CERIAS Tech Report 2005-20

ON SAFETY IN DISCRETIONARY ACCESS CONTROL

by Ninghui Li and Mahesh V. Tripunitara

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907-2086

On Safety in Discretionary Access Control

Ninghui Li Mahesh V. Tripunitara
Center for Education and Research in Information Assurance and Security
and Department of Computer Sciences
Purdue University
656 Oval Drive, West Lafayette, IN 47907-2086
{ninghui, mtripuni}@cs.purdue.edu

Abstract

An apparently prevailing myth is that safety is undecidable in Discretionary Access Control (DAC); therefore, one needs to invent new DAC schemes in which safety analysis is decidable. In this paper, we dispel this myth. We argue that DAC should not be equated with the Harrison-Ruzzo-Ullman scheme, in which safety is undecidable. We present an efficient (running time cubic in its input size) algorithm for deciding safety in the Graham-Denning DAC scheme, which subsumes the DAC schemes used in the literature on comparing DAC with other access control models. We also refute several claims made in recent work by Solworth and Sloan [26], in which the authors present a new access control scheme based on labels and relabelling and claim that it can "implement the full range of DAC models". We present a precise characterization of their access control scheme and show that it does not adequately capture a simple DAC scheme.

1 Introduction

Safety analysis, first formulated by Harrison, Ruzzo, and Ullman [11] for the access matrix model [13, 9], has been recognized as a fundamental problem in access control. Safety analysis decides whether rights can be leaked to unauthorized principals in future states. Safety analysis was shown to be undecidable in the HRU scheme. Since then, considerable research effort has gone into designing access control schemes in which safety analysis is decidable [1, 2, 5, 12, 16, 18, 19, 22, 23, 24, 25, 26, 28, 29]. Safety analysis is particularly interesting in DAC [6, 7, 9, 10], in which a subject gets rights to resources at the discretion of other subjects. Recently, there appears to be renewed interest in the topic of safety in DAC, as evidenced by the work by Solworth and Sloan [26], which was published at the IEEE Symposium on Security and Privacy in 2004. In that work, the authors assert that "in general", safety is undecidable in DAC, and use this assertion as the motivation for introducing a new access control scheme based on labels and relabelling that has decidable safety properties.

Our goals in this paper are to present a clear picture of safety in DAC and to refute several erroneous claims in Solworth and Sloan [26]. The work in Solworth and Sloan [26] is based on the myth that "safety is undecidable in DAC; therefore, one needs to design new schemes for DAC so that safety analysis is decidable". We conjecture that the basis for this myth is that DAC is sometimes erroneously equated to the HRU scheme [11] (for instance, in work such as [17, 21]). As we discuss in Section 3, DAC cannot be equated to HRU for the following reasons. First, the HRU scheme can be used to encode schemes that are not DAC schemes; therefore, the fact that safety is undecidable in the HRU scheme should not lead one to conclude that safety is undecidable in DAC. Second, features in DAC cannot always be encoded in the HRU scheme. For example, some DAC schemes require that each object be owned by exactly one other subject; thus removal of a subject who has the ownership of some objects requires the transfer of ownership to some other subject (often times the owner of the subject being removed) so that this property is maintained. Both the removal of the subject and the transfer

of ownership of objects it owns occur in a single state-change. A single HRU command cannot capture these features, because it cannot loop over all objects owned by a subject.

We dispel the myth that safety is undecidable in DAC by presenting an efficient algorithm for deciding safety in the DAC scheme proposed by Graham and Denning [9]. Our algorithm runs in time cubic in the size of the input. The Graham-Denning scheme is, to our knowledge, the first DAC scheme to have been proposed, and several other DAC schemes proposed subsequently are either subsumed by or are simple extensions of the Graham-Denning scheme. Examples of such DAC schemes include those used in Osborn et al. [20] to show that RBAC can be used to implement DAC. The same schemes were used in Solworth and Sloan [26] to show that the Solworth-Sloan scheme can implement DAC. Our algorithm suggests that safety in these DAC schemes can be efficiently decided and there is no need to invent new access control schemes.

Some may hold the view that safety can be trivially decided in DAC schemes. For instance, if the owner of an object is untrusted, then he can grant rights over the object to any other subject. Therefore, if such an owner exists, then the system will be unsafe for that object. While it may be easy to identify one or two such conditions that make a DAC system unsafe, identifying all such conditions may not be trivial. To our knowledge, algorithms for deciding safety in the Graham-Denning or other derived DAC schemes have not appeared in the literature before. The proof that our algorithm is correct, which is in Appendix A, was not trivial for us.

We have also developed an algorithm for deciding safety in the DAC scheme developed Griffiths and Wade [10] in the context of database access control. This scheme is the basis for DAC schemes used in most relational database systems currently in use. Owing to space limitations, we are unable to include a detailed description and analysis of the Griffiths-Wade scheme, but present them separately in a technical report [8]. We summarize our results for the Griffiths-Wade scheme in Section 5. We point out that safety analysis is this scheme is more involved than in the Graham-Denning scheme. Our conclusion is that safety can be decided in time quartic in the size of the input for the Griffiths-Wade scheme.

We also refute several erroneous claims in Solworth and Sloan [26], in which the authors claim:

> "We note that ours is the first general access control model which both has a decidable safety property and is able to implement the full range of DAC models."

We show that the proposed implementation of DAC schemes in the Solworth-Sloan scheme is incorrect. Two particular limitations that we discuss are the lack of support for removing subjects and objects and the inability to ensure that an object has only one owner, as required by DAC schemes such as Strict DAC with Change of Ownership (SDCO), which is a simplified version of the Graham-Denning scheme.

We observe that the presentation in [26] does not clearly specify what information is maintained in a state, how states may change, and the precise construction to implement DAC in their scheme. Many details are scattered in the paper and need to be inferred from descriptions in several places. This makes the understanding of the scheme and the study of implementation of DAC in this scheme very difficult. In this paper we give a precise characterization of the Solworth-Sloan scheme and an "implementation" of the SDCO scheme [20] in it. We observe the "implementation" incurs considerable overhead. Essentially for each new object to be created, a data structure of the size at least as large as the total number of subjects needs to be created. Furthermore, the "implementation" is incorrect as it does not preserve the property that every object has only one owner in every state. We believe that a precise characterization of the Solworth-Sloan scheme is of independent interest. The publication of two papers [26, 27] based on this scheme in recent major security conferences reflects that there is interest in such a access control scheme based on labels and relabelling.

The rest of this paper is organized as follows. We discuss related work in Section 2 and give precise definitions of safety analysis in DAC in Section 3. In Section 4, we study safety analysis in the Graham-Denning scheme. In Section 5, we briefly summarize our results on safety analysis for the Griffiths-Wade scheme. We analyze the Solworth-Sloan scheme in Section 6 and conclude in Section 7.

2 Related Work

There is considerable work on DAC and safety analysis. To our knowledge, Graham and Denning [9] proposed the first DAC scheme. Their scheme is based on the work by Lampson on the access matrix model [13]. Subsequently, Griffiths and Wade proposed their DAC scheme for relational database systems [10]. Downs et al. [7] discussed salient aspects of DAC, and their work was subsequently subsumed by the NCSC's guide to DAC [6]. In her work on issues in access control, Lunt [17] examined various issues in DAC as well. Samarati and de Capitani di Vimercati [21] included discussions on DAC in their treatment of access control. Osborn et al. [20] discussed several DAC schemes that are sub-cases or variants of the Graham-Denning scheme in their comparison of DAC to RBAC. DAC was extended to include temporal constructs by Bertino et al. [3, 4]. Solworth and Sloan [26] presented a new DAC scheme based on labels and relabelling rules. The same scheme was also used by Solworth and Sloan in [27].

Safety is a fundamental property that was first proposed in the context of access control by Harrison et al. [11]. As we mention in the previous section, subsequently, there has been considerable work on safety in various contexts related to security [1, 2, 5, 12, 14, 15, 16, 18, 19, 22, 23, 24, 25, 26, 28, 29]. Recent work by Li et al. [14, 15] perceived various forms of safety as special cases of more general security properties, and safety analysis is subsumed by security analysis. In this paper, we adopt this perspective in defining safety analysis in the next section. To our knowledge, the work by Solworth and Sloan [26] is the first to directly address safety in DAC. Other work on safety has been on specific schemes such as the HRU scheme [11], the ESPM scheme [1] and a trust management scheme [15]. Furthermore, to our knowledge, there is no prior work on safety analysis in the context of specific DAC schemes such as the Graham-Denning scheme [9] and the Griffiths-Wade scheme [10].

3 Defining Safety Analysis in DAC

In this section, we define access control schemes and systems, and the general problem of security analysis in the context of such schemes and systems. We then define safety analysis as a special case of security analysis. In our definitions, we adopt the meta-formalism introduced by Li et al. [15, 14].

Definition 1 (Access Control Schemes and Systems) An access control scheme is a four-tuple $\langle \Gamma, \Psi, Q, \vdash \rangle$, where Γ is a set of states, Ψ is a set of state-change rules, Q is a set of queries and $\vdash\colon \Gamma \times Q \rightarrow \{true, false\}$ is the entailment function, that specifies whether a propositional logic formula of queries is true or not in a state.

A state-change rule, $\psi \in \Psi$, determines how the access control system changes state. Given two states γ and γ_1 and a state-change rule ψ, we write $\gamma \mapsto_\psi \gamma_1$ if the change from γ to γ_1 is allowed by ψ, and $\gamma \overset{*}{\mapsto}_\psi \gamma_1$ if a sequence of zero or more allowed state changes leads from γ to γ_1.

An access control system based on a scheme is a state-transition system specified by the four-tuple $\langle \gamma, \psi, Q, \vdash \rangle$, where $\gamma \in \Gamma$ is the start (or current) state, and $\psi \in \Psi$ specifies how states may change.

We recognize that our formalism for schemes and systems is fairly abstract. Nonetheless, we need such an formalism to be able to represent disparate access control schemes, such as those based on the access matrix, role-based access control and trust management. When we specify a particular access control scheme, we specify each component precisely, using constructs that are well-understood.

An example of an access control scheme is the HRU scheme [11], in which the state is maintained in an access matrix. Examples of queries, $q_1, q_2 \in Q$ in the HRU scheme are "$q_1 = r \in M[s, o]$" and "$q_2 = r' \in M[s, o]$". The queries q_1 and q_2 ask whether the subject s has the right r and r' over the object o, respectively. Given a state, γ, and a state-change rule, ψ, in an HRU system, let S_γ be the set of subjects that exist in the state, γ, O_γ be the set of objects that exist, $M_\gamma[\,]$ be the access matrix, and R_ψ be the set of rights in the system. Then, $\gamma \vdash q_1 \wedge \neg q_2$ if and only if $s \in S_\gamma \wedge o \in O_\gamma \wedge r \in M_\gamma[s, o] \wedge r' \notin M_\gamma[s, o]$.

Definition 2 (Security Analysis) Given an access control scheme $\langle \Gamma, \Psi, Q, \vdash \rangle$, a security analysis instance is of the form $\langle \gamma, \psi, \Box\phi \rangle$, where ϕ is a propositional logic formula of queries. Given such an instance, we say that the instance is true if for all states γ' such that $\gamma \overset{*}{\mapsto}_{\psi} \gamma'$, $\gamma' \vdash \phi$. That is, ϕ represents a security invariant that must be satisfied in all states reachable from γ under ψ for the instance to be true. Otherwise, the instance is false.

Harrison et al. [11] informally characterize safety as the condition "that a particular system enables one to keep one's own objects 'under control' ". This informal characterization seems to be appropriate as a security property of interest in DAC systems, as the very purpose of DAC is that subjects should be able to keep objects that they own, under their control. More formally, safety analysis is a special case of the security analysis, where the invariant is such that an unauthorized subject should not have a particular right to a given object.

Definition 3 (Safety Analysis) Given an access control scheme $\langle \Gamma, \Psi, Q, \vdash \rangle$, let the set of subjects that can exist in a system based on the scheme be $\mathcal{S}$, let the set of objects be $\mathcal{O}$, and let the set of rights be $\mathcal{R}$. Assume that there exists a function $\mathsf{hasRight}$: $\mathcal{S} \times \mathcal{O} \times \mathcal{R} \rightarrow \{true, false\}$ that returns $true$ if in the current state, s and o exist, r is a right in the system, and s has the right r over o, and false otherwise. A safety analysis instance is $\langle \gamma, \psi, \Box\neg\mathsf{hasRight}(s, o, r) \rangle$ for some $s \in \mathcal{S}$, $o \in \mathcal{O}$ and $r \in \mathcal{R}$. That is, safety analysis is security analysis with ϕ instantiated to $\neg\mathsf{hasRight}(s, o, r)$. The safety analysis instance is true if $\mathsf{hasRight}(s, o, r)$ is false in any reachable state, and true otherwise.

