is the length of its encoding in the input alphabet. The *size* of a set of integers is the sum of the size of its elements.

The *size* of a certificate $\mathcal{C}$ for Φ is the sum of all the sizes of $[\![t]\!]_{\mathcal{C}}$, where t ranges over terms occurring in Φ.

From now on, we write t and t' to denote any single-valued term, c and c' to denote any integer constants, T and T' to denote any set-valued term, C and C' to denote any set of integer constants, $f(e)$ to denote any single-valued term with f a unary function and e a variable, and $F(e)$ to denote any set-valued term with F a binary relation and e a variable.

To show that the set of satisfiable existential FORBAC formulas is in NP, it suffices to show that for a satisfiable existential FORBAC formula Φ, there exists a certificate $\overline{\mathcal{C}}$ that satisfies Φ and is of size at most $n^2 \log(d+n)$, where d is the size of the largest integer constant occurring in Φ and n is the length of Φ. If Φ is satisfiable, then there exists a certificate $\mathcal{C}$ that satisfies Φ; however $\mathcal{C}$ does not need to have a size polynomial in the length of Φ for two reasons. First, the size of $[\![f(e)]\!]_{\mathcal{C}}$ might be large. Second, the set $[\![F(e)]\!]_{\mathcal{C}}$ might have a large number of elements or some element in $[\![F(e)]\!]_{\mathcal{C}}$ might have a large size.

The rest of the proof shows how to build a small certificate $\overline{\mathcal{C}}$ from $\mathcal{C}$. But before that, we build an auxiliary set σ of atomic formulas. For every atomic formula ψ occurring in Φ, put ψ in σ, if $\mathcal{C}$ satisfies ψ, and put $\neg\psi$ in σ, otherwise. There are many kinds of atomic formulas in σ. We can reduce this by rewriting the following formulas as follows:

1) $f(e) \leq t$. In this case, replace $f(e) \leq t$ with $f(e) = t$, if $[\![f(e)]\!]_{\mathcal{C}} = [\![t]\!]_{\mathcal{C}}$, and with $f(e) < t$, otherwise.
2) $f(e) = t$. In this case, replace every occurrence of $f(e)$ in σ with t. After $\overline{\mathcal{C}}$ is built, define $[\![f(e)]\!]_{\overline{\mathcal{C}}}$ as $[\![t]\!]_{\overline{\mathcal{C}}}$.
3) $c \sim c'$. In this case, remove the formula from σ.
4) $F(e) = T$. In this case, replace every occurrence of $F(e)$ in σ with T. After $\overline{\mathcal{C}}$ is built, define $[\![F(e)]\!]_{\overline{\mathcal{C}}}$ as $[\![T]\!]_{\overline{\mathcal{C}}}$.
5) $F(e) \neq T$. In this case, replace $F(e) \neq T$ with $F(e) \not\subseteq T$ if $[\![F(e)]\!]_{\mathcal{C}} \not\subseteq [\![T]\!]_{\mathcal{C}}$ and $T \not\subseteq F(e)$ otherwise.
6) $c \in F(e)$. In this case, replace this formula in σ with $\{c\} \subseteq F(e)$.
7) $c \notin F(e)$. In this case, replace this formula in σ with $F(e) \not\subseteq \mathbb{Z} \setminus \{c\}$.
8) $f(e) \notin C$. In this case, replace $f(e) \notin C$ with $f(e) \in \mathbb{Z} \setminus C$.
9) $T \not\subseteq T'$. In this case, take a new fresh integer variable t and replace $T \not\subseteq T'$ with the two formulas $t \in T$ and $t \notin T'$. The value of $[\![t]\!]_{\mathcal{C}}$ is a number that belongs to $[\![T]\!]_{\mathcal{C}}$ but not to $[\![T']\!]_{\mathcal{C}}$.

After this, only the following kinds of formulas occur in σ: $c < f(e)$, $f(e) < c$, $f(e) < f'(e')$, $C \subseteq F(e)$, $F(e) \subseteq C$, $F(e) \subseteq F'(e')$, $f(e) \in F(e)$, $f(e) \notin F(e)$, and $f(e) \in C$. We sometimes write $[\psi] \in \sigma$ instead of $\psi \in \sigma$ to prevent awkward notation like $f(e) \in F(e) \in \sigma$. For two set-valued terms T and T', we say that $T \subseteq_\sigma T'$ if $[T \subseteq T'] \in \sigma$. We denote with $\subseteq_\sigma^*$ the reflexive-transitive closure of $\subseteq_\sigma$.

Note that any certificate that satisfies all formulas in σ will also satisfy Φ. In particular, $\mathcal{C}$ satisfies all formulas in σ.

