Scalable and Precise Automated Analysis of Administrative Temporal Role-Based Access Control

Silvio Ranise
Security & Trust, FBK-Irst, Trento, Italia
ranise@fbk.eu

Anh Truong
DISI, U. of Trento, Italia
tatruong@fbk.eu

Alessandro Armando
DIBRIS, U. of Genova, Italia
armando@fbk.eu

ABSTRACT

Extensions of Role-Based Access Control (RBAC) policies taking into account contextual information (such as time and space) are increasingly being adopted in real-world applications. Their administration is complex since they must satisfy rapidly evolving needs. For this reason, automated techniques to identify unsafe sequences of administrative actions (i.e., actions generating policies by which a user can acquire permissions that may compromise some security goals) are fundamental tools in the administrator’s tool-kit.

In this paper, we propose a precise and scalable automated analysis technique for the safety of administrative temporal RBAC policies. Our approach is to translate safety problems for this kind of policy to (decidable) reachability problems of a certain class of symbolic transition systems. The correctness of the translation allows us to design a precise analysis technique for the safety of administrative RBAC policies with a finite but unknown number of users. For scalability, we present a heuristic that allows us to reduce the set of administrative actions without losing the precision of the analysis. An extensive experimental analysis confirms the scalability and precision of the approach also in comparison with a recent analysis technique developed for the same class of temporal RBAC policies.

Categories and Subject Descriptors
D.2 [Software Engineering]: Software/Program Verification

Keywords
Administrative Access Control; Temporal Role-Based Access Control; Automated Safety Analysis

1. INTRODUCTION

Today, the administration of access control policies is key to the security of many IT systems that need to evolve in rapidly changing environments and dynamically finding the best trade-off among a variety of needs. Permissions to perform administrative actions must be restricted since security officers can only be partially trusted. In fact, some of them may collude to, inadvertently or maliciously, modify the policies so that untrusted users can gather sensitive permissions. Taking into consideration the effect of all possible sequences of administrative actions is a difficult task.

Thus, push-button analysis techniques are needed to identify safety issues, i.e., administrative actions generating policies by which a user can acquire permissions that may compromise some security goals. This is known as the safety problem, which amounts to establish whether there exists a (finite) sequence of administrative actions, selected from a set of available ones, that applied to a given initial policy, yield a policy in which a user gets a certain permission. Several important policy analysis problems can be reduced to safety problems, e.g., deciding if the set of users having a given permission is a sub-set of that having another permission (containment), computing the minimal sets of permissions that a user should have so as to make another policy reachable by means of a finite sequence of administrative actions (weakest preconditions), and others (see, e.g., [9]).

This makes automated techniques capable of solving safety problems even more valuable to understand the subtle interplay among the actions performed by several administrators.

In general, the safety problem is undecidable [12]. A first step towards the development of automated techniques is to identify classes of policies for which the safety problem is decidable. Several such techniques have been proposed for administrative models of Role-Based Access Control (RBAC) policies [22]. This is so because RBAC is one of the most widely adopted access control models in the real world. The reason for this is the fact that the notion of role allows for simplifying policy management by decomposing user-permission assignment into user-role and role-permission assignments. For example, a new employee joining an organization is assigned to a role and this is sufficient for him/her to automatically acquire all the permissions associated to that role. Similarly, when someone is promoted or demoted, it is sufficient to update the roles with which he/she is associated to make available the permissions required by the new position. The role-permission assignment rarely changes since this implies a change in the organization. These observations lead researchers to study safety problems for administrative RBAC (ARBAC) models—the most important one is the URA97 model [21]—in which administrative actions can only update the user-role assignment relation (see, e.g., [15, 24, 13, 5, 9]).
Available analysis techniques for RBAC are not readily applicable to extensions of RBAC in which authorizations depend also on contextual information, such as time, that are widely used in real-world applications. For instance, two temporal extensions of RBAC are reported in [7, 14]. These models impose temporal constraints on roles being enabled or for them to be assigned to users. In these models, the executability of administrative actions is also restricted by temporal constraints. To the best of our knowledge, only two works [16, 26] have proposed techniques for the automatic analysis of safety problems for Administrative Temporal RBAC (ATRBAC) models. In [16], the safety problem for ATRBAC policies is reduced to verification problems of timed automata [2], whose solution is supported by the model checker UPPAAL.\footnote{http://www.uppaal.org} The approach supports the verification of a variety of properties, not only those for safety, but has the drawback of assuming a fixed set of users; every time the set of users changes, the analysis must be re-run. Additionally, the size of the state space to be explored by the model checker grows exponentially in the number of users, making the technique difficult to scale. The approach proposed in [26] amounts to decomposing safety of ATRBAC policies into reachability problems for policies that can be expressed in a model that is (close to) URA97. In this way, existing tools (such as RBAC-PAT [11] or VAC [9]) for the safety of URA97 administrative policies can be re-used. One of the main advantages of [26] is the possibility to leverage recent advances in the analysis of ARBAC policies. The main disadvantage is the state-space explosion since several safety problems for URA97 must be solved and the complexity of many restricted versions of this problem is known to be NP-hard [23]. To cope with this problem, [26] identifies situations under which the generated safety problems for ARBAC policies can be simplified, either by reducing the number of administrative actions or by simplifying their applicability conditions. For example, separate administration is assumed whereby the set of administrative roles are separate from that of regular roles, an administrative action can only be executed by a user with an administrative role and can modify the membership of regular roles only. Such an assumption is known to be unrealistic in many real-world scenarios (see e.g., [24] for a discussion of this and related issues), thereby making the results of the analysis less useful. On the other hand, available implementations of safety analysis techniques for temporal RBAC (TRBAC) policies are dramatically confronted with the state-space explosion problem and performing simplifying assumptions, even at the price of a loss of precision, is somehow necessary for the analysis to scale up. In this paper, we present an automated technique for solving safety problems for ATRBAC policies that goes beyond the limitations of available approaches by reusing the symbolic model checking technique in [19]. Preliminary (Section 2), we discuss and formalize a temporal RBAC model, the administrative actions, and the safety problem along the lines of [26]. We also briefly overview (Section 3) the symbolic model checking technique in [19] that uses a class of first-order formulae, called Bernays-Shönfinkel-Ramsey (BSR) [18], to represent transition systems and compute the fix-point of the set of (backward) reachable states by Satisfiability Modulo Theories (SMT) solvers (see, e.g., [6]).

Our first contribution (Section 4) is a translation from safety problems for ATRBAC policies to reachability problems of BSR transition systems. By re-using the decidability result in [19] for BSR-transition systems and arguing the correctness of the translation, we are able to state the first— to the best of our knowledge—decidability result for the safety of ATRBAC policies (Theorem 2). The result paves the way to a precise automated analysis technique since it assumes neither separate administration nor any other simplifying (unrealistic) assumption. In fact, we solve the safety problem considering a finite but unknown number of users in the TRBAC policies manipulated by the administrative actions. In other words, our technique for safety analysis is capable of certifying safety by taking into account that users may join or leave the organization in which the TRBAC policies are administered since the certificate holds for any (finite) number of users. Similarly, it can discover the number of users required for a certain sequence of administrative actions to turn the initial policy into one violating a security goal. This dramatically enlarges the scope of applicability of the analysis and thus the usefulness of its results.

