Decidability of Safety in Graph-Based Models for Access Control

Manuel Koch[1], Luigi V. Mancini[2], and Francesco Parisi-Presicce[2,3]

[1] Freie Universität Berlin, Berlin (DE)
mkoch@inf.fu-berlin.de
[2] Univ. di Roma La Sapienza, Rome (IT)
{lv.mancini,parisi}@dsi.uniroma1.it
[3] George Mason University, Fairfax VA (USA)
fparisi@ise.gmu.edu

Abstract. Models of Access Control Policies specified with graphs and graph transformation rules combine an intuitive visual representation with solid semantical foundations. While the expressive power of graph transformations leads in general to undecidable models, we prove that it is possible, with reasonable restrictions on the form of the rules, to obtain access control models where safety is decidable. The restrictions introduced are minimal in that no deletion and addition of a graph structure are allowed in the same modification step. We then illustrate our result with two examples: a graph based DAC model and a simplified decentralized RBAC model.

1 Introduction

Safety analysis determines, for a given set of policy rules and an initial state, whether or not it is possible to reach a state in which a particular access right is acquired by a subject that did not previously possess it. A system state is said to be *safe* with respect to an access right r if no sequence of commands can transform the state into a state that leaks r. Safety analysis was first formalized by Harrison, Ruzzo and Ullman [HRU76] in a model commonly known as HRU model. The HRU model captures security policies in which access rights may change and subjects as well as objects may be created and deleted [HRU76]. In general, safety of a system state with respect to an access right r is an undecidable problem, but decidability can be gained by restricting the models. For example, an authorization system is decidable if one considers mono-operational commands only [HRU76] or if the number of subjects is finite [LS78]. Sandhu and Suri have shown in [SS92] that their model of non-monotonic transformations (i.e., no deletion of access rights from the system), is subsumed by the model of [LS78] if object creation is possible but no subject creation. The approach of defining limited access control models with sufficient expressive power for practical use has not been followed recently because of complexity (both conceptual and algorithmic).

D. Gollmann et al. (Eds.): ESORICS 2002, LNCS 2502, pp. 229–244, 2002.

More recent research has focused on the alternative of defining constraint languages for the treatment of safety requirements. An example of a limited logical language is RSL99 [AS00]. Unfortunately, the languages proposed so far seem to lack either in simplicity or in expressive power.

We proposed a graph-based security framework to specify access control models [KMPP00, KMPP01a, KMPP01b] and we investigate in the present paper the safety issue of this graph-based framework. Compared to the HRU model, safety in our framework is decidable if each graph rule either deletes or adds graph structure but does not do both.

Graph rules do not have to be mono-operational and object creation is possible. Subject nodes chosen from a predefined finite set can be added to the system graph using the graph rules. The graph transformations do not have to be non-monotonical in the sense of [SS92].

There are other approaches [NO99, JT01] to the graphical representation of constraints in access control. The first defines a graphical model to combine role inheritance and separation of duty constraints. The second one uses a graphical model to express constraints based on set identification and set comparison. Neither paper deals with decidability of safety. In the context of the Take-Grant model, decidability is discussed in [Sny77]. The model presented is graph based, in the sense that the state is represented by a graph and state changes by graph transformations as rewriting rules. Safety is described as the (in)ability to derive, using the rewrite rules, a graph containing a specified edge.

In this paper, we illustrate our result with two simple examples: a graph based DAC model and a decentralized RBAC model. These examples are oversimplified in order to focus on the methodology to carry on the safety analysis. Note that the graph-based security framework proposed can specify also more expressive (and realistic) access control models for which the safety property remains decidable.

The result can be seen also as a new mechanism to compute safety in the presence of propositional constraints.

In section 2 of this paper, we review the basic notion of graph transformations; section 3 presents the decidability results and section 4 and section 5 show examples of decidability in a graph-based DAC model and a decentralized RBAC model, respectively. Section 6 contains a summary and points to future work.

2 Graph-Based Models

This section recalls some basic definitions and notation for graph transformations [Roz97]. A *graph* $G = (G_V, G_E(e)_{e \in ETyp}, t_V^G, l^G)$ consists of a set of nodes G_V, a relation $G_E(e) \subseteq G_V \times G_V$ for each edge type e in the set of edge types $ETyp$, a typing mapping $t_V^G : G_V \to VTyp$ and a label mapping $l^G : G_V \to VLabel$ for nodes. Node labels are elements of a disjoint union $VLabel = X \cup C$, where X is a set of variables and C is a set of constants. A reflexive binary relation $< \subseteq VLabel \times VLabel$ is defined as $\alpha < \beta$ if and only if $\alpha \in X$ or $\alpha = \beta$.

Figure 1 on the left-hand side depicts a graph with three nodes and three edges. The node types are U (for user) and O (for objects). There is an edge type for edges without any label and an edge type for edges with label r (for a read access right). The labels for nodes are the user names *Jackie* and *Thomas*, the object label is the object name *newProject.pdf*. We omit the explicit presentation of the typing and labeling morphisms in the figures and attach types and labels at the nodes and edges.

A *graph morphism* $f = (f_V, f_E(e)_{e \in ETyp}) : G \to H$ between graphs G and H is given by partial mappings $f_V : G_V \to H_V$ between nodes and $f_E(e) : G_E(e) \to H_E(e)$ for $e \in ETyp$ between edges so that

- f preserves the graph structure, i.e., $f_E(e)((v, v')) = (f_V(v), f_V(v'))$ for all $(v, v') \in dom(f_E(e))$ and $e \in ETyp$,
- f preserves types, i.e., $t_V^G(v) = t_V^H(f_V(v))$ for each $v \in dom(f_V)$ and
- f respects the label order, i.e. $l^G(v) < l^H(f_V(v))$ for each $v \in dom(f_V)$.