Let $f_1(e_1), \ldots, f_m(e_m)$ be all the non-constant single-valued terms that occur in Φ. We suppose that they are enumerated in a way that $[\![f_j(e_j)]\!]_{\mathcal{C}} \leq [\![f_{j+1}(e_{j+1})]\!]_{\mathcal{C}}$, for $j < m$. We build from $\mathcal{C}$ a sequence of certificates $\mathcal{C} = \mathcal{C}_0, \mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_m$ that satisfies the following invariant: for $\mathcal{C}_j$, all $[\![f_1(e_1)]\!]_{\mathcal{C}_j}, [\![f_2(e_2)]\!]_{\mathcal{C}_j}, \ldots, [\![f_j(e_j)]\!]_{\mathcal{C}_j}$ are at most $j+d$, where d is the largest constant that occurs in Φ. Therefore, they all have size at most $\log(j+d) \leq \log(n+d)$. After that, we build from $\mathcal{C}_m$ a certificate $\overline{\mathcal{C}}$ that satisfies Φ and where for every set-valued term $F(e)$ the size of $[\![F(e)]\!]_{\mathcal{C}}$ is polynomial in the length of Φ.

For $j \leq m$, suppose that $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_{j-1}$ are already built. We explain how to build $\mathcal{C}_j$. For any set-valued term $F(e)$, let $[\![F(e)]\!]_{\mathcal{C}_j} = [\![F(e)]\!]_{\mathcal{C}}$. For $i \neq j$, let $[\![f_i(e_i)]\!]_{\mathcal{C}_j} = [\![f_i(e_i)]\!]_{\mathcal{C}_{j-1}}$ and for $i > j$, let $[\![f_i(e_i)]\!]_{\mathcal{C}_j} = [\![f_i(e_i)]\!]_{\mathcal{C}}$. Next, we compute a value for $[\![f_j(e_j)]\!]_{\mathcal{C}_j}$ small enough that fits all the constraints in σ that $f_j(e_j)$ must fulfill. First, we compute the set A_j of possible values for $[\![f_j(e_j)]\!]_{\mathcal{C}_j}$. Let A_j be the set of integers a such that

1) $[\![f_{j-1}(e_{j-1})]\!]_{\mathcal{C}_{j-1}} \leq a$, if $j > 1$.
2) $c < a$, for any integer constant c such that $[c < f_j(e_j)] \in \sigma$.
3) $a \in C$, for any set of constants C and any set-valued term $F(e)$ such that $[f_j(e_j) \in F(e)] \in \sigma$ and $F(e) \subseteq_\sigma^* C$.
4) $a \notin C$, for any set of constants C and any set-valued term $F(e)$ such that $[f_j(e_j) \notin F(e)] \in \sigma$ and $C \subseteq_\sigma^* F(e)$.
5) $a \neq [\![f_i(e_i)]\!]_{\mathcal{C}_{j-1}}$, for any single-valued term $f_i(e_i)$ with $i < j$ and any set-valued term T such that
   a) $[f_i(e_i) \in T'] \in \sigma$, $[f_j(e_j) \notin T] \in \sigma$ and $T' \subseteq_\sigma^* T$, or
   b) $[f_i(e_i) \notin T] \in \sigma$, $[f_j(e_j) \in T'] \in \sigma$ and $T' \subseteq_\sigma^* T$.

Note that A_j is an intersection of sets of integer intervals, one set for every item above. Furthermore, an extreme point in that interval is either one of $f_i(e_i)$ with $i < j$ or an integer constant occurring in σ. Therefore, an extreme point in any interval of A_j is at most one more than the maximum among $[\![f_1(e_1)]\!]_{\mathcal{C}_{j-1}}, \ldots, [\![f_{j-1}(e_{j_1})]\!]_{\mathcal{C}_{j-1}}$ and d, the largest integer constant occurring in σ. By our invariant, $[\![f_i(e_i)]\!]_{\mathcal{C}_{j-1}} \leq d + j - 1$. Therefore, an extreme point in any interval of A_j is at most $d + j$.

Let $f_j(e_j)$ be the minimum value of A_j. Note that (i) A_j is not empty, as $[\![f_j(e_j)]\!]_{\mathcal{C}_{j-1}} \in A_j$ and (ii) for $j > 1$, A_j has a minimum, as it is bounded below by $[\![f_{j-1}(e_{j-1})]\!]_{\mathcal{C}_{j-1}}$. A_j might be unbounded below when $j = 1$. In that case, let $[\![f_1(e_1)]\!]_{\mathcal{C}_1}$ be an extreme point of A_1 and if A_1 is the entire set of integers, then let it be 0.

Recall that $[\![f_j(e_j)]\!]_{\mathcal{C}_{j-1}} \in A_j$. Therefore, $[\![f_j(e_j)]\!]_{\mathcal{C}_j} \leq [\![f_j(e_j)]\!]_{\mathcal{C}_{j-1}}$.

