On the Suitability of Dissemination-centric Access Control Systems for Group-centric Sharing

Full Proofs

William C. Garrison III
bill@cs.pitt.edu

Yechen Qiao
yeq1@cs.pitt.edu

Adam J. Lee
adamlee@cs.pitt.edu

Department of Computer Science
University of Pittsburgh
Pittsburgh, Pennsylvania 15260

Revision bd5bbeb

1 g-SIS Instantiations

1.1 g-SIS$_0$ Model

States States in g-SIS$_0$ systems have the following fields.

- S, the set of subjects
- O, the set of objects
- G, the set of groups
- T, the set of times
- $>_T$, the total order on T
- $Time \in T$, the current time
- $StrictJoin \subseteq S \times G \times T$, the record of strict join events
- $LiberalJoin \subseteq S \times G \times T$, the record of liberal join events
- $StrictLeave \subseteq S \times G \times T$, the record of strict leave events
- $LiberalLeave \subseteq S \times G \times T$, the record of liberal leave events
- $StrictAdd \subseteq O \times G \times T$, the record of strict add events
- $LiberalAdd \subseteq O \times G \times T$, the record of liberal add events
- $StrictRemove \subseteq O \times G \times T$, the record of strict remove events
- $LiberalRemove \subseteq O \times G \times T$, the record of liberal remove events

Requests

- s, o, g for whether subject s has access to o through group g

Queries g-SIS$_0$ includes queries $Member$, $Assoc$, and $auth$. Below, we define these queries and several helper predicates that simplify their definition.

- $Join(s,g,t) \triangleq StrictJoin(s,g,t) \vee LiberalJoin(s,g,t)$
- $Leave(s,g,t) \triangleq StrictLeave(s,g,t) \vee LiberalLeave(s,g,t)$
- $Add(o,g,t) \triangleq StrictAdd(o,g,t) \vee LiberalAdd(o,g,t)$
- $Remove(o,g,t) \triangleq StrictRemove(o,g,t) \vee LiberalRemove(o,g,t)$
- $Member(s,g) \triangleq \exists t_1.($
  $Join(s,g,t_1) \wedge$
  $\forall t_2.($
  $Leave(s,g,t_2) \Rightarrow t_1 > t_2$
  $)$
  $)$
- $Assoc(o,g) \triangleq \exists t_1.($
  $Add(o,g,t_1) \wedge$
  $\forall t_2.($
  $Remove(o,g,t_2) \Rightarrow t_1 > t_2$
  $)$
  $)$
- $authForward(s,o,g) \triangleq \exists t_1,t_2.($
  $Join(s,g,t_1) \wedge$
  $Add(o,g,t_2) \wedge$
  $t_2 > t_1 \wedge$
  $\forall t_3.($
  $Leave(s,g,t_3) \Rightarrow (t_1 > t_3 \vee t_3 > t_2) \wedge$
  $StrictLeave(s,g,t_3) \Rightarrow t_2 > t_3 \wedge$
  $StrictRemove(o,g,t_3) \Rightarrow t_2 > t_3$
  $)$
  $)$
- $authBackward(s,o,g) \triangleq \exists t_1,t_2.($
  $LiberalJoin(s,g,t_1) \wedge$
  $LiberalAdd(o,g,t_2) \wedge$
  $t_1 > t_2 \wedge$
  $\forall t_3.($
  $Remove(o,g,t_3) \Rightarrow (t_2 > t_3 \vee t_3 > t_1) \wedge$
  $StrictLeave(s,g,t_3) \Rightarrow t_1 > t_3 \wedge$
  $StrictRemove(o,g,t_3) \Rightarrow t_1 > t_3$
  $)$
  $)$
- $auth(s,o,g) \triangleq authForward(s,o,g) \vee authBackward(s,o,g)$

1.2 g-SIS Systems

1.2.1 Role-like g-SIS

Labels

- $addS(s)$: Add $S(s)$
- $delS(s)$: Remove $S(s)$

- $addG(g)$: Add $G(g)$
- $delG(g)$: Remove $G(g)$
- $addO(o)$: Add $O(o)$
- $delO(o)$: Remove $O(o)$
- $liberalJoin(s, g)$: Remove $Time(t)$, add $LiberalJoin(s, g, t), Time(t + 1)$
- $strictLeave(s, g)$: Remove $Time(t)$, add $StrictLeave(s, g, t), Time(t + 1)$
- $liberalAdd(o, g)$: Remove $Time(t)$, add $LiberalAdd(o, g, t), Time(t + 1)$
- $strictRemove(o, g)$: Remove $Time(t)$, add $StrictRemove(o, g, t), Time(t + 1)$