Although desirable, precision hinders performances. Suitable heuristics must be devised in order to make the analysis scalable. Our second contribution is a heuristics designed as a pre-processing phase to the computation of the set of (backward) reachable states aiming at reducing the number of administrative actions that must be taken into account (Section 5). The heuristics identifies useful actions by transitively propagating dependencies on the roles that are mentioned in a security goal. The check to establish the usefulness of an action is computationally cheap but does not guarantee that an action considered useful will be retained when computing the set of backward reachable states. However, if an action is not useful, it is guaranteed not to contribute to the computation of the set of backward reachable states and can thus be discarded without degrading the precision of the analysis. The third contribution of the paper is an extensive experimental evaluation (Section 5) confirming that our technique is scalable. We also perform a comparison with the implementation of the approach in [26] showing how our approach is both more precise and scalable. In particular, the experiments confirm the crucial role played by the heuristics for identifying useful actions to alleviate the state-space explosion problem.

2. TEMPORAL RBAC POLICIES AND ADMINISTRATION

The idea underlying RBAC [22] is to regulate access by assigning users to roles which, in turn, are granted permissions to perform certain operations. Formally, we assume a (finite) set $U$ of users, a (finite) set $R$ of roles, a (finite) set $P$ of permissions, a user-role assignment relation $UA \subseteq U \times R$, and a permission-role assignment relation $PA \subseteq P \times R$. For simplicity, we ignore the role hierarchy (i.e. a partial order on $R$). A user $u$ is a member of role $r$ when $(u, r) \in UA$. A user $u$ has permission $p$ if there exists a role $r \in R$ such that $(p, r) \in PA$ and $u$ is a member of $r$. A RBAC policy is a tuple $(U, R, P, UA, PA)$.

In many scenarios, authorization conditions depend on contextual information such as the time of the day. For instance, a part time employee of an enterprise should be authorized to access the IT system only during working hours,
e.g., between 8am to 12pm. Before being able to specify authorization conditions that depend on temporal constraints, we need to introduce a model of time. As observed in [26], a simple model is sufficient for expressing time-dependent authorization conditions. In fact, temporal constraints are usually specified by means of intervals periodically repeating time intervals, such as day/night-time (two intervals repeating daily), each hour (twenty-four intervals again repeating daily), or each day (seven intervals repeating weekly).

**A model of time.** Let \( T_{\text{MAX}} \) be a positive integer and \( a \) a non-negative integer such that \( a + 1 \leq T_{\text{MAX}} \). A time slot is a pair \((a, a + 1)\); to ease the readability of the examples in the following of the paper, we will use, e.g., \((8am, 4pm)\), \((4pm, 12am)\), and \((12am, 8am)\) to denote time slots \( (0, 1) \), \((1, 2) \), and \((2, 3) \), respectively. The set of all time slots is 

\[
T S_{T_{\text{MAX}}} = \{(a, a + 1) \mid 0 \leq a < T_{\text{MAX}}\}
\]

We will often write \( TS \) in place of \( T S_{T_{\text{MAX}}} \). A time instant is a non-negative real number. A time instant \( t \) belongs to a time slot \((a, a + 1)\) if \( a \leq (t \mod T_{\text{MAX}}) < a + 1 \) where \( \mod \) is the usual modulo operator, i.e. \( t' = t \mod T_{\text{MAX}} \) if there exists a non-negative integer \( k \) such that \( t = t' + k \cdot T_{\text{MAX}} \).

We are now ready to formalize a simplified version of the Temporal RBAC model along the lines of [26]. The idea is to make RBAC policies depend on periodic constraints based on the notion of time introduced above.

**Temporal RBAC.** From now on, we assume that \( T_{\text{MAX}} \) is given so that the set \( TS \) of all time slots is fixed. TRBAC extends RBAC by adding the role status relation \( RS \subseteq R \times TS \) and replacing the user-role assignment \( UA \) with the temporal user-role assignment relation \( TUA \subseteq U \times R \times TS \). For the sake of simplicity, following [26], we neglect role hierarchies.

A role \( r \) is enabled at time instant \( t \) iff there exists a time slot \( ts \) such that \( t \) belongs to \( ts \) and \((r, ts) \in RS \). A user \( u \) is a member of role \( r \) at time instant \( t \) iff \( r \) is enabled at \( t \) and there exists a time slot \( ts \) such that \( t \) belongs to \( ts \) and \((u, r, ts) \in TUA \). A user \( u \) has permission \( p \) at time instant \( t \) iff \( u \) has role \( r \) at \( t \) and there exists a role \( r' \) such that \((p, r') \in PA \) and \( u \) is a member of \( r' \) at \( t \). (The fact that \( u \) is a member of \( r \) at \( t \) implies that \( r \) is enabled at \( t \).) A TRBAC policy is a tuple \((U, R, P, RS, TUA, PA)\).

Following [26], we extend ARBAC to Administrative TRBAC policies by considering two groups of administrative actions: those that enable or disable a role \( r \) by modifying the time slots associated to \( r \) in the \( RS \) relation and those that change the time slots associated to a user-role pair in the \( TUA \) relation.

**Administrative TRBAC.** A signed role is an expression of the form \( r \) or of the form \( \lnot r \). A condition is a (finite) set of signed roles. A signed role \( \sigma \) in a condition \( C \) is positive (negative, resp.) when there exists a role \( r \) such that \( \sigma = r \) (\( \sigma = \lnot r \), resp.). A schedule is a set of time slots. A time instant \( t \) belongs to schedule \( s \) iff there exists a time slot \( ts \) in \( s \) such that \( t \) belongs to \( ts \).

Let \( C \) be a condition, \( RS \) be a role status relation, and \( TUA \) be a temporal user-role assignment relation. A time slot \( ts \) satisfies \( C \) under \( RS \) (or \( TUA \)) iff \((r, ts) \in RS \) (or \((u, r, ts) \in TUA \)) for each \( r \) in \( C \) and \((u, r, ts) \notin TUA \) for each \( r \) in \( C \); i.e. when the positive roles in \( C \) are enabled in \( ts \) and the negative roles in \( C \) are not enabled in \( ts \).

We illustrate the definitions with an example inspired to the hospital scenario in [26].