A graph morphism f is total (injective) if all underlying mappings f_V and $f_E(e)$ ($e \in ETyp$) are total (injective). Nodes in the domain of a graph morphism need not have variable labels. But if they do, they can be replaced by other variables or constants; if they are not variables, the labels must be preserved.

A *graph rule* $p(V) : (r, A(p))$ consists of a rule name p, a tuple of variables $V = (x_1, ..., x_n), x_i \in X$, a label preserving injective graph morphism $r : L \to R$ and a set $A(p)$ of *negative application conditions*. The graph L, *left-hand side*, describes the elements a graph must contain for p to be applicable. The morphism r is undefined on nodes/edges that are intended to be deleted, defined on nodes/edges that are intended to be preserved. Nodes and edges of R, *right-hand side*, without a pre-image are newly created. The actual deletions/additions are performed on the graphs to which the rule is applied. A Negative Application Condition (NAC) consists of a set $A(p)$ of pairs (L, N), where the graph L is a subgraph of N. The part $N \setminus L$ represents a structure that must not occur in a graph G for the rule to be applicable. In the figures, we depict (L, N) by the graph N, where the subgraph L is drawn with solid and $N \setminus L$ with dashed lines

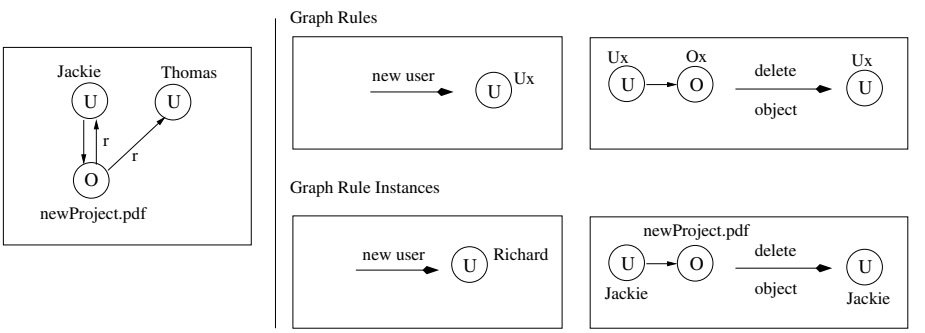


Fig. 1. An example of a graph (left), graph rules and graph rule instances (right)

(e.g., rule *remove user* in figure 4). A rule p with NAC $A(p)$ is applicable to G if L occurs in G and it is not possible to extend L to N for each (L, N) in $A(p)$.

A *rule instance* $p(c_1, ..., c_n) : (r_I, A(p))$ with $c_i \in C$ for a rule $p(x_1, ..., x_n) : (r, A(p))$ instantiates the rule variables x_i by constants c_i $(i = 1...n)$, so that the constants in the left-hand side and the right-hand side are unique.

Figure 1 on the right-hand side shows examples for rules and rule instances. The rule *new user* creates a new user, the rule *delete object* deletes an object which belongs to a user. The rules contain the variables Ux and Ox which can be instantiated by constants. The bottom of the figure shows two possible rule instances where, on the one hand, Ux is instantiated by *Richard* (rule *new user*), on the other hand Ux is instantiated by *Jackie* and Ox by *newProject.pdf*.

A *match* for a rule instance $p(C) : (r, A(p))$ in a graph G is a label preserving total graph morphism $m : L \rightarrow G$. The application of a rule instance $p(C) : (r, A(p))$ to a graph G is given by a match for p in G which satisfies the NAC $A(p)$. The direct derivation $G \overset{p,m}{\Rightarrow} H$ is given by the pushout of r and m in the category of graphs [Roz97]. The pushout is constructed by deleting all elements $m(L \backslash dom(r))$ and adding afterwards all elements $R \backslash r(L)$ to the part $m(dom(r))$. A *derivation sequence* $\rho = (G_0 \overset{p_0,m_0}{\Rightarrow} G_1 \overset{p_1,m_2}{\Rightarrow} G_2 \overset{p_2,m_2}{\Rightarrow} ...)$ is a sequence of direct derivations $G_i \overset{p_i,m_i}{\Rightarrow} G_{i+1}$. The length of a derivation sequence is the number of direct derivations it consists of. We denote a derivation sequence (possibly of length 0) from G_0 to G_n using rule instances of $\mathcal{R}$ by $G_0 \overset{\mathcal{R}}{\Rightarrow}_* G_n$.

Figure 2 depicts an example of a derivation sequence of length 2. First, a rule instance for the rule *delete object* is applied. The rule *delete object* specifies the deletion of an object which belongs to a user, in this rule instance, the deletion of the object *newProject.pdf* of user *Jackie*. We can find these graphical elements into graph G and delete the *newProject.pdf* node. Since no dangling edges must

