Mohawk+T: Efficient Analysis of Administrative Temporal Role-Based Access Control (ATRBAC) Policies

Jonathan Shahen
University of Waterloo
jmshahen@uwaterloo.ca

Jianwei Niu
University of Texas at San Antonio
niu@cs.utsa.edu

Mahesh Tripunitara
University of Waterloo
tripunit@uwaterloo.ca

ABSTRACT

Safety analysis is recognized as a fundamental problem in access control. It has been studied for various access control schemes in the literature. Recent work has proposed an administrative model for Temporal Role-Based Access Control (TRBAC) policies called Administrative TRBAC (ATRBAC). We address ATRBAC-safety. We first identify that the problem is **PSPACE**-complete. This is a much tighter identification of the computational complexity of the problem than prior work, which shows only that the problem is decidable. With this result as the basis, we propose an approach that leverages an existing open-source software tool called Mohawk to address ATRBAC-safety. Our approach is to efficiently reduce ATRBAC-safety to ARBAC-safety, and then use Mohawk. We have conducted a thorough empirical assessment. In the course of our assessment, we came up with a "reduction toolkit," which allows us to reduce Mohawk+T input instances to instances that existing tools support. Our results suggest that there are some input classes for which Mohawk+T outperforms existing tools, and others for which existing tools outperform Mohawk+T. The source code for Mohawk+T is available for public download.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection—*Access controls*; D.2.4 [**Software Engineering**]: Software/Program Verification

General Terms

Security, Verification

Keywords

Role-Based Access Control; Administration; Temporal; Safety Analysis

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SACMAT'15, June 01–03, 2015, Vienna, Austria.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
978-1-4503-3556-0/15/06 $15.00 .
DOI: http://dx.doi.org/10.1145/2752952.2752966.

1. INTRODUCTION

Access control deals with whether a principal, such as a user, may exercise a privilege, such as read or write, on a resource, such as a file. It is an important aspect of the security of a system. Whether an attempted access is permitted is customarily specified in a policy.

Effecting and intuiting the consequences of changes to an access control policy is called administration. An aspect of administration is delegation, with which a trusted administrator empowers another principal to change the policy in limited ways. Delegation is used so administrative efficiency can scale with the size of an access control system.

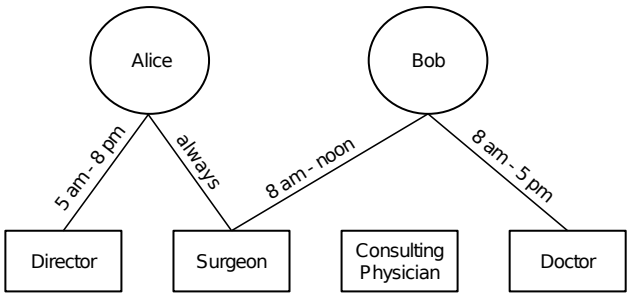
With delegation arises the need for safety analysis, which has been recognized as a fundamental problem in access control since the work of Harrison et al. [3]. The safety analysis problem takes three inputs: (1) a start-state, which is an instance of an access control policy, (2) a state-change rule, which is the set of administrative rules by which a policy can change, and, (3) a query, typically whether a particular user has a particular privilege.

The output is 'true' if the query can never become true, and 'false' otherwise. That is, the system is deemed to be unsafe if there exists a reachable state in which the user indeed has the privilege. The reason is that the user's acquisition of the privilege is presumably undesirable.

Safety analysis has been addressed for various access control schemes in the literature. Our focus is safety analysis in the context of Administrative Temporal Role-Based Access Control (ATRBAC) [12]. ATRBAC is an administrative scheme for Temporal Role-Based Access Control (TRBAC). TRBAC is Role-Based Access Control (RBAC) with temporal-extensions. In RBAC, rather than assigning a user directly to a permission, we adopt the indirection of a role. A user is authorized to a role, which in turn is authorized to a permission. In TRBAC, a user's authorization to a role is tempered by time-intervals during which the authorization is valid.

Furthermore, in TRBAC, a role may be annotated with time-intervals. The role is then said to be active during those time-intervals. A user is able to exercise permissions she acquires via a role when the role is active only.

In Figure 1, we show an example of an TRBAC policy. It can be seen as an example of an RBAC policy by simply ignoring the references to time periods such as "5 am – 8 pm." We show also an instances of administrative rules in ARBAC and ATRBAC. We discuss ATRBAC more precisely in Section 2.1. ATRBAC is an extension of Administrative RBAC (ARBAC) [7]. ARBAC is used to administer RBAC,



ARBAC rules:
can_assign:
⟨Director, Surgeon ∧ ¬Doctor, Consulting Physician⟩
can_revoke:
⟨Director, Doctor⟩

ATRBAC rules:
t_can_assign:
⟨Director, 8 am – 9 am, Surgeon ∧ ¬Doctor,
8 am – noon, Consulting Physician⟩
t_can_revoke:
⟨Director, 8 am – 9 am, true, 8 am – noon, Doctor⟩
t_can_enable:
⟨true, 6 am – 8 am, true, 8 am – 5 pm, Director⟩
t_can_disable is empty

Figure 1: An example of the *TUA* component of a TRBAC policy is to the left. We assume that no role is enabled. Ignoring the time periods on the edges gives us an example of the *UA* component of an RBAC policy. Example ARBAC and ATRBAC administrative rules are to the right. Figures 2 and 3 contain examples of safety queries.

i.e., without any temporal extensions. ATRBAC specifies how two components of a TRBAC access control policy can change: (1) the temporal assignment of a user to a role, and, (2) the temporal activation of a role.

In the example in Figure 1, as the caption for the figure says, with the RBAC policy in the figure as the start-state, the ARBAC safety query, "could Bob become a member of Consulting Physician?" is true. However, the safety query, "could Bob simultaneously be a member of Consulting Physician and Doctor?" is false.

Also, for the example in Figure 1, with the TRBAC policy as the start-state, the ATRBAC-safety query, "could Bob become a member of Consulting Physician between 8 am and noon?" is true. The sequence of actions that must occur for that query to become true, however, is somewhat different from the ARBAC case. In particular, the Director role must first be enabled, as the *t_can_assign* and *t_can_revoke* rules that must be exercised to make that query true have Director as the administrative role.

Prior work There is considerable prior work on safety analysis in various contexts. See, for example, the work of Harrison et al. [3]. It is beyond the scope of this paper to discuss all of those pieces of work. We are aware of two pieces of prior work on ATRBAC-safety. The work of Uzun et al. [12] is, to our knowledge, the first work to propose ATRBAC and pose the safety-analysis problem for it. In addition, that work discusses the design of two software tools, TredRole and TredRule to address instances in practice.

The work of Ranise et al. [6] syntactically generalizes some aspects of the version of ATRBAC from Uzun et al. [12]. It then presents a result on the computational-complexity of ATRBAC-safety — it proves that the problem is decidable. It then discusses the design, construction and evaluation of a different software tool, AsaspTime, to address problem instances in practice.

Our work We make both theoretical and practical contributions in the context of ATRBAC-safety. We observe that prior work refers to a number of different versions of ATRBAC-safety. We carefully distinguish the various versions. This is important to validate prior claims regarding the non-existence of an efficient reduction from one version to another, and for a meaningful empirical assessment. For our theoretical analysis, for each feature of the problem, we adopt the more general form across the two versions from prior work [6, 12].

Our main theoretical result is a much tighter identification of the complexity-class in which ATRBAC-safety lies than prior work — we show that it is **PSPACE**-complete. **PSPACE** is the class of decision problems that can be solved by a (deterministic) Turing machine given space only polynomial in the size of the input.

We show the upper-bound, i.e., that the problem is in **PSPACE**, by constructing a non-deterministic Turing machine that decides instances and uses space only polynomial in the size of the input, and then leveraging the corollary to Savitch's theorem that **NSPACE** = **PSPACE** [9]. **NSPACE** is the class of decision problems that can be solved by a non-deterministic Turing machine given space only polynomial in the size of the input. We infer the lower-bound, i.e., that the problem is **PSPACE**-hard, by observing that ATRBAC generalizes ARBAC, and user-role safety in ARBAC is known to be **PSPACE**-hard [5].