What is DAC? The NCSC guide titled 'A Guide To Understanding Discretionary Access Control in Trusted Systems' [6], portions of which were published as a research paper [7], states that "the basis for (DAC) is that an individual user, or program operating on the user's behalf, is allowed to specify explicitly the types of access other users (or programs executing on their behalf) may have to information under the user's control." We point out two specific properties from this characterization of DAC: (1) The notion of "control" – there is a notion that users exercise control over resources in that a user that controls a resource gets to dictate the sorts of rights other users have over the resource, and (2) the notion of initiation of an action by a user to change the protection state – such state changes occur because particular users initiate such changes. A representation of a DAC scheme needs to capture both these properties.

Some literature (for example, [17, 21]) appears to equate DAC with the HRU scheme [11]. This is incorrect, as there exist some systems based on the HRU scheme that are not DAC systems. For instance, consider an HRU system in which there is only one command, and that command has no condition. This system is not a DAC system as it does not have the first property from above on the control of resources by a subject. In addition, there are DAC schemes that do not have natural representations as HRU schemes. For instance, the Graham-Denning scheme [9] (see Section 4.1) is a DAC scheme in which a subject may be 'owned' or 'controlled' by at most one other subject. An system based on the HRU scheme cannot capture this feature in a natural way.

Trusted subjects in safety analysis In considering the safety property discussed above, each instance of the analysis is associated with a set $\mathcal{T}$ of trusted subjects. The meaning of a trusted subject is that we preclude state-changes initiated by any subject from $\mathcal{T}$ in our analysis. The intuition is that we expect these subjects to be "well-behaved". That is, while such subjects may effect state-changes, they do so in such a way that the state that results from the state-changes they effect is safe. Harrison et al. [11] do consider trusted subjects as part of their safety analysis. Nonetheless, as pointed out previously by Li et al. [15], the way they deal with trusted subjects is incorrect. They require that we delete the rows and columns corresponding to trusted subjects prior to the analysis. While a trusted subject is not allowed to initiate a state-change, she may be used as an intermediary, and the way Harrison et al. [11] deal with trusted subjects does not consider this possibility. In this paper, we require only that a member of the set of trusted subjects not initiate a state-change. In all other ways, these subjects continue to be part of the system.

4 Safety Analysis in the Graham and Denning Scheme

In this section, we study safety analysis in the Graham-Denning DAC scheme [9]. We first present a description of the scheme in the following section. Our description clearly describes the states and state-change rules in the scheme. In Section 4.2, we present a correct algorithm to decide safety in the scheme. We also assert that the algorithm is efficient.

4.1 The Graham-Denning Scheme

In this section, We present a precise representation for the Graham-Denning scheme. We define what data are stored in a protection state, and how a state-change rule changes a state.

States, Γ We postulate the existence of the following countably infinite sets: $\mathcal{O}$, the set of objects; $\mathcal{S}$, the set of subjects ($\mathcal{S} \subset \mathcal{O}$); and $\mathcal{R}$, the set of rights.

Note that the set of objects (or subjects) in any given state is finite; however, the number of objects that could be added in some future state is unbounded. Similarly, the set of rights in any given access control system is finite; however, different access control systems may use different set of rights. Therefore, we assume $\mathcal{S}$, $\mathcal{O}$, and $\mathcal{R}$ are countably infinite.

We assume a naming convention so that we can determine, in constant time, whether a given object, o, is a subject (i.e., $o \in \mathcal{S}$) or not (i.e., $o \in \mathcal{O} - \mathcal{S}$). There exists a special "universal subject" $\mathcal{U}$ in $\mathcal{S}$; the role of $\mathcal{U}$ will be explained later. The set of rights $\mathcal{R}$ contains two special rights own and $control$, a countably infinite set $\mathcal{R}_b$ of "basic" rights, and a countably infinite set $\mathcal{R}_b^*$ of basic rights with the copy flag, i.e., $\mathcal{R}_b^* = \{r^* | r \in \mathcal{R}_b\}$. In other words, $\mathcal{R} = \{own, control\} \cup \mathcal{R}_b \cup \mathcal{R}_b^*$. The meaning of the copy flag is clarified when we discuss the state-change rules for the scheme. An access control system based on the Graham-Denning scheme is associated with a protection state, and a state-change rule.

A state in the Graham-Denning scheme, γ, is associated with the tuple $\langle O_\gamma, S_\gamma, M_\gamma[\,]\rangle$, where $O_\gamma \subset \mathcal{O}$ is a finite set of objects that exist in the state γ, $S_\gamma \subset \mathcal{S}$ is a finite set of subjects that exist in γ, and S_γ is a subset of O_γ. $M_\gamma[\,]$ is the access matrix, and $M_\gamma[\,]$: $S_\gamma \times O_\gamma \rightarrow 2^{\mathcal{R}}$. That is, $M_\gamma[s, o] \subset \mathcal{R}$ is the finite set of rights the subject $s \in S_\gamma$ has over the object $o \in O_\gamma$.

Every state, $\gamma = \langle O_\gamma, S_\gamma, M_\gamma[\,]\rangle$, in the Graham-Denning scheme satisfies the following seven properties.

1. Every object must be owned by at least one subject, i.e., $\forall\, o \in O_\gamma\, \exists s \in S_\gamma(own \in M_\gamma[s, o])$.
2. Objects are not controlled, only subjects are, i.e., $\forall\, o \in O_\gamma - S_\gamma \forall\, s \in S_\gamma(control \notin M_\gamma[s, o])$.
3. The special subject $\mathcal{U}$ exists in the state, is not owned by any subject, and is not controlled by any other subject, i.e., $\mathcal{U} \in S_\gamma \wedge \forall\, s \in S_\gamma(own \notin M_\gamma[s,\mathcal{U}]) \wedge \forall\, s \in S_\gamma - \{\mathcal{U}\}(control \notin M_\gamma[s,\mathcal{U}])$.
4. A subject other than $\mathcal{U}$ is owned by exactly one other subject, i.e., for every $s \in S_\gamma - \{\mathcal{U}\}$, there exists exactly one $s' \in S_\gamma$ such that $own \in M_\gamma[s', s]$;
5. Every subject controls itself, i.e., $\forall s \in S_\gamma(control \in M_\gamma[s, s])$.
6. A subject other than $\mathcal{U}$ is controlled by at most one other subject, i.e., for every $s \in S_\gamma - \{\mathcal{U}\}$, there exists at most one $s' \in S_\gamma$ such that $s' \neq s$ and $control \in M_\gamma[s', s]$.
7. There exists no set of subjects such that they form a "cycle" in terms of ownership of each other (and in particular, a subject does not own itself), i.e., $\neg(\exists\, \{s_1, \ldots, s_n\} \subseteq S_\gamma(own \in M_\gamma[s_2, s_1] \wedge own \in M_\gamma[s_3, s_2] \wedge \cdots \wedge own \in M_\gamma[s_n, s_{n-1}] \wedge own \in M_\gamma[s_1, s_n]))$.

These state invariants are maintained by the state-change rules.

State-Change Rules, Ψ Each member, ψ, of the set of state-change rules, Ψ, in the Graham-Denning scheme, is a set of commands parameterized by a set of rights, R_ψ. These commands are shown in Figure 1. Where possible, we use the syntax for commands from the HRU scheme [11], but as we mention in Section 3, we cannot represent all aspects of DAC schemes using only constructs from commands in the HRU scheme. We use some additional well-known constructs such as $\forall$ and $\exists$ in these commands. A state-change is the successful execution of one of the commands. We assume that the state subsequent to the execution of a command is γ'. We denote such a state-change as $\gamma \mapsto_{\psi(s)} \gamma'$, where s is the initiator of the command. We point out that for each command, unless specified otherwise, $S_{\gamma'} = S_\gamma$, $O_{\gamma'} = O_\gamma$, and $M_{\gamma'}[s, o] = M_\gamma[s, o]$ for every $s \in S_\gamma$ and $o \in O_\gamma$. We use $\leftarrow$ to denote assignment, i.e., $a \leftarrow b$ means that the value in a is replaced with the value in b. The commands in the Graham-Denning scheme are the following.

- transfer_r(i, s, o) This command is used to grant the right r by an initiator that has the right r^* over o. There is one such command for every $r \in R_\psi \cap \mathcal{R}_b$. The initiator, i, must possess the right r^* over o, and the subject s must exist for this command execution to succeed.
- transfer_r*(i, s, o) This command is used to grant the right r^* by an initiator that has the right r^* over o. There is one such command for every $r^* \in R_\psi \cap \mathcal{R} * b^*$. The initiator, i, must possess the right r^* over o, and the subject s must exist for this command execution to succeed.
- transfer_own(i, s, o) This command is used to transfer ownership over o from i to s. For this command to succeed, i must have the own right over o, s must exist, and the transfer of ownership must not violate invariant (7) from the list of state invariants we discuss above. After the execution of this command, i will no longer have the own right over o (but s will).
- grant_r(i, s, o) This command is used to grant the right r over o by the owner of o. There is one such command for every $r \in R_\psi \cap \mathcal{R}_b$. For this command execution to succeed, i must have the own right over o, and s must exist.
- grant_r*(i, s, o) This command is very similar to the previous command, except the the owner grants $r^* \in R_\psi \cap \mathcal{R}_b^*$.
- grant_control(i, s, o) This command is used to grant the $control$ right over o by its owner. For the execution of this command to succeed, i must have the right $control$ over o, s must exist, o must be a subject, and another subject must not already have the right $control$ over o. These checks are needed to maintain the state invariants related to the $control$ right that we discuss above.
- grant_own(i, s, o) This command is used to grant the own right over o. This is different from the transfer_own command in that in this case, i retains (joint) ownership over o. For the execution of this command to succeed, i must have the right own over o, o must not be a subject, and s must exist.
- delete_r(i, s, o) This command is used to delete a right a subject has over o. There is one such command for every $r \in R_\psi \cap \mathcal{R}_b$. For the execution of this command to succeed, i must have the right own over o, and s must exist.
- delete_r*(i, s, o) This command is similar to the previous command, except that a right $r^* \in R_\psi \cap \mathcal{R}_b^*$ is deleted.
- create_object(i, o) This command is used to create an object that is not a subject. For the execution of this command to succeed, i must exist, and o must be an object that is not a subject, that does not exist. An effect of this command is that i gets the own right over o in the new state.
- destroy_object(i, o) This command is used to destroy an object that exists. For the execution of this command to succeed, i must have the right own over o, and o must be an object that is not a subject.
- create_subject(i, s) This command is used to create a subject. For the execution of this command to succeed, i must exist, and s must be a subject that does not exist. In the new state, i has the own right over s, and s has the $control$ right over itself.

- destroy_subject(i, s) This command is used to destroy a subject. For the execution of this command to succeed, i must have the own right over s. An effect of this command is that ownership over any object owned by s is transferred to i.

4.2 Safety analysis

An algorithm to decide whether a system based on the Graham-Denning scheme is safe is shown in Figure 2. A system based on the Graham-Denning scheme is characterized by a start-state, γ, and state-change rule, ψ (which is a set of commands). The algorithm takes as input γ, ψ, a triple, $\omega = \langle s, o, x \rangle \in \mathcal{S} \times \mathcal{O} \times \mathcal{R}$, and a finite set, $\mathcal{T} \subset \mathcal{S}$, of trusted subjects. The algorithm outputs "true" if the system satisfies the safety property with respect to the subject s, object o and right x, and "false" otherwise. We first discuss the algorithm, and then its correctness and time-complexity.

In lines 5-8 of the algorithm, we check the cases for which we do not have to consider potential state-changes before we are able to decide whether the system is safe or not. In line 7, we check that the right x is indeed in the system. In line 8, we check whether we are being asked whether s can get the $control$ right over o, where o is an object that is not a subject (we know s does not have and cannot get the right, by property (2) of the six properties we discuss in the previous section). In line 9, we check whether the right x has already been acquired by s over o. In line 10, we check that if the right y has already been acquired by s over o (the check in line 10 is needed when $x \in \mathcal{R}_b$, as then, the possession of x^* implies the possession of x; in the case that $x \in \mathcal{R}_b^*$, the lines 9 and 10 are identical). When $x = own$ or $x = control$, the condition of line 10 will never be true, and we will not return from that line. In the remainder of the algorithm, we consider those cases in which a state-change is needed before s can get x over o (if it can at all). In line 11, we check whether there is at least one subject that can initiate state-changes, and if not, we know that the system is safe. In line 12, we check whether o exists, and if it does not, given that there exists a subject that can create o (from our check in line 11), the subject can then grant x to s over o. In line 13, we check whether there is a subject that can initiate state-changes, and that has x with the copy-flag (or x itself, if $x \in \mathcal{R}_b^*$). If $x = own$ or $x = control$, the condition of line 13 cannot be true. In lines 14-16, we check whether there is a sequence of subjects with the particular property that each owns the next in the sequence, and the last subject in the sequence owns o. If any one of those subjects can initiate state-changes, then we conclude that the system is not safe and return false. In all other cases, we conclude that the system is safe, and return true.

The following lemma asserts that the algorithm is correct. Theorem 2 summarizes our results with respect to safety analysis in the Graham-Denning scheme.