Simplified $auth$ **Definition**

- $auth(s, o, g) \triangleq \exists t_1, t_2.($
  $LiberalJoin(s, g, t_1) \wedge$
  $LiberalAdd(o, g, t_2) \wedge$
  $\forall t_3.($
  $StrictLeave(s, g, t_3) \Rightarrow t_1 > t_3 \wedge$
  $StrictRemove(o, g, t_3) \Rightarrow t_2 > t_3$
  $)$
  $)$

1.2.2 Top g-SIS

Labels

- $addS(s)$: Add $S(s)$
- $delS(s)$: Remove $S(s)$
- $addG(g)$: Add $G(g)$
- $delG(g)$: Remove $G(g)$
- $addO(o)$: Add $O(o)$
- $delO(o)$: Remove $O(o)$
- $strictJoin(s, g)$: Remove $Time(t)$, add $StrictJoin(s, g, t), Time(t + 1)$
- $strictLeave(s, g)$: Remove $Time(t)$, add $StrictLeave(s, g, t), Time(t + 1)$
- $strictAdd(o, g)$: Remove $Time(t)$, add $StrictAdd(o, g, t), Time(t + 1)$
- $strictRemove(o, g)$: Remove $Time(t)$, add $StrictRemove(o, g, t), Time(t + 1)$

Simplified $auth$ **Definition**

- $auth(s, o, g) \triangleq \exists t_1, t_2.($
  $StrictJoin(s, g, t_1) \wedge$
  $StrictAdd(o, g, t_2) \wedge$
  $t_2 > t_1 \wedge$
  $\forall t_3.($
  $StrictLeave(s, g, t_3) \Rightarrow t_1 > t_3 \wedge$
  $StrictRemove(o, g, t_3) \Rightarrow t_2 > t_3$
  $)$
  $)$

1.2.3 Bottom g-SIS

Labels

- $addS(s)$: Add $S(s)$
- $delS(s)$: Remove $S(s)$
- $addG(g)$: Add $G(g)$
- $delG(g)$: Remove $G(g)$
- $addO(o)$: Add $O(o)$
- $delO(o)$: Remove $O(o)$
- $liberalJoin(s, g)$: Remove $Time(t)$, add $LiberalJoin(s, g, t), Time(t + 1)$
- $liberalLeave(s, g)$: Remove $Time(t)$, add $LiberalLeave(s, g, t), Time(t + 1)$
- $liberalAdd(o, g)$: Remove $Time(t)$, add $LiberalAdd(o, g, t), Time(t + 1)$
- $liberalRemove(o, g)$: Remove $Time(t)$, add $LiberalRemove(o, g, t), Time(t + 1)$

Simplified *auth* Definition

- $auth(s, o, g) \triangleq \exists t_1, t_2.($
  $LiberalJoin(s, g, t_1) \wedge$
  $LiberalAdd(o, g, t_2) \wedge$
  $($
  $t_2 > t_1 \wedge$
  $\forall t_3.(LiberalLeave(s, g, t_3) \Rightarrow t_1 > t_3 \vee t_3 > t_2)$
  $) \vee ($
  $t_1 > t_2 \wedge$
  $\forall t_3.(LiberalRemove(o, g, t_3) \Rightarrow t_2 > t_3 \vee t_3 > t_1)$
  $)$
  $)$

1.3 g-SIS Workloads

1.3.1 Program Committee

Labels

- $addS(s)$: Add $S(s)$
- $delS(s)$: Remove $S(s)$
- $addG(g)$: Add $G(g)$
- $delG(g)$: Remove $G(g)$
- $addO(o)$: Add $O(o)$
- $strictJoin(s, g)$: Remove $Time(t)$, add $StrictJoin(s, g, t), Time(t + 1)$
- $liberalJoin(s, g)$: Remove $Time(t)$, add $LiberalJoin(s, g, t), Time(t + 1)$
- $strictLeave(s, g)$: Remove $Time(t)$, add $StrictLeave(s, g, t), Time(t + 1)$
- $liberalLeave(s, g)$: Remove $Time(t)$, add $LiberalLeave(s, g, t), Time(t + 1)$
- $liberalAdd(o, g)$: Remove $Time(t)$, add $LiberalAdd(o, g, t), Time(t + 1)$