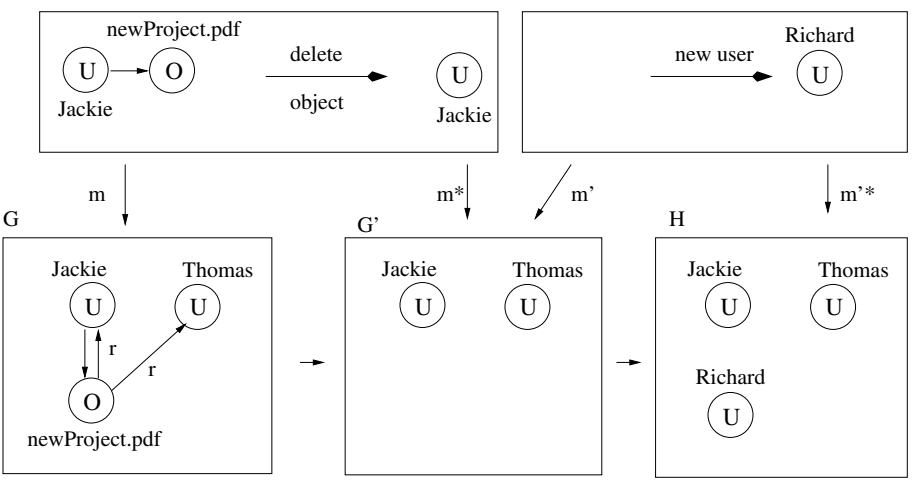


Fig. 2. A derivation sequence

occur in graphs, dangling edges are removed as well. In the second derivation step, a rule instance for the rule *new user* is applied to the result graph of the first derivation step. The rule *new user* adds a new user, in this instance the user *Richard*.

Graph transformations are the basic component for security policy frameworks used to model access control policies (a role-based model in [KMPP00], a lattice-based model and a discretionary model in [KMPP01a]).

3 Decidability in Graph Transformations

Definition 1 (safety). *Let Q_0 and A be graphs and $\mathcal{R}_I$ be a set of graph rule instances. The graph Q_0 is* safe *w.r.t. $\mathcal{R}_I$ and A if and only if there does not exist a graph Q so that $A \subseteq Q$ and $Q_0 \overset{\mathcal{R}_I}{\Rightarrow}_* Q$.*

In other words, an initial system state represented by Q_0 is safe, with respect to the access right represented by the graph A and a set of policy rule instances represented by $\mathcal{R}_I$, if and only if no derivation sequence exists from Q_0 to a graph Q such that $A \subseteq Q$, that is no sequence of graph rule instances can transform Q_0 in a state Q that leaks A.

Proposition 1. *Safety for graph transformations in general is undecidable.*

This is not surprising since graph transformation rules can model the rules of a type 0 grammar, but safety is decidable if each rule is either *expanding* or *deleting*. Deleting rules delete nodes or edges, but add nothing (i.e., $r(L) = R$) and expanding rules may add nodes and edges, but do not delete anything (i.e., $dom(r) = L \subset R$) and do not have a negative application condition.

Proposition 2 (upper bound). *Let Q_0 and A be graphs, $\mathcal{R}_I^+$ be a finite set of expanding graph rule instances and $Der = \{Q_0 \overset{\mathcal{R}_I^+}{\Rightarrow}_* Q | A \subseteq Q\}$ be the set of all derivation sequences starting at Q_0 that use the rule instances in $\mathcal{R}_I^+$ and ending in a graph Q which contains A. Then, the minimal derivation ρ^{min} in Der (i.e., for all $\rho \in Der$, the length of ρ is greater or equal the length of ρ^{min}) has an upper bound that depends only on $\mathcal{R}_I^+, Q_0$ and A.*

Proof. Let R_0 be the set of rule instances in $\mathcal{R}_I^+$ that are applicable to Q_0 and $M(p) = \{m : L \to Q_0 | m \text{ total}\}$ the set of all matches of rule instance $p \in R_0$ in Q_0. Furthermore, let R_f be the set of rule instances in $\mathcal{R}_I^+$ that construct parts of A, i.e., $(p : L \overset{r}{\to} R) \in R_f$ if and only if there is a nonempty injective partial graph morphism $h : R \to A$ so that $h(R \setminus r(L)) \cap A \neq \emptyset$. The set $R_f(p)$ contains all these partial morphisms h in A for the rule instance $p \in R_f$.

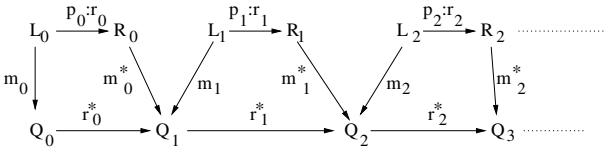
If all rule instances in R_f are applicable to a graph Q_f, the number of rule applications to construct graph A is at most $\sum_{p \in R_f} card(R_f(p))$[1]. Then, all

[1] $card(A)$ of a set A denotes the number of elements in A.

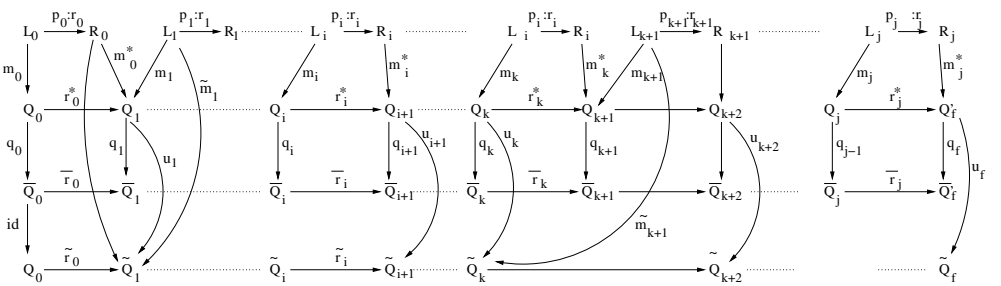
elements of A are constructed that can be constructed by the rule instances in $\mathcal{R}_I^+$ and one can decide whether A occurs or not.

We show now, that there is an upper bound for the number of rule applications to construct a graph Q_f (i.e., a graph to which all rule instances are applicable) from Q_0 by the rule instances in $\mathcal{R}_I^+$. We consider all paths $\sigma = (p_0 \overset{m_0}{\rightarrow} p_1 \rightarrow p_2 \rightarrow ...p_{n-2} \rightarrow p_{n-1})$, where $p_0 \in R_0$, $m_0 \in M(p_0)$, $p_i \in \mathcal{R}_I^+$ for each $0 \leq i < n$, so that each p_i occurs at most once. The paths have a length not greater than $card(\mathcal{R}_I^+)$. Let $\mathcal{P}$ be the set that contains all the paths σ.