**Example 1** Let \( U = \{u_1, u_2, u_3\} \), \( R = \{\text{EMP (Employee)}, \text{DDR (Day Doctor)}, \text{NDR (Night Doctor)}, \text{PRC (Practitioner)}, \text{NRS (Nurse)}, \text{SEC (Secretary)}, \text{CHR (Chairman)}\} \).
\[ TS = \{(8am, 4pm), (4pm, 12am), (12am, 8am)\}, \]

\[
(\text{CHR}, \{t_1\}, \{\text{NDR}\}, \{t_3\}, \text{PRC}) \in \text{can_enable (1)}
\]

\[
(\text{SEC}, \{t_2\}, \{\text{EMP}, \text{NDR}\}, \{t_3\}, \text{NRS}) \in \text{can_disable (2)}
\]

\[
(\text{CHR}, \{t_2\}, \{\text{EMP}, \text{NRS}\}, \{t_3\}, \text{DNR}) \in \text{can_assign (3)}
\]

\[
(\text{CHR}, \{t_3\}, \{\text{EMP}, \text{NRS}\}, \{t_3\}, \text{NDR}) \in \text{can_assign (4)}
\]

\[
(\text{CHR}, \{t_3\}, \emptyset, \{t_2\}, \text{SEC}) \in \text{can_revoke (5)}
\]

where \(t_1\) stands for \((8am, 4pm)\), \(t_2\) for \((4pm, 12am)\), and \(t_3\) for \((12am, 8am)\). Assume that the initial state is \(\alpha_0 = (RS_0, TUA_0, 8am)\) where

\[
RS_0 = \{(\text{CHR}, t_1), (\text{CHR}, t_3), (\text{DNR}, t_2), (\text{NDR}, t_3)\}
\]

\[
TUA_0 = \{(u_1, \text{CHR}, t_1), (u_1, \text{CHR}, t_3), (u_2, \text{EMP}, t_2)\}
\]

Let us consider actions (1) and (3). It is easy to check that: (i) schedule \(s_{\text{rule}} = \{t_1\} \) satisfies condition \(C_a = \{\text{CHR}\} \) under \(RS_0\) (e.g., \((\text{CHR}, t_1) \in RS_0\)); (ii) user \(u_1\) and \(s_{\text{rule}}\) satisfy \(C_a\) under \(TUA_0\) since \((u_1, \text{CHR}, t_1) \in TUA_0\); (iii) user \(u_1\) and \(s_{\text{rule}}\) satisfy \(C_a\) at 9am (under \(RS_0\) and \(TUA_0\)) because 9am belongs to \(s_{\text{rule}}\) and (i) and (ii); (iv) schedule \(s_1 = \{t_1\} \) satisfies condition \(C_1 = \{\text{NDR}\}\) under \(RS_0\) (e.g., \((\text{NDR}, t_3) \in RS_0\)) and \(u_1\) user \(u_2\) together with \(s_2 = \{t_2\}\) satisfies \(C_2 = \{\text{EMP}, \text{NRS}\}\) since \((u_2, \text{EMP}, t_2) \in TUA_0\) and \((u_2, \text{NRS}, t_3) \notin TUA_0\).

Action (1) \((3, \emptyset)\) resp. can be executed at 9 am w.r.t. schedule \(s_1\) \((u_2\) and schedule \(s_2\) resp.) because of (iii) and (iv) \(\emptyset\)-\(\emptyset\) resp. The effect of executing action (1) on the initial state is that \((RS_0, TUA_0, 9am) \rightarrow \psi (RS', TUA', 9am)\) where \(RS' = RS_0 \cup \{(\text{PRC}, t_3)\}\) and \(TUA' = TUA_0\). Similarly, we also have the relation \((RS_0, TUA_0, 9am) \rightarrow \psi (RS'', TUA'', 9am)\) where \(RS'' = RS_0\) and \(TUA'' = TUA_0 \cup \{(u_2, \text{DNR}, t_2)\}\) by executing (3).

Administrative actions do not modify the current time of the state of the ATRBAC system, i.e. administrative actions are assumed to occur instantaneously. To model the passing of time, we extend the definition above by adding the following clause: \(\alpha, t \rightarrow \psi (\alpha, t')\) if \(t' > t\) for any TRBAC policy \(\alpha\) and any set \(\psi\) of administrative actions. Notice that the passing of time does not modify the TRBAC policy in the state of the ATRBAC system.

A run of an ATRBAC system \((\alpha_0, \psi)\) is a \((\text{possibly infinite})\) sequence \((\alpha_0, 0), \ldots, (\alpha_i, t_i), (\alpha_{i+1}, t_{i+1}), \ldots\) of states such that \((\alpha_i, t_i) \rightarrow \psi (\alpha_{i+1}, t_{i+1})\) and \(t_i \leq t_{i+1}\) for \(i = 1, \ldots, n-1\) with \(n > 1\). If the run is finite, i.e. it is of the form \((\alpha_0, 0), \ldots, (\alpha_n, t_n)\) for some \(n \geq 0\), we say that \((\alpha_0, \alpha_n)\) is the final state of the run.

Despite the fact that administrators can only execute a given set of administrative actions, it is still quite difficult to foresee all possible interleavings of actions that a group of administrators can perform together with their effect on an initial TRBAC policy. As a consequence, it may be the case that an untrusted user can acquire, in some time interval, a permission that he/she should not acquire. Being able to identify this situation amounts to solving the following analysis problem.

A reachability problem for an ATRBAC system \((\alpha_0, \psi)\) is identified by a tuple \((u, C_f, s_f)\) and amounts to checking if there exists a finite run of the ATRBAC system whose final state \((\alpha_f, t_f)\) is such that user \(u\) and schedule \(s_f\) satisfy condition \(C_f\) under \(TUA_f\) and \(s_f\) satisfies \(C_f\) under \(RS_f\), where \(\alpha_f = (RS_f, TUA_f)\).

Our definitions of administrative action and reachability problem generalize those in [26], which assumes \(C_a = \{r_a\}\) in \(\tau = (C_a, s_{\text{rule}}, C_s, r)\) and \(C = \{r\}\) in \((u, C, s)\). In general, there is no efficient reduction from our version of the reachability problem to that in [26]. This is so because \(C_a\) can be any \((\text{finite})\) sub-set of the set \(\bigcup (\{r | r \in R\})\) of \((\text{signed})\) roles; thus, there are at most \(2^{|R|}\) of such conditions. To encode this by a singleton \(C_a\) as in [26], we must introduce a "fresh" role for each such condition, i.e. the number of new roles is in \(O(2^{|R|})\).

Finally, we observe that [26] introduces a timed version of the reachability problem in which the final state should be reached within a given time instant. For lack of space, we do not consider this problem here but we claim that our approach can be easily extended to handle also this problem.

### 3. REACHABILITY OF A CLASS OF SYMBOLIC TRANSITION SYSTEMS

Following [19], we reduce reachability problems for ATRBAC systems to a \((\text{finite})\) sequence of constraint satisfaction problems. Here, we briefly recall the main notions underlying the approach in [19].

A well-known constraint satisfaction problem is Boolean satisfiability. It consists of establishing whether a formula—obtained by combining Boolean variables with logical connectives—can be made true (equivalently, is satisfiable) by assigning appropriate values to its Boolean variables. Our approach reduces reachability problems for ATRBAC systems to sequences of constraint satisfaction problems that are more easily and compactly described by using a richer language, called Bernays-Schönfinkel-Ramsey (BSR) formulæ (see, e.g., [19]). The constraint satisfaction problem for BSR formulæ is to determine the satisfiability of formulæ of the form \(\exists \bar{x} \varphi(\bar{x}, \bar{y})\), where \(\varphi\) is a quantifier-free formula, i.e. a Boolean combination of atomic sub-formulæ built out of equality, predicates, constants (functions are not allowed), and variables in the tuples \(\bar{x}, \bar{y}\). When \(\varphi(\bar{x}, \bar{y})\) is empty and \(\varphi(\bar{x}, \bar{y})\) is not, the BSR formula is universal (existential, resp.). When both \(\varphi(\bar{x}, \bar{y})\) and \(\varphi(\bar{x}, \bar{y})\) are empty, the BSR formula is quantifier-free.