Our result that ATRBAC-safety is **PSPACE**-complete is also of practical consequence. It immediately suggests to us an approach that we can use for instances that arise in practice — model-checking that is complete for **PSPACE**. Mohawk [4] is an open-source tool for ARBAC-safety that leverages a state of the art model checker, and has additional features that are customized for the ARBAC-safety problem. As ARBAC-safety is **PSPACE**-hard and ATRBAC-safety is in **PSPACE**, we know that there exists an efficient, i.e., polynomial-time, reduction from the latter to the former.

Mohawk has been shown to scale to problem-instances that comprise tens of thousands of roles and hundreds of thousands of administrative rules. A thesis we seek to validate empirically is that Mohawk can be extended via an efficient reduction from ATRBAC-safety to ARBAC-safety to address ATRBAC-safety with scalability similar to what it demonstrates for ARBAC-safety instances. We call the tool that we have constructed in this manner, Mohawk+T. The version of ATRBAC-safety that Mohawk+T supports is the most general for each feature across the different tools

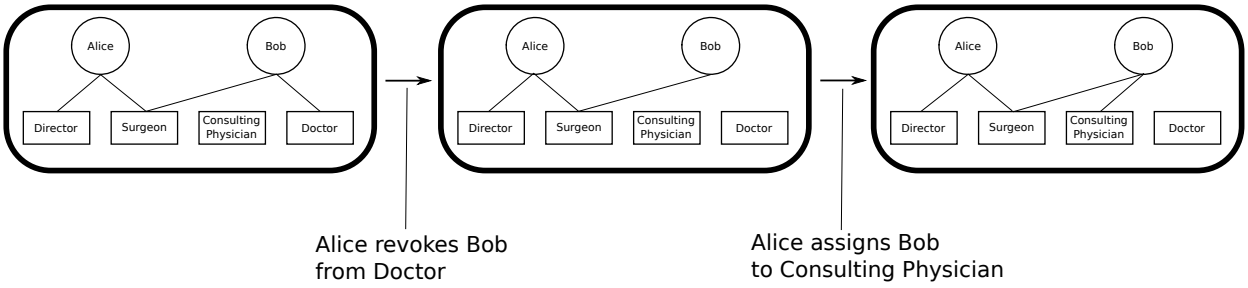


Figure 2: An ARBAC safety query for the example in Figure 1. We ignore the labels on edges that pertain to time-intervals to get an example of *UA* in RBAC. The ARBAC safety query, "could Bob become a member of the role Consulting Physician?," is true. Alice first revokes him from the role Doctor and then assigns him to the role Consulting Physician.

from prior work. Mohawk+T is available for public download [10].

We provide a thorough empirical assessment of Mohawk+T, and compare its performance to tools from prior work*. The versions of the problem that the tools from prior work support are syntactically incomparable to one another. As we adopt the most general from across those tools for each feature for Mohawk+T, to be able to meaningfully empirically compare the different tools on the same inputs, we efficiently reduce the Mohawk+T version to the others. Towards this, we present a "reduction toolkit". The toolkit comprises mappings to efficiently reduce from a version of the problem to another.

A composition of steps from the toolkit is also an efficient reduction, and different such compositions reduce the version of the problem that Mohawk+T supports to the other versions. This includes the version of ARBAC-safety that Mohawk supports. In this manner, we are able to perform an apples-for-apples empirical comparison with prior tools.

2. ATRBAC-SAFETY

In this section, we describe ATRBAC, and then pose the ATRBAC-safety problem. We do this in stages. We first introduce RBAC, ARBAC and a version of ARBAC-safety that is relevant to ATRBAC-safety. Then, we describe TRBAC, ATRBAC and ATRBAC-safety. Figure 4 shows the relationship between RBAC, ARBAC, TRBAC and ATRBAC.

We then clarify that various versions of ATRBAC- and ARBAC-safety are addressed in the literature, and discuss the choices we have made with regards to the various features of the problem. Specifically, that we have chosen the most general of each feature.

2.1 RBAC, ARBAC and ARBAC-Safety

ATRBAC addresses temporal extensions to RBAC and ARBAC. In this section we discuss RBAC, ARBAC and the version of safety analysis in ARBAC that we call ARBAC-safety that is relevant to our work on ATRBAC-safety.

RBAC RBAC is used to specify an authorization policy — who has access to what. An RBAC policy, in the context of this work, is a set UA, the user–role assignment relation. An instance of UA is a set of pairs of the form $\langle u, r\rangle$, where u is a user and r is a role. A user u is authorized to the role r if and only if $\langle u, r\rangle \in UA$. RBAC has other constructs, such as role-permission assignment and a role-hierarchy, that are not relevant to ATRBAC-safety with which we deal in this paper. Indeed, a role-hierarchy can be flattened as a preprocessing step without affecting the correctness or efficiency of our techniques.

ARBAC ARBAC is a syntax for specifying the ways in which an RBAC policy may change. As our work deals with the UA component of an RBAC policy only, by ARBAC we mean its URA portion [7], via which users are authorized to and revoked from roles.

There are only two ways in which an instance of UA may change. One is the addition of an entry $\langle u, r\rangle$ to UA, which is the authorization of u to r. The other is the removal of an entry $\langle u, r\rangle$, which is the revocation of u's authorization to r. An instance of ARBAC is a collection of *rules*, and addresses two issues with regards to such changes to UA: who may carry out one of those operations, and under what conditions.

A set of *can_assign* rules controls additions to UA, and a set of *can_revoke* rules controls removals from UA. A *can_assign* rule is of the form $\langle a, C, t\rangle$, where a, t are roles and C is a precondition. The precondition C is a set in which each entry is either a role r, or its negation, $\neg r$. The semantics of the *can_assign* rule $\langle a, C, t\rangle$ is that a member of the role a may assign a user u to the role t provided u is already a member of every non-negated role in C and is not a member of any negated role in C.

In the *can_assign* rule in Figure 1 for example, a member of the role Director, e.g., Alice, may assign another user, e.g., Bob, to the role Consulting Physician provided he is already a member of the role Surgeon and is not a member of the role Doctor.

In a *can_assign* rule $\langle a, C, t\rangle$, the role a is called an administrative role and the role t is called a target role. A *can_revoke* rule has the form $\langle a, t\rangle$ where both a and t are roles. The semantics is that a member of the administrative role a is allowed to revoke a user's authorization from the target role t. The reason that a *can_revoke* rule has no

*We thank the creators of the prior tools [6, 12] for making their tools available to us and helping us with their use. We thank also Ranise et al. [6] for making all of their inputs from their empirical assessment available to us.

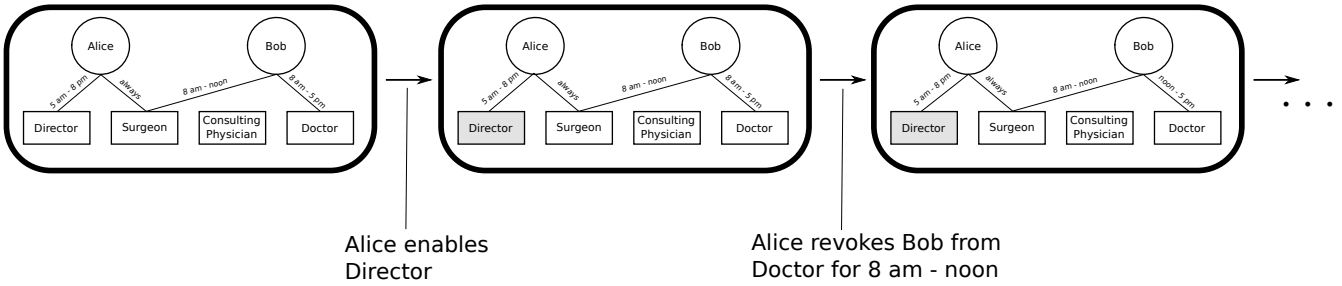


Figure 3: An ATRBAC safety query for the example in Figure 1. The ATRBAC safety query "could Bob become a member of the role Consulting Physician between 8 am and noon?," is true, provided we adopt a version of the problem that does not require the Consulting Physician role to be enabled. The role Director is first enabled (shown shaded in the second state in the figure). This allows Alice to carry out administrative tasks. Alice then exercises the *t_can_revoke* rule so Bob is revoked from the Doctor role for 8 am − noon. She then is able to assign him to the Consulting Physician role for 8 am − noon. This last state-change is not shown in the above figure. "Could Bob become a member of the role Consulting Physician between 1 pm and 5 pm?," is an example of a safety query that is not true.

precondition is that revocation is seen as an inherently safe operation [7].