A derivation $\rho = (Q_0 \overset{p_0,m_0}{\Rightarrow} Q_1 \overset{p_1,m_1}{\Rightarrow} ... \overset{p_{n-1},m_{n-1}}{\Rightarrow} Q_n)$ is a transformation of a path $\sigma = (p_0 \overset{m_0}{\rightarrow} p_1 \rightarrow p_2... \rightarrow p_{n-1})$, if it applies the rule instances in the order given in the path, starting at match m_0. Let Ω be the set of derivations transformed from the paths in $\mathcal{P}$. All derivations $\rho \in \Omega$ have the maximal length of $maxDerivation = card(\mathcal{R}_I^+)$.



We claim now: if a graph Q_f can be constructed by the rule instances in $\mathcal{R}_I^+$, there is a derivation $\rho = (Q_0 \overset{p_0,m_0}{\Rightarrow} Q_1 \overset{p_1,m_1}{\Rightarrow} ... \Rightarrow Q_f)$ in Ω. If we assume the opposite, there is a derivation $\rho' = (Q_0 \overset{p_0,m_0}{\Rightarrow} Q_1 \overset{p_1,m_1}{\Rightarrow} ... \overset{p_j,m_j}{\Rightarrow} Q'_f) \notin \Omega$, which constructs a graph Q'_f to which all rule instances are applicable. Then, the path $\sigma' = (p_0 \overset{m_0}{\rightarrow} p_1 \rightarrow p_2... \rightarrow p_j)$ can not be in $\mathcal{P}$, i.e., there must be at least one rule instance p_i that occurs more than once in the path σ'. In the diagram below, we assumed that the rule instance p_i is applied at position i and k.



In the derivation ρ', we identify in each graph Q_i $(0 \leq i \leq j)$ all nodes with the same label and get total surjective morphisms $q_i : Q_i \rightarrow \bar{Q}_i$. The sequence $(\bar{Q}_0 \rightarrow \bar{Q}_1... \rightarrow \bar{Q}'_f)$ is generally not a derivation sequence, since the diagrams $\bar{r}_i \circ$

$q_i \circ m_i = q_{i+1} \circ m_i^* \circ r_i$ are generally not pushout diagrams. But the identification ensures that $q_{i+1} \circ m_i^*(R_i) = q_{k+1} \circ m_k^*(R_i)$, so that in particular $q_{i+1} \circ m_i^*(R_i) = q_{k+1} \circ m_k^*(R_i) \subseteq \bar{Q}_{i+1} \subseteq \bar{Q}_k \subseteq \bar{Q}_{k+1}$.

We construct now the derivation sequence $\tilde{\rho} = (\tilde{Q}_0 \overset{p_0,\tilde{m}_0}{\Rightarrow} \tilde{Q}_1 \overset{p_1,\tilde{m}_1}{\Rightarrow} ...\tilde{Q}_{k-1} \overset{p_{k-1},\tilde{m}_{k-1}}{\Rightarrow} \tilde{Q}_k \overset{p_{k+1}}{\Rightarrow} \tilde{Q}_{k+2} \Rightarrow .... \Rightarrow \tilde{Q}_f)$, where $\tilde{Q}_0 = Q_0$, $\tilde{m}_0 = q_0 \circ m_0$ and $\tilde{m}_i = u_i \circ m_i$ where u_i is the universal pushout morphism w.r.t. the pushout diagram $m_i^* \circ r_i = r_i^* \circ m_i$ $(0 < i < j,\ i \neq k+1)$. We define $\tilde{m}_{k+1} = u_k \circ q_k^{-1} \circ q_{k+1} \circ m_{k+1}$, what is possible since q_k is surjective and $\bar{r}_k$ is the identity on Q_k, since $q_{i+1} \circ m_i^*(R_i) = q_{k+1} \circ m_k^*(R_i)$.

By assumption, there is a match $m_i : L_i \to Q_f$ for each rule instance $p_i : r_i \in \mathcal{R}_I^+$. Then, we have also a match $u_f \circ m_i : L_i \to \tilde{Q}_f$, i.e., all rule instances in $\mathcal{R}_I^+$ are applicable to $\tilde{Q}_f$. In such a way, we can remove each repeated rule instance application from ρ' and get a path $\tilde{\sigma} \in \Omega$, what is a contradiction to our assumption.

To conclude, the upper bound for constructing a graph Q_f from Q_0 using the rule instances in $\mathcal{R}_I^+$ in which all rule instances of R_f are applicable is $card(\mathcal{R}_I^+)$. The necessary rule applications for constructing A from Q_f are at most $\sum_{p \in R_f} card(R_f(p))$. Together, we get an upper bound of $card(\mathcal{R}_I^+) + \sum_{p \in R_f} card(R_f(p))$.