As shown in [17], BSR formulæ can be used in many verification scenarios. For instance, the following BSR formulæ

\[
e_i \neq e_j \text{ for distinct } i, j \in \{1, \ldots, n\}
\]

\[
\forall x. (x = e_1 \lor \cdots \lor x = e_n)
\]

characterize an enumerated data-type with \(n\) elements. The formulæ in (6) constrain the constants \(e_1, \ldots, e_n\) to be pairwise distinct \((e_i \neq e_j\) abbreviates \(\neg(e_i = e_j))\) while (7) considers at most \(n\) distinct elements.

The satisfiability problem for BSR formulæ can be reduced to Boolean satisfiability by an instantiation procedure [18]. Unfortunately, this process may yield a Boolean formula that is exponentially larger than the original. (The satisfiability problem for BSR formulæ is known to be NEXPTIME complete.) To alleviate this problem, alternative approaches (see, e.g., [17]) have been proposed that avoid the up-front reduction by reasoning on the extended (with respect to Boolean logic) language. These techniques have been implemented in Satisfiability Modulo Theories (SMT) solvers, that are extensions of Boolean solvers capable of establishing the satisfiability of formulæ in decidable fragments of first-order logic (e.g., the BSR fragment) and de-
SMT solvers are receiving a lot of attention because of their effectiveness in solving SMT problems derived from several application areas, such as hardware verification, program analysis, and scheduling; see, e.g., [6] for an introduction to SMT solving and its applications. In the following, we show how SMT solvers can be used to support the reachability analysis of a class of symbolic transition systems.

An adequate BSR symbolic transition system (adequate BSR-STS, for short) is a tuple \((s, Ax, In, Tr)\), where \(s\) is a (finite) sequence of predicates, called the state variables, \(Ax\) is a (finite) set of BSR formulae, called axioms, \(In(s)\) is a universal BSR formula,\(^2\) called the initial state formula, and \(Tr\) is a (finite) disjunction of BSR formulae of the form

\[
\exists \bar{x}. \left( G(s) \land \bigwedge_{x \in \bar{x}} \forall y. \left( s'(y) \Leftrightarrow U_s(s, y) \right) \right),
\]

called transition formulae, where \(s'\) is the sequence obtained from \(s\) by priming each element, \(x\) and \(y\) are tuples of variables, \(G(s)\) is a quantifier-free BSR formula—called the guard, containing the variables in \(\bar{x}\) as free variables, and where each occurrence of the predicate symbols in \(s\) are applied to variables in \(\bar{x}\), and \(U_s(s, y)\) is a quantifier-free BSR formula—called the update, containing the variables in \(x, y\) as free variables, the length of \(y\) matches the arity of \(s\), and each occurrence of the predicate symbols in \(s\) is applied to variables in \(x, y\).

The reachability problem for an adequate BSR-STS \((s, Ax, In, Tr)\) and an existential BSR formula \(\gamma(s)\), called the goal, consists of establishing whether there exists an integer \(n \geq 0\) such that

\[
In(s_0) \land \tau(s_0, s_1) \land \cdots \land \tau(s_{n-1}, s_n) \land \gamma(s_n) \land Ax
\]

is satisfiable where \(\tau(s, s') := \bigvee_{i \in \mathbb{N}} \text{tr}(s, s')\) and the sequence \(s_i\) is obtained from \(s\) by uniquely renaming each of its element by appending the suffix \(i\) (for \(i = 0, \ldots, n\)). For clarity, if \(\phi(s)\) is a formula containing symbols from \(s\), then \(\phi(s_i)\) is the formula obtained from \(\phi\) by pairwise replacement of each element in \(s\) with the corresponding one in \(s_i\).

A monadic BSR-STS is an adequate BSR-STS whose predicates are unary.

**Theorem 1 ([19])** The reachability problem for monadic BSR-STSs is decidable.

The proof of this result amounts to proving the termination of a symbolic backward reachability procedure. The idea is to find the value of \(n\) for which the formula (9) is satisfiable (if possible) by computing increasingly precise under-approximations \(R_0(s), \ldots, R_n(s)\) of the formula \(R(s)\) representing the set of states from which it is possible to reach a state of the goal \(\gamma(s)\) by applying \(0, \ldots, n\) times, respectively, the transition \(Tr(s, s')\). In order to stop computing formulæ in the sequence \(R_0(s), \ldots, R_n(s)\), there are two criteria. First, we can check whether \(R_n(s) \wedge In(s) \wedge Ax\) is satisfiable: in this case, the sequence \(R_0(s), \ldots, R_n(s)\) has reached a fix-point at \(n\). SMT solvers can be used to automatically solve the SMT problems underlying these two criteria. The tool described in Section 5 implements the procedure above and use an SMT solver to tackle the SMT problems encoding the two termination criteria.

To simplify the technical development, we consider a simple extension of the notion of monadic BSR-STSs. Preliminarily, we introduce the many-sorted version of BSR formulæ: each symbol is associated with sorts denoting the sets of values over which the arguments of the symbol range. It is well-known that many-sorted first-order logic is as expressive as first-order logic without sorts (see, e.g., [8]). Thus, all the results above carry over to sorted BSR formulæ. Notationally, if \(S\) is a sort symbol, then \(\text{Enum}(\{e_1, \ldots, e_n\}, S)\) stands for the set of formulæ (6) and (7) above with \(x, e_1, \ldots, e_n\) of the enumerated sort \(S\). An effectively monadic BSR-STS is an adequate BSR-STS whose predicate symbols are \(n\)-ary for \(n \geq 0\) and such that at least \(n - 1\) arguments range over enumerated datatypes. It is always possible to transform an effectively monadic BSR-STS to a monadic BSR-STS.

**Corollary 1** The reachability problem for effectively monadic BSR-STSs is decidable.

In [19], this result has been applied to the safety analysis of ARBAC policies. Section 4 will explain how to apply it to the analysis of ATRBC systems. There are two main advantages in developing safety analysis based on Corollary 1. First, the user-role reachability problem is solved with respect to a finite but unknown number of users in the policies manipulated by the administrative actions; see [19] for a detailed discussion on this issue. Thus, when the goal is unreachable, the safety certification takes into account the fact that users may join or leave the organization in which the policies are administered. Instead, when the goal is reachable, the technique is capable of establishing the number of users needed for some sequence of administrative actions to transform the initial policy into one satisfying the goal (usually obtained by negating a security goal). In this way, our safety analysis technique can go beyond the separate administration assumption adopted by many approaches available in the literature, such as [11, 13]. The second advantage of developing a safety analysis on the backward reachability procedure on which Corollary 1 is based is the possibility to integrate heuristics that dramatically reduce the state-space explosion problem as shown in [20] for ARBAC policies. Section 5 will discuss an adaptation of an heuristics in [20] to ATRBC systems whose impact on performances will be illustrated by the experiments of Section 5.1.

### 4. Solving Reachability Problems for ATRBC Systems

Let \(U\) be a set of users, \(R\) of roles, and \(\text{TST}_{\text{MAX}}\) of time slots. We now show how to translate a reachability problem \((u, C, s)\) for an ATRBC system \((u, C, s)\) to a reachability problem of an effectively monadic BSR-STS \((\mathcal{A}_{\text{ATRBC}}, \text{Ax}_{\text{ATRBC}}, \text{In}_{\text{ATRBC}}, \text{Tr}_{\text{ATRBC}})\) as specified in Figure 1. In the following, we first argue its correctness and then use Corollary 1 to derive the decidability of reachability problems for ATRBC systems. Preliminary, we make a simple but important observation on the role of time.