ARBAC-safety We now discuss a version of safety analysis in ARBAC that is relevant to our work. We call it ARBAC-safety. As our work deals with user-role authorization only, ARBAC-safety refers to that aspect only. More general versions of safety analysis for ARBAC have been considered in the literature [8], that reconcile not only the user-role authorizations, but also role-role relationships. Nevertheless, all the versions of safety analysis in ARBAC of which we are aware lie in the same complexity-class — they are all **PSPACE**-complete.

ARBAC-safety is a state-reachability problem. It takes three inputs. (1) A *query*, which is a pair $\langle u, r\rangle$, where u is a user and r is a role. (2) A current- or start-state, which is an instance of UA. (3) A state-change specification, which is an instance of ARBAC, i.e., instances of *can_assign* and *can_revoke* rules. The output of the ARBAC-safety instance is 'false,' if there exists a state that is reachable from the start-state in which the user u from the query is a member of the role r from the query. Otherwise, the output is 'true.' Figure 2 extends the example from Figure 1 with an example of ARBAC-safety.

ARBAC-safety is known to be **PSPACE**-complete [5]. Several techniques have been proposed to address instances that are likely to arise in practice. For example, Gofman et al. [2] propose a tool called RBAC-PAT, and Jayaraman et al. [4] propose a tool called Mohawk. We adopt the latter as the basis for the tool we build for ATRBAC-safety as it has been shown to scale well with the size of the input.

2.2 TRBAC, ATRBAC and ATRBAC-Safety

We now discuss the temporal extensions to RBAC and ARBAC that give us TRBAC and ATRBAC respectively. We discuss also the problem we address, ATRBAC-safety. We first present a model and encoding of time that is the basis for the syntax for temporality in ATRBAC. The version we adopt is the same as prior work [12].

Time An instant in time, m, can be thought of as represented by a real number. A time-slot represents some duration of time, and is represented as a non-negative integer. In an instance of ATRBAC-safety, no two distinct time-slots overlap in time. Given time-slots i, j where $i < j$, the time-slot j is associated with a duration of time that occurs later than time-slot i. A time-instant m falls within a time-slot.

We assume that the earliest time-slot with which an instance of ATRBAC-safety is associated is 0, and there is some integer, $T_{\max}$, such that $T_{\max} - 1$ is the latest time-slot that pertains to the ATRBAC-safety instance. We discuss how time progresses under ATRBAC-safety below.

A generalization of a time-slot is a time-interval. A time-interval is a pair of integers $\langle i, j\rangle$ where $i \leq j$. It represents the set of time-slots $\{i, i+1, \ldots, j\}$. We say that a time-instant m falls within a time-interval if m falls within one of the time-slots in that time-interval.

The mindset that underlies the above notions for time is that each time-slot represents some realistic, recurring, fixed time period, such as "9 AM – 10 AM." The particular such actual time periods to which time-slots in an instance of ATRBAC-safety map is irrelevant to the analysis.

TRBAC From the standpoint of our work, TRBAC generalizes RBAC in two, temporal ways. (1) The set UA is generalized to TUA, each of whose elements is a triple $\langle u, r, l_{u,r}\rangle$, where $l_{u,r}$ is a time-interval. The semantics is that u is a member of r during the time-interval $l_{u,r}$ only. (2) Each role r that appears in TUA is annotated with a time-interval, l_r. We say that l_r is the time-interval during which the role r is active. The semantics is that outside of the time-interval l_r, no user can exercise a permission that she acquires via the role r. The set of all pairs, $\langle r, l_r\rangle$, is denoted RS.

Thus, a TRBAC policy, and therefore a state in the verification problem we consider, is a 3-tuple, $\langle TUA, RS, m\rangle$, where TUA and RS are as described above, and m is a time-instant. A user u is authorized to a role r at the time-instant, m, if and only if there exists an entry $\langle u, r, l_{u,r}\rangle \in TUA$ such that m is within $l_{u,r}$. The entries in RS matter when a user attempts to make an administrative change, i.e., a change to the authorization state. We discuss this under ATRBAC below.

ATRBAC ATRBAC generalizes ARBAC by providing rules for changes to TRBAC policies. As we discuss under "Versions of the problem" below, the version we discuss generalizes prior versions. Under ATRBAC, there are two ways

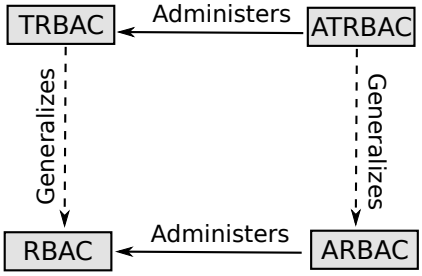


Figure 4: The relationship between RBAC, ARBAC, TRBAC and ATRBAC. TRBAC syntactically generalizes RBAC with temporal extensions. Similarly, ATRBAC syntactically generalizes ARBAC. ARBAC is used to administer RBAC, and ATRBAC is used to administer TRBAC.

in which a state, which is a TRBAC policy, can change: (1) via an administrative action, or, (2) the passage of time.

Under (1), four kinds of administrative actions are possible to a TRBAC policy, $\langle TUA, RS, m\rangle$. It is possible to add an entry to, and remove an entry from TUA, and it is possible to add an entry to, and remove an entry from RS. The first two kinds of changes are called assign and revoke administrative actions, and the next two are called role enabling and disabling administrative actions. We have the corresponding sets of tuples *t_can_assign*, *t_can_revoke*, *t_can_enable*, and *t_can_disable*. (As in Uzun et al. [12], we employ the prefix "*t_*" to distinguish clearly that these are rules for ATRBAC, rather than ARBAC.)

Each such set contains 5-tuples. Each tuple is of the form $\langle C_a, L_a, C_t, L_t, t\rangle$. The first two components, C_a, L_a are conditions on the administrator that seeks to effect the action. The next two components, C_t, L_t, are conditions on the user or role to which the rule pertains. The last component, t, is the target-role; the role that is affected by the action. C_a is either the mnemonic 'true,' or a condition, i.e., a set of negated and non-negated roles. L_a is a set of time-intervals. We specify their semantics below for each kind of administrative rule. The entry t is the target role, i.e., the role that is affected by the firing of the rule. C_t is a role-condition similar to C_a above. There are some important differences between C_a and C_t, however, and we discuss these below for each kind of rule. L_t is a set of time-intervals, similar to L_a above. We discuss the semantics of L_t below for each kind of rule as well.

Such a 5-tuple $\langle C_a, L_a, C_t, L_t, t\rangle$ applies when an administrator, say, Alice, attempts an administrative action at a particular time-instant, m. Each administrative action takes inputs, one of which is the administrator that attempts it, i.e., Alice, and others that we discuss below.

Role enabling: the inputs are Alice, a target role t, and a set of time-intervals, L. Alice succeeds in her attempt at enabling the role t if and only if there exists an entry $\langle C_a, L_a, C_t, L_t, t\rangle \in$ *t_can_enable* for which all of the following are true. (1) The time-instant, m, at which Alice attempts the action falls within some time-interval in L_a. (2) Alice and the current time-instant m together satisfy the administrative condition, C_a. That is, if p is a non-negated role in C_a, then Alice is a member of p at time-instant m in the current state, $\langle TUA, RS, m\rangle$, and p is active at the time-instant m. If n is a negated role in C_a, then either Alice is not a member of n at time-instant m in the current state, or the role n is not active, or both. If C_a is the mnemonic 'true,' then the rule may fire provided m is within some time-interval in L_a. (3) The set of time-intervals L is contained within the set of time-intervals L_t. That is, for every time-interval $l \in L$, there exists a time-interval $l_t \in L_t$ such that l is within l_t. (4) The set of time-intervals L satisfies the target condition C_t for every $l \in L$. That is, if p is a non-negated role in C_t, then for every $l \in L$, p is active during the time-interval l, in the current-state, i.e., RS. And if n is a negated role in C_t, then for every $l \in L$, n is not active during l, in the current-state.