auth **Definition**

- $authForward(s, o, g) \triangleq \exists t_1, t_2.($
  $Join(s, g, t_1) \wedge$
  $LiberalAdd(o, g, t_2) \wedge$
  $t_2 > t_1 \wedge$
  $\forall t_3.($
  $Leave(s, g, t_3) \Rightarrow (t_1 > t_3 \vee t_3 > t_2) \wedge$
  $StrictLeave(s, g, t_3) \Rightarrow t_2 > t_3$
  $)$
  $)$
- $authBackward(s, o, g) \triangleq \exists t_1, t_2.($
  $LiberalJoin(s, g, t_1) \wedge$
  $LiberalAdd(o, g, t_2) \wedge$
  $t_1 > t_2 \wedge$
  $\forall t_3.($
  $StrictLeave(s, g, t_3) \Rightarrow t_1 > t_3$
  $)$
  $)$
- $auth(s, o, g) \triangleq authForward(s, o, g) \vee authBackward(s, o, g)$

Traces Valid traces include three phases. In the first phase, the creation phase, program committee groups are created. In the join phase, which follows, users liberal join discussion groups while papers are made available to these groups via liberal add. Occasionally, a user must resign with strict leave. Finally, during the review phase, users discuss the papers, posting reviews and messages to PC groups. When a user has a conflict of interest with upcoming discussion, she liberal leaves for a period of time before strict joining again.

1.3.2 PlayStation Plus

Labels

- $addS(s)$: Add $S(s)$
- $delS(s)$: Remove $S(s)$
- $addO(o)$: Add $O(o)$
- $delO(o)$: Remove $O(o)$
- $liberalJoin(s, g)$: Remove $Time(t)$, add $LiberalJoin(s, g, t), Time(t + 1)$
- $strictLeave(s, g)$: Remove $Time(t)$, add $StrictLeave(s, g, t), Time(t + 1)$
- $liberalAdd(o, g)$: Remove $Time(t)$, add $LiberalAdd(o, g, t), Time(t + 1)$
- $strictRemove(o, g)$: Remove $Time(t)$, add $StrictRemove(o, g, t), Time(t + 1)$
- $liberalRemove(o, g)$: Remove $Time(t)$, add $LiberalRemove(o, g, t), Time(t + 1)$

auth **Definition** Note that, due to its having restorative rejoin, this system is not a member of the g-SIS$_0$ model and therefore the following *auth* definition is not a simplified or special case of the one used in g-SIS$_0$.

- $auth(s, o, g) \triangleq Member(s, g) \wedge$
  $\exists t_1, t_2.($
  $LiberalJoin(s, g, t_1) \wedge$

$LiberalAdd(o, g, t_2) \wedge$
$\forall t_3.(StrictRemove(o, g, t_3) \Rightarrow t_2 > t_3) \wedge$
$($
$t_2 > t_1 \wedge$
$\forall t_3.(StrictLeave(s, g, t_3) \Rightarrow (t_1 > t_3 \vee t_3 > t_2))$
$) \vee ($
$t_1 > t_2 \wedge$
$\forall t_3.(Remove(o, g, t_3) \Rightarrow (t_2 > t_3 \vee t_3 > t_1))$
$)$
$)$

Traces Initial states include those with 2–5 groups, which represent regions (e.g., Sony's PlayStation Plus includes regions US, Europe, and Japan). One subject represents the service administrator, while others represent subscribers. Some objects represent free games, while others represent discounts. Several times per week (3, on average), the administrator will remove one free game, and replace it with another. In addition, also several times weekly (9 times per week, on average), the administrator will remove one discount from availability, and with the same rate adds new discounts. All objects are added to groups with liberal add. Discounts are removed with strict remove, while free games are removed with liberal add.

Subscribers join for fixed subscription periods—Sony offers 3-month and 1-year subscriptions.

2 Candidate Dissemination-centric Systems

2.1 Role-Based Access Control

This role-based system, $RBAC_0$, is based on the system of the same name in the RBAC standard [2].

States States in $RBAC_0$ have the following fields.