**Abstracting away time.** We observe that, in order to solve reachability problems for ATRBC systems, we do...
not need to keep track of the current time \( t \) in the state \( (RS, TUA,t) \) of \((\alpha_0, \psi)\) but only the time slot \( ts \) to which \( t \) belongs to. In fact, for any time instants \( t, t' \) belonging to the same time slot \( ts \), we can easily show that

- a role enabling/disabling action \((C_a, s_{rule}, C, s_r, r)\) can be executed at \( t \) with respect to a schedule \( s_r \) iff \((C_a, s_{rule}, C, s_r, r)\) can be executed at \( t' \) with respect to \( s_r \), and
- an assign/revoke action \((C_a, s_{rule}, C, s_r, r)\) can be executed at \( t \) with respect to a user \( u \) and a schedule \( s_r \) iff \((C_a, s_{rule}, C, s_r, r)\) can be executed at \( t' \) with respect to \( u \) and \( s_r \).

For this reason, without loss of generality, we assume that the state of any ATRBAC system is of the form \((RS, TUA, ts)\) where \( ts \) is a time slot.

We consider many-sorted BSR formulae built out of the following symbols: sorts \( User, Role \); a constant of sort \( User \) for each element in \( U \); a constant of sort \( Role \) for each element in \( R \); and a constant of sort \( TimeSlot \) for each element in \( TS_{MAX} \). We also assume that \( Ax_{ATRBAC} = En(R, Role) \cup En(TS_{MAX}, TimeSlot) \).

**Representing states.** Typically, the relations \( RS_0 \) and \( TUA_0 \) in an initial state \( \alpha_0 \) are finite since only finitely many users, roles, and time slots are considered. Thus, \( \alpha_0 \) can be represented by a universal BSR formula, denoted with \( [\alpha_0] \), as shown in Figure 1. In general, \([\alpha]\) denotes a universal BSR formula representing the state \( \alpha \) of an ATRBAC system.

**Example 2** We recall that the initial state of the system in Example 1 is \( \alpha_0 = (RS_0, TUA_0, 8am) \) and that 8am belongs to time slot \( ts_1 = (8am, 4pm) \). Then, the formula \([\{RS_0, TUA_0, ts_1\}]\) is

\[
\begin{align*}
\forall x, y, z. & \\
rs(y, z) & \iff (y = CHR \land z = ts_1) \lor (y = DDR \land z = ts_2) \lor (y = NDR \land z = ts_3) \lor \\
tua(x, y, z) & \iff (x = A \land y = CHR \land z = ts_3) \lor (x = B \land y = EMP \land z = ts_3) \lor (z = ts_3) \\
at(z) & \iff z = ts_3
\end{align*}
\]

(We recall from Example 1 that \( ts_3 \) stands for \((4pm, 12am)\) and \( ts_3 \) for \((12am, 8am)\).) Clearly, this is a universal BSR formula.

\[\square\]
Before describing how administrative actions are translated to transition formulae of the form (8), we observe that it is possible to consider, without loss of generality, actions in which time schedules contain a single time slot, i.e. actions of the form \((C_a, \{ts_{rule}\}, C, \{ts_r\}, r)\), for \(ts_{rule}\) and \(ts_r\) time slots. In fact, an action whose time schedules contain more than one time slot can be easily transformed to a finite set of actions whose time schedules are singleton sets. We illustrate how this can be done on a simple example since the generalization to arbitrary actions is easy. The administrative action \((C_a, \{ts_1, ts_3\}, \{r_1, r_2\}, \{ts_5\}, r)\) can be transformed into the following two actions: \((C_a, \{ts_1\}, \{r_1, r_2\}, \{ts_5\}, r)\) and \((C_a, \{ts_3\}, \{r_1, r_2\}, \{ts_5\}, r)\) for \(ts_i\) a time slot \((i = 1, 3, 5)\). This means that using schedules for the specification of administrative TRBAC actions does not increase expressiveness, it only allows for more compact specifications. For this reason, in the following, we consider administrative actions of the form \((C_a, \{ts_{rule}\}, C, \{ts_r\}, r)\).

**Representing transitions.** Figure 1 shows that an administrative action \((C_a, \{ts_{rule}\}, C, \{ts_r\}, r)\) can be mapped to a transition formula (8). The guard \(G\) is obtained as the conjunction of three formulae: \(G_1\) requires the action to be executed in the time slot \(ts_{rule}\), \(G_a\) that the chosen administrator satisfies the condition \(C_a\), and \(G_t\) that the chosen time slot \(ts_r\) satisfies the condition \(C\) in case of an enabling/disabling action or the selected user with the time slot \(ts_r\) satisfies the condition \(C\) in case of assign/revoke action. The updates of an enabling/disabling action modify the state variable \(rs\) only by adding/deleting the pair \((r, ts_r)\) whereas the updates of an assign/revoke action modify the state variable \(rs\) only by adding/deleting the triple \((u, r, ts_r)\). We explain this more in detail below.

Consider the abbreviations \(ts\text{-sat}_{RS}\) and \(ts\text{-sat}_{TUA}\) at the bottom of Figure 1. Given a state \(\alpha = (RS, TUA, ts)\), it is easy to see that

- \(ts\) satisfies a condition \(C\) under \(RS\) iff
  \[
  [\alpha] \land ts\text{-sat}_{RS}(ts, C) \land Ax_{ATRBAC}\] is satisfiable and
- a user \(u\) and \(ts\) satisfy a condition \(C\) under \(TUA\) iff
  \[
  [\alpha] \land ts\text{-sat}_{TUA}(u, ts, C) \land Ax_{ATRBAC}\] is satisfiable.

These imply that an administrative action \((C_a, \{ts_{rule}\}, C, \{ts_r\}, r)\) can be executed in state \(\alpha\) at any time instant \(t\) of the time slot \(ts_{rule}\) if there exists user \(u_{ad}\) such that

\[
[\alpha] \land ts = ts_{rule} \land ts\text{-sat}_{RS}(ts_{rule}, C_a) \land ts\text{-sat}_{TUA}(u_{ad}, ts_{rule}, C_a) \land Ax_{ATRBAC} \tag{10}
\]

is satisfiable. Notice how—according to Figure 1—(10) can be re-written as

\[
[\alpha] \land G_1 \land G_a \land Ax_{ATRBAC}. \tag{11}
\]

Additionally, we have that a role enabling/disabling action can be executed in \(\alpha\) (w.r.t. \(ts_r\)) iff

\[
(10) \land [\alpha] \land ts\text{-sat}_{RS}(ts_r, C) \tag{11}
\]

is satisfiable and an assign/revoke action can be executed in \(\alpha\) (w.r.t. some user \(u\) and \(ts_r\)) iff

\[
(10) \land [\alpha] \land ts\text{-sat}_{TUA}(u, ts_r, C). \tag{12}
\]

As already observed in Section 2, the administrative actions do not modify time. Correspondingly, the state variable \(at\) is updated identically by the transition formulae in Figure 1 associated to an administrative action \((C_a, \{ts_{rule}\}, C, \{ts_r\}, r)\). To model the passing of time, Figure 1 shows two transition formulae of the form (8) that update identically the state variables \(rs\) and \(ts\) while modifying the value of \(at\) so that it stores time slot \((j + 1)\) after storing time slot \((j + 1)\).