The effect of a successful role enabling by Alice is that the component RS of the current-state is updated as follows to get a new state: $RS \leftarrow RS \cup \{\langle t, l\rangle : l \in L\}$.

Example: In Figure 1 Alice must first enable the role of "Director" so that she can later be allowed to exercise rules where the "Director" role is required by the administrative condition, C_a. Alice may exercise the *t_can_enable* rule during 6 am – 8 am as she satisfies C_a during that time. Once she enables it, Alice must wait before she exercises the *t_can_revoke* rule, where the "Director" role is required by *t_can_revoke*'s C_a, until the current time falls within 8 am – noon.

Role disabling: the inputs are Alice, a target role t, and a set of time-intervals, L. Alice succeeds in disabling t via her action at time-instant m if and only if there exists an entry $\langle C_a, L_a, C_t, L_t, t\rangle \in$ *t_can_disable* for which all of the following are true. (1) The current time-instant, m, falls within some time-interval in L_a. (2) Alice and the current time-instant, m, together satisfy the administrative condition, C_a. (3) The set of time-intervals, L, is contained within the set of time-intervals, L_t. (4) The set of time-intervals L satisfies the target condition C_t for every $l \in L$.

The effect of a successful role disabling by Alice is that the component RS of the current-state is updated as follows to get a new state: $RS \leftarrow RS \setminus \{\langle t, l\rangle : l \in L\}$.

Example: Figure 1 does not provide a rule for *t_can_disable*. But an example case of this rule is to have a *t_can_disable* rule: ⟨true, 6 am – 6 pm, true, noon – 1 pm, Director⟩. This rule allows for the "Director" role to be disabled during lunch time, which means that no changes to TUA can be done during lunch. Exercising the *t_can_enable* rule after this *t_can_disable* rule will overwrite the changes and have the "Director" role enabled during noon – 1 pm.

User-role assignment: the inputs are the administrator, Alice, a user u, a target role t to which she seeks to assign u, and a set of time-intervals L. The assignment action that she attempts at time-instant m succeeds if and only if there exists an entry $\langle C_a, L_a, C_t, L_t, t\rangle \in$ *t_can_assign* for which all of the following are true. (1) The current time-instant, m, falls within some time-interval in L_a. (2) Alice and the current time-instant, m, together satisfy the administrative condition, C_a. (3) The set of time-intervals, L, is contained within the set of time-intervals, L_t. (4) The user u and the set of time-intervals L satisfy the target condition C_t for every $l \in L$. That is, if p is a non-negated role in C_t, then u is a member of p during every time-interval $l \in L$. If n is a negated role in C_t, then u is not a member of n in any time-interval $l \in L$. If C_t is the mnemonic 'true,' then there

are no constraints on the current role-memberships of the user u.

The effect of a successful assignment by Alice is that the component TUA of the current-state is updated as follows to get a new state: $TUA \leftarrow TUA \cup \{\langle u, t, l\rangle : l \in L\}$.

Example: The example in Figure 1 shows that Alice is able to assign the "Consulting Physician" role to Bob during 8 am – noon. She is able to exercise this rule because Bob has the role "Surgeon," and does not have the role "Doctor" during 8 am – noon, and Alice satisfies the administrative condition by having the role "Director."

User-role revocation: the inputs are an administrator Alice, a user u that she seeks to revoke from a role, a target role, t from which she seeks to revoke u, and a set of time-intervals, L. The revocation action she attempts at some time-instant m succeeds if and only if there exists an entry $\langle C_a, L_a, C_t, L_t, t\rangle \in$ *t_can_revoke* for which all of the following are true. (1) The current time-instant, m, falls within some time-interval in L_a. (2) Alice and the current time-instant, m, together satisfy the administrative condition, C_a. (3) The set of time-intervals, L, is contained within the set of time-intervals, L_t. (4) The user u and the set of time-intervals L satisfy the target condition C_t for every $l \in L$. The effect of a successful revocation by Alice is that the component TUA of the current-state is updated as follows to get a new state: $TUA \leftarrow TUA \setminus \{\langle u, t, l\rangle : l \in L\}$.

Example: Alice may revoke Bob from the "Doctor" role during 8 am – noon using the *t_can_revoke* rule in Figure 1. If she does so, Bob retains membership of "Doctor" during noon – 5 pm, as we show in Figure 3.

Time-change: Another way that a state, $\langle TUA, RS, m\rangle$, can change is in its time component, m. The manner in which passage of time is modelled [6, 12] is simply by allowing the m component to increase without any change to the other two components, TUA and RS. That is, a possible state-change is from $\langle TUA, RS, m\rangle$ to a new state, $\langle TUA, RS, m'\rangle$, where $m' > m$.

An issue we clarify in this regard of passage of time is whether, once we reach the time-slot $T_{\max} - 1$ to which an instance of ATRBAC-safety pertains, the time-slot 0 recurs, followed by time-slot 1 and so on, forever. The assumption in prior work [12] is that it does. The reason regards the semantics of a time-slot — it maps to some realistic, recurring period of time. We refer to this property as periodicity, and revisit it in the context of the software tools.

Example: Time periodicity is what allows the rules in Figure 1 to be described by just the time of day. The intention of the rules is that they are contained within a day. Thus when a day ends and the next day begins, the rules should still apply to the new day.

ATRBAC-safety The safety analysis problem for ATRBAC takes three inputs. (1) A query, $\langle u, C, L, t\rangle$, where u is a user, C is a condition (set of negated and non-negated roles), L is a set of time-intervals and t is some units of time. (2) A start-state, $\langle TUA, RS, m\rangle$, which is an instance of TRBAC. (3) A state-change specification, which is an instance of ATRBAC, i.e., four sets of rules, *t_can_assign*, *t_can_revoke*, *t_can_enable*, and *t_can_disable*.

The output is 'false,' if there exists a TRBAC state $\langle TUA', RS', m'\rangle$ that is reachable from the start-state in which: (i) the user u is a member of every non-negated role in C in every time-interval in L, and is not a member of any negated role in C in any time-interval in L, (ii) every non-negated role in C is active for every time-interval in L, and no negated role in C is active in any time-interval in L, and, (iii) the time-instant m' of this state is within t time-units of the time-instant of the start-state. Otherwise, the output is 'true.' We point out that it is possible to specify t that is large enough that the query pertains to any time-slot that pertains to the problem instance.

In Figure 3 we discuss two ATRBAC safety questions: "could Bob become a member of the role Consulting Physician between 8 am and noon?" and "could Bob become a member of the role Consulting Physician between 1 pm and 5 pm?". As the caption of the figure discusses, the former is true, provided we do not require the role Consulting Physician to be enabled when Bob becomes a member of it. The latter question is not true.

Versions of the problem Prior work that is relevant to our work refers to different versions of ATRBAC-safety. We broadly classify the various versions along two axes. One comprises the versions that are discussed theoretically, i.e., in prose, assertions and proofs only. The other comprises versions supported by software tools.

Correspondingly, from the work of Uzun et al. [12] we have the versions of ATRBAC-safety that we call TRED-THEORY, and the versions supported by their tools, TREDROLE and TREDRULE. They are so named because Ranise et al. [6] refer to their two software tools with the prefix TRED. Similarly, from Ranise et al. [6] we have ASASPTIME-THEORY, and the versions supported by their tools ASASPTIME-NSA and ASASPTIME-SA. The acronym "NSA" stands for Non-Separate Administration, and "SA" for "Separate Administration." They pertain to whether administrative roles are distinct from user roles. Finally, we refer to the version of ATRBAC-safety we discuss above as Mohawk+T-THEORY. We also have the version that is supported by our tool, Mohawk+T.

We address the differences between ASASPTIME-THEORY and TRED-THEORY, and our choices for Mohawk+T-THEORY here. We address the differences between the versions of the software tools in Section 4. A recognition of the differences of the theoretical versions is important from two standpoints. One is that Mohawk+T-THEORY syntactically generalizes both ASASPTIME-THEORY and TRED-THEORY. Thus, an upper-bound for the computational complexity of ATRBAC-safety for Mohawk+T-THEORY is an upper-bound for each of the other two as well.