Lemma 1 *A system based on the Graham-Denning scheme, that is characterized by the start-state, γ, and state-change rule, ψ, is safe with respect to $\omega = \langle s, o, x \rangle$ and $\mathcal{T} \subset \mathcal{S}$ (where $\mathcal{T}$ is finite) if and only if* isSafeGD$(\gamma, \psi, \omega, \mathcal{T})$ *returns true.*

Proof. Sketch: the proof is quite lengthy, and we present it in Appendix A. We present a sketch of the proof here. For the "if" part, we need to show that if the system is not safe with respect to ω and $\mathcal{T}$, then isSafeGD returns false on input $(\gamma, \psi, \omega, \mathcal{T})$. If the system is not safe, then we know that there exists a state-change sequence $\gamma \vdash_{\psi(s_1)} \gamma_1 \vdash_{\psi(s_2)} \cdots \vdash_{\psi(s_n)} \gamma_n$, such that $x \in M_{\gamma_n}[s, o]$. If such a sequence exists with $n = 0$, then this can only be because s already has the right, and we show that in this case the algorithm returns false. If $n = 1$, then the right has to appear in $M_{\gamma_1}[s, o]$ in only one state-change, and we show that in this case as well, the algorithm returns false. For the general case, we use induction on n, with $n = 1$ as the base case.

For the "only if" part, we need to show that if the algorithm returns false, then the system is not safe with respect to ω and $\mathcal{T}$. We consider each case in which the algorithm returns false (lines 9, 10, 12, 13 and 16). In each case, we construct a state-change sequence such that in the final state of the sequence, γ', $x \in M_{\gamma'}[s, o]$. ∎

$command$ transfer_r(i, s, o)
  $if\ r^* \in M_\gamma[i, o] \wedge s \in S_\gamma\ then$
    $M_{\gamma'}[s, o] \leftarrow M_\gamma[s, o] \cup \{r\}$

$command$ transfer_own(i, s, o)
  $if\ own \in M_\gamma[i, o] \wedge o \in S_\gamma \wedge s \in S_\gamma\ then$
    $if\ \nexists \{s_1, \ldots, s_n\} \in S_\gamma\ such\ that$
    $own \in M_\gamma[s_1, s] \wedge own \in M_\gamma[s_2, s_1]$
    $\wedge \cdots \wedge own \in M_\gamma[s_n, s_{n-1}]$
    $\wedge\ own \in M_\gamma[o, s_n]\ then$
      $M_{\gamma'}[s, o] \leftarrow M_\gamma[s, o] \cup \{own\}$
      $M_{\gamma'}[i, o] \leftarrow M_\gamma[i, o] - \{own\}$

$command$ grant_control(i, s, o)
  $if\ own \in M_\gamma[i, o] \wedge o \in S_\gamma \wedge s \in S_\gamma\ then$
    $if \nexists\ s' \in S_\gamma\ such\ that$
    $s' \neq o \wedge control \in M_\gamma[s', o]\ then$
      $M_{\gamma'}[s, o] \leftarrow M_\gamma[s, o] \cup \{control\}$

$command$ delete_r(i, s, o)
  $if\ \ (own \in M_\gamma[i, o] \wedge s \in S_\gamma)$
  $\vee\ control \in M_\gamma[i, s]\ then$
    $M_{\gamma'}[s, o] \leftarrow M_\gamma[s, o] - \{r\}$

$command$ create_object(i, o)
  $if\ o \notin O_\gamma \wedge i \in S_\gamma \wedge o \in \mathcal{O} - \mathcal{S}\ then$
    $O_{\gamma'} \leftarrow O_\gamma \cup \{o\}$
    $M_{\gamma'}[i, o] \leftarrow own$

$command$ create_subject(i, s)
  $if\ s \notin O_\gamma \wedge i \in S_\gamma \wedge s \in \mathcal{S}\ then$
    $O_{\gamma'} \leftarrow O_\gamma \cup \{s\}$
    $S_{\gamma'} \leftarrow S_\gamma \cup \{s\}$
    $M_{\gamma'}[i, s] \leftarrow \{own\}$
    $M_{\gamma'}[s, s] \leftarrow \{control\}$

$command$ transfer_r$^*(i, s, o)$
  $if\ r^* \in M_\gamma[i, o] \wedge s \in S_\gamma\ then$
    $M_{\gamma'}[s, o] \leftarrow M_\gamma[s, o] \cup \{r^*\}$

$command$ grant_r(i, s, o)
  $if\ own \in M_\gamma[i, o] \wedge s \in S_\gamma\ then$
    $M_{\gamma'}[s, o] \leftarrow M_\gamma[s, o] \cup \{r\}$

$command$ grant_r$^*(i, s, o)$
  $if own \in M_\gamma[i, o] \wedge s \in S_\gamma\ then$
    $M_{\gamma'}[s, o] \leftarrow M_\gamma[s, o] \cup \{r^*\}$

$command$ grant_own(i, s, o)
  $if\ own \in M_\gamma[i, o] \wedge o \notin S_\gamma$
  $\wedge\ s \in S_\gamma\ then$
    $M_{\gamma'}[s, o] \leftarrow M_\gamma[s, o] \cup \{own\}$

$command$ delete_r$^*(i, s, o)$
  $if\ \ (own \in M_\gamma[i, o] \wedge s \in S_\gamma)$
  $\vee\ control \in M_\gamma[i, s]\ then$
    $M_{\gamma'}[s, o] \leftarrow M_\gamma[s, o] - \{r^*\}$

$command$ destroy_object(i, o)
  $if\ own \in M_\gamma[i, o] \wedge o \notin S_\gamma\ then$
    $O_{\gamma'} \leftarrow O_\gamma - \{o\}$

$command$ destroy_subject(i, s)
  $if\ own \in M_\gamma[i, s] \wedge s \in S_\gamma\ then$
    $\forall\ o \in O_\gamma, if\ own \in M_\gamma[s, o]\ then$
      $M_{\gamma'}[i, o] \leftarrow M_\gamma[i, o] \cup \{own\}$
    $O_{\gamma'} \leftarrow O_\gamma - \{s\}$
    $S_{\gamma'} \leftarrow S_\gamma - \{s\}$

Figure 1: The set of commands that constitutes the state-change rule, ψ, for a system based on the Graham-Denning scheme. Each command has a name (e.g., transfer_own), and a sequence of parameters. The first parameter is always named i, and is the initiator of the command, i.e., the subject that executes the command. There is one transfer_r, grant_r, and delete_r command for each $r \in R_\psi \cap \mathcal{R}_b$, and one transfer_r*, grant_r*, and delete_r* command for each $r^* \in R_\psi \cap \mathcal{R}_b^*$.

```
 1 Subroutine isSafeGD(γ, ψ, ω, T)
 2   /* inputs: γ,  ψ,  ω = ⟨s, o, x⟩,  T ⊆ S  */
 3   /* output: true or false */
 4   if x ∈ R*_b then let y ← x
 5   else if x ≠ own ∧ x ≠ control then let y ← x*
 6   else let y ← invalid /* No copy flags for own or control */
 7   if x ∉ R_ψ then return true
 8   if x = control ∧ o ∈ O − S then return true
 9   if x ∈ M_γ[s, o] then return false
10   if y ∈ M_γ[s, o] then return false
11   if T ⊇ S_γ then return true
12   if o ∉ O_γ then return false
13   if ∃ŝ ∈ S_γ − T such that y ∈ M_γ[ŝ, o] then return false
14   for each sequence U, s_n, ..., s_2, s_1 such that
15   own ∈ M_γ[s_1, o] ∧ ··· ∧ own ∈ M_γ[s_n, s_{n−1}] ∧ own ∈ M_γ[U, s_n] do
16     if ∃s_i ∈ {s_1, ..., s_n} such that s_i ∈ S_γ − T then return false
17   return true
```

Figure 2: The subroutine isSafeGD returns "true" if the system based on the Graham-Denning scheme, characterized by the start-state, γ, and state-change rule, ψ, satisfies the safety property with respect to ω and $\mathcal{T}$. Otherwise, it returns "false". In line 6, we assign some invalid value to y, as there is not corresponding right with the copy flag for the rights own and $control$. In this case, the algorithm will not return in line 10 or 13.

Theorem 2 *Safety is efficiently decidable in a system based on the Graham-Denning scheme. In particular,* isSafeGD *runs in time at worst cubic in the size of the components of the start state and the set of rights in the system.*

Proof. We make the following observations about the running time of isSafeGD in terms of its input, namely, $S_\gamma, O_\gamma, R_\psi$, $M_\gamma[\,]$, ω and $\mathcal{T}$, by considering each line in the algorithm as follows. Each of the lines 5-10 runs in time at worst linear in the size of the input. In particular, as we mention in the previous section, we adopt a naming convention for subjects and objects that enables us to perform the check $o \in \mathcal{O} - \mathcal{S}$ in line 8, in constant time. Line 11 runs in time at worst quadratic in the size of the input ($|S_\gamma| \times |\mathcal{T}|$), line 12 runs in time at worst linear ($|O_\gamma|$), and line 13 runs in time at worst quadratic ($|S_\gamma| \times |R_\psi|$). As each subject is owned only by one other subject, each sequence to which line 14 refers is of size at most $|S_\gamma|$. Furthermore, there are at most $|S_\gamma|$ such sequences. Therefore, lines 14-16 run in time at worst cubic in the size of the input. The fact that isSafeGD$(\gamma, \psi, \omega, \mathcal{T})$ runs in time polynomial in the size of the input in conjunction with Lemma 1 proves our assertion. ∎

5 The Griffiths-Wade Scheme

Griffiths and Wade [10] present a DAC scheme for relational database systems. Their scheme is different from the Graham-Denning scheme in important respects. A key difference is that in the Griffiths-Wade scheme, we have three kinds of objects, base relations, views and rows, and safety analysis must consider all three kinds of objects. Furthermore, rights over views depend on rights over base relations, and rights over rows depend on rights over relations and views.

As we mention in Section 1, owing to space limitations, we are unable to include a detailed description and analysis of the Griffiths-Wade scheme. We include these details in [8], and present only the results here. The algorithm to decide safety in the Griffiths-Wade scheme (in [8]) is more involved that the algorithm for

deciding safety in the Graham-Denning scheme (in Section 4.2), but we adopt a similar strategy in its proof of correctness.

Theorem 3 *Safety is efficiently decidable in a system based on the Griffiths-Wade scheme. In particular, there exists an algorithm that returns true if a system based on the Griffiths-Wade scheme is safe and false otherwise, and the algorithm runs in time quartic in the size of the components of the start state.*

6 The Solworth-Sloan Scheme, Revisited

Solworth and Sloan [26] present a new DAC scheme based on labels and relabelling rules, and we call it the Solworth-Sloan scheme. While the presentation in [26] does not clearly specify what information is maintained in a state and how states may change, we were able to infer what is intended after considerable effort.

In this section, we give a precise characterization of the Solworth-Sloan scheme as a state transition system. Our objective in doing so is to represent the Solworth-Sloan scheme sufficiently precisely to enable comparisons to other DAC schemes. In particular, our intent is to assess the mapping of DAC schemes to the Solworth-Sloan scheme that is discussed by Solworth and Sloan [26]. Solworth and Sloan [26] refer to the DAC schemes discussed by Osborn et al. [20] and assert that "...we present a general access control model which is sufficiently expressive to implement each of these DAC models..." In this section, we show that this claim is incorrect.

We reiterate that the DAC schemes discussed by Osborn et al. [20] are either subsumed by, or are minor extensions of the Graham-Denning scheme that we discuss in Section 4. We have shown in Section 4.2 that safety is efficiently decidable in the Graham-Denning scheme, and our algorithm can be used with relatively minor modifications to decide safety in these schemes. Thereby, Solworth and Sloan's [26] other assertion in reference to the DAC schemes discussed by Osborn et al. [20], that "...every published general access control model...either is insufficiently expressive to represent the full range of DACs or has an undecidable safety problem...", has been rendered invalid.

6.1 The Solworth-Sloan Scheme

Overview There exists the following countably infinite sets of constants: a set $\mathcal{S}$ of subjects, a set $\mathcal{O}$ of objects, a set $\mathcal{R}$ of rights, a set $\mathcal{G}$ of groups, a set $\mathcal{T}^o$ of object tags, and a set $\mathcal{T}^g$ of group tags. An *object label* is a pair $\langle s, t\rangle$, where $s \in \mathcal{S}$ is a subject and $t \in \mathcal{T}^o$ is a object tag.

Which rights a subject has over a particular object are determined indirectly in the following three steps.

1. There is a labelling function label that assigns an object label to each object.

   An object's label may be changed by object relabelling rules, which determine whether an action rewriting one object label into another succeeds or not. For example, when the object label $\ell_1 = \langle s_1, t_1\rangle$ is relabelled to $\ell_2 = \langle s_2, t_2\rangle$, all objects that originally have the label ℓ_1 now have the label ℓ_2.

2. There is an authorization function auth that decides which rights a group has over a particular object label. For each object label ℓ and each right r, there is one group who has the right r over the label ℓ. Members of the group have right r over objects that are assigned the label ℓ.

3. Which subjects are members of a group is determined by native group sets (NGS's), which are complicated structures that we describe below. We define a function $members$ that maps each group to a set of subjects.