Theorem 1 (safety). *Safety of a graph Q_0 with respect to a graph A and a finite set of graph rule instances $\mathcal{R}_\mathcal{I}$ is decidable if $\mathcal{R}_\mathcal{I}$ contains only expanding and deleting graph rules.*

Proof. We show first that the minimal length derivation has only expanding rules. Suppose $Q_0 \overset{p_1}{\Rightarrow} Q_1 \overset{p_2}{\Rightarrow} ... \overset{p_n}{\Rightarrow} Q$ is a minimal length derivation reaching graph Q that contains A. Then, each rule instance p_i for $1 \leq i \leq n$ is an expanding rule. If we assume not, let p_j be a deleting rule. The removal of the rule p_j from the derivation does not affect the construction of the graph A in Q: If we remove p_j, we get the derivation sequence $Q_0 \overset{p_1}{\Rightarrow} Q_1 \overset{p_2}{\Rightarrow} ... \overset{p_{j-1}}{\Rightarrow} Q_{j-1} \overset{p_{j+1}}{\Rightarrow} Q'_{j+1}... \overset{p_n}{\Rightarrow} Q'$. Since the graph Q could be constructed also without the structure $Q_{j-1} \setminus Q_j$ and expanding rules do not have application conditions forbidding some structure in a graph, the rule p_{j+1} can be applied to Q_{j-1} as well. Analog, the rules p_k for $k > j$ can be applied constructing a graph Q' containing A. Hence, we got a shorter derivation what is a contradiction to the assumption of a minimal length derivation sequence.

Thus, the minimal sequence for constructing a graph containing A starting at Q_0 contains only expanding rules. Proposition 2 shows that the minimal sequence to construct A has an upper bound m. A decision procedure would try all derivation sequences of expanding rules of length up to m.

Proposition 3. *Let A be a graph and $p_I(C) : L_I \overset{r_I}{\to} R_I$ be an instance rule of rule $p(V) : L \overset{r}{\to} R$. Furthermore, let*

- *$R_f(p)$ be the set of nonempty partial injective morphisms $g : R \to A$ so that $g(R \setminus r(L)) \cap A \neq \emptyset$ and*

– $R_f(p_I)$ *the set of nonempty partial injective morphisms* $g : R_I \to A$ *so that* $g(R_I \setminus r_I(L_I)) \cap A \neq \emptyset$.

Then, $card(R_f(p_I)) \leq card(R_f(p))$.

Proof. Bases on the fact that the variables in rules can be mapped to any appropriate constant, but constants in rule instances can be mapped only to the same constants: For each morphism in $R_f(p_I)$ there is a morphism in $R_f(p)$. A morphism in $R_f(p)$, however, does not need to have a counterpart on the instance level, since nodes labeled with a constant can be mapped only to nodes labeled with the same constant.

Corollary 1 (upper bound). *Let* Q_0 *and* A *be graphs,* $\mathcal{R}^+$ *a set of expanding rules,* $\mathcal{R}_I^+(p)$ *a finite set of rule instances for* $p \in \mathcal{R}^+$ *and* $Der = \{Q_0 \overset{\mathcal{R}_I^+}{\Rightarrow}_* Q | A \subseteq Q\}$ *be the set of all derivation sequences starting at* Q_0 *and ending in a graph* Q *which contains* A. *Then,*

$$UB = card(\mathcal{R}_I^+) + R_F$$

is an upper bound for the minimal derivation in Der, *where* $R_F = \sum_{p \in \mathcal{R}^+} card(R_f^+(p))$.

Proof. In the proof of proposition 2, the upper bound was given by

$$card(\mathcal{R}_I^+) + \sum_{p_I \in R_f} card(R_f(p_I)).$$

By proposition 3, we have

$$\sum_{p_I \in R_f} card(R_f(p_I)) \leq \sum_{p_I \in R_I^+} card(R_f(p_I)) \leq \sum_{p \in R^+} card(R_f(p)).$$

Figure 3 depicts the decidability algorithm.The method *setOfOverlaps(rule,graph)* returns the set of partial injective mappings from the right-hand side of the *rule* to the *graph*, where the domain of the mapping contains at least an element constructed by *rule* (i.e. in $R \setminus r(L)$). The method *setOfDerivationSequences(ruleSet,startGraph, length)* constructs all derivation sequences up to *length* starting from *startGraph* using rule instances of *ruleSet*. The method *card*(*set*) returns the number of elements in *set*. The method *subgraph(graph1,graph2)* returns *true* if *graph*1 is a subgraph of *graph*2, otherwise it returns *false*.

The decidability algorithm presented above can probably be optimized. The apparent complexity is exponential because it requires generating a set of overlaps and checking the subgraph property. The effective complexity of the algorithm is probably much lower than the computed worst case since labels (or unique names) on nodes reduce considerably the number of possible matchings. Graph transformation tools [EEKR99] can be used to automate the process of generating all the mappings and all the derivation sequences to be checked against the unwanted configuration.

input: sets $\mathcal{R}_I(p)$ ($p \in \mathcal{R}$) of expanding rule instances of rules in $\mathcal{R}$, graph Q_0 and graph A
output: **true**, if there is a derivation sequence ($Q_0 \Rightarrow ... \Rightarrow Q$) where $A \subseteq Q$, **false** otherwise

begin
  //get the rules which construct parts of A
  $R_F = 0$;
  for each $p \in \mathcal{R}$ {
    $R_f(p) = \text{setOfOverlaps}(p, A)$;
    $R_F = R_F + card(R_f(p))$;
  }
  //construct the upper bound for derivation sequences
  $length = \text{card}(R_I) + R_F$;
  $Der = \text{setOfDerivationSequences}(\mathcal{R}_I, Q_0, length)$;
  for each $(Q_0 \Rightarrow ... \Rightarrow Q) \in Der$
    if subgraph(A,Q) **then return true**;
  return false;
end

Fig. 3. The decidability algorithm

4 Example: Decidability in a Graph-Based Model for Discretionary Access Control

We describe a graph-based DAC model for which the safety property is decidable. This simplified DAC model provides subject and object creation/removal and the granting of rights to access objects from one subject to another subject. The terms subject and user are synonymies in this section. For simplicity, the access rights considered are *read* and *write* only.