- U, the set of users
- R, the set of roles
- P, the set of permissions
- $UR \subseteq U \times R$, the user-role relation
- $PA \subseteq R \times P$, the role-permission relation

Requests

- u, p for whether user u has access to permission p

Queries

- $UR(u, r)$
- $PA(r, p)$
- $R(r)$
- $auth(u, p) \triangleq \exists r_1.(UR(u, r_1) \wedge PA(r_1, p))$

Labels

- $addU(u)$: Add $U(u)$
- $delU(u)$: Remove $U(u)$
- $addR(r)$: Add $R(r)$
- $delR(r)$: Remove $R(r)$
- $addP(p)$: Add $P(p)$
- $delP(p)$: Remove $P(p)$
- $assignUser(u, r)$: Add $UR(u, r)$
- $revokeUser(u, r)$: Remove $UR(u, r)$
- $assignPermission(r, p)$: Add $PA(r, p)$
- $revokePermission(r, p)$: Remove $PA(r, p)$

2.2 Role-Based Access Control with Role Hierarchy

This hierarchical role-based system, $RBAC_1$, is based on the system of the same name in the RBAC standard [2].

States States in $RBAC_1$ have the following fields.

- U, the set of users
- R, the set of roles
- P, the set of permissions
- $UR \subseteq U \times R$, the user-role relation
- $PA \subseteq R \times P$, the role-permission relation
- $RH \subseteq R \times R$, a partially ordered role hierarchy (written $\geq$ in infix notation)

Requests

- u, p for whether user u has access to permission p

Queries

- $UR(u, r)$
- $PA(r, p)$
- $R(r)$
- $RH(r_1, r_2)$
- $Senior(r_1, r_2) \triangleq RH(r_1, r_2) \vee \exists r_3.($
  $Senior(r_1, r_3) \wedge Senior(r_3, r_2)$
  $)$
- $auth(u, p) \triangleq \exists r_1, r_2.($
  $UR(u, r_1) \wedge PA(r_2, p) \wedge (r_1 = r_2 \vee Senior(r_1, r_2))$
  $)$

Labels

- $addU(u)$: Add $U(u)$
- $delU(u)$: Remove $U(u)$
- $addR(r)$: Add $R(r)$
- $delR(r)$: Remove $R(r)$
- $addP(p)$: Add $P(p)$
- $delP(p)$: Remove $P(p)$
- $assignUser(u, r)$: Add $UR(u, r)$
- $revokeUser(u, r)$: Remove $UR(u, r)$
- $assignPermission(r, p)$: Add $PA(r, p)$
- $revokePermission(r, p)$: Remove $PA(r, p)$
- $addHierarchy(r_1, r_2)$: Add $RH(r_1, r_2)$
- $removeHierarchy(r_1, r_2)$: Remove $RH(r_1, r_2)$

2.3 *ugo* System

The *ugo* system is based on UNIX's traditional user-group-other discretionary access control system.

States States in *ugo* have the following fields.

- S, the set of subjects
- O, the set of objects
- G, the set of groups
- $R = \{read, write, execute\}$, the set of rights
- $Member \subseteq S \times G$, the group-membership relation
- $Owner : O \to S$, the object-ownership record
- $Group : O \to G$, the object-group-membership record
- $OwnerRight \subseteq O \times R$, the granted owner rights for objects
- $GroupRight \subseteq O \times R$, the granted group rights for objects
- $OtherRight \subseteq O \times R$, the granted global rights for objects

Requests

- s, o, r for whether subject s has access to object o with right r.

Queries

- $G(g)$
- $Member(s, g)$
- $Group(o, g)$
- $OwnerAccess(s, o) \triangleq Owner(o, s)$
- $GroupAccess(s, o) \triangleq \neg Owner(o, s) \wedge \exists g_1.(Group(o, g_1) \wedge Member(s, g_1))$
- $OtherAccess(s, o) \triangleq \neg Owner(o, s) \wedge \forall g_1.(\neg Group(o, g_1) \vee \neg Member(s, g_1))$

- $auth(s, o, r) \triangleq$
  $OwnerAccess(s, o) \land OwnerRight(o, r) \lor$
  $GroupAccess(s, o) \land GroupRight(o, r) \lor$
  $OtherAccess(s, o) \land OtherRight(o, r)$