We schematically illustrate the steps to determine whether a subject can access an object or not as follows.

$$\text{objects} \xrightarrow{\mathsf{label}} \text{object labels} \xrightarrow{\mathsf{auth}} \text{groups} \xrightarrow{members} \text{subjects}$$

States, Γ A state, γ, is characterized by a 9-tuple $\langle S_\gamma, O_\gamma, R_\gamma, G_\gamma, L_\gamma, \mathsf{label}_\gamma, \mathsf{auth}_\gamma, \mathsf{ORS}_\gamma, E_\gamma\rangle$.

- S_γ is the set of subjects in the state γ; O_γ is the set of objects in the state γ; R_γ is the set of rights in the state γ, and G_γ is the set of groups in state γ.

  There is a distinguished right wr, which exists in every state, i.e., $\mathsf{wr} \in R_\gamma$.

- $L_\gamma \subset S_\gamma \times \mathcal{T}^o$ is the finite set of object labels in the state γ.

- label_γ: $O_\gamma \longrightarrow L_\gamma$ assigns a unique object label to each object in the current state.

- auth_γ: $(L_\gamma \times \mathcal{R}_\gamma) \longrightarrow G_\gamma$ maps each pair of an object label and a right to a group. For example, $\mathsf{auth}_\gamma[\ell, \mathsf{re}] = g_1$ means that the group g_1 has the re right over all objects labelled ℓ.

- ORS_γ is an ordered sequence of object relabelling rules, each rule has the form of $\mathsf{rl}(p_1, p_2) = h$, where rl is a keyword, and p_1, p_2 are object patterns. An *object pattern* is a pair, where the first element is a subject in $\mathcal{S}$ or one of the three special symbols $*$, $*\mathsf{u}$, and $*\mathsf{w}$, and the second element is an object tag in $\mathcal{T}^o$ or the special symbol $*$. In the rule $\mathsf{rl}(p_1, p_2) = h$, h is a group, a subject, or one of the four following sets: $\{\}, \{*\}, \{*\mathsf{u}\}, \{*\mathsf{w}\}$. When h is $\{*\mathsf{u}\}$ (resp., $\{*\mathsf{w}\}$), $\{*\mathsf{u}\}$ (resp., $\{*\mathsf{w}\}$) must appear in p_1 or p_2.

  For example, the following is an RLS, in which s_1 is a subject, t_1 is an object tag, and g_1 is a group:

  $$\begin{array}{lcl} \mathsf{rl}(\langle *\mathsf{u}, t_1\rangle,\ \langle s_1, *\rangle) & = & g_1 \\ \mathsf{rl}(\langle s_1, *\rangle,\ \langle *\mathsf{u}, t_1\rangle) & = & \{*\} \\ \mathsf{rl}(\langle *\mathsf{u}, *\rangle,\ \langle *\mathsf{u}, *\rangle) & = & \{*\mathsf{u}\} \\ \mathsf{rl}(\langle *\mathsf{u}, *\rangle,\ \langle *\mathsf{w}, *\rangle) & = & \{\} \end{array}$$

- E_γ is a finite set of native group sets (NGS's) that exist in the state, γ. Each $e \in E_\gamma$ is characterized by the 7-tuple $\langle e.G,\ e.T^g,\ e.\mathsf{gtag},\ e.nt^g,\ e.\mathsf{admin},\ e.\mathsf{patterns},\ , e.\mathsf{GRS}\rangle$.

  - $e.G \subseteq G_\gamma$ is the set of groups that are defined in this NGS.
  - $e.T^g \subseteq \mathcal{T}^g$ is the set of group tags that are used in this NGS.
  - The function $e.\mathsf{gtag} : S_\gamma \longrightarrow e.T^g$ assigns a unique tag to each subject in the current state.
  - $e.nt^g$ is a group tag in $e.T^g$; it determines when a new subject is added to the state, which tag is assigned to that subject. That is, if a subject s is added, then $e.\mathsf{gtag}[s]$ would be set to $e.nt^g$.
  - $e.\mathsf{admin}$ points to one NGS in E_γ; it identifies a NGS in the current state as the administrative group set of the NGS e; $e.\mathsf{admin}$ could be e, in which case e is the administrative group set for itself.
  - $e.\mathsf{patterns}$ is a function mapping each group in $e.G$ to a (possibly empty) set of group patterns. Each *group pattern* is a pair where the first element is either a subject in the current state or a special symbol $*\mathsf{u}$, and the second element is a group tag in $e.T^g$. In other words, the set of all group patterns that are can be used in e, denoted by $e.P^g$, is $(S_\gamma \cup \{*\mathsf{u}\}) \times e.T^g$, and the signature of $e.\mathsf{patterns}$ is $e.G \longrightarrow 2^{e.P^g}$, where $2^{e.P^g}$ denote the powerset of $e.P^g$.

    For any group $g \in e.G$, $e.\mathsf{patterns}[g]$ gives a set of patterns for determining memberships of the group. Intuitively, the label $\langle *\mathsf{u}, t^g\rangle$ is in $e.\mathsf{patterns}[g]$ means that any subject who is assigned (via the e.gtag function) the group tag t^g is a member of the group; and the label $\langle s, t^g\rangle$ is in $e.\mathsf{patterns}[g]$ means that the subject s is a member of the group if it is assigned the group tag t^g.
  - e.GRS is a set of group relabelling rules, each has the form $Relabel(t_1^g, t_2^g) = g$, where $Relabel$ is a keyword, $t_1^g, t_2^g \in e.T^g$ are two group tags used in this NGS, and g is a group defined in the administrative group set $e.\mathsf{admin}$ (i.e., $g \in e.\mathsf{admin}.G$).

We define the following auxiliary function $e.members[\,] : e.G \longrightarrow S_\gamma$ such that $e.members[g]$ is the set of all subjects that are members of the group g. A subject $s \in e.members[g]$ if and only if the tag t^g assigned to s (via $e.\mathsf{gtag}$) satisfies the condition that at least one of the two group labels $\langle s, t^g\rangle$ and $\langle *\mathsf{u}, t^g\rangle$ are in the patterns for g, i.e.,

$$\exists\, t^g \in e.T^g\ (\ e.\mathsf{gtag}(s) = t^g\ \wedge\ (\ \langle s, t^g\rangle \in e.\mathsf{patterns}[g]\ \vee\ \langle *\mathsf{u}, t^g\rangle \in e.\mathsf{patterns}[g]\ )\ )$$

An additional constraint on the state γ is that each group is defined in exactly one NGS and each group tag can be used in at most one NGS, i.e.,

$$\forall e_1 \in E_\gamma \forall e_2 \in E_\gamma\ (\ e_1.G \cap e_2.g = \emptyset\ \wedge\ e_1.T^g \cap e_2.T^g = \emptyset\ )$$

State-Change Rules, Ψ There is a single state transition rule ψ in this scheme; ψ consists of six actions that can result in state changes. These actions are mentioned in Section 3.4 of [26] without precise definition. (We break up the "Relabel an object" operation in [26] into two relabelling actions.) We describe the actions and their effects when applying them to a state $\gamma = \langle S_\gamma, O_\gamma, R_\gamma, G_\gamma, L_\gamma, \mathsf{label}_\gamma, \mathsf{auth}_\gamma, \mathsf{ORS}_\gamma, E_\gamma\rangle$. We use γ' to denote the state after the change.

1. $\mathsf{create\_object}(s, o, \ell = \langle s_1, t_1^o\rangle)$: the subject s creates the object o and assigns the object label ℓ to the object o.

   This action succeeds when $s \in S_\gamma$, $o \notin O_\gamma$, $\ell \in L_\gamma$ and the subject s has the wr right on the object label ℓ, i.e., $s \in members[\mathsf{auth}_\gamma(\ell, \mathsf{wr})]$.

   Effects of the action are $O_{\gamma'} = O_\gamma \cup \{o\}$ and the function label is extended so that $\mathsf{label}_{\gamma'}(o) = \langle s_1, t_1^o\rangle$.

2. $\mathsf{create\_label}(s, \ell = \langle s, t_1\rangle, g_1, g_2, \cdots, g_k)$, where $k = |R_\gamma|$ is the number of rights in γ: the subject s creates the new object label ℓ, and assigns the groups $g_1, g_2, \cdots, g_k$ to have the rights over ℓ, .

   This action succeeds when $s \in S_\gamma$, $\ell \notin L_\gamma$, the subject in ℓ is s, and $g_1, \cdots, g_k \in G_\gamma$.

   The effects of this action are follows. Let $r_1, r_2, \cdots, r_k$ be the k rights in R_γ. Then $L_{\gamma'} = L_\gamma \cup \{\ell\}$ and the function auth is extended such that $\mathsf{auth}_{\gamma'}(\ell, r_i) = g_i$ for $1 \le i \le k$.

3. $\mathsf{create\_subject}(s, s')$: the subject s creates a new subject s'.

   This action succeeds when $s \in S_\gamma$ and $s' \notin S_\gamma$.

   The effects of this action are $S_{\gamma'} = S_\gamma \cup \{s'\}$ and for every NGS $e \in E_\gamma$, $e.\mathsf{gtag}$ is extended so that in γ', $e.\mathsf{gtag}(s') = e.nt^g$.

4. $\mathsf{object\_relabel}(s, \ell_1 = \langle s_1, t_1\rangle, \ell_2 = \langle s_2, t_2\rangle)$: the subject s relabels objects having label ℓ_1 to have the label ℓ_2.

   This action succeeds when the first relabelling rule in the object relabelling rule sequence ORS_γ that *matches* (ℓ_1, ℓ_2) is $\mathsf{rl}(p_1, p_2) = h$ and $s \in value[h]$. The rule $\mathsf{rl}(p_1, p_2) = h$ matches (ℓ_1, ℓ_2) when p_1 matches ℓ_1 and p_2 matches ℓ_2 at the same time. When the pattern $\langle *\mathsf{u}, *\rangle$ matches the label $\langle s_1, t_1\rangle$, we say that $*\mathsf{u}$ is unified with the subject s_1. Note that when $*\mathsf{u}$ occurs more than one times in p_1, p_2, they should be unified with the same subject. Recall that h maybe a group g, a subject s', or one of the four sets: $\{\}, \{*\}, \{*u\}, \{*w\}$. The function $value$ is defined as follows: $value[g] = e.members[g]$, where e is the NGS in which g is defined; $value[s'] = \{s'\}$; $value[\{\}] = \emptyset$, $value[\{*\}] = S_\gamma$, $value[\{*\mathsf{u}\}]$ is the subject that is unified with $*\mathsf{u}$.

   Consider the following RLS.

$$\begin{array}{lcl} \mathsf{rl}(\langle *\mathsf{u}, t_1\rangle,\ \langle s_1, *\rangle) & = & g_1 \\ \mathsf{rl}(\langle s_1, *\rangle,\ \langle *\mathsf{u}, t_1\rangle) & = & \{*\} \\ \mathsf{rl}(\langle *\mathsf{u}, *\rangle,\ \langle *\mathsf{u}, *\rangle) & = & \{*\mathsf{u}\} \\ \mathsf{rl}(\langle *\mathsf{u}, *\rangle,\ \langle *\mathsf{w}, *\rangle) & = & \{\} \end{array}$$

The action $\mathsf{object\_relabel}(s, \langle s_2, t_1\rangle, \langle s_1, t_2\rangle)$ would match the first relabelling rule and succeeds when s is a member of the group g_1. The action $\mathsf{object\_relabel}(s, \langle s_1, t_1\rangle, \langle s_2, t_2\rangle)$ would match the second relabelling rule and always succeeds. The action $\mathsf{object\_relabel}(s, \langle s_2, t_2\rangle, \langle s_2, t_1\rangle)$ would match the third relabelling rule and fail, because $*\mathsf{u}$ is unified with s_2. The action $\mathsf{object\_relabel}(s, \langle s_2, t_2\rangle, \langle s_1, t_1\rangle)$ would match the fourth relabelling rule and fail.

The effect of the relabel action is in the function label. For every object o such that $\mathsf{label}_\gamma[o] = \ell_1$, in the new state, $\mathsf{label}_{\gamma'}[o] = \ell_2$.

5. $\mathsf{group\_tag\_relabel}(s, s', t_1^g, t_2^g)$: the subject s relabels the group tag for the subject s' from t_1^g to t_2^g.

   This action succeeds when there is an NGS $e \in E_\gamma$ such that t_1^g and t_2^g are used in e, the subject s' has the group tag t_1^g in e, there is a corresponding group relabelling rule in $e.\mathsf{GRS}$, and s is a member of the group that can use the relabelling rule. More precisely, the action succeeds when

$$\exists e \in E_\gamma \left(e.\mathsf{gtag}[s'] = t_1^g \ \wedge\ \text{“}\mathit{Relabel}(t_1^g, t_2^g) = g\text{”} \in e.\mathsf{GRS} \ \wedge\ s \in e.\mathit{members}[g] \right)$$

   Note that the tags t_1^g and t_2^g can appear only in one NGS and they must appear in the same NGS for the action to succeed. The effect of this action is such that the function $e.\mathsf{gtag}$ is changed such that in γ', $e.\mathsf{gtag}[s'] = t_2^g$.

6. $\mathsf{create\_ngs}(s, e)$: the subject s creates a new NGS e.