Another reason a recognition of these different versions is important regards an assertion about the non-existence of an efficient reduction from ASASPTIME-THEORY to TRED-THEORY in prior work [6]. As we point out in the next section on computational complexity, the assertion is in error.

The differences between ASASPTIME-THEORY and TRED-THEORY pertain to (1) whether time-intervals are allowed, or only time-slots, (2) whether an administrative condition may be specified in a rule, or an administrative role only, and, (3) the kind of query that may be specified in an instance of safety. The differences are the following.

1. ASASPTIME-THEORY allows time-slot only, and not time-intervals. TRED-THEORY allows time-intervals. So, for this feature, ASASPTIME-THEORY is less gen-

eral than TRED-THEORY. We point out that naively encoding a time-interval as a set of time-slots is inefficient in the worst-case. Recall from our discussion above under "Time" that a time-interval is a set of consecutive time-slots $\{i, i+1, \ldots, i+n\}$, where $n \geq 0$. Encoding a time-interval as the pair $\langle i, i+n \rangle$ takes space $\leq 2\log(i+n)$ only. Encoding it as the set $\{i, i+1, \ldots, i+n\}$, however, takes space $\geq (n+1)\log(i)$. The latter is exponential in the former, in the worst-case.

2. ASASPTIME-THEORY allows a condition for the administrator in a rule, rather than an administrative role only. TRED-THEORY allows an administrative role only. For this feature, therefore, ASASPTIME-THEORY generalizes TRED-THEORY.

3. A query in ASASPTIME-THEORY is of the form $\langle u, C, L \rangle$ where u is a user, C is a role condition, and L is a set of time-slots. The semantics is: does there exist a reachable state in which (1) u satisfies C for L, i.e., is a member of every non-negated role for every time-slot in L and not a member of any negated role in C for any time-slot in L, and, (2) roles are enabled and disabled as C specifies for the time-slots in L.

   TRED-THEORY proposes two kinds of queries. One is of the form $\langle u, r, L \rangle$, which asks whether u can become a member of r in all the time-intervals L. This is less syntactically general than ASASPTIME-THEORY in that C is allowed to be a role only, but more general in that L is allowed to be a set of time-intervals, and not only a set of time-slots. The other kind of query is of the form $\langle u, r, L, t \rangle$, which asks whether u can become a member of r in all the time-intervals in L within t time-units of the start-state. This syntactically generalizes the version of the safety problem that allows the first kind of query only — there is a straightforward reduction from the safety problem that allows the first kind of query only, to one that allows the second kind of query.

   Thus, from the standpoint of the query, the two versions are incomparable to one another.

For Mohawk+T-THEORY that we describe earlier, for each of the three features above, we have chosen the more general. For example, the query allows a role condition, a set of time-intervals and the extra parameter t that limits the number of time units in which the query must become true. Also, we assume that all possible users that are allowed to exist in the system are part of the *TUA* component in the start-state. (It is easy to incorporate users that are not assigned to any role into *TUA*. Create a "dummy" role that does not appear in any of the administrative rules and assign all users to it in the start-state.)

The reason we have adopted such a general version is to emphasize our complexity result that we present in the next section — such a choice has no consequence to the upper-bound computational complexity of ATRBAC-safety.

3. COMPUTATIONAL COMPLEXITY

We now identify the computational complexity of ATRBAC-safety, for the version we call Mohawk+T-THEORY. In Theorem 1 below, we identify an upper-bound. Then, in Theorem 2, we identify a tight bound. We then identify that all versions of ATRBAC-safety we address are **PSPACE**-complete. We then address the expressive power of ATRBAC and an assertion in prior work on the non-existence of a reduction.

THEOREM 1. *ARBAC-safety for Mohawk+T-THEORY is in **PSPACE**.*

The above theorem asserts an upper-bound on the hardness of safety analysis in ATRBAC. To prove it, we construct a non-deterministic Turing machine that terminates on every input with the correct 'safe' or 'unsafe' output, and runs with space only polynomial in the size of the input. Then, from the fact that **PSPACE** = **NSPACE**, which is a corollary to Savitch's theorem [9], we immediately infer that the problem is in **PSPACE**.

We point out that a similar non-deterministic Turing machine is constructed by Jha et al. [5] to show that their version of ARBAC-safety is in **PSPACE**. A main difference for us is that we need to reconcile the temporal aspect of ATRBAC-safety as well. Our non-deterministic Turing machine, M, is provided the three inputs: (1) a query, $q = \langle u, C, L, t \rangle$, (2) A start-state, $\langle TUA, RS, m \rangle$, and, (3) A set of state-change rules in ATRBAC, i.e., the four sets *t_can_assign*, *t_can_revoke*, *t_can_enable* and *t_can_disable*.

M first assembles a set S of all the time-intervals (some of which may be time-slots) that appear in any of the three components of the input. M then breaks up the time-intervals in S so none of them overlaps with any other. It does this using the algorithm in Figure 5. The input set S is at worst linear in the size of the input ARBAC-safety instance. The output from the algorithm in Figure 5, call it S', is at worst quadratic in the size of input S – the caption in Figure 5 provides a reasoning.

The reason M does this is that it only has to maintain one of these time-intervals as the current time. The exact value of the current time, or even time-slot, no longer matters. For convenience, M could rewrite the input ATRBAC rules so each mentions only entries from S' and not S. M then maintains the following state. (1) The current values for the sets *TUA* and *RS*. (2) The current time-interval from S' within which the current time instant falls. (3) The number of time units that have elapsed.

M performs its moves as follows. M first checks whether the query q is satisfied. It can do so from the state information it maintains. If yes, it halts with output 'unsafe.' If not, it checks whether it has exceeded t from the query q. If yes, it halts with output 'safe.' Otherwise, it assembles all the administrative rules that are enabled, i.e., for which the administrative and role conditions are met. It also includes an update of the current time-interval to the next time-interval as an option. It non-deterministically picks an option from those, updates its state, and continues.

M ensures that it terminates — we know that the input instance is unsafe if and only if the non-deterministic Turing machine halts with an output of 'unsafe' within 2^n transitions, where n is the size of the input. The Turing machine can keep a count of its transitions with space $\log_2(2^n) = n$. Thus, if M has not determined that the instance is unsafe after 2^n transitions, it outputs 'safe' and halts. We point out also that M maintains state that is at worst quadratic in the size of the input.

THEOREM 2. *ATRBAC-safety for Mohawk+T-THEORY is **PSPACE**-complete.*

Input: a set of time-intervals, S

1: From the entries of S that have not been considered before, pick one of smallest duration.
2: If any entry in S overlaps with the entry chosen in Line (1) other than itself, break it up into at most 3 non-overlapping entries and add those back to S.
3: If all entries have been considered, halt.
4: Else goto Line 1.

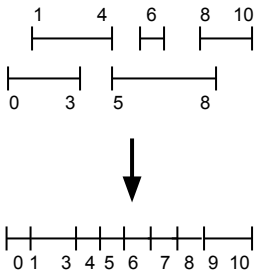


Figure 5: To the left is an algorithm to break up input time-intervals into non-overlapping time-intervals. An example of an input and output is shown to the right. The algorithm is used by the non-deterministic Turing machine in Section 3, and as part of our Reduction Toolkit in Section 4. The algorithm takes as input S, a set of time-intervals. The algorithm is guaranteed to terminate as no entry is chosen more than once in Line (1), and at most a constant number of entries is added in Line (2) for every entry chosen in Line (1). The algorithm runs in time at-worst quadratic in the size of the input S because for every entry chosen in Line (1), each entry in S is broken up into at most a constant number of entries in Line (2). In the example to the right, the input is the 5 time-intervals, $\{\langle 0,3\rangle, \langle 1,4\rangle, \langle 5,8\rangle, \langle 6,6\rangle, \langle 8,10\rangle\}$. The output is the 8 non-overlapping time-intervals, $\{\langle 0,0\rangle, \langle 1,3\rangle, \langle 4,4\rangle, \langle 5,5\rangle, \langle 6,6\rangle, \langle 7,7\rangle, \langle 8,8\rangle, \langle 9,10\rangle\}$. To represent the time interval $\langle 0,3\rangle$, from the input set S, in non-overlapping time intervals we would use: $\{\langle 0,0\rangle, \langle 1,3\rangle\}$.