Labels

- $addS(s)$: Add $S(s)$
- $delS(s)$: Remove $S(s)$
- $addO(o)$: Add $O(o)$
- $delO(o)$: Remove $O(o)$
- $addG(g)$: Add $G(g)$
- $delG(g)$: Remove $G(g)$
- $changeOwner(o, s)$: Set $Owner(o) = s$
- $changeGroup(o, g)$: set $Group(o) = g$
- $grantOwner(o, r)$: Add $OwnerRight(o, r)$
- $revokeOwner(o, r)$: Remove $OwnerRight(o, r)$
- $grantGroup(o, r)$: Add $GroupRight(o, r)$
- $revokeGroup(o, r)$: Remove $GroupRight(o, r)$
- $grantOther(o, r)$: Add $OtherRight(o, r)$
- $revokeOther(o, r)$: Remove $OtherRight(o, r)$

3 Preliminary Proofs

3.1 Derived *auth* Definitions

Here, we prove the simplified *auth* definition of role-like g-SIS by simple logical deduction. The other *auth* definitions for systems belonging to the g-SIS$_0$ model (i.e., excluding the Playstation Plus system) can be proved similarly.

Lemma 1 *The $rgSIS$-specific definition of auth ($auth_r$) is equivalent to the general definition of auth for g-SIS$_0$ ($auth_0$) within the restricted context of the role-like g-SIS system $rgSIS$.*

$$
\begin{aligned}
&auth_0(s,o,g) \triangleq (\\
&\quad \exists t_1, t_2.(\\
&\quad\quad Join(s,g,t_1)\wedge\\
&\quad\quad Add(o,g,t_2)\wedge\\
&\quad\quad t_2 > t_1\wedge\\
&\quad\quad \forall t_3.(\\
&\quad\quad\quad Leave(s,g,t_3) \Rightarrow (t_1 > t_3 \vee t_3 > t_2)\wedge\\
&\quad\quad\quad StrictLeave(s,g,t_3) \Rightarrow t_2 > t_3\wedge\\
&\quad\quad\quad StrictRemove(o,g,t_3) \Rightarrow t_2 > t_3\\
&\quad\quad)\\
&\quad)\\
&) \vee (\\
&\quad \exists t_1, t_2.(\\
&\quad\quad LiberalJoin(s,g,t_1)\wedge\\
&\quad\quad LiberalAdd(o,g,t_2)\wedge\\
&\quad\quad t_1 > t_2\wedge\\
&\quad\quad \forall t_3.(\\
&\quad\quad\quad Remove(o,g,t_3) \Rightarrow (t_2 > t_3 \vee t_3 > t_1)\wedge\\
&\quad\quad\quad StrictLeave(s,g,t_3) \Rightarrow t_1 > t_3\wedge\\
&\quad\quad\quad StrictRemove(o,g,t_3) \Rightarrow t_1 > t_3\\
&\quad\quad)\\
&\quad)\\
&)
\end{aligned}
$$

$$
\begin{aligned}
&auth_r(s,o,g) \triangleq \exists t_1, t_2.(\\
&\quad LiberalJoin(s,g,t_1)\wedge\\
&\quad LiberalAdd(o,g,t_2)\wedge\\
&\quad \forall t_3.(\\
&\quad\quad StrictLeave(s,g,t_3) \Rightarrow t_1 > t_3\wedge\\
&\quad\quad StrictRemove(o,g,t_3) \Rightarrow t_2 > t_3\\
&\quad)\\
&)
\end{aligned}
$$

Proof Since role-like g-SIS includes only liberal join and add, and only strict leave and remove, we have:

$$
\begin{aligned}
&Join_r(s,g,t) \triangleq LiberalJoin(s,g,t)\\
&Leave_r(s,g,t) \triangleq StrictLeave(s,g,t)\\
&Add_r(o,g,t) \triangleq LiberalAdd(o,g,t)\\
&Remove_r(o,g,t) \triangleq StrictRemove(o,g,t)
\end{aligned}
$$

First, inspect $authForward$, the first half of the $auth_0$ definition.

$$
\begin{aligned}
&authForward_0(s,o,g) \triangleq \exists t_1, t_2.(\\
&\quad Join(s,g,t_1)\wedge\\
&\quad Add(o,g,t_2)\wedge\\
&\quad t_2 > t_1\wedge\\
&\quad \forall t_3.(\\
&\quad\quad Leave(s,g,t_3) \Rightarrow (t_1 > t_3 \vee t_3 > t_2)\wedge\\
&\quad\quad StrictLeave(s,g,t_3) \Rightarrow t_2 > t_3\wedge\\
&\quad\quad StrictRemove(o,g,t_3) \Rightarrow t_2 > t_3\\
&\quad)\\
&)
\end{aligned}
$$

Substitute $rgSIS$-specific expansions for $Join$, Add, and $Leave$.