   To perform this action, one must provide the complete description of a new NGS e, i.e., the 7-tuple $\langle e.G,\ e.T^g,\ e.\mathsf{gtag},\ e.nt^g,\ e.\mathsf{admin},\ e.\mathsf{patterns},\ , e.\mathsf{GRS}\rangle$. For this action to succeed, the groups defined in e and the group tags in e must be new, i.e., they do not appear in any existing NGS's in γ.

   The effects are that $G_{\gamma'} = G_\gamma \cup e.G$ and $E_{\gamma'} = E_\gamma \cup e$.

Given the above state transition rule, we make the following observations. No removal of subjects, objects, labels, or groups is defined. Given a state $\langle S_\gamma, O_\gamma, R_\gamma, G_\gamma, L_\gamma, \mathsf{label}_\gamma, \mathsf{auth}_\gamma, \mathsf{ORS}_\gamma, E_\gamma\rangle$, S_γ (the set of subjects), O_γ (the set of objects), and G_γ (the set of groups) may change as a result of create_subject, create_object, and create_label, respectively. R_γ, the set of rights, is fixed for the system and does not change. G_γ, the set of groups, may change when a new NGS is added by the create_ngs action. The function label_γ: $O_\gamma \longrightarrow L_\gamma$ is extended when a new object is added and is changed when an object relabelling action object_relabel happens. The function auth_γ is extended when a new object label is created; existing assignments do not change. ORS_γ, the object relabelling rule sequence, always stay the same. E_γ is extended when a new NGS is added.

6.2 Encoding a simple DAC scheme in the Solworth-Sloan scheme

In this section, we encode a relatively simple DAC scheme in the Solworth-Sloan scheme. The DAC scheme we consider is a sub-scheme of the Graham-Denning scheme. It is called Strict DAC with Change of Ownership (SDCO) and is one of the DAC schemes discussed by Osborn et al. [20]. Our construction is based on comments by Solworth and Sloan [26] on how various DAC schemes can be encoded in the Solworth-Sloan scheme. As the presentation in that paper is not detailed, we offer a more detailed construction. Our constructions lets us assess the utility of the Solworth-Sloan scheme in encoding SDCO. After we present our encoding, we discuss the overhead introduced by mapping SDCO to the Solworth-Sloan scheme and the correctness of this mapping.

Strict DAC with Change of Ownership (SDCO) As we mention above, SDCO is a sub-scheme of the Graham-Denning scheme (see Section 4.1). In SDCO, there is a distinguished right, own, but no $control$ right. Also, there are no rights with the copy flag. The state-change rules in SDCO are the commands grant_r (for each $r \in R_\psi$), delete_r (for each $r \in R_\psi$), grant_own, create_object and create_subject. We do not consider commands to destroy subjects or objects as their counterparts are not specified for the Solworth-Sloan scheme.

For simplicity, we consider an SDCO scheme that has only three rights own, re, wr. In the Solworth-Sloan scheme, if two objects o_1 and o_2 have the same label, then o_1 and o_2 always have the same access characteristics. That is, in every state, the set of subjects having a right r over o_1 is the same as the set of subjects having the right r over o_2. In SDCO, one can reach states in which o_1 and o_2 have different access characteristics. Therefore, each object needs to be assigned a distinct label, we use $\langle s, t(o)\rangle$ to denote such an label.

Therefore, before creating an object, one has to create a new label. When creating a new label ℓ, one has to assign a group to $\mathsf{auth}(\ell, \mathsf{own})$ and a group to $\mathsf{auth}(\ell, \mathsf{re})$; and a group to $\mathsf{auth}(\ell, \mathsf{wr})$. Each pair $\langle \ell, r\rangle$ determines a unique access class. Therefore, a distinct group needs to be created. We use $g(o, r)$ to denote the group that will be assigned to have the right r over object o.

In order to keep track of which subset of rights a subject has over an object, we need 8 group tags, one corresponding to each subset of $\{own, \mathsf{re}, \mathsf{wr}\}$, we use $t^g(o, x)$, where x is a 3-bit string to denote these tags.

In order for a subject s to create an object o, s needs to do the following:

1. Create an NGS $e = \langle e.G,\ e.T^g,\ e.\mathsf{gtag},\ e.nt^g,\ e.\mathsf{admin},\ e.\mathsf{patterns},\ , e.\mathsf{GRS}\rangle$ as follows.

   - $e.G = \{g(o, \mathsf{own}), g(o, \mathsf{re}), g(o, \mathsf{wr})\}$
   - $e.T^g = \{t^g(o, 000), t^g(o, 001), t^g(o, 010), t^g(o, 011), t^g(o, 100), t^g(o, 101), t^g(o, 110), t^g(o, 111)\}$.
   - $e.\mathsf{gtag}[s] = t^g(o, 100)$ and $e.\mathsf{gtag}[s'] = t^g(o, 000)$ for every $s' \in S_\gamma$ s.t. $s' \neq s$.
   - $e.nt^g = t^g(o, 000)$
   - $e.\mathsf{admin} = e$
   - $e.\mathsf{patterns}[g(o, \mathsf{own})] = \{\langle *u, t^g(o, 100)\rangle, \langle *u, t^g(o, 101)\rangle, \langle *u, t^g(o, 110)\rangle, \langle *u, t^g(o, 111)\rangle\}$
     $e.\mathsf{patterns}[g(o, \mathsf{re})] = \{\langle *u, t^g(o, 010)\rangle, \langle *u, t^g(o, 011)\rangle, \langle *u, t^g(o, 110)\rangle, \langle *u, t^g(o, 111)\rangle\}$
     $e.\mathsf{patterns}[g(o, \mathsf{wr})] = \{\langle *u, t^g(o, 001)\rangle, \langle *u, t^g(o, 011)\rangle, \langle *u, t^g(o, 101)\rangle, \langle *u, t^g(o, 111)\rangle\}$
   - $e.\mathsf{GRS} = \{Relabel(g(o, b_1b_2b_3), g(o, b'_1b'_2b'_3)) = g(o, \mathsf{own})$
     $\mid b_1b_2b_3, b'_1b'_2b'_3 \in \{0,1\}^3 \wedge b_1b_2b_3$ and $b'_1b'_2b'_3$ differ in exactly one bit$\}$

2. Use the action $\mathsf{create\_label}(s, \langle s, t(o)\rangle, g(o, \mathsf{re}), g(o, \mathsf{wr}))$ to create the label $\ell(o)$.

3. Use the action $\mathsf{create\_object}(s, o, \langle s, t(o)\rangle)$ to create the object o and label it with $\ell(o)$.

To grant or revoke a right, one uses group relabelling. For instance, suppose s is a subject, and for the NGS, e, $e.\mathsf{gtag}[s] = t^g(o, 000)$. Then, we know that s is not a member of any of the groups $g(o, \mathsf{own})$, $g(o, \mathsf{re})$ or $g(o, \mathsf{wr})$. The subject would be granted the right re by relabelling $\langle s, t^g(o, 000)\rangle$ to the label $\langle s, t^g(o, 010)\rangle$. The execution of this relabelling results in the subject becoming a member of the group $g(o, \mathsf{re})$, thereby giving him the right re over the object o. Similarly, the subject would have the right re revoked by relabelling $\langle s, t^g(o, 010)\rangle$ to the label $\langle s, t^g(o, 000)\rangle$. These operations can be carried out only by a subject that is a member of the group $g(o, \mathsf{own})$.

We make the following observations about the above mapping.

- There is considerable overhead in implementing a relatively simple DAC scheme (SDCO) in the Solworth-Sloan scheme. For each object, we need to create a set of labels whose size is linear in the number of the subjects in the state. We also need to create a set of tags whose size is exponential in

the number rights in the system. These tags are used to define groups, and the therefore, the number of entries in all the sets of patterns is also exponential in the number of rights in the system. This is considerable overhead considering the simplicity of SDCO, and the fact that we can "directly" implement it, with efficiently decidable safety.

- We are unable to capture destruction of subjects and objects as such constructs have not been specified for the Solworth-Sloan scheme. Destruction of subjects and objects is generally considered to be an important component of any access control system. In particular, it is unclear how and with what overhead we can capture in the Solworth-Sloan scheme, the notion of transfer of ownership over objects owned by a subject that is being destroyed.

- The above mapping does not capture the state invariant in SDCO that in every state, there is exactly one owner for every object that exists. In the Solworth-Sloan system that results from the above mapping, one can perform relabelling operations and reach states in which there are mutiple owners for an object, or no owner for an object. For instance, suppose that there already exists a subject s such that $s \in e.members[g(o, \mathsf{own})]$. Given the above relabelling rules, there is nothing that precludes another subject from also becoming a member of the group $g(o, \mathsf{own})$ while s continues to maintain membership in that group. It is also possible to remove the membership of s in the group $g(o, \mathsf{own})$ thereby leaving the object with no owner. It is unclear how we would prevent such situations from occuring in a system based on the Solworth-Sloan scheme.

Our conclusion is that several of the claims made by Solworth and Sloan [26] are incorrect. In particular, not only is the motivation (decidable safety) for the creation of their new scheme invalid, but it is also not effective in implementing relatively simple DAC schemes.

7 Conclusions

The focus of this paper is to provide a clear picture of safety analysis in DAC. We have used a state-transition-system-based meta-formalism to precisely model access control schemes and systems and have studied safety analysis in two general DAC schemes from the literature, the Graham-Denning scheme [9], and the Griffiths-Wade scheme [10]. For the Graham-Denning scheme, we have presented an algorithm for deciding safety with running time $O(n^3)$ and proved that the algorithm is correct. Because of space limitations, we were unable to include details of our analysis of the Griffiths-Wade scheme, but we have summarized that there exists an $O(n^4)$ algorithm for deciding safety in the scheme. We have also refuted several claims made by Solworth and Sloan [26]. In particular, we have refuted the claim that the mapping presented there encodes all DAC schemes by considering a relatively simple DAC scheme and demonstrating that the mapping has several deficiencies. We conclude by asserting that safety in existing general DAC schemes is decidable and there is no need to invent new DAC schemes with decidable safety as the goal.

References

[1] Paul Ammann and Ravi S. Sandhu. Safety analysis for the extended schematic protection model. In *Proceedings of the 1991 IEEE Symposium on Security and Privacy*, pages 87–97, May 1991.

[2] Paul Ammann and Ravi S. Sandhu. The extended schematic protection model. *Journal of Computer Security*, 1(3-4):335–383, 1992.

[3] Elisa Bertino, Claudio Bettini, Elena Ferrari, and Pierangela Samarati. An access control model supporting periodicity constraints and temporal reasoning. *ACM Transactions on Database Systems*, 23(3):231–285, 1998.

[4] Elisa Bertino, Claudio Bettini, and Pierangela Samarati. A temporal authorization model. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, pages 126–135. ACM Press, 1994.

[5] T. Budd. Safety in grammatical protection systems. *International Journal of Computer and Information Sciences*, 12(6):413–430, 1983.

[6] National Computer Security Center. A guide to understanding discretionary access control in trusted systems, September 1987. NCSC-TG-003.

[7] Deborah D. Downs, Jerzy R. Rub, Kenneth C. Kung, and Carole S. Jordan. Issues in discretionary access control. In *Proceedings of IEEE Symposium on Research in Security and Privacy*, pages 208–218, April 1985.

[8] Anonymous (for blind review). On safety in discretionary access control. Technical report, 2004.

[9] G. Scott Graham and Peter J. Denning. Protection — principles and practice. In *Proceedings of the AFIPS Spring Joint Computer Conference*, volume 40, pages 417–429. AFIPS Press, May 16–18 1972.

[10] Patricia P. Griffiths and Bradford W. Wade. An authorization mechanism for a relational database system. *ACM Transactions on Database Systems*, 1(3):242–255, 1976.

[11] Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, August 1976.

[12] Anita K. Jones, Richard J. Lipton, and Lawrence Snyder. A linear time algorithm for deciding security. In *17th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 33–41, October 1976.

[13] Butler W. Lampson. Protection. In *Proceedings of the 5th Princeton Conference on Information Sciences and Systems*, 1971. Reprinted in ACM Operating Systems Review, 8(1):18-24, Jan 1974.

[14] Ninghui Li and Mahesh V. Tripunitara. Security analysis in role-based access control. In *Proceedings of the Ninth ACM Symposium on Access Control Models and Technologies (SACMAT 2004)*, pages 126–135, June 2004.

[15] Ninghui Li, William H. Winsborough, and John C. Mitchell. Beyond proof-of-compliance: Safety and availability analysis in trust management. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 123–139. IEEE Computer Society Press, May 2003.

[16] Richard J. Lipton and Lawrence Snyder. A linear time algorithm for deciding subject security. *Journal of the ACM*, 24(3):455–464, 1977.

[17] Teresa Lunt. Access control policies: Some unanswered questions. In *Proceedings of the 2nd IEEE Computer Security Foundations Workshop*, pages 227–245. IEEE Computer Society Press, June 1988.

[18] Naftaly H. Minsky. Selective and locally controlled transport of privileges. *ACM Transactions on Programming Languages and Systems*, 6(4):573–602, October 1984.

[19] Rajeev Motwani, Rina Panigrahy, Vijay A. Saraswat, and Suresh Ventkatasubramanian. On the decidability of accessibility problems (extended abstract). In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, pages 306–315. ACM Press, May 2000.