4.1 Graph Model for the DAC Policy

Users are represented by nodes of type U, objects to which users may have access by nodes of type O. Labels attached to user nodes specify the user name, labels attached to objects the object name. The variables used in rules are denoted by $Ux, Uy, Ox, Oy, ...$

The rule *new user* in figure 4 introduces a user Ux. The creation of new objects (e.g., files and directories) on behalf of a user Ux is specified in the rule *new object*. Every object has an owner, namely the user who has created the object. The ownership of an object to a user is modeled by an edge from the user node to the object node. The owner of the new object has read and write access to the object. A permission on an object o for the owner is modeled by an edge from o to the owner with a label specifying the permitted access right (r for read and w for write). The owner of an object can simply remove the object (modeled in rule *delete object*). Users can be removed with rule *remove user*.

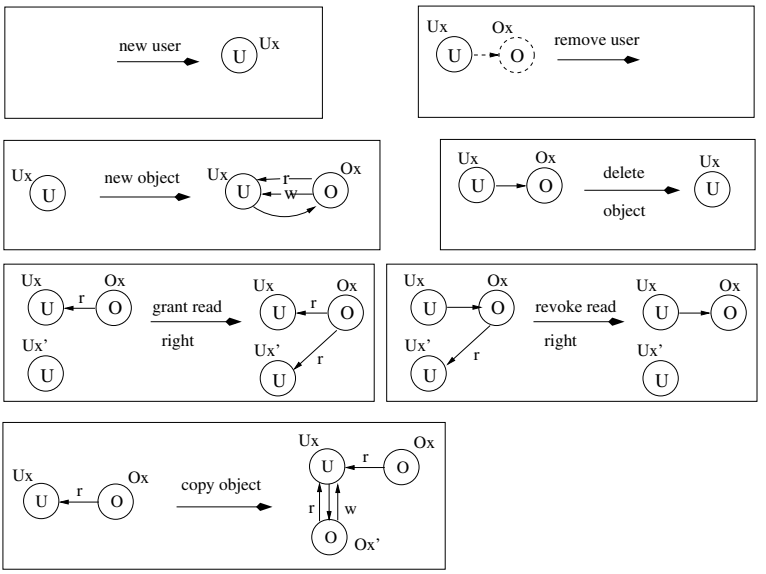


Fig. 4. The graph rules for the DAC model

To prevent objects without owner, the deletion of users is possible only if they do not own objects (specified in the NAC of rule *remove user* which forbids the application of *remove user* to a user who is connected to an object).

A user may grant its right on an object to other users. The rule *grant read right* specifies how a user Ux grants a read right on an object Ox (the right is modeled by an r-labeled edge from Ox to Ux) to another user Ux'. The r-labeled edge from Ox to Ux' specifies the read access right for Ux' on Ox. The rule for granting the write permission is analogously. Whereas an access right on an object can be granted by anybody who has the right, the revocation of a right on an object can be only executed by the object's owner. This is specified in rule *revoke read right*. The revocation of the write right is similar to rule *revoke read right*.

If a user Ux has read access to an object (i.e., there exists an edge with label r from the object to the user), and Ux copies the object, then Ux becomes the owner of the copy with read and write permission. The rule *copy object* specifies the copying of objects.

4.2 Safety of the DAC Model

This section carries on a safety analysis of the initial system state Q_0 shown in figure 5 with users *Richard*, *Jackie* and *Thomas*. *Jackie* is the owner of the object *newProject.pdf*. *Thomas* has read access to the object *newProject.pdf* and *Richard* has no right to access the object at all. *Jackie* has read and write access to the object because she is the object owner.

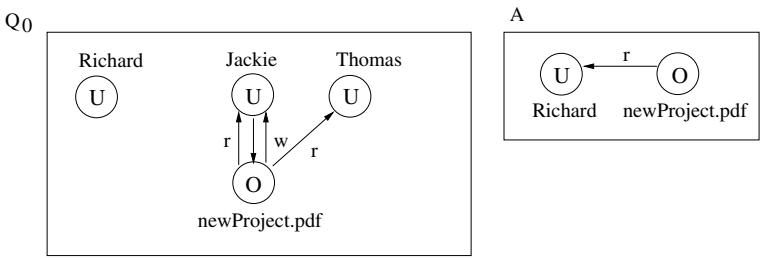


Fig. 5. An example for safety

Suppose that we would like to know now if it is possible for Richard to read *newProject.pdf* unbeknown to Jackie. That means, is it possible to construct a graph Q starting from Q_0 using the rules in figure 4 so that graph A of figure 5 is a subgraph of Q? Theorem 1 and corollary 1 state that it is sufficient to check all the derivation sequences of rule applications using rule instances of expanding rules up to a length of the upper bound $UB = card(\mathcal{R}_I^+) + R_F$. The expanding rules in figure 4 are: *new user*, *new object*, *grant read right* and *copy object*.

The number of rule instances $card(\mathcal{R}_I^+)$ depends on the (finite) set of lables for user names, where UN denotes the number of elements, and of the (finite) set of labels for object names, where we denote by ON the number of elements. For example, the number of rule instances for *new user* is equal to UN since *new user* has one variable for a user. The number of rule instances for rule *new object* is $UN \cdot ON$ since both a user and an object variable must be instantiated. The number of rule instances for rule *grant read right* is $UN \cdot (UN-1) \cdot ON$, since two user and one object node must be instantiated. Since node labels for nodes of the same type must be unique in the left-hand side of the rule (as well as in the right-hand side), we have $UN \cdot (UN-1)$ possibilities to instantiate the two user nodes with different labels. For the rule *copy object*, we get $UN \cdot ON \cdot (ON-1)$ possible rule instances. Table 1 shows the number of possible instance rules.