$$
\begin{aligned}
& authForward_r(s, o, g) \Leftrightarrow \exists t_1, t_2.(\\
& \quad LiberalJoin(s, g, t_1) \wedge \\
& \quad LiberalAdd(o, g, t_2) \wedge \\
& \quad t_2 > t_1 \wedge \\
& \quad \forall t_3.(\\
& \quad\quad StrictLeave(s, g, t_3) \Rightarrow (t_1 > t_3 \vee t_3 > t_2) \wedge \\
& \quad\quad StrictLeave(s, g, t_3) \Rightarrow t_2 > t_3 \wedge \\
& \quad\quad StrictRemove(o, g, t_3) \Rightarrow t_2 > t_3 \\
& \quad) \\
&)
\end{aligned}
$$

Remove redundancy.

$$
\begin{aligned}
& authForward_r(s, o, g) \Leftrightarrow \exists t_1, t_2.(\\
& \quad LiberalJoin(s, g, t_1) \wedge \\
& \quad LiberalAdd(o, g, t_2) \wedge \\
& \quad t_2 > t_1 \wedge \\
& \quad \forall t_3.(\\
& \quad\quad StrictLeave(s, g, t_3) \Rightarrow t_1 > t_3 \wedge \\
& \quad\quad StrictRemove(o, g, t_3) \Rightarrow t_2 > t_3 \\
& \quad) \\
&)
\end{aligned}
$$

Next, we follow the same procedure for $authBackward$.

$$
\begin{aligned}
& authBackward_r(s, o, g) \Leftrightarrow \exists t_1, t_2.(\\
& \quad LiberalJoin(s, g, t_1) \wedge \\
& \quad LiberalAdd(o, g, t_2) \wedge \\
& \quad t_1 > t_2 \wedge \\
& \quad \forall t_3.(\\
& \quad\quad Remove(o, g, t_3) \Rightarrow (t_2 > t_3 \vee t_3 > t_1) \wedge \\
& \quad\quad StrictLeave(s, g, t_3) \Rightarrow t_1 > t_3 \wedge \\
& \quad\quad StrictRemove(o, g, t_3) \Rightarrow t_1 > t_3 \\
& \quad) \\
&)
\end{aligned}
$$

$$
\begin{aligned}
&authBackward_r(s,o,g) \Leftrightarrow \exists t_1, t_2.(\\
&\quad LiberalJoin(s,g,t_1)\wedge \\
&\quad LiberalAdd(o,g,t_2)\wedge \\
&\quad t_1 > t_2\wedge \\
&\quad \forall t_3.(\\
&\quad\quad StrictLeave(s,g,t_3) \Rightarrow t_1 > t_3\wedge \\
&\quad\quad StrictRemove(o,g,t_3) \Rightarrow t_2 > t_3\wedge \\
&\quad) \\
&)
\end{aligned}
$$

Now, since $auth_0(s,o,g) \triangleq authForward_0(s,o,g) \vee authBackward_0(s,o,g)$,

$$
\begin{aligned}
&auth_r(s,o,g) \Leftrightarrow (\\
&\quad \exists t_1, t_2.(\\
&\quad\quad LiberalJoin(s,g,t_1)\wedge \\
&\quad\quad LiberalAdd(o,g,t_2)\wedge \\
&\quad\quad t_2 > t_1\wedge \\
&\quad\quad \forall t_3.(\\
&\quad\quad\quad StrictLeave(s,g,t_3) \Rightarrow t_1 > t_3\wedge \\
&\quad\quad\quad StrictRemove(o,g,t_3) \Rightarrow t_2 > t_3 \\
&\quad\quad) \\
&\quad) \\
&) \vee (\\
&\quad \exists t_1, t_2.(\\
&\quad\quad LiberalJoin(s,g,t_1)\wedge \\
&\quad\quad LiberalAdd(o,g,t_2)\wedge \\
&\quad\quad t_1 > t_2\wedge \\
&\quad\quad \forall t_3.(\\
&\quad\quad\quad StrictLeave(s,g,t_3) \Rightarrow t_1 > t_3\wedge \\
&\quad\quad\quad StrictRemove(o,g,t_3) \Rightarrow t_2 > t_3\wedge \\
&\quad\quad) \\
&\quad) \\
&)
\end{aligned}
$$