We infer the above theorem from the fact that ARBAC-safety is **PSPACE**-hard [5]. As ATRBAC-safety for Mohawk+T-Theory generalizes that version of ARBAC-safety, the **PSPACE**-hard lower bound applies to ATRBAC-safety for Mohawk+T-Theory as well. Given Theorem 1, we thus prove Theorem 2.

Theorem 3. *All the versions of ATRBAC-safety we consider in this paper are **PSPACE**-complete. They are: AsaspTime-Theory, Tred-Theory, Mohawk+T-Theory, and the versions supported by the tools AsaspTime-Nsa, AsaspTime-Sa, TredRole, TredRule, Mohawk+T and Mohawk.*

To prove the theorem, we first oberve that the version of ARBAC-safety, which can be perceived as a special case of ATRBAC-safety with only 1 time-slot, supported by Mohawk, is **PSPACE**-complete. We comment on this more in Section 4 under Item (4), "Admin role is 'true' only," in our Reduction Toolkit. All the other versions generalize the version supported by Mohawk, and therefore are **PSPACE**-hard. And, all the versions are at most as general as Mohawk+T-Theory, and therefore are in **PSPACE**.

3.1 Expressive power of ATRBAC

We can interpret the hardness of ATRBAC-safety as a measure of the expressive power of ATRBAC. The fact that ATRBAC-safety is no harder than ARBAC-safety, within a polynomial factor, suggests that ATRBAC is no more expressive than ARBAC.

Indeed, in this context, one can point to some seeming deficiencies in the syntax of ATRBAC. It is not possible, for example, to directly express a rule such as the following: "if a user is a member of the role Surgeon between 8 am and 9 am, then allow the user to be assigned to the role Consulting Physician between noon and 5 pm." The reason is that the set of time-intervals L_t in a rule $\langle C_a, L_a, C_t, L_t, t\rangle$ serves two purposes. It is used as a precondition to check the current temporal role-memberships of the user along with C_t, and, it is used to limit the time-intervals for which the user can acquire membership in the target role t.

A straightforward way to extend the syntax of ATRBAC to account for such use cases is to allow C_t to be a set of pairs, each of the form $\langle c, l\rangle$, where c is either a non-negated or negated role, and l is a time-interval. This separates the two purposes mentioned above. Nonetheless, ATRBAC-safety for this version of ATRBAC is also **PSPACE**-complete. That is, from the standpoint of computational complexity of safety analysis, nothing has changed. This means that this new version can be reduced efficiently to the original.

Another issue in this context regards an assertion from prior work [6] regarding the non-existence of an efficient reduction from AsaspTime-Theory to Tred-Theory. Theorem 3 invalidates the assertion. The erroneous reasoning in that work is that to support an administrative condition in an ATRBAC rule rather than only an administrative role, one must introduce exponentially many new roles.

An efficient reduction, however, can be constructed as follows. We introduce a new role for every administrative condition. Thus, the number of new roles is at worst linear in the ATRBAC-safety instance. We then introduce *t_can_assign* rules with the condition as a role precondition to assign a user to the new role. There are additional details we need which we omit here, for example to account for when a user no longer satisfies a condition in a future state.

4. EMPIRICAL ASSESSMENT

We have designed and built a software tool that we call Mohawk+T for ATRBAC-safety. Mohawk+T has been built as a wrapper to Mohawk [4], an open-source tool for ARBAC-safety. Mohawk reduces ARBAC-safety to model-checking and employs an off-the-shelf model checker, NuSMV. In addition, Mohawk employs within it domain-specific heuristics called abstraction-refinement and bound-estimation for increased efficiency. The empirical results that have been reported for Mohawk suggest that it scales well for large input instances, for example, 40,000 roles and 200,000 rules.

Our intent here is to validate the thesis that we can wrap Mohawk in a manner that we preserve its scalability for ATRBAC policies. Thus, Mohawk+T would scale significantly better than what has been shown for existing tools for

	Time-intervals or time-slots only	Enable/disable rules supported	Query	Administrative roles supported or 'true' only
AsaspTime-Nsa	Time-slots only	Yes	$\langle r, l\rangle$ — can some user become a member of r in time-slot l?	Yes
AsaspTime-Sa	Time-slots only	No	$\langle r, l\rangle$ — see above	No — 'true' only
TredRole, TredRule	Allows time-interval in administrative condition	No	$\langle R\rangle$ — can the same user become a member of all roles in R in the same time-slot?	No — 'true' only
Mohawk+T	Allows time-interval in administrative condition	Yes	$\langle R, l\rangle$ — can the same user become a member of all roles in R in the time-slot l?	Yes
Mohawk	No temporal support	No	$\langle u, r\rangle$ — can the user u become a member of role r?	No — 'true' only

Table 1: Feature-support of the various existing tools, and our design choice for Mohawk+T. For Mohawk+T, we have chosen the most general version of a feature from amongst existing tools for ATRBAC-safety. We show also the support Mohawk offers. As a tool for ARBAC-safety only, Mohawk expectedly has no support for temporality. There is also no notion of enable/disable rules in ARBAC-safety. When we say "'true' only" for administrative roles, we mean that every rule is enabled in every state.

ATRBAC-safety [6, 12]. The manner in which Mohawk+T wraps Mohawk is an efficient reduction from ATRBAC-safety to ARBAC-safety. That is, given as input an instance of ATRBAC-safety, the wrapper in Mohawk+T maps this to an instance of ARBAC-safety that it provides as input to Mohawk. The mapping is a reduction, and therefore Mohawk's 'safe' or 'unsafe' output can immediately be adopted as Mohawk+T's output.

We had to address some technical challenges in realizing and assessing Mohawk+T. One is that we had to choose a version of ATRBAC-safety that Mohawk+T would support. So far in this paper, we have discussed what we have called theoretical versions of ATRBAC-safety. None of the prior tools, AsaspTime-Nsa, AsaspTime-Sa, TredRole and TredRule supports its corresponding theoretical version. AsaspTime-Nsa, for example, supports administrative roles only, and not administrative conditions. Another example is that TredRole allows only a set of time-slots, and not a time-interval, as the role condition.

As with their theoretical counterparts, the two sets of existing tools are incomparable to one another from the standpoint of generality. We have investigated the various features that the existing tools allow for their input, and chosen for Mohawk+T the more general for each feature. We discuss this in more detail below.

Another technical challenge we had to address is that we had to devise an efficient reduction from the version of ATRBAC-safety that Mohawk+T supports to the version of ARBAC-safety that Mohawk supports. In this context, we point out that Mohawk also has a corresponding theoretical version [5] that is more general. Mohawk, for example, allows only 'true' for the administrative role in a rule. We discuss our reduction in Section 4, once we introduce our Reduction Toolkit.

We devised our Reduction Toolkit not only to reduce the version of ATRBAC-safety that Mohawk+T supports to the version of ARBAC-safety that Mohawk supports, but also so we can conduct a meaningful empirical comparison with existing tools for ATRBAC-safety. As Mohawk+T syntactically generalizes all existing tools, we cannot directly provide as input to an existing tool an input designed for Mohawk+T.

So we can perform apples-for-apples comparisons on inputs, we have devised and implemented efficient reductions from the version of ATRBAC-safety that Mohawk+T supports to the version that each of the existing tools supports. Thus, we need 4 reductions in total — to (1) AsaspTime-Nsa, (2) AsaspTime-Sa, (3) TredRole and TredRule, and, (4) Mohawk. We observed that the reductions have commonalities, which motivated us to devise our Reduction Toolkit. Composing particular reductions from the toolkit provides us with each of the 4 reductions we seek.

Versions that tools support Table 1 expresses the support for various features in existing tools, and our design choice for Mohawk+T. We list only those features for which the tools differ. As the table shows, for Mohawk+T, we have chosen the most general for each feature in existing tools. We show in the table also the support in Mohawk. We show this to indicate what our reduction from Mohawk+T's version of ATRBAC-safety must address. We point out, in addition to the information in the table, that none of the existing tools for ATRBAC-safety supports administrative conditions. They also do not support periodicity of time. Therefore, neither does Mohawk+T. These are features that are supported by the theoretical versions we discuss earlier in the paper.