To see why it is sufficient to model the flow of time from time slot \((j, j+1)\) to \((j+1, j+2)\), it is sufficient to observe that the set of enabled administrative actions is fixed for any time instant in a given time slot.

**Example 4 Consider again Example 1 and recall that the set \(TS_{\text{MAX}}\) of time slots is \(\{ts_1, ts_2, ts_3\}\), where \(ts_1\) is (8am,
generates the following instances of (8) to model the passing of time:
\[ \exists t_1, t_2, t_3 \exists z. (z = t_1) \land ID(rs, tua) \]
\[ \exists t_1, t_2, t_3 \exists z. (z = t_2) \land ID(rs, tua) \]
\[ \exists t_1, t_2, t_3 \exists z. (z = t_3) \land ID(rs, tua) \]

where \( ID(rs, tua) \) is the conjunction of \( \forall y, z. rs(y, z) \land tua(w, y, z) \). The first formula above formalizes the passing of time from a time instant in \( ts_1 \) to one in \( ts_2 \), and the second from an instant in \( ts_2 \) to one in \( ts_3 \), and the last formula from an instant in \( ts_3 \) to one in \( ts_1 \) after \( T_{MAX} \) units of time have elapsed.

Reachability problems. The translation in Figure 1 generates an adequate BSR-STS: the axioms are BSR formulae, the initial formula is a universal BSR formula, and the transitive formulae are all of the form (8). It is not difficult to see that it is effectively monadic since the arguments of both \( rs \) and \( at \) range over sorts whose cardinality is bounded while \( tua \) has two, out of three, arguments that range over sorts whose cardinality is bounded. In fact, only the first argument of \( tua \) may assume an unknown number of values since the interpretation of User is left unconstrained by the axioms in \( A\_ATRBAC \). In terms of the original model, this allows our safety analysis technique to consider a finite but unknown number of users in the temporal RBAC policies manipulated by the administrative actions.

To be able to apply Corollary 1, we are left with the problem of checking that the goal formula obtained by applying the translation described in Figure 1 to the reachability problem \((u, C_f, s_f)\) is an existential BSR formula. Indeed, this is the case as a simple inspection of the penultimate row of the figure reveals.

**Theorem 2** The reachability problem \((u, C_f, s_f)\) for ATRBAC systems is decidable (even when considering a finite but unknown number of users).

At a closer look, it is possible to solve a generalized reachability problem \((C_f, s_f)\) that requires to establish whether there exists a user \( u \) (we are not interested in who is the one, a priori) such that \( u \) and \( s_f \) satisfy \( C_f \). To see why this is the case, it is sufficient to observe that the translation of \((C_f, s_f)\) is the existential BSR formula
\[ \exists x.ts-sat_{TUA}(x, t_f, C_f) \land ts-sat_{BS}(t_f, C_f) \]

obtained from that in Figure 1 by simply dropping the conjunction \( x = u \). This version of the problem is useful to establish—among other things—availability, i.e., ensuring that an authorized user is able to acquire a certain set of permissions (via a given set of roles).

**Theorem 3** The generalized reachability problem \((C_f, s_f)\) for ATRBAC systems is decidable (even when considering a finite but unknown number of users).

Both Theorems 2 and 3 are immediate consequences of Corollary 1 since the translation in Figure 1 yields BSR-STSs. Since the proof of Theorem 1 (and thus also Corollary 1) is constructive by showing the backward reachability procedure sketched in Section 3 terminates on every BSR-STSs, we design the following two-phase technique for the solution of a reachability problem for an ATRBAC system: first use the translation in Figure 1 and then run the backward reachability procedure in Section 3 on the resulting BSR-STSs. Since the interpretation of User is left unconstrained by the axioms in \( A\_ATRBAC \), the backward reachability procedure supports the solution of reachability problems for a finite but unknown number of users. As it is common for model checking procedures, backward reachability suffers from the state-space explosion problem. In Section 5, we discuss a general heuristics to alleviate this problem and refine the two-phase technique by adding a pre-processing phase whose goal is to reduce the number of administrative actions to consider while not degrading the precision of the analysis. We also illustrate the impact of the heuristics on the performance of an implementation of our approach in Section 5.1.

## 5. Technique and Experiments

Our safety analysis technique is depicted in Figure 2. It takes as input a reachability problem \((u, C, s)\) for an ATRBAC system \((\alpha, \psi)\) and returns either ‘reachable’ or ‘unreachable’. When returning ‘reachable’, it outputs a finite sequence of actions in \( \psi \) that leads the ATRBAC system from \( \alpha \) to a state in which \((u, C, s)\) is satisfied.

The modules ‘Translator’ and ‘BR’ in Figure 2 correspond to the translation in Figure 1 and the backward reachability procedure sketched in Section 3, respectively. The module ‘Action Filter’ in Figure 2 corresponds to a heuristics aiming to identify a (hopefully small) sub-set \( \psi' \) of the administrative actions in \( \psi \), that is sufficient to show whether \((u, C, s)\) is reachable or not. The intuition underlying the heuristics is to identify—in a computationally cheap way—the actions in \( \psi' \) that are likely to be applied in successive iterations of the backward reachability procedure, i.e, first those giving the set \( S_1 \) of states from which it is possible to reach the goal in one step, then those giving set \( S_2 \) of states from which a state of \( S_1 \) can be reached in one step, and so on. This is formalized by the notion of useful action. Let \((u, C, s)\) be a reachability problem for an ATRBAC system \( \psi \), an administrative action \( \tau = (C_u, \{t_{\text{rule}}\}, C_r, \{t_{\text{r}}\}, r) \) in \( \psi \) is useful iff there exists a non-negative integer \( k \) such that \( \tau \) is \( k \)-useful, where \( k \) is

- 0-useful iff \( r \in C \) and \( t_s \) is in \( s \) or
- \( k \)-useful with \( k > 0 \) iff it is \((k-1)\)-useful or there exists a \((k-1)\)-useful action \((C_u', \{t_{\text{rule}}'\}, C_r', \{t_{\text{r}}'\}, r') \) such that \( r' \) occurs (possibly negated) either in \( C_r \) and \( t_s' = t_s \), or in \( C_u' \) and \( t_{\text{rule}}' = t_{\text{r}} \).

Since there are finitely many administrative actions in \( \psi \), there must exist a non-negative integer \( k^* \) such that an administrative action in \( \psi \) is useful iff it is \( k^* \)-useful. The ‘Action Filter’ in Figure 2 computes \( k^* \) by first identifying
the set of all 0-useful actions, then the set of all 1-useful actions, and so on until the set of \( (k^* + 1) \)-useful actions is the same of that of \( k^* \)-useful actions, i.e. the fix-point \( \psi' \) has been obtained.

**Property 1** Let \((u, C, s)\) be a reachability problem for an ATRBAC system \((\alpha, \psi)\) and \(\psi'\) be the set of useful actions computed by the ‘Action Filter’ in Figure 2. Then, \((u, C, s)\) is reachable by using the actions in \(\psi\) iff it is so by using the actions in \(\psi'\).