Let σ_S be the subset of σ that contains all atomic formulas where a set-valued term of the form $F(e)$ occurs.

Lemma 16: If $\mathcal{C}$ satisfies all formulas in σ, then $\mathcal{C}_m$ satisfies all formulas in $\sigma \setminus \sigma_S$.

Proof. Suppose that $\mathcal{C}$ satisfies all formulas in σ. It suffices to show by induction on $j \leq m$ that $\mathcal{C}_j$ satisfies all formulas in σ. The base case is obvious, since $\mathcal{C}_0 = \mathcal{C}$ by definition.

So suppose that $\mathcal{C}_j$ satisfies all formulas in σ. We show that $\mathcal{C}_{j+1}$ satisfies all formulas in σ. $\mathcal{C}_j$ and $\mathcal{C}_{j+1}$ differ only on the interpretation of $f_j(e_j)$, so it suffices to show that $\mathcal{C}_{j+1}$ satisfies any atomic formula $\psi \in \sigma$ that involves $f_j(e_j)$. We evaluate the possible cases for ψ:

- $c < f_j(e_j)$. Since $[\![f_j(e_j)]\!]_{\mathcal{C}_j} \in A_j$, we have by Item 2 of the definition of A_j that $c < [\![f_j(e_j)]\!]_{\mathcal{C}_j}$.
- $f_j(e_j) < c$. Recall that $[\![f_j(e_j)]\!]_{\mathcal{C}_{j-1}} \in A_j$. Therefore, $[\![f_j(e_j)]\!]_{\mathcal{C}_j} \leq [\![f_j(e_j)]\!]_{\mathcal{C}_{j-1}}$. Also, note that $[\![f_j(e_j)]\!]_{\mathcal{C}_{j-1}} < c$ as, by the induction hypothesis, $\mathcal{C}_{j-1}$ satisfies all formulas in σ. Therefore, $[\![f_j(e_j)]\!]_{\mathcal{C}_j} < c$.
- $f_j(e_j) < f'(e')$. First, recall that $[\![f_j(e_j)]\!]_{\mathcal{C}_j} \leq [\![f_j(e_j)]\!]_{\mathcal{C}_{j-1}}$. Second, note that $[\![f_j(e_j)]\!]_{\mathcal{C}_{j-1}} < [\![f'(e')]\!]_{\mathcal{C}_{j-1}}$ as, by the induction hypothesis, $\mathcal{C}_{j-1}$ satisfies all formulas in σ. Therefore, $[\![f_j(e_j)]\!]_{\mathcal{C}_j} < [\![f'(e')]\!]_{\mathcal{C}_{j-1}}$.
- $f_j(e_j) \in F(e)$ and $f_j(e_j) \notin F(e)$. This kind of formulas do not belong to $\sigma \setminus \sigma_S$.
- $f(e) \in C$. Since $[\![f_j(e_j)]\!]_{\mathcal{C}_j} \in A_j$, we have by Item 3 of the definition of A_j that $[\![f_j(e_j)]\!]_{\mathcal{C}_j} \in C$.

■

Once $\mathcal{C}_m$ is built, $[\![f_i(e_i)]\!]_{\mathcal{C}_m} \leq d + m \leq d + n$, for $i \leq m$. Recall that d is the largest constant occurring in Φ. Therefore, the size of $[\![f_i(e_i)]\!]_{\mathcal{C}_m}$, for $i \leq m$, is at most $\log(d + n)$.

We address now the problem that for a set-valued term of the form $F(e)$, the size of $[\![F(e)]\!]_{\mathcal{C}_m}$ might be very large. We define a final certificate $\overline{\mathcal{C}}$ where $[\![f(e)]\!]_{\overline{\mathcal{C}}} = [\![f(e)]\!]_{\mathcal{C}_m}$ and $[\![F(e)]\!]_{\overline{\mathcal{C}}}$ is the least set that satisfies the following:

1) it contains the union of all sets of constants C such that $C \subseteq^*_\sigma F(e)$ and
2) it contains every $[\![f(e)]\!]_{\overline{\mathcal{C}}}$ such that $f(e) \in F'(e')$ for some set-valued term $F'(e')$ and $F'(e') \subseteq^*_\sigma F(e)$.

Note that there is at least one set that satisfies this: $[\![F(e)]\!]_{\mathcal{C}_m}$. Therefore, $[\![F(e)]\!]_{\overline{\mathcal{C}}} \subseteq [\![F(e)]\!]_{\mathcal{C}_m}$.

Lemma 17: If $\mathcal{C}_m$ satisfies all formulas in $\sigma \setminus \sigma_S$, then $\overline{\mathcal{C}}$ satisfies all formulas in σ.