[20] Sylvia Osborn, Ravi S. Sandhu, and Qamar Munawer. Configuring role-based access control to enforce mandatory and discretionary access control policies. *ACM Transactions on Information and System Security*, 3(2):85–106, May 2000.

[21] Pierangela Samarati and Sabrina de Capitani di Vimercati. Access control: Policies, models, and mechanisms. In Ricardo Focardi and Roberto Gorrieri, editors, *Foundations of Security Analysis and Design*, volume 2171 of *Lecture Notes in Computer Science*, pages 137–196. Springer, 2001.

[22] Ravi S. Sandhu. The schematic protection model: Its definition and analysis for acyclic attenuating systems. *Journal of the ACM*, 35(2):404–432, 1988.

[23] Ravi S. Sandhu. Expressive power of the schematic protection model. *Journal of Computer Security*, 1(1):59–98, 1992.

[24] Ravi S. Sandhu. The typed access matrix model. In *Proceedings of the 1992 IEEE Symposium on Security and Privacy*, pages 122–136. IEEE Computer Society Press, May 1992.

[25] Ravi S. Sandhu. Undecidability of the safety problem for the schematic protection model with cyclic creates. *Journal of Computer and System Sciences*, 44(1):141–159, February 1992.

[26] Jon A. Solworth and Robert H. Sloan. A layered design of discretionary access controls with decidable safety properties. In *Proceedings of IEEE Symposium on Research in Security and Privacy*, May 2004.

[27] Jon A. Solworth and Robert H. Sloan. Security property based administrative controls. In *Proceedings of the Ninth European Symposium on Research in Computer Security (ESORICS 2004)*, pages 244–259. Springer, September 2004.

[28] Masakazu Soshi. Safety analysis of the dynamic-typed access matrix model. In *Proceedings of the Sixth European Symposium on Research in Computer Security (ESORICS 2000)*, pages 106–121. Springer, October 2000.

[29] Masakazu Soshi, Mamoru Maekawa, and Eiji Okamoto. The dynamic-typed access matrix model and decidability of the safety problem. *IEICE Transactions on Fundamentals*, E87-A(1):190–203, January 2004.

A Proof for Lemma 1

Proof. The "if" part: we need to show that if $\mathsf{isSafeGD}(\gamma, \psi, \omega, \mathcal{T})$ returns true, then the system is safe with respect to ω and $\mathcal{T}$. We show, equivalently, that if the system is not safe with respect to ω and $\mathcal{T}$, then $\mathsf{isSafeGD}(\gamma, \psi, \omega, \mathcal{T})$ returns false. Assume that the system is not safe with respect to ω and $\mathcal{T}$. We have two cases. The first case is that in the start-state, γ, s has x over o. This case consists of two subcases: either (1) $x \in M_\gamma[s, o]$, or (2) $x \in \mathcal{R}_b$ and $x^* \in M_\gamma[s, o]$ (possession of x^* implies possession of x). If both (1) and (2) are true, we consider either one of those two subcases. If subcase (1) is true, then we know that $x \in R_\psi$, and if $x = \mathit{control}$ and $o \in \mathcal{O} - \mathcal{S}$, then $x \notin M_\gamma[s, o]$ (by property (2) from the previous section that objects that are not subjects cannot have the $\mathit{control}$ right over them). Therefore, the 'if' conditions of lines 7 and 8 are not satisfied, and line 9 of the algorithm returns false, and we are done. For subcase (2), in line 5 we instantiate y to x^*. We know that $x, y \in R_\psi$, and that $x \neq \mathit{control}$. Therefore, the 'if' conditions for lines 7 and 8 are not

satisfied. The 'if' condition for line 9 may be satisfied and if it is, the algorithm returns false and we are done. Otherwise, the algorithm returns false in line 10.

The second case is that s does not have x over o in the start-state, i.e., $x \notin M_\gamma[s, o]$ and if $x \in \mathcal{R}_b$, then $x^* \notin M_\gamma[s, o]$. In this case, as the system is not safe, there exists a finite sequence of state-changes $\gamma \mapsto_{\psi(s_1)} \gamma_1 \mapsto_{\psi(s_2)} \cdots \mapsto_{\psi(s_n)} \gamma_n$ where n is an integer and $n \geq 1$, such that either $x \in M_{\gamma_n}[s, o]$, or if $x \in \mathcal{R}_b$, then $x^* \in M_{\gamma_n}[s, o]$. Each $s_i \in S_{\gamma_{i-1}} - \mathcal{T}$ and the s_i's are not necessarily distinct from one another. We point out also that if $s_i \in S_{\gamma_j} - \mathcal{T}$ for some i and j, and $s_i \in S_{\gamma_k}$ for some $k \neq j$, then $s_i \in S_{\gamma_k} - \mathcal{T}$, because $\mathcal{T}$ is specified a-priori and does not change with changes in the state. We now show that if such a sequence of state-changes exists, then the algorithm returns false. We show this by induction on n. For the base case, if there exists a sequence of length 1, then $\gamma \mapsto_{\psi(s_1)} \gamma_1$, and $x \notin M_\gamma[s, o]$ and $x^* \notin M_\gamma[s, o]$ if $x \in \mathcal{R}_b$, and $x \in M_{\gamma_1}[s, o]$, or $x \in \mathcal{R}_b$ and $x^* \in M_{\gamma_1}[s, o]$. In this case, the state-change is the execution of one of the following commands, and we show that the algorithm returns false in each case. The state-change has to be the execution of one of these commands because these are the only commands that enter a right in to a cell of the access matrix.

transfer_r – in this case we know that $x \in \mathcal{R}_b \cap R_\psi$, $x^* \in R_\psi$, $x^* \in M_\gamma[s_1, o]$ for some $s_1 \in S_\gamma - \mathcal{T}$, and $s \in S_\gamma$. The algorithm will not return in any of the lines 7-11 as the respective 'if' conditions are not satisfied. If $o \notin O_\gamma$, then the algorithm returns false in line 12, and we are done. If $o \in O_\gamma$, then the conditions for line 13 are met (y is instantiated to x^*), and the algorithm returns false.

transfer_r* – we have two subcases to consider: either (1) $x \in \mathcal{R}_b^* \cap R_\psi$, or, (2) $x \in \mathcal{R}_b \cap R_\psi$. In case (2), let y be x^*, and in case (1), let y be x. We know in either case that $y \in M_\gamma[s_1, o]$ for some $s_1 \in S_\gamma - \mathcal{T}$, and $s \in S_\gamma$ (otherwise s would not get the right x over o after the execution of the command). The algorithm will not return in any of the lines 7-11 as the respective 'if' conditions are not satisfied. If $o \notin O_\gamma$, then the algorithm returns false in line 12, and we are done. If $o \in O_\gamma$, then the conditions for line 13 are met and the algorithm returns false.

transfer_own – in this case we know that $x = own$, $own \in M_\gamma[i, o]$ for some $i \in S_\gamma - \mathcal{T}$, $o \in S_\gamma$ and $s \in S_\gamma$. The 'if' conditions for each of lines 7-13 are not met (for line 11, we know that $own^* \notin R_\psi$). Consider lines 14-16. We know that such a sequence of subjects exists (as i has the own right over o in S_γ), and furthermore, $i \in S_\gamma - \mathcal{T}$. Therefore, the conditions to return false in lines 14-16 are met, and the algorithm returns false.

grant_r – in this case, we know that $own \in M_\gamma[i, o]$ for some $i \in S_\gamma - \mathcal{T}$ and $x \in \mathcal{R}_b \cap R_\psi$ (in particular, $x \neq control$ and $x \neq own$ – there are other commands to grant those rights). The 'if' conditions for each of lines 7-11 are not met. If $o \notin O_\gamma$, the algorithm returns false in line 12, and we are done. If $o \in O_\gamma$, the conditions for line 13 may be met, and if they are, the algorithm returns false and we are done. If the conditions in line 13 are not met, then we observe that the conditions for lines 14-16 are met (the sequence of subjects contains i, as i has the own right over o in S_γ), and the algorithm returns false.

grant_r* – we have two subcases to consider. Either (1) $x \in \mathcal{R}_b \cap R_\psi$, or, (2) $x \in \mathcal{R}_b^* \cap R_\psi$. For case (1), let y be x^* and for case (2), let y be x. In either case, we know that $own \in M_\gamma[i, o]$ for some $i \in S_\gamma - \mathcal{T}$. The 'if' conditions for lines 7-11 are not met. If $o \notin O_\gamma$, then the algorithm returns false in line 12, and we are done. Otherwise, the conditions for line 13 may be met, and if they are, the algorithm returns false, and we are done. Otherwise, we observe that the conditions for lines 14-16 are met (the sequence of subjects contains i, as i has the own right over o in S_γ), and the algorithm returns false.

grant_control – in this case, we know that $x = control$, $own \in M_\gamma[i, o]$ for some $i \in S_\gamma - \mathcal{T}$ and $o \in S_\gamma$. Therefore, the 'if' conditions for lines 7-12 are not met. The 'if' conditions for line 13 are not met because we know that $y \notin R_\psi$. But, we observe that the conditions for lines 14-16 are met, because the

subject i that is not trusted exists in γ, and i has the own right over o. Therefore, the algorithm returns false in line 16.

grant_own – in this case, we know that $x = own$ and $own \in M_\gamma[i, o]$ for some $i \in S_\gamma - \mathcal{T}$. The 'if' conditions for lines 7-11 are not satisfied. If $o \notin O_\gamma$, then the algorithm returns false in line 12 and we are done. Otherwise, the condition in line 13 is not satisfied, but, we observe that the conditions for lines 14-16 are satisfied, and the algorithm returns false.

create_object – in this case, we know that $x = own$ and $o \notin O_\gamma$. The 'if' conditions for lines 7-11 are not met, but the 'if' condition for line 12 is met, and the algorithm returns false.

create_subject – in this case, we know that $\exists i \in S_\gamma - \mathcal{T}$, and either $x = own$ or $x = control$. Furthermore, we know that $o \notin O_\gamma$. The reason is that in the body of the command, we enter a right only in the column corresponding to the subject that is created in the execution of the command, and not any other object. Therefore, for $\omega = \langle s, o, x\rangle$, we know that o must be the subject that is created in the execution of the create_subject command. We know also that $o \notin \mathcal{O} - \mathcal{S}$, because the object that is created is a subject. Therefore, the respective 'if' conditions for lines 7-11 are not satisfied, but the 'if' condition for line 12 is satisfied, and the algorithm returns false.

destroy_subject – in this case, we know that $x = own$, and $own \in M_\gamma[s, s']$, where $\omega = \langle s, o, x\rangle$ and s' is the subject that is destroyed in the execution of the command. The reason is that we enter a right only in the row corresponding to such a subject s. Furthermore, we know that $o \in O_\gamma$ and $own \in M_\gamma[s', o]$, because the only columns in which a right is entered in the execution of the command are columns with that property. We know also that $s \in S_\gamma - \mathcal{T}$ as s is the initiator of the command-execution. Given these facts, we know that the 'if' conditions for lines 7-12 are not satisfied. The conditions for line 13 may be met, and if they are, the algorithm returns false and we are done. Otherwise, we observe that the conditions for lines 14-16 are satisfied; the sequence of subjects contains s and s' with s' being the last member of the sequence, and s immediately preceding s' in the sequence. As $s \in S_\gamma - \mathcal{T}$, the algorithm returns false in line 16.

For the induction hypothesis, we assume that if there exists a state-change sequence $\gamma \mapsto_{\psi(s_1)} \gamma_1 \mapsto_{\psi(s_2)} \cdots \mapsto_{\psi(s_{k-1})} \gamma_{k-1}$ of length $k-1$ (for $k-1 \geq 1$) such that $x \notin M_\gamma[s, o]$ and if $x \in \mathcal{R}_b$, $x^* \notin M_\gamma[s, o]$, and either $x \in M_{\gamma_{k-1}}[s, o]$ or, if $x \in \mathcal{R}_b$, $x^* \in M_{\gamma_{k-1}}[s, o]$, then the algorithm returns false. Now assume that there exists a state-change sequence $\gamma \mapsto_{\psi(s_1)} \cdots \mapsto_{\psi(s_k)} \gamma_k$ of length k (for $k \geq 2$) such that $x \notin M_\gamma[s, o]$ and if $x \in \mathcal{R}_b$, $x^* \notin M_\gamma[s, o]$, and either $x \in M_{\gamma_k}[s, o]$ or, if $x \in \mathcal{R}_b$, $x^* \in M_{\gamma_k}[s, o]$. We need to show that the algorithm returns false for $\omega = \langle s, o, x\rangle$.