This heuristics is an extension of the one designed for ARBAC policies in [20]. The proof of Property 1 is an adaptation of that in [20] of a similar result for ARBAC policies; it is omitted for space constraints. The correctness of the translation (discussed in Section 4) and Property 1 imply that \((u, C, s)\) is reachable by the ATRBAC system \((\alpha, \psi)\) iff the technique in Figure 2 returns ‘reachable’.

The heuristics for useful actions and the related discussion extends straightforwardly to the case in which the input to our technique is a generalized reachability problem \((C, s)\), as defined in Section 4. To see why this is the case, it is sufficient to observe that the definition of useful action does not depend on the user \(u\) in the reachability problem \((u, C, s)\).

### 5.1 Implementation and Evaluation

We have implemented the technique of Figure 2 in a tool called asasptime; the module ‘BR’ re-uses (off-the-shelf) the SMT-based model checker mcmt [10] and the modules ‘Action Filter’ and ‘Translator’ have been implemented in Python.

**Description of the benchmark problems.** We consider three benchmark classes—identified in the following by \((a), (b), \) and \((c)\)—comprising synthetic reachability problems: \((a)\) and \((b)\) assume separate administration whereas \((c)\) does not. Benchmark class \((a)\) contains the problems used in [26] whereas benchmark classes \((b)\) and \((c)\) contain reachability problems for ATRBAC systems that we have created by “temporalizing” (i.e. randomly adding the rule and role schedule to administrative actions) of the ARBAC policies in [24] and [9], respectively. Although our technique is capable of handling an arbitrary reachability problem \((u, C_f, s_f)\), the three benchmark classes contain problems in which both \(C_f\) and \(s_f\) are singletons. The reason for this choice are two-fold. First, it simplifies the comparison with the tool described in [26], denoted in the following with Tred, whereby reachability problems with only singleton condition and schedule set are considered. Second, observe that it is possible to reduce a reachability problem \((u, C_f, s_f)\) with non-singleton set \(s_f\) of schedules to the set of problems \(\{(u, C_f, \{t \}}\mid t \in s_f\}. When the condition \(C_f = \{\sigma\}\) is a singleton set for some signed role \(\sigma\), this reduction enables our analysis technique to report whether \(\sigma\) can (or cannot, according to its sign) be assigned to the user \(u\) at each time slot in \(s_f\). This is similar to what is done in one of the variants (called the “role schedule approach”) of the technique in [26] and provides more detailed information on a certain reachability problem than simply reporting that the user \(u\) can (or cannot) be assigned to \(\sigma\) in some of the time slots in \(s_f\). Indeed, the price to pay for the higher precision of the analysis is a linear increase—in the number of time slots in \(s_f\)—of the reachability problems to be solved. Also in this case, the behavior of asasptime on the problems in benchmark class \((a)\) is similar to those in Figure 3 thanks to the heuristics based on useful actions. When the condition \(C_f\) is not a singleton set, the situation is more complex since we need to establish if it is possible to reach a TRBAC policy in which the user \(u\) can (or cannot) be assigned to each (signed) role of \(C_f\). This situation is already computationally more expensive for ARBAC policies as observed in [1, 19]. To cope with this, we believe it is possible to adapt the sub-goaling technique in [1] for ARBAC policies to the ATRBAC systems considered in this paper. We leave this and the related experimental evaluation to future work. Here, instead, we prefer to study the behavior of our technique for different values of the parameters characterizing the size of (the set of) administrative actions. The values of such parameters for the three benchmark classes are shown in Table 1. The first column lists the parameters of an ATRBAC system \((\alpha_0, \psi)\) for a reachability problem in any one of the three benchmark classes: \(Ro\) is the number of roles in the TRBAC policy \(\alpha_0\), \(Ru\) is the number of administrative actions in \(\psi\), \(TS\) is the number of time slots of the ATRBAC system \((\alpha_0, \psi)\), \(AC\) is the cardinality of \(C_0\) and \(C\) is the cardinality of \(C\) for an administrative action \((C_a, s_{rule}, C, s_r)\) in \(\psi\). The remaining three columns of Table 1, each one corresponding to a benchmark class, contains the value \(Z\) or the range \([X..Y]\) of values that a parameter \(p\) can take.

**Experimental evaluation.** All the experiments were performed on an Intel QuadCore (3.6 GHz) CPU with 16 GB Ram running Ubuntu 11.10. Benchmark class \((a)\) contains instances of the reachability in which the role status and the timed user-assignment relations in the initial TRBAC policy are empty and the administrative actions are randomly generated under separate administration [26]. Because of this assumption, we can safely ignore the condition \(C_0\) in each administrative action \((C_a, s_{rule}, C, s_r)\), i.e. \(AC = 0\) in Table 1, as there exists an administrator capable of executing any action anytime. The goal of these reachability problems is to understand how performances vary for increasing values (namely, 100, 300, 500, 700, and 1,000) of one of the three parameters \(Ro\), \(Ru\), and \(TS\) and leaving the other two fixed values (both at 200). For increasing values of each parameter, 15 instances are considered and the average of the times taken to solve each problem is shown in the three plots of Figure 3. Blue squares with solid lines identifies the performances of asasptime. Dark blue diamonds with dashed lines and dark blue triangles with dash-dotted lines are the performances of two versions of Tred, resulting from two strategies with which it reduces a reachability problem for an ATRBAC system to several instances of the reachability problem for (variants of) ARBAC policies and then re-use (adapt) existing tools for ARBAC policies.
analysis. We denote by TRED_RULE and TRED_ROLE the two versions of TRED that work by splitting the reachability problem according to $s_{rule}$ and $s_r$, respectively, in each administrative action ($C_a, s_{rule}, C, s_r, r$). For details on the techniques underlying TRED, the reader is pointed to [26].

The plots of Figure 3 shows that the growth of ASASP-TIME performance closely follows that of the best version of TRED: it is linear when the number of roles or of time slots increases and it is linear (not exponential as it is the case for TRED, according to what has been already observed in [26]) when the number of administrative actions increases. The heuristic based on useful actions described above is key to this result because it helps in greatly reducing (even of 99%) the number of administrative actions. We observe that both tools find the problems in this class rather easy, as all timings are under 2 seconds.

Benchmark class (b) contains problems obtained by temporally splitting the ARBAC policies in [24] under separate administration ($AC = 0$ in Table 1). For the values of the parameters Ro and Ru fixed by the underlying ARBAC policies, we generate an ATRBAC system by considering a value for $TS$ from the set \{5, 10, 20, 30, 40, 60, 80, 100\}. The cluster bars plot in Figure 4 shows the performances of TRED_ROLE (black bar) and ASASP-TIME (white bar) on this benchmark class: the y-axis shows average time (in seconds) taken by the tools to solve 15 user-role reachability problems obtained by randomly generated goals for each ATRBAC system considered above while the x-axis reports the tuples of values $\{Ro, Ru, TS\}$ for $Us\$ the number of useful actions computed by the heuristics illustrated above. (Notice how more than one values of the parameters Ro, Ru, and TS may vary from one data point to the next one on the x-axis.) Although the problems in this benchmark class are more difficult than those in benchmark class (a)—compare the values on the y-axis of the plots in Figure 3 with those in the plot of Figure 4—clearly, ASASP-TIME performs better than TRED_ROLE and shows a linear behavior for increasing values of the parameter TS instead of the exponential behavior of TRED_ROLE. (We do not report the results of TRED_RULE because it seg-faults for all but one problem in class (b); we have informed the authors of [26] about this problem and they are currently fixing it.) The heuristic based on useful actions is—as before—the key to explain the superior performances of ASASP-TIME over TRED—compare the value of Ru with that of Us for each point on the x-axis: the alleviation of the state-space explosion problem is quite substantial.