Proof. Suppose that $\mathcal{C}_m$ satisfies all formulas in σ. Note that $\mathcal{C}_m$ and $\overline{\mathcal{C}}$ agree on the interpretation of single-valued terms. Therefore, it suffices to show that $\overline{\mathcal{C}}$ satisfies all atomic formulas $\psi \in \sigma$ that involve set-valued terms. We proceed by evaluating the possible cases for ψ:

- $C \subseteq F(e)$. By construction, $C \subseteq [\![F(e)]\!]_{\overline{\mathcal{C}}}$.
- $F(e) \subseteq C$. Recall that $[\![F(e)]\!]_{\overline{\mathcal{C}}} \subseteq [\![F(e)]\!]_{\mathcal{C}_m}$ and that $\mathcal{C}_m$ interprets set-valued terms in the same way $\mathcal{C}$ does. Therefore $[\![F(e)]\!]_{\overline{\mathcal{C}}} \subseteq [\![F(e)]\!]_{\mathcal{C}}$. Since $\mathcal{C}$ satisfies all formulas in σ, we have $[\![F(e)]\!]_{\mathcal{C}} \subseteq C$. Therefore $[\![F(e)]\!]_{\overline{\mathcal{C}}} \subseteq C$.
- $F(e) \subseteq F'(e')$. The same argument holds. Recall that $[\![F(e)]\!]_{\overline{\mathcal{C}}} \subseteq [\![F(e)]\!]_{\mathcal{C}_m}$ and that $\mathcal{C}_m$ interprets set-valued terms in the same way $\mathcal{C}$ does. Therefore $[\![F(e)]\!]_{\overline{\mathcal{C}}} \subseteq [\![F(e)]\!]_{\mathcal{C}}$. Since $\mathcal{C}$ satisfies all formulas in σ, we have $[\![F(e)]\!]_{\mathcal{C}} \subseteq [\![F'(e')]\!]_{\mathcal{C}}$. Therefore $[\![F(e)]\!]_{\overline{\mathcal{C}}} \subseteq [\![F'(e')]\!]_{\mathcal{C}}$.
- $f_j(e_j) \notin F(e)$, with $j \leq m$. By the definition of $[\![F(e)]\!]_{\overline{\mathcal{C}}}$, it suffices to show that
  1) $[\![f_j(e_j)]\!]_{\overline{\mathcal{C}}} \notin C$, for any set of integer constants C such that $C \subseteq^*_\sigma F(e)$. Recall that $[\![f_j(e_j)]\!]_{\overline{\mathcal{C}}} = [\![f_j(e_j)]\!]_{\mathcal{C}_m} = [\![f_j(e_j)]\!]_{\mathcal{C}_j} \in A_j$, which implies, by Item 4 of the definition of A_j, that $[\![f_j(e_j)]\!]_{\overline{\mathcal{C}}} \notin C$.
  2) $[\![f_j(e_j)]\!]_{\overline{\mathcal{C}}} \neq [\![f_\ell(e_\ell)]\!]_{\overline{\mathcal{C}}}$ for any single-valued term $f_\ell(e_\ell)$ such that $[f_\ell(e_\ell) \in F'(e')] \in \sigma$ and $F'(e') \subseteq^*_\sigma F(e)$. We consider two subcases. If $\ell < j$, then since $[\![f_j(e_j)]\!]_{\overline{\mathcal{C}}} = [\![f_j(e_j)]\!]_{\mathcal{C}_j} \in A_j$, $[f_j(e_j) \notin F(e)] \in \sigma$, $[f_\ell(e_\ell) \in F'(e')] \in \sigma$ and $F'(e') \subseteq^*_\sigma F(e)$, by Item 5a of the definition of A_j, we have that $[\![f_j(e_j)]\!]_{\mathcal{C}_j} \neq [\![f_\ell(e_\ell)]\!]_{\mathcal{C}_j}$. If $\ell > j$, then since $[\![f_\ell(e_\ell)]\!]_{\overline{\mathcal{C}}} = [\![f_\ell(e_\ell)]\!]_{\mathcal{C}_\ell} \in A_\ell$, $[f_j(e_j) \notin F(e)] \in \sigma$, $[f_\ell(e_\ell) \in F'(e')] \in \sigma$ and $F'(e') \subseteq^*_\sigma F(e)$, by Item 5b of the definition of A_ℓ, we have that $[\![f_\ell(e_\ell)]\!]_{\mathcal{C}_\ell} \neq [\![f_j(e_j)]\!]_{\mathcal{C}_\ell}$.

  We conclude then that $[\![f_j(e_j)]\!]_{\overline{\mathcal{C}}} \notin [\![F(e)]\!]_{\overline{\mathcal{C}}}$.

■

This concludes the proof of Theorem 15.