We have two cases. The first case has two subcases: either (a) $x \in M_{\gamma_{k-1}}[s, o]$, or, (b) $x \in \mathcal{R}_b$ and $x^* \in M_{\gamma_{k-1}}[s, o]$. In either case, we have a state-change sequence of length $k-1$ with the appropriate properties, and by the induction hypothesis, we know that the algorithm returns false. In the second case, we assume that $x \notin M_{\gamma_{k-1}}[s, o]$ and if $x \in \mathcal{R}_b$, $x^* \notin M_{\gamma_{k-1}}[s, o]$, and either $x \in M_{\gamma_k}[s, o]$ or $x \in \mathcal{R}_b$ and $x^* \in M_{\gamma_k}[s, o]$. We need to show that the algorithm returns false in this case. We consider the state-change $\gamma_{k-1} \mapsto_{\psi(s_k)} \gamma_k$. It must be the execution of one of the following commands (the same as those we considered for the base case), as those are the only commands that add a right to a cell in the access matrix. We consider each in turn. We point out that as $k \geq 2$, we have at least 3 states in our state-change sequence, including the start-state, i.e., we know that at least the states $\gamma_{k-2}, \gamma_{k-1}$ and γ_k (where the start-state, $\gamma = \gamma_0$) exist in the state-change sequence.

transfer_r – in this case, we know that $x \in \mathcal{R}_b \cap R_\psi$ and $x^* \in M_{\gamma_{k-1}}[s_k, o]$. Let $\omega^k = \langle s_k, o, x^*\rangle$. Then, we know by the induction hypothesis that $\mathsf{isSafeGD}(\gamma, \psi, \omega^k, \mathcal{T})$ returns false (as there exists a state-change sequence of length $k-1$ with the appropriate properties). We refer to the execution of the algorithm for the input $(\gamma, \omega, \mathcal{T})$ as e, and for the input $(\gamma, \omega^k, \mathcal{T})$ as e^k. Consider the following cases.

- e^k returns in line 9: in this case, we know that $x^* \in M_\gamma[s_k, o]$. Now, e cannot return in lines 7 or 8 (because $x \in \mathcal{R}_b \cap R_\psi$). e may return false in line 9 or line 10, in which case we are done. If not, e will not return in lines 11-12 as $s_k \in S_\gamma - \mathcal{T}$ and $o \in O_\gamma$. Finally, e will return false in line 13, because $s_k \in S_\gamma - \mathcal{T}$, and $y \in M_\gamma[s_k, o]$.
- e^k returns in line 10: this cannot happen as, in this case, e^k would have returned in line 9. Therefore, the arguments for the previous case apply.
- e^k returns in line 12: in this case, e will not return in any of the lines 7-11, but will return false in line 12.
- e^k returns in line 13: in this case, we know that $\exists\, \widehat{s} \in S_\gamma - \mathcal{T}$ such that $y \in M_\gamma[\widehat{s}, o]$ where $y = x^*$. e will not return in lines 7-8, but may return false in one of the lines 9 or 10, in which case we are done. Otherwise, e will not return in line 11 (as $\widehat{S}$ exists in γ) or in line 12 ($o \in O_\gamma$). But, e will return false in line 13, as the condition is met ($\widehat{S}$ is such a subject).
- e^k returns in line 16: in this case, e will not return in lines 7-8 but may return in line 9, in which case we are done. Otherwise, e will not return in lines 10-13. We know that e will return false in line 16, just as e^k does, because the same condition is true for e as well.

transfer_r* – in this case, we know that $x \in \mathcal{R}_b^* \cap R_\psi$, and $x \in M_{\gamma_{k-1}}[s_k, o]$ where $s_k \in S_{\gamma_{k-1}} - \mathcal{T}$. Let $\omega^k = \langle s_k, o, x\rangle$, e^k be the execution of the algorithm isSafeGD for the input $(\gamma, \psi, \omega^k, \mathcal{T})$, and e be the execution for the input $(\gamma, \omega, \mathcal{T})$. Then we know that e^k returns false by the induction hypothesis. We now have exactly the same arguments as in the previous case for why e returns false.

transfer_own – in this case we know that $x = own$ and $own \in M_{\gamma_{k-1}}[s_k, o]$ where $s_k \in S_{\gamma_{k-1}} - \mathcal{T}$. For $\omega^k = \langle s_k, o, own\rangle$, we know that, e^k, the execution of the algorithm on input $(\gamma, \omega^k, \mathcal{T})$, returns false, by the induction hypothesis. We consider all the cases in which e^k can return false.

- e^k returns in line 9: in this case, we know that $own \in M_\gamma[s_k, o]$ and $s_k \in S_\gamma - \mathcal{T}$. Now, e does not return in any of the lines 7-8. e may return in line 9, in which case we are done. e cannot return in line 10 (as $y \notin R_\psi$), or in line 11, but may return in line 12, in which case we are done. e cannot return in line 13. Finally, we observe that the conditions in lines 14-16 are satisfied, and therefore, e returns in line 16.
- e^k returns in line 10: this cannot happen because when $x = own$, $y \notin R_\psi$.
- e^k returns in line 12: in this case, we know that e does not return in lines 7-11, but returns false in line 12.
- e^k returns in line 13: this cannot happen because when $x = own$, $y \notin R_\psi$.
- e^k returns in line 16: in this case, e does not return in lines 7-8, but may return in line 9, in which case we are done. Otherwise, e cannot return in lines 10-13, but returns false in line 16 based on the same conditions that e^k satisfies to return in line 16.

grant_r – in this case, we know that $x \in \mathcal{R}_b \cap R_\psi$ and $own \in M_{\gamma_{k-1}}[s_k, o]$, where $s_k \in S_{\gamma_{k-1}} - \mathcal{T}$. We know also that e^k, the execution of the algorithm, on input $(\gamma, \omega^k, \mathcal{T})$ returns false, where $\omega^k\ tuples_k, o, own$. Let e be the execution of the algorithm for the input $(\gamma, \omega, \mathcal{T})$. We have the following cases.

- e^k returns in line 9: in this case, we know also that $own \in M_\gamma[s_k, o]$ where $s_k \in S_\gamma - \mathcal{T}$. Therefore, e does not return in lines 7-8, but may return false in either line 9 or line 10, in which case we are done. Otherwise, e does not return in lines 11-12, but may return false in line 13, in which case we are done. Finally, e returns false in line 16, because the conditions for returning in line 16 are satisfied (s_k is such a subject).

- e^k returns in line 10: this is not possible as when $x = own$, $y \notin R_\psi$.
- e^k returns in line 12: in this case, e does not return in lines 7-11, but returns false in line 12.
- e^k returns in line 13: this is not possible as when $x = own$, $y \notin R_\psi$.
- e^k returns in line 16: in this case, we know that e does not return in lines 7-8, but may return in one of the lines 9-10, in which case we are done. Otherwise, e does not return in lines 11-12, but may return in line 13, in which case we are done. Finally, e returns in line 16 as the conditions for which e^k returns in line 16 apply to e as well.

grant_r* – in this case, we know that $x \in \mathcal{R}_b^* \cap R_\psi$ and $own \in M_\gamma[s_k, o]$ for $s_k \in S_{\gamma_{k-1}} - \mathcal{T}$. The argument now proceeds exactly as for the previous case, and we are able to show that isSafeGDreturns false on the input $(\gamma, \psi, \omega, \mathcal{T})$.

grant_control – in this case, we know that $x = control$ and $own \in M_{\gamma_{k-1}}[s_k, o]$ for $s_k \in S_{\gamma_{k-1}} - \mathcal{T}$. Let $\omega^k = \langle s_k, o, own \rangle$, and e^k be the execution of the algorithm on the input $(\gamma, \omega^k, \mathcal{T})$. We know, by the induction hypothesis, that e^k returns false. Let e be the execution of the algorithm on the input $(\gamma, \omega, \mathcal{T})$. We have the following cases.

- e^k returns in line 9: in this case we know also that $own \in M_\gamma[s_k, o]$ and $s_k \in S_\gamma - \mathcal{T}$. Therefore, e does not return in lines 7-8 (for line 8, we know that $o \notin \mathcal{O} - \mathcal{S}$, as otherwise, we would not be able to grant the $control$ right to s over o in the final state-change in our sequence), and e may return false in line 9, in which case we are done. Otherwise, e does not return in lines 10-13 (for lines 10 and 13, $y \notin R_\psi$). Finally, e returns false in line 16 because we know that s_k, a subject that is not trusted, exists in γ, and has the own right over o.
- e^k returns in line 10: this is not possible as when $x = own$, $y \notin R_\psi$.
- e^k returns in line 12: in this case, e does not return in lines 7-11, but returns false in line 12.
- e^k returns in line 13: this is not possible as when $x = own$, $y \notin R_\psi$.
- e^k returns in line 16: in this case, e does not return in lines 7-8, but may return in lines 9-10, in which case we are done. Otherwise, e does not return in lines 11-12, but may return in line 13, in which case we are done. Finally, e returns in line 16 as the conditions for which e^k returns in line 16 apply to e as well.

grant_own – in this case, we know that $x = own$ and $own \in M_{\gamma_{k-1}}[s_k, o]$ for $s_k \in S_{\gamma_{k-1}} - \mathcal{T}$. We show that the execution of the algorithm on input $(\gamma, \omega, \mathcal{T})$ returns false using the same arguments as the ones we use for the previous case.

create_object – in this case, we know that $x = own$, $s = s_k$ and $s_k \in S_{\gamma_{k-1}} - \mathcal{T}$. We consider the following cases (and sub-cases).

- $s \in S_{\gamma_{k-2}}$: in this case we need to consider the following two sub-cases.
  - $o \in O_{\gamma_{k-2}}$: in this case, we know that the state-change $\gamma_{k-2} \mapsto_{\psi(s_{k-1})} \gamma_{k-1}$ is destroy_object of object o by s_{k-1}. Therefore, we know that $own \in M_{\gamma_{k-2}}[s_{k-1}, o]$ and $s_{k-1} \in S_{\gamma_{k-2}} - \mathcal{T}$. If $s = s_{k-1}$, then we have a state-change sequence of length $k-2$ with the appropriate properties, and we know that the algorithm returns false. Otherwise, we have a state-change $\gamma_{k-2} \mapsto_{\psi(s_{k-1})} \gamma'_{k-1}$ which is the execution of either the command transfer_own (if $o \in \mathcal{S}$), or the command grant_own (if $o \in \mathcal{O} - \mathcal{S}$), by s_{k-1} to s, which results in $own \in M_{\gamma'_{k-1}}[s, o]$. As there exists a state-change sequence of length $k-1$, we know that the algorithm returns false by the induction hypothesis.

– $o \notin O_{\gamma_{k-2}}$: in this case, there exists a state-change $\gamma_{k-2} \mapsto_{\psi(s)} \gamma'_{k-1}$ which is the execution of the command create_object of o by s, which results in $own \in M_{\gamma'_{k-1}}[s, o]$. As there exists a state-change sequence of length $k-1$, we know that the algorithm returns false by the induction hypothesis.

- $s \notin S_{\gamma_{k-2}}$: in this case, we know that the state-change $\gamma_{k-2} \mapsto_{\psi(s_{k-1})} \gamma_{k-1}$ is the execution of create_subject to create s. Also, we know that $o \notin O_{\gamma_{k-2}}$. If $\gamma_{k-2} = \gamma$, then we know that, on input $(\gamma, \omega, \mathcal{T})$, the algorithm will not return in lines 7-11, but will return false in line 12, and we would be done in this case. Otherwise, there exists at least one prior state, γ_{k-3} in the sequence of state-changes. We have the following sub-cases.
  - $s \in S_{\gamma_{k-3}}$, but $o \notin O_{\gamma_{k-3}}$: in this case, we know that the state-change $\gamma_{k-3} \mapsto_{\psi(s_{k-2})} \gamma_{k-2}$ is the execution of destroy_subject of s by s_{k-2}. Consider the alternate state-changes $\gamma_{k-3} \mapsto_{\psi(s_{k-2})} \gamma'_{k-2} \mapsto_{\psi(s_{k-2})} \gamma'_{k-1}$, where the first state-change is the execution of create_object of o by s_{k-2}, and the second is the execution of transfer_own (if $o \in \mathcal{S}$) or grant_own (if $o \in \mathcal{O} - \mathcal{S}$) of the object o by s_{k-2} to s. We have a desired state-change sequence of length $k-1$, and the algorithm returns false by the induction hypothesis.
  - $s \notin S_{\gamma_{k-3}}$, but $o \in O_{\gamma_{k-3}}$: in this case, we know that the state-change $\gamma_{k-3} \mapsto_{\psi(s_{k-2})} \gamma_{k-2}$ is the execution of destroy_object of o by s_{k-2}. Consider instead the state-changes $\gamma_{k-3} \mapsto_{\psi(s_{k-2})} \gamma'_{k-2} \mapsto_{\psi(s_{k-2})} \gamma'_{k-1}$, where the first state-change is the execution of create_subject of s by s_{k-2} and the second is the execution of transfer_own (if $o \in \mathcal{S}$) or grant_own (if $o \in \mathcal{O} - \mathcal{S}$) of the object o to s by s_{k-2}. We have the desired state-change sequence of length $k-1$, and the algorithm returns false by the induction hypothesis.
  - $s \notin S_{\gamma_{k-3}}$, and $o \notin O_{\gamma_{k-3}}$: we know that $s \notin \mathcal{T}$ (otherwise s would not be able to execute create_object as the last state-change in our state-change sequence of length k). We know also that $s_{k-2} \in S_{\gamma_{k-3}} - \mathcal{T}$. Consider the following state-changes: $\gamma_{k-3} \mapsto_{\psi(s_{k-2})} \gamma'_{k-2} \mapsto_{\psi(\psi(s)} \gamma'_{k-1}$ where the first state-change is the execution of create_subject of s by s_{k-2} and the second is the execution of create_object of o by s. We have the desired state-change sequence of length $k-1$, and the algorithm return false.
  - $s \in S_{\gamma_{k-3}}$, and $o \in O_{\gamma_{k-3}}$: this case cannot happen, as then, we would need to first destroy each of s and o, which requires two state-changes (we know that $s \neq o$, because otherwise, s would not be able to create o in the last state-change in our sequence of length k). We have already fixed two additional state-changes (create_subject of s, and create_object of o as our last two steps in our state-change sequence of length k). As there do not exist four state changes between γ_{k-3} and γ_k, we know that this case cannot happen.

create_subject – in this case, we know that $o \in S_{\gamma_k}$, and either $s = o$ (and $x = control$), or $s = s_k$ (and $x = own$). We know also that $o \notin S_{\gamma_{k-1}}$. We have the following cases.