Benchmark class (c) contains problems obtained by temporally splitting the ARBAC policies in [9] without assuming separate administration ($AC = 1$ in Table 1). The problems are generated in the same way of those in benchmark class (b). The average time taken by ASASP-TIME to solve the 15 problem instances per generated ATRBAC system are shown in Table 2. It is not possible to perform a comparison with TRED on this benchmark class since it does not fully support non-separate administration [25]. This is only an implementation issue, not a theoretical limitation of the approach in [26]. A first step in the direction of tackling reachability problems without separate administration is reported in [26] by considering so-called “multi-user reachability problems.” These seem to be more general than problems under separate administration but not yet fully beyond it. We conclude by observing the importance of the heuristics based on useful actions for the scalability of ASASP-TIME also on benchmark class (c)—compare the value of Ru with that in the column ‘Useful actions:’ as for the previous two benchmark classes, the alleviation of the state-space explosion problem is substantial.

6. CONCLUSIONS

We have described a precise and scalable automated analysis technique for solving reachability problems of ATRBAC systems. The approach amounts to translating safety problems to (decidable) reachability problems of BSR-STSs [19] and leveraging an existing symbolic model checking tool, McMT [10], to solve them. We have also argued the correct-
ness of the translation and shown that the analysis technique built on top of it can solve reachability problems for ATRBAC systems with a finite but unknown number of users. In this way, our technique is capable of certifying safety by taking into account that users may join or leave the organization in which the TRBAC policies are administered since the certificate holds for any (finite) number of users. Similarly, it can discover the number of users required for a certain sequence of administrative actions to turn the initial policy into one violating a security goal. This dramatically enlarges the scope of applicability of the analysis and thus the usefulness of its results. For example, it allows us to go beyond unrealistic assumptions, such as separate-administration. To the best of our knowledge, it is the first time that the decidability of the reachability problem for this class of ATRBAC systems is proved.

Concerning complexity, reachability problems of ATRBAC systems are PSPACE-hard. In fact, it is easy to reduce user-role reachability problems of administrative URA97 policies—which are in PSPACE [23]—to ATRBAC reachability problems, by ensuring that the temporal constraints are always satisfied. A more exhaustive characterization of the complexity of the general reachability problem for ATRBAC systems and its restrictions (e.g., under separate administration) is left as future work. In this paper, we have presented an extensive experimental analysis of the performances of our approach, that confirms its scalability and better behavior in comparison with the recent approach in [26]. Key to this is the heuristics (based on useful actions) that permits a substantial reduction of the set of administrative actions to be considered during model checking without losing precision. Since our heuristics is applied as a pre-processing step, we believe that it can be used also by other approaches to solve reachability problems for ATRBAC systems.

Another interesting line of research for future work is to study how to incorporate in our approach the notion of temporal role hierarchies [27]. Here, only time-independent role hierarchies have been considered [recall the beginning of Section 2] since they can be pre-processed away [23]. This can no more be done when the hierarchy is time-dependent, rising a new challenge for our safety analysis technique.

As a final remark, we would like to point out the problem of finding adequate benchmark sets that we have faced several times in our efforts for this paper and previous works [3, 1, 5, 4, 20, 19] for the design of safety analysis techniques in access control. As in many other papers in the literature (see, e.g., [24, 13, 9]), the evaluation is based on benchmarks derived from synthetic policies. In many cases, these are generated by identifying a realistic policy (e.g., for a bank or a hospital) together with some parameters that can be increased so that larger and larger instances of the same policy can be generated. Indeed, the goal is to evaluate the scalability of the proposed techniques. Unfortunately, the significance of the experimental results obtained in this way is debatable. The results reported in this paper suffer from the same problem. We believe that a community effort is needed to build up a common database of benchmarks, derived from real-world policies, that can be used to evaluate and compare old and new analysis techniques. Similar initiatives in other fields (e.g., SAT/SMT solving, Planning and Verification) have greatly contributed to their advance. We believe this is a great opportunity also for increasing the impact of the safety analysis of access control policies in security. We hope that these remarks will stimulate further discussion and work in the community.

### 7. REFERENCES


---

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Ro/Ru/TS</th>
<th>ASAP TIME</th>
<th>Verification time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hospital 1</td>
<td>15/127/5</td>
<td>6</td>
<td>0.45</td>
</tr>
<tr>
<td>Hospital 2</td>
<td>15/149/5</td>
<td>9</td>
<td>0.56</td>
</tr>
<tr>
<td>Hospital 3</td>
<td>15/267/10</td>
<td>8</td>
<td>0.63</td>
</tr>
<tr>
<td>Hospital 4</td>
<td>15/287/10</td>
<td>12</td>
<td>0.89</td>
</tr>
<tr>
<td>Hospital 5</td>
<td>15/365/20</td>
<td>16</td>
<td>1.93</td>
</tr>
<tr>
<td>Hospital 6</td>
<td>15/392/20</td>
<td>14</td>
<td>2.06</td>
</tr>
<tr>
<td>Hospital 7</td>
<td>15/367/30</td>
<td>13</td>
<td>2.15</td>
</tr>
<tr>
<td>Hospital 8</td>
<td>15/552/30</td>
<td>22</td>
<td>6.58</td>
</tr>
<tr>
<td>Hospital 9</td>
<td>15/479/40</td>
<td>17</td>
<td>5.19</td>
</tr>
<tr>
<td>Hospital 10</td>
<td>15/725/40</td>
<td>19</td>
<td>6.81</td>
</tr>
<tr>
<td>University 1</td>
<td>34/220/5</td>
<td>5</td>
<td>0.59</td>
</tr>
<tr>
<td>University 2</td>
<td>34/224/5</td>
<td>8</td>
<td>0.83</td>
</tr>
<tr>
<td>University 3</td>
<td>34/483/10</td>
<td>9</td>
<td>0.95</td>
</tr>
<tr>
<td>University 4</td>
<td>34/491/10</td>
<td>7</td>
<td>1.39</td>
</tr>
<tr>
<td>University 5</td>
<td>34/542/20</td>
<td>13</td>
<td>2.71</td>
</tr>
<tr>
<td>University 6</td>
<td>34/560/20</td>
<td>13</td>
<td>2.21</td>
</tr>
<tr>
<td>University 7</td>
<td>34/783/30</td>
<td>18</td>
<td>6.79</td>
</tr>
<tr>
<td>University 8</td>
<td>34/802/30</td>
<td>17</td>
<td>4.01</td>
</tr>
<tr>
<td>University 9</td>
<td>34/938/40</td>
<td>23</td>
<td>6.96</td>
</tr>
<tr>
<td>University 10</td>
<td>34/994/40</td>
<td>27</td>
<td>9.92</td>
</tr>
</tbody>
</table>