The last column of table 1 depicts the number of partial nonempty injective mappings from the right-hand side R of rule p into the graph A, so that at least some newly created elements in R are mapped to A. By the information of this last column, the value of R_F can be calculated (cf. corollary 1).

Table 1.

	number of instances	$card(R_f(p))$
new user(Ux)	UN	1
new object(Ux,Ox)	$UN \cdot ON$	3
grant read right(Ux,Ox)	$UN \cdot (UN-1) \cdot ON$	1
copy object(Ux,Ox)	$UN \cdot ON \cdot (ON-1)$	3

In this example, the upper bound is: $UN + UN \cdot ON + UN \cdot (UN - 1) \cdot ON + UN \cdot ON \cdot (ON - 1) + 8$. After having calculated the upper bound, only a finite number of derivation sequences with length less or equal than the upper bound are checked to decide the safety of the model. In particular, in this DAC example the derivation sequence which applies first the rule instance *copy object(Thomas,newProject.pdf)* to the initial state Q_0 and then the rule instance *grant read right(Thomas,Richard,newProject.pdf)* to the resulting graph constructs a graph which contains A. Therefore, the algorithm in figure 3 returns *true*, meaning that Q_0 leaks A.

5 Example: Decidability in a Graph-Based Model for Role-Based Access Control

The example in this section considers a simple graph model for decentralized RBAC (i.e., with more than one administrator for roles). In this model, where the safety property is decidable, a user is assigned to at most one role at a time and the assignment of a user to a role is static in the sense that it can only be changed by deleting the assignment and inserting a new one. The deletion of user-role assignment implies the loss of authorization for roles that were granted by the deleted assignment edge. This simplified RBAC graph model was originally presented in [KMPP00], and is reported in figure 6.

Users are created by the rule *add user* and are deleted by the rule *remove user*. A user can be assigned to a role if (s)he is not yet a member of a role. Membership is indicated by the color of the u node. A white u node indicates that the user is not yet in a role; a black one indicates that (s)he is a member of a role. The rule *add to role* turns a white user node to black when an administrator sets the assignment edge from the user to the role node. Since the roles authorization for each user is known, namely all junior roles reachable from the unique assignment edge, the deletion of a user-role assignment is simpler than in the general decentralized RBAC model. In particular, the deletion of the unique assignment edge ensures that the user does not have authorization for any role. In this case, all sessions are deactivated since all active roles for a session are authorized by the removed assignment edge, and the user node is changed to white.

A session is graphically presented by a node of type s and has always a connection to one user. The rules for the creation and deletion of sessions are *new session* and *remove session*. A session can be deleted at any time regardless of the presence of active roles of the session. The session is deleted by deleting the session node. This implies that all session-to-role assignments are deleted as well.

A user may activate any role r for which she/he is authorized. A user is authorized for r if there is a path starting from an assignment edge and ending in r. The corresponding graph rule is *activate role*. Rule *deactivate role* specifies the deactivation of a role from a session by deleting the edge between the session and the role node.

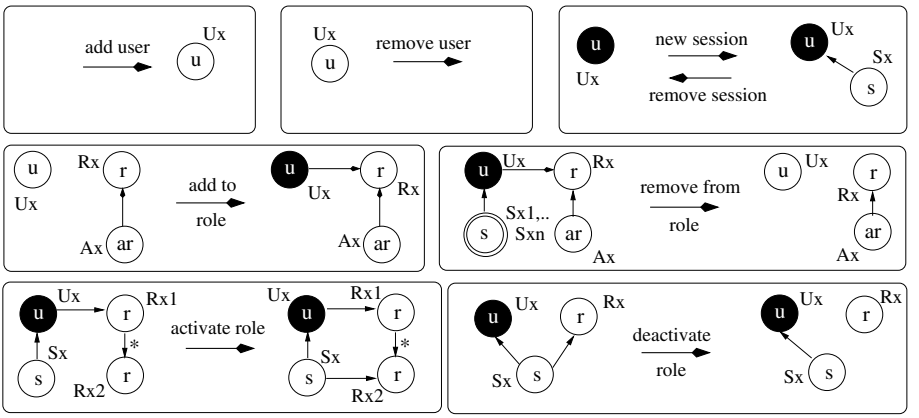


Fig. 6. Graph rules for the RBAC model

5.1 Safety of the RBAC Model

Suppose that we want to know if the initial state Q_0 in figure 7 can leak A, that is we would like to know whether *Elena* may play the role *President*. Table 2 shows the number of possible rule instances for the expanding rules of the RBAC example and the number of mappings from the right-hand side of the rules into the graph A. The number of rule instances depends on the finite label sets for user, session, role and administrator role variables. The label set for user includes the label *Elena*, the set of role labels is $\{President, ChiefManager, Manager\}$, the set of administrator role labels is $\{Bart, Anna\}$. We denote by UN the number of elements in the user label set, by SN the number of elements in the session label set, by $RN = 3$ the number of elements in the role label set and by $AN = 2$ the number of elments in the administrator role set.