4.1 A "Reduction Toolkit"

There are some common transformations we effect in our efficient reductions from Mohawk+T to AsaspTime-Nsa, AsaspTime-Sa, TredRole and TredRule, and Mohawk. In this section, we discuss six mappings, each of which transforms some component of the problem efficiently (in polynomial time). Then, in subsequent sections, we mention which ones we need for each of the four reductions. Each of our transformations from an input to Mohawk+T to one of the other tools composes some of these transformations from our toolkit. As each of these transformations is an efficient re-

duction, so is any of those compositions. We provide more details in [11].

Reduction (1) in our toolkit maps a Mohawk+T query, which is of the form $\langle R, l\rangle$, to what we call a Type 1, 2 or 3 query. Type 1 is of the form $\langle r', l'\rangle$, which is supported by AsaspTime-Nsa and AsaspTime-Sa. We create a new role r' and a new *t_can_assign* rule with it as target. The roles in R are the precondition. We also identify to which time-slot l', the time-slot l maps from Reduction (2) of the toolkit. A query of Type 2 is of the form $\langle R'\rangle$, which is supported by TredRole and TredRule. We do something similar to Type 1 — create a new role r', a new *t_can_assign* rule for it, and set $R' = \{r'\}$. The Type 3 query is of the form $\langle u', r'\rangle$ and is supported by Mohawk. For this mapping, we first remove temporality using Reduction (5) from the toolkit. Then, we create a *can_assign* rule as for Types 1 and 2, and also ensure that a user u' exists by mentioning it in Mohawk's input.

Reduction (2) maps time-intervals to time-slots only. Recall from our discussion in Section 2.2 that naively enumerating the time-slots that comprise a time-interval is not efficient. For this reduction, we run the algorithm from Figure 5. We then adopt each time-interval in the output as a time-slot.

Reduction (3) produces an output with no *t_can_enable* or *t_can_disable* rules. This is needed to map Mohawk+T inputs to AsaspTime-Sa, TredRole, TredRule and Mohawk. (For Mohawk, we also need to remove temporality using Reduction (5) below.) The manner in which we remove *t_can_enable* and *t_can_disable* rules is to introduce, for every role r, a new role r_e. The user's membership in r_e indicates enablement. A *t_can_enable*/*t_can_disable* rule with r as target or precondition becomes a *t_can_assign*/*t_can_revoke* rule with r_e as target or precondition respectively.

Reduction (4) produces an output with admin role set to the mnemonic 'true' only. That is, every rule is enabled in every state. It may seem surprising that an efficient reduction exists from Mohawk+T's version of the problem that allows arbitrary administrative roles in rules to this one. We point out that the **PSPACE**-hardness reduction of Jha et al. [5], maps to instances of ARBAC-safety that has such rules only. The way to do this is to first observe that the maximum number of administrative users we need is $A \times L$, where A is the set of all administrative roles and L is the set of time-intervals in the input. We transform the administrator in a rule, which can be seen as a constraint, to a role precondition. The way to do this is to allow the same, single user to act as each administrator. We need to do this carefully, however, to ensure that it is a reduction. See [11] for more details.

Reduction (5) produces an output with no temporality. This can be seen as a version of the problem with one time-slot only. We need this to reduce to Mohawk's version, which is for ARBAC-safety. We map every entry in $R \times L$ to a role in Mohawk, where R is the set of all roles and L is the set of all time-intervals in the Mohawk+T input.

Reduction (6) produces an output with no preconditions in *can_revoke* rules. This is needed to reduce to Mohawk — we assume that temporality has been removed using Reduction (5) above. Given a *can_revoke* rule $\langle a, C, t\rangle$, we first rewrite it as $\langle a, t\rangle$. Then, we add a new role t' and a *can_assign* rule, $\langle a, \neg t \wedge C, t'\rangle$. We replace a role precondition $\neg t$ with $\neg t \wedge t'$. See [11] for the reasoning.

Reductions to versions To reduce from Mohawk+T's version to AsaspTime-Nsa, we compose Reductions (1) and (2). To reduce to AsaspTime-Sa, we compose Reductions (1)–(4). To reduce to TredRole and TredRule, we compose Reductions (1), (3) and (4). To reduce to Mohawk's version, we compose (1), and (3)–(6).

5. EMPIRICAL RESULTS

We have conducted empirical assessments on the three benchmark classes from prior work [6]. In addition, we have converted the input instances that were used in the empirical assessment of Mohawk [4] to ATRBAC-safety instances, and tried them as well on the tools. The conversion of Mohawk inputs is trivial — we adopt a single time-slot for the policy. Our results are shown in Figures 6, 7 and 8. The curves interpolate the average of 5 runs. The error-bars show the standard deviation from the average.

We observe that the results are mixed with regards to favoring Mohawk+T. For Benchmark Class (a), which is shown in Figure 6, the existing tools for ATRBAC-safety outperform Mohawk+T. Mohawk+T completes in a few seconds, while existing tools complete in less than a second each. Our investigation reveals that the policies in this benchmark are always safe, given the way they are generated. Mohawk's static slicing does a good job of paring large policies (e.g., 2,000 roles, 100,000 rules) down to a much smaller size (e.g., 150 roles and rules). However, its bound estimator estimates a bound that is linear in the number of roles for these policies, e.g., 150. While this is certainly much better than the worst-case estimate of 2^{150}, it still results in Mohawk+T taking several seconds.

For Benchmark Class (b), which is shown in Figure 7, and Mohawk inputs, which is shown in Figure 8, however, Mohawk+T significantly outperforms the existing tools. Furthermore, the existing tools are unable to withstand the input instances from Mohawk [4] beyond a certain threshold. For the polynomial-time verifiable sub-class, for example, which is Test Suite 1 in Figure 8, none of the existing tools is able to handle inputs beyond 20,000 roles and 80,000 rules.

For Benchmark Class (c), which is shown in Figure 7, AsaspTime-Nsa outperforms Mohawk+T. Note that as in prior work [6], we did not try this Benchmark Class on the other existing tools.

Thus, we have the somewhat interesting situation that no single tool can be said to be good with all the input ATRBAC-safety instances we have tried. Further investigation is warranted to carefully identify the structure of input instances, and what features a universally good tool needs to have.

6. CONCLUSIONS

We have addressed the safety analysis problem in the context of Administrative Temporal Role-Based Access Control (ATRBAC-safety). We have shown that the problem is **PSPACE**-complete, and also that various versions of the problem from the literature are **PSPACE**-complete. As the complexity class is the same as ARBAC-safety, we have investigated an approach to dealing with practical instances via reduction to ARBAC-safety, and the leveraging an exist-

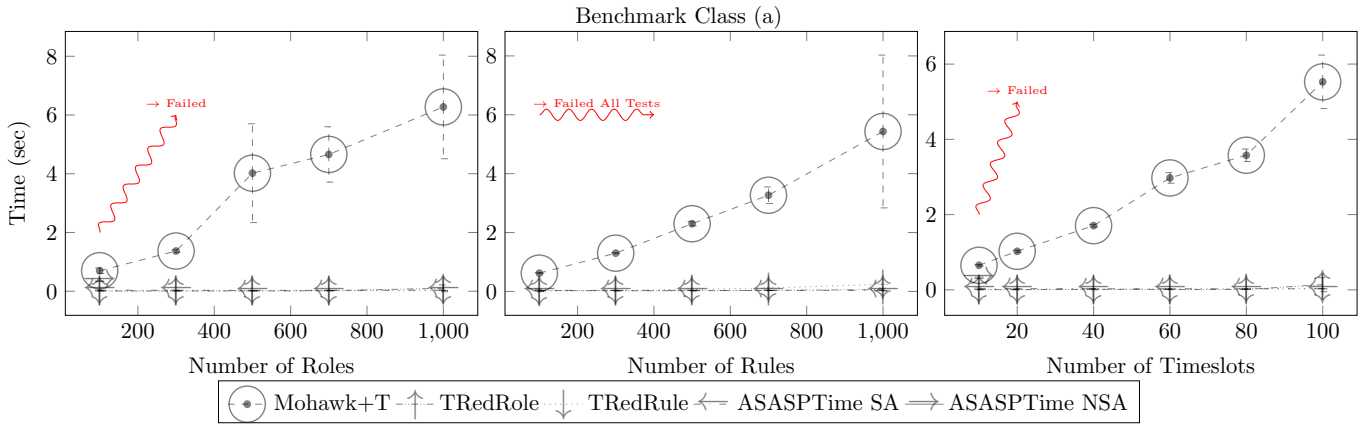


Figure 6: Results on all tools for Benchmark Class (a). It comprises random input instances from a generator from Uzun et al. [12]. The curves interpolate averages, and the error-bars show the standard deviation.