$$
\begin{aligned}
& auth_r(s,o,g) \Leftrightarrow \exists t_1, t_2.(\\
& \quad LiberalJoin(s,g,t_1) \wedge \\
& \quad LiberalAdd(o,g,t_2) \wedge \\
& \quad (\\
& \qquad t_2 > t_1 \wedge \\
& \qquad \forall t_3.(\\
& \qquad\quad StrictLeave(s,g,t_3) \Rightarrow t_1 > t_3 \wedge \\
& \qquad\quad StrictRemove(o,g,t_3) \Rightarrow t_2 > t_3 \\
& \qquad) \\
& \quad) \vee (\\
& \qquad t_1 > t_2 \wedge \\
& \qquad \forall t_3.(\\
& \qquad\quad StrictLeave(s,g,t_3) \Rightarrow t_1 > t_3 \wedge \\
& \qquad\quad StrictRemove(o,g,t_3) \Rightarrow t_2 > t_3 \wedge \\
& \qquad) \\
& \quad) \\
&)
\end{aligned}
$$

$$
\begin{aligned}
& auth_r(s,o,g) \Leftrightarrow \exists t_1, t_2.(\\
& \quad LiberalJoin(s,g,t_1) \wedge \\
& \quad LiberalAdd(o,g,t_2) \wedge \\
& \quad \forall t_3.(\\
& \qquad StrictLeave(s,g,t_3) \Rightarrow t_1 > t_3 \wedge \\
& \qquad StrictRemove(o,g,t_3) \Rightarrow t_2 > t_3 \\
& \quad) \wedge \\
& \quad t_2 > t_1 \vee t_1 > t_2 \\
&)
\end{aligned}
$$

$$
\begin{aligned}
& auth_r(s,o,g) \Leftrightarrow \exists t_1, t_2.(\\
& \quad LiberalJoin(s,g,t_1) \wedge \\
& \quad LiberalAdd(o,g,t_2) \wedge \\
& \quad \forall t_3.(\\
& \qquad StrictLeave(s,g,t_3) \Rightarrow t_1 > t_3 \wedge \\
& \qquad StrictRemove(o,g,t_3) \Rightarrow t_2 > t_3 \\
& \quad) \\
&)
\end{aligned}
$$

Which is the definition as proposed. □

3.2 Weak AC-Preservation

Definition 1 Given a workload $\mathcal{W}$, a system $\mathcal{Y}$, and an implementation $\mathcal{I} = \langle \alpha, \sigma, \pi \rangle$, π (and thus $\mathcal{I}$) is *weak AC-preserving* if there exists a request transformation $f : \mathit{Requests}(\mathcal{W}) \rightarrow \mathit{Requests}(\mathcal{Y})$ such that for any workload state w, workload request r, and system request r', the following conditions hold.

1. $\pi_{auth(r)}(Th(\sigma(w))) = \text{TRUE} \Rightarrow \sigma(w) \vdash auth(f(r))$
2. $\sigma(w) \vdash auth(r') \Rightarrow \exists r.(\pi_{auth(r)}(Th(\sigma(w))) = \text{TRUE} \wedge f(r) = r')$ $\Diamond$

Lemma 2 *Given two weak AC-preserving mappings, π^1 and π^2, $\pi^1 \circ \pi^2$ is weak AC-preserving.*

Proof Assume π^1 is a query mapping from $\mathcal{X}$ to $\mathcal{Y}$ (i.e., a mapping from $\mathcal{Y}$ theories to $\mathcal{X}$ theories) and π^2 is a query mapping from $\mathcal{Y}$ to $\mathcal{Z}$ (i.e., a mapping from $\mathcal{Z}$ theories to $\mathcal{Y}$ theories).

Given π^1 is weak AC-preserving, there exists a request transform f^1 such that, for any $\mathcal{Y}$ theory $T^{\mathcal{Y}}$, $\mathcal{X}$ request r, and $\mathcal{Y}$ request r':

$$auth(r) \in \pi^1(T^{\mathcal{Y}}) \Rightarrow auth(f^1(r)) \in T^{\mathcal{Y}}$$
$$auth(r') \in T^{\mathcal{Y}} \Rightarrow \exists r.(auth(r) \in \pi^1(T^{\mathcal{Y}}) \wedge f^1(r) = r')$$

Given π^2 is weak AC-preserving, there exists a request transform f^2 such that, for any $\mathcal{Z}$ theory $T^{\mathcal{Z}}$, $\mathcal{Y}$ request r, and $\mathcal{Z}$ request r':

$$auth(r) \in \pi^2(T^{\mathcal{Z}}) \Rightarrow auth(f^2(r)) \in T^{\mathcal{Z}}$$
$$auth(r') \in T^{\mathcal{Z}} \Rightarrow \exists r.(auth(r) \in \pi^2(T^{\mathcal{Z}}) \wedge f^2(r) = r')$$

Then, we show that $\pi^1 \circ \pi^2$ is a weak AC-preserving query mapping from $\mathcal{X}$ to $\mathcal{Z}$ (i.e., mapping from $\mathcal{Z}$ theories to $\mathcal{X}$ theories).