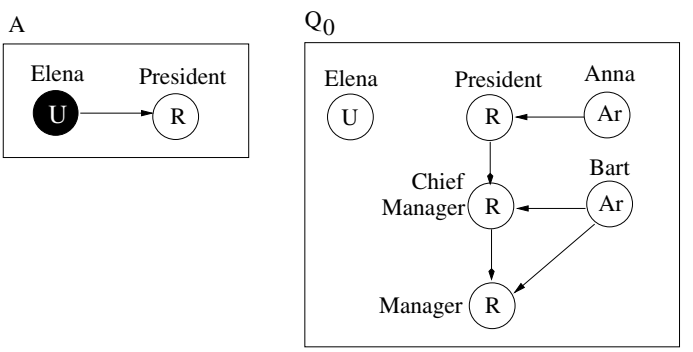


Fig. 7. A safety example for the RBAC model

Table 2.

	number of instances	$card(R_f(p))$
add user(Ux)	UN	1
new session(Ux,Sx)	$UN \cdot SN$	0
activate role(Ux,Rx1,Rx2,Sx)	$UN \cdot SN \cdot RN \cdot (RN-1)$	0
add to role	$UN \cdot RN \cdot AN$	1

The last column of table 2 depicts the number of nonempty partial injective mappings from the right-hand side R of the rule into graph A so that at least some newly created elements of R are mapped to A. Here, only the rules *add user* and *add to role* construct parts of A. After having calculated the upper bound, all derivation sequences with the given rule instances are checked up to the length given by the upper bound. In this example, the decidability algorithm returns *true* since the application of the rule instance *add to role(Elena, President, Anna)* constructs graph A.

Note that if we remove *Anna* from the administrator set AN (e.g., because we have confidence in *Anna*, but *Bart* must be checked explicitly), this RBAC example turns to be safe. Indeed, in the modified example, it will result that A cannot be constructed from Q_0. We will not list here all derivation sequences up to the upper bound, this can be automated with the help of a graph transformation tool [EEKR99]. However, we would like to informally convince the reader, that none of these derivation sequences constructs the graph A. The reason is that the rule *add to role(Elena,President,Anna)* cannot be instantiated, since the administrator label *Anna* is not in the label set AN for administrators. Indeed, only *Anna* can assign *Elena* to the president role, and *Elena* will never become president. Therefore, the decidability algorithm returns *false*.

6 Conclusion

The safety problem (can a state be reached in which a subject is granted a permission not previously possessed) in an arbitrary access control model based on graph transformations is in general undecidable.

This is not surprising since graph transformation rules can easily model the rules of a type 0 grammar. By imposing reasonable restrictions on the form of the rules, we have shown that the problem becomes decidable. The restrictions imposed still allow for an expressive framework as illustrated by the two examples of a graph-based DAC model and a graph-based single assignment decentralized RBAC. Moreover, it is possible in our framework to specify the Take-Grant model so that the rewrite rules *take, grant, create, remove* in [Sny77] satisfy the restrictions of our Theorem 1. Our notion of safety is more general than the one in [Sny77], since we test the presence of a general subgraph instead of a simple edge.

The safety problem "there exists a subject with a permission not previously possessed" can be handled, with a finite number of subjects, by defining a graph A for each specific subject. Alternatively, the problem can be handled by allowing the nodes in the graph A to be labelled with variables and checking if there exist an instantiation that is a subgraph of a state Q generated by the graph rules (similar to resolution in logic programming, shown in [CRPP91]).

The effective complexity of the decidability algorithm is probably much lower than the computed worst case since labels (or unique names) on nodes reduce considerably the number of possible matchings.

References

[AS00] G. Ahn and R. Sandhu. Role-based Authorization Constraint Specification. *ACM Trans. of Info. and System Security*, 3(4), 2000. 230

[CRPP91] A. Corradini, F. Rossi, and F. Parisi-Presicce. Logic programming as hypergraph rewriting. In *Proc. of CAAP91*, volume 493 of *LNCS*, pages 275–295. Springer, 1991. 243

[EEKR99] H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformations. Vol. II: Applications, Languages, and Tools*. World Scientific, 1999. 236, 242

[HRU76] M. A. Harrison, M. L. Ruzzo, and J. D. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, 1976. 229

[JT01] Trent Jaeger and Jonathan E. Tidswell. Practical safety in flexible access control models. *ACM Trans. of Info. and System Security*, 4(2), 2001. 230

[KMPP00] M. Koch, L. V. Mancini, and F. Parisi-Presicce. A Formal Model for Role-Based Access Control using Graph Transformation. In F.Cuppens, Y.Deswarte, D.Gollmann, and M.Waidner, editors, *Proc. of the 6th European Symposium on Research in Computer Security (ESORICS 2000)*, number 1895 in Lect. Notes in Comp. Sci., pages 122–139. Springer, 2000. 230, 233, 240

[KMPP01a] M. Koch, L. V. Mancini, and F. Parisi-Presicce. On the Specification and Evolution of Access Control Policies. In S. Osborne, editor, *Proc. 6th ACM Symp. on Access Control Models and Technologies*, pages 121–130. ACM, May 2001. 230, 233

[KMPP01b] M. Koch, L. V. Mancini, and F. Parisi-Presicce. Foundations for a graph-based approach to the Specification of Access Control Policies. In F.Honsell and M.Miculan, editors, *Proc. of Foundations of Software Science and Computation Structures (FoSSaCS 2001)*, Lect. Notes in Comp. Sci. Springer, March 2001. 230

[LS78] R. J. Lipton and L. Snyder. On synchronization and security. In Demillo et al., editor, *Foundations of Secure Computation*. Academic Press, 1978. 229

[NO99] M. Nyanchama and S. L. Osborne. The Role Graph Model and Conflict of Interest. *ACM Trans. of Info. and System Security*, 1(2):3–33, 1999. 230

[Roz97] G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformations. Vol. I: Foundations*. World Scientific, 1997. 230, 232

[Sny77] L. Snyder. On the Synthesis and Analysis of Protection Systems. In *Proc. of 6th Symposium on Operating System Principles*, volume 11 of *Operating System Review*, pages 141–150. ACM, 1977. 230, 242

[SS92] Ravi S. Sandhu and Gurpreet S. Suri. Non-Monotonic Transformation of Access Rights. In *Proc. IEEE Symposium on Research and Privacy*, pages 148–161, 1992. 229, 230