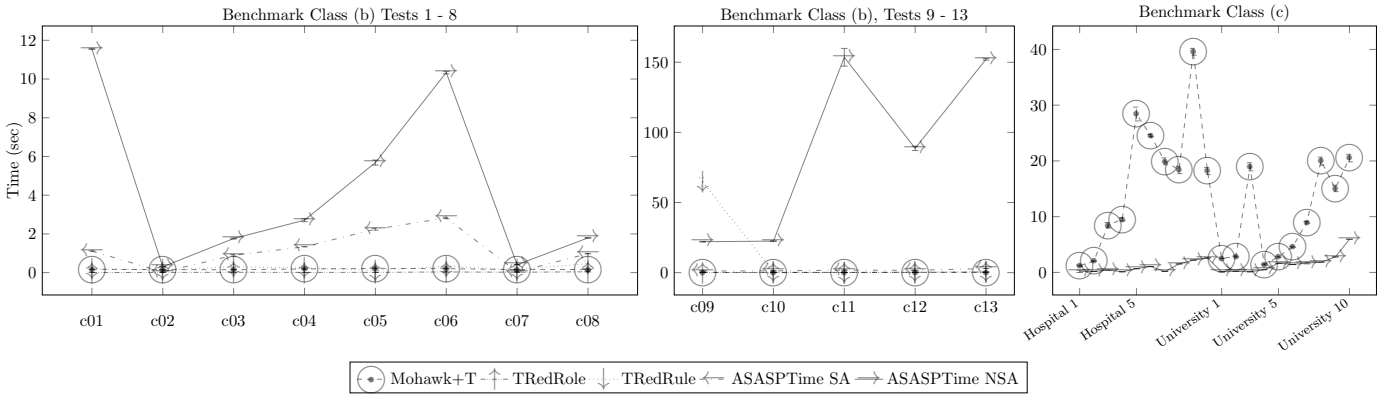


Figure 7: Results for Benchmark Class (b) (two graphs to the left), and Benchmark Class (c) (right). These comprise input instances from the work of Ranise et al. [6].

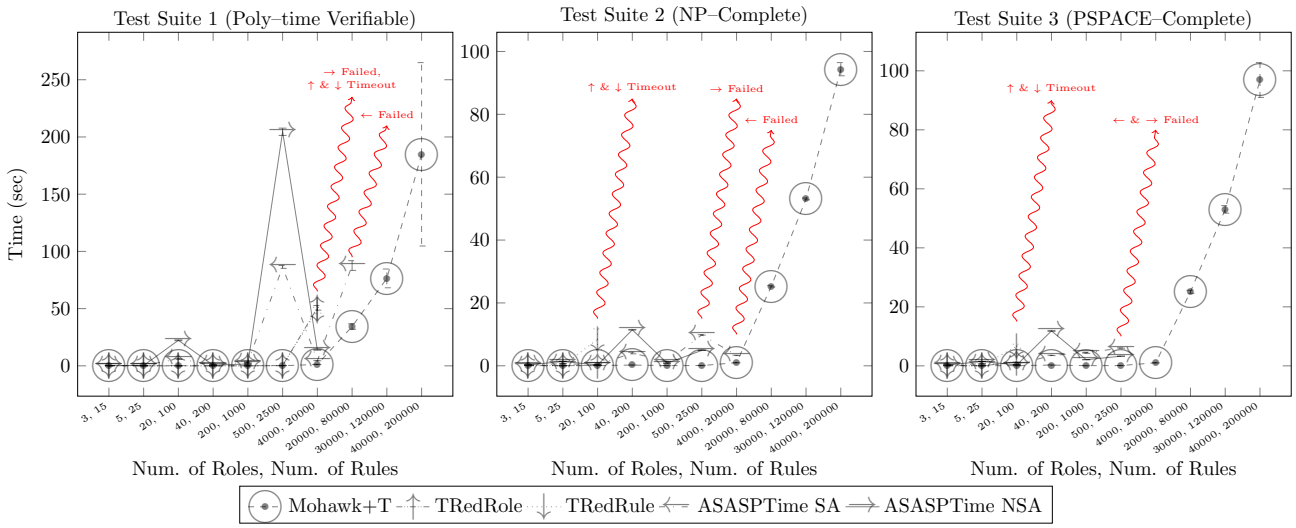


Figure 8: Results for inputs used in the empirical assessment of Mohawk [4]. We trivially converted the Mohawk inputs to ATRBAC-safety instances — there is one time-slot only. The graph to the far left is for inputs with non-negated preconditions only. The middle graph is for inputs with no revoke rules. To the right is the hardest class, in which both negated and non-negated preconditions are allowed, as are revoke rules. Some of the curves are truncated because the corresponding tools crash at those inputs sizes and beyond.

ing tool that has been shown to scale well — Mohawk. For an apples-for-apples comparison with existing tools, we have also come up with a Reduction Toolkit. Compositions of reductions from the toolkit allow us to reduce Mohawk+T's more general version of ATRBAC-safety to other versions. We have conducted a thorough empirical assessment. Our results are that there are some classes of inputs for which exiting tools outperform Mohawk+T, and others for which Mohawk+T outperforms existing tools.

7. REFERENCES

[1] MiniSat. `http://minisat.se/`, Feb 2015.

[2] M. Gofman, R. Luo, A. Solomon, Y. Zhang, P. Yang, and S. Stoller. Rbac-pat: A policy analysis tool for role based access control. In S. Kowalewski and A. Philippou, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 5505 of *Lecture Notes in Computer Science*, pages 46–49. Springer Berlin Heidelberg, 2009.

[3] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman. Protection in operating systems. *Commun. ACM*, 19(8):461–471, Aug. 1976.

[4] K. Jayaraman, M. Tripunitara, V. Ganesh, M. Rinard, and S. Chapin. Mohawk: Abstraction-refinement and bound-estimation for verifying access control policies. *ACM Trans. Inf. Syst. Secur.*, 15(4):18:1–18:28, Apr. 2013.

[5] S. Jha, N. Li, M. Tripunitara, Q. Wang, and W. Winsborough. Towards formal verification of role-based access control policies. *Dependable and Secure Computing, IEEE Transactions on*, 5(4):242–255, Oct 2008.

[6] S. Ranise, A. Truong, and A. Armando. Scalable and Precise Automated Analysis of Administrative Temporal Role-based Access Control. In *Proceedings of the 19th ACM Symposium on Access Control Models and Technologies*, SACMAT '14, pages 103–114, New York, NY, USA, 2014. ACM.

[7] R. Sandhu, V. Bhamidipati, and Q. Munawer. The arbac97 model for role-based administration of roles. *ACM Trans. Inf. Syst. Secur.*, 2(1):105–135, Feb. 1999.

[8] A. Sasturkar, P. Yang, S. D. Stoller, and C. R. Ramakrishnan. Policy analysis for administrative role based access control. *Theoretical Computer Science*, 412(44):6208–6234, Oct. 2011.

[9] W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177 – 192, 1970.

[10] J. Shahen. Mohawk+T: Source Code. `https://ece.uwaterloo.ca/~jmshahen/mohawk+t/`, Jun 2015.

[11] J. Shahen. Mohawk+T: Efficient Analysis of Administrative Temporal Role-Based Access Control (ATRBAC) Policies. Master's thesis, University of Waterloo, 2016. Available from `https://uwspace.uwaterloo.ca/`.

[12] E. Uzun, V. Atluri, S. Sural, J. Vaidya, G. Parlato, A. L. Ferrara, and M. Parthasarathy. Analyzing Temporal Role Based Access Control Models. In *Proceedings of the 17th ACM Symposium on Access Control Models and Technologies*, SACMAT '12, pages 177–186, New York, NY, USA, 2012. ACM.