- $s = o$: we have the following sub-cases.
  - $o \in S_{\gamma_{k-2}}$: in this case, we know that $s = o \in S_{\gamma_{k-2}}$ and $control \in M_{\gamma_{k-2}}[s, o]$, and therefore we have a state-change sequence of length $k-2$ with the appropriate properties, and therefore by the induction hypothesis, the algorithm returns false.
  - $o \notin S_{\gamma_{k-2}}$: in this case, consider the state-change $\gamma_{k-2} \mapsto_{\psi(s_{k-1})} \gamma'_{k-1}$ which is the execution of create_subject of $o = s$ by s_{k-2} (we know that $s_{k-2} \in S_{\gamma_{k-2}} - \mathcal{T}$). We have the desired state-change sequence of length $k-1$ and the algorithm returns false by the induction hypothesis.
- $s = s_k$: we have the following sub-cases.

– $o \in S_{\gamma_{k-2}}$: in this case, we know that the state-change $\gamma_{k-2} \mapsto_{\psi(s_{k-2})} \gamma_{k-1}$ is the execution of destroy_subject of o by $s_{k-1} \in S_{\gamma_{k-2}} - \mathcal{T}$. We know also, in this case, that $s \in S_{\gamma_{k-2}}$, where $s = s_k$. Therefore, we have the state-change $\gamma_{k-2} \mapsto_{\psi(s)} \gamma'_{k-1}$ which is the execution of create_subject of o by s. We have the desired state-change sequence of length $k-1$, and by the induction hypothesis, the algorithm returns false.

– $o \notin S_{\gamma_{k-2}}$: in this case, if $\gamma_{k-2} = \gamma$, then the algorithm does not return in lines 7-11, but returns false in line 12, and we are done. Otherwise, we know that there exists a prior state, γ_{k-3}. We have the following sub-sub-cases.

∗ $s \in S_{\gamma_{k-2}}$: in this case, consider the state-change $\gamma_{k-2} \mapsto_{\psi(s)} \gamma'_{k-1}$ which is the execution of create_subject of o by s. We have the desired state-change sequence of length $k-1$, and the algorithm returns false by the induction hypothesis.

∗ $s \notin S_{\gamma_{k-2}}$, $s \in S_{\gamma_{k-3}}$ and $o \in S_{\gamma_{k-3}}$: this cannot happen as we know that $o \notin S_{\gamma_{k-2}}$ and $s \notin S_{\gamma_{k-2}}$, and we cannot create both o and s in a single state-change.

∗ $s \notin S_{\gamma_{k-2}}$, $s \notin S_{\gamma_{k-3}}$ and $o \in S_{\gamma_{k-3}}$: in this case, we know that the state-change $\gamma_{k-3} \mapsto_{\psi(s_{k-2})} \gamma_{k-2}$ is the execution of destroy_subject of o by s_{k-2}. We consider, instead the state-changes $\gamma_{k-3} \mapsto_{\psi(s_{k-2})} \gamma'_{k-2} \mapsto_{\psi(s_{k-2})} \gamma'_{k-1}$, where the first state-change is the execution of create_subject of s by s_{k-2}, and the second is the execution of transfer_own of o to s by s_{k-2}. We have the desired state-change sequence of length $k-1$, and the algorithm returns false by the induction hypothesis.

∗ $s \notin S_{\gamma_{k-2}}$, $s \in S_{\gamma_{k-3}}$ and $o \notin S_{\gamma_{k-3}}$: in this case, consider the state-change $\gamma_{k-3} \mapsto_{\psi(s)} \gamma'_{k-2}$ which is the execution of create_subject of o by s. We have the desired state-change sequence of length $k-2$, and the algorithm returns false by the induction hypothesis.

∗ $s \notin S_{\gamma_{k-2}}$, $s \notin S_{\gamma_{k-3}}$ and $o \notin S_{\gamma_{k-3}}$: in this case, we know that $s_{k-2} \in S_{\gamma_{k-3}} - \mathcal{T}$. Consider the following state-changes: $\gamma_{k-3} \mapsto_{\psi(s_{k-2})} \gamma'_{k-2} \mapsto_{\psi(s)} \gamma'_{k-1}$, where the first state-change is the execution of create_subject of s by s_{k-2}, and the second is the execution of create_subject of o by s. We have the desired state-change sequence of length $k-1$, and the algorithm returns false by the induction hypothesis.

destroy_subject – in this case, we know that $x = own$, $s = s_k$, $s \neq o$ (as in state γ_k, s has the own right over o), $own \in M_{\gamma_{k-1}}[\widehat{s}, o]$ for some $\widehat{s} \in S_{\gamma_{k-1}}$ with $\widehat{s} \neq s$, and $own \in M_{\gamma_{k-1}}[s, \widehat{s}]$. The state-change is the execution of destroy_subject of $\widehat{s}$ by s to aquire own over o. Let $\widehat{\omega} = \langle \widehat{s}, o, own \rangle$, and $\widehat{e}$ be the execution of the algorithm for the input $(\gamma, \widehat{\omega}, \mathcal{T})$. Then we know that $\widehat{e}$ returns false, by the induction hypothesis. We observe that $\widehat{e}$ cannot return either in line 10 or line 13, because when in $\widehat{e}$, $y \notin R_\psi$. Similarly, let $\omega^s = \langle s, \widehat{s}, own \rangle$, and e^s be the execution of the algorithm for the input $(\gamma, \omega^s, \mathcal{T})$. Then, we know that e^s returns false by the induction hypothesis, but not in line 10 or line 13 (as in the case of e^s as well, $y \notin R_\psi$). Let e be the execution of the algorithm for the input $(\gamma, \omega, \mathcal{T})$. We have the following cases and sub-cases.

- $\widehat{e}$ returns in line 9: in this case, we know that e^s cannot return in line 12, because $\widehat{s} \in O_\gamma$. Therefore, we have the following two sub-cases.
  – e^s returns in line 9: in this case, e does not return in lines 7-8, but may return false in line 9, in which case we are done. Otherwise, e does not return in lines 10-13, but e returns false in line 16, because the conditions are satisfied: we have $\widehat{s}$ that owns o, and $s \in S_\gamma - \mathcal{T}$ that owns $\widehat{s}$.
  – e^s returns in line 16: in this case, e does not return in lines 7-8, but may return false in line 9, in which case we are done. Otherwise, e does not return in lines 10-13. Finally, e returns false in line 16, because the conditions are satisfied: we know that $\widehat{s}$ owns o in γ, and that we have a sequence of subjects as needed in lines 14-16, the first of which owns $\widehat{s}$.

- $\widehat{e}$ returns in line 12: in this case e does not return in lines 7-11, but returns false in line 12 (in particular, we know that e does not return in line 11 because e^s either returns in line 9, which means that $s \in S_\gamma - \mathcal{T}$, or returns in either line 12 or 16, which means that $\exists\ s' \in S_\gamma - \mathcal{T}$).
- $\widehat{e}$ returns in line 16: in this case, e does not return in lines 7-8, but may return in line 9, in which case we are done. Otherwise, e does not return in lines 10-13, but returns in line 16, because the same conditions that cause $\widehat{e}$ to return in line 16 cause e to return in line 16 as well.

The "only if" part: we need to show that if the system is safe with respect to ω and $\mathcal{T}$, then isSafeGD$(\gamma, \psi, \omega, \mathcal{T})$ returns true. We show, equivalently, that if isSafeGD$(\gamma, \psi, \omega, \mathcal{T})$ returns false, then the system is not safe with respect to ω and $\mathcal{T}$. We do this by considering each case that the algorithm returns false, and showing (by construction) that a sequence of state-changes $\gamma \mapsto_{\psi(s_1)} \gamma_1 \mapsto_{\psi(s_2)} \cdots \mapsto_{\psi(s_n)} \gamma_n$ such that $x \in M_{\gamma_n}[s, o]$ exists (each $s_i \in S_{\gamma_{i-1}} - \mathcal{T}$, and the s_i's may not be distinct from one another). We have the following cases.

- The algorithm returns in line 9: in this case, we have a state-change sequence of length 0 (i.e., simply γ), as we know that $x \in M_\gamma[s, o]$.
- The algorithm returns in line 10: in this case, we again have a state-change sequence of length 0 (i.e., simply γ), as we know that if $x \in \mathcal{R}_b \cap R_\psi$, then $x^* \in M_\gamma[s, o]$ (and possession of x^* implies possession of x), and if $x \in \mathcal{R}_b^* \cap R_\psi$, then $x \in M_\gamma[s, o]$. There are no other cases that the algorithm returns in line 10.
- The algorithm returns in line 12: in this case, we know from the check on line 11 that $\exists\ s' \in S_\gamma - \mathcal{T}$. Therefore, if $s \notin S_\gamma$, we have the following state-change sequence: $\gamma \mapsto_{\psi(s')} \gamma_1 \mapsto_{\psi(s')} \gamma_2 \mapsto_{\psi(s')} \gamma_3$, where the first state-change is the execution of create_subject of s by s', the second state-change is the execution of create_object of o (if $o \in \mathcal{O} - \mathcal{S}$) or create_subject of o (if $o \in \mathcal{S}$) by s', and the last state-change is the execution of one of the following:
  - transfer_own, if $o \in \mathcal{S}$ and $x = own$
  - grant_own, if $o \in \mathcal{O} - \mathcal{S}$ and $x = own$
  - grant_control, if $o \in \mathcal{S}$ and $x = control$
  - grant_r, if $x \in \mathcal{R}_b \cap R_\psi$
  - grant_r*, if $x \in \mathcal{R}_b^* \cap R_\psi$

  If $s \in S_\gamma$, then we simply use the same sequence as above, but without the first state-change (i.e., $\gamma \mapsto_{\psi(s')} \gamma_2 \mapsto_{\psi(s')} \gamma_3$).
- The algorithm returns in line 13: in this case, we know that $x \neq own$ and $x \neq control$. If $s \notin S_\gamma$, our state-change sequence is $\gamma \mapsto_{\psi(\widehat{s})} \gamma_1 \mapsto_{\psi(\widehat{s})} \gamma_2$, where the first state-change is the execution of create_subject of s by $\widehat{s}$, and the second state-change is the execution of transfer_r of x to s over o if $x \in \mathcal{R}_b \cap R_\psi$, or transfer_r* to s over o if $x \in \mathcal{R}_b^* \cap R_\psi$. If $s \in S_\gamma$, then we have simply exclude the first state-change (creation of s) from our state-change sequence.
- The algorithm returns in line 16: Let $\sigma = \{s_1, \ldots, s_n\}$ be the set of subjects alluded to in line 16, and let $s_i \in \sigma$ be such that $s_i \in S_\gamma - \mathcal{T}$, for some integer i such that $1 \leq i \leq n$. We know that $o \in O_\gamma$. If $s \notin S_\gamma$, then the first state-change in our state-change sequence is the execution of create_subject of s by s_i. If $s \in S_\gamma$, we exclude this state-change.

  We then have $i - 1$ executions of destroy_subject of each subject s_j such that $j < i$, so that if γ' is the state at the end of the $i - 1$ executions, we have $own \in M_{\gamma'}[s_i, o]$. Finally, we have the following cases.

- $o \in \mathcal{S}$ and $x = own$: in this case, we have the execution of transfer_own of o by s_i to s.
- $o \in \mathcal{O} - \mathcal{S}$ and $x = own$: in this case, we have the execution of grant_own of o by s_i to s.
- $o \in \mathcal{S}$, $x = control$ and $\exists\, s'$ such that $control \in M_{\gamma'}[s', o]$: in this case, we have two state-changes, both initiated by s_i. We first have the execution of delete_r of the $control$ right over o from s', and then the execution of grant_control over o to s.
- $o \in \mathcal{S}$, $x = control$ and $\nexists\, s'$ such that $control \in M_{\gamma'}[s', o]$: in this case, we have the execution of grant_control over o to s by s_i.
- $x \in \mathcal{R}_b \cap R_\psi$: in this case we have the execution of grant_r of x over o to s by s_i.
- $x \in \mathcal{R}_b^* \cap R_\psi$: in this case we have the execution of grant_r* of x over o to s by s_i.

∎