Choose an arbitrary $\mathcal{Z}$ theory, $T^{\mathcal{Z}}$, and $\mathcal{X}$ request, r. Assume $auth(r) \in \pi^1(\pi^2(T^{\mathcal{Z}}))$. By weak AC-preservation of π^1, $auth(f^1(r)) \in \pi^2(T^{\mathcal{Z}})$. By weak AC-preservation of π^2, $auth(f^2(f^1(r))) \in T^{\mathcal{Z}}$. Thus, $(\pi^1 \circ \pi^2)_{auth(r)}(T^{\mathcal{Z}}) = \text{TRUE} \Rightarrow auth(f^2(f^1(r))) \in T^{\mathcal{Z}}$, and we have proved condition (1) for weak AC-preservation of $\pi^1 \circ \pi^2$ with request transform $f^2 \circ f^1$.

Next, choose an arbitrary $\mathcal{Z}$ theory, $T^{\mathcal{Z}}$, and $\mathcal{Z}$ request, r''. Assume $auth(r'') \in T^{\mathcal{Z}}$. By weak AC-preservation of π^2, $\exists r'.(auth(r') \in \pi^2(T^{\mathcal{Z}}) \wedge f^2(r') = r'')$. By weak AC-preservation of π^1, $\exists r.(auth(r) \in \pi^1(\pi^2(T^{\mathcal{Z}}))) \wedge f^2(f^1(r)) = r'')$. Thus, $auth(r'') \in T^{\mathcal{Z}} \Rightarrow \exists r.((\pi^1 \circ \pi^2)_{auth(r)}(T^{\mathcal{Z}}) = \text{TRUE} \wedge f^2(f^1(r)) = r'')$, and we have proved condition (2) for weak AC-preservation of $\pi^1 \circ \pi^2$ with request transform $f^2 \circ f^1$.

Thus, if π^1 is weak AC-preserving with transform f^1, and π^2 is weak AC-preserving with transform f^2, then $\pi^1 \circ \pi^2$ is weak AC-preserving with transform $f^2 \circ f^1$. $\square$

Theorem 3 *Given a correctness-preserving reduction $\langle \sigma, \pi \rangle$ from $\mathcal{Y}$ to $\mathcal{Z}$ where π is weak AC-preserving, then $\mathcal{Y} \leq^{Ca} \mathcal{Z}$.*

Proof Since $\langle \sigma, \pi \rangle$ is correctness-preserving, we know already that $\mathcal{Y} \leq^{C} \mathcal{Z}$ (this can be proved using either Theorem 1 from [1] or Corollary 4 from Section 3.3 of the present work). Thus, we must simply show that, given a reduction $\langle \sigma, \pi \rangle$ from $\mathcal{Y}$ to $\mathcal{Z}$ where π is weak AC-preserving; and a weak AC-preserving implementation $\langle \alpha^{\mathcal{Y}}, \sigma^{\mathcal{Y}}, \pi^{\mathcal{Y}} \rangle$ of workload $\mathcal{W}$ in $\mathcal{Y}$: the implementation $\langle \alpha^{\mathcal{Z}}, \sigma^{\mathcal{Z}}, \pi^{\mathcal{Z}} \rangle$ of $\mathcal{W}$ in $\mathcal{Z}$ is weak AC-preserving.

Since $\langle \sigma, \pi \rangle$ and $\langle \alpha^{\mathcal{Y}}, \sigma^{\mathcal{Y}}, \pi^{\mathcal{Y}} \rangle$ are weak AC-preserving, π and $\pi^{\mathcal{Y}}$ are weak AC-preserving, say with request transforms f and $f^{\mathcal{Y}}$, respectively. Thus, by Lemma 2, $\pi^{\mathcal{Z}} = \pi^{\mathcal{Y}} \circ \pi$ is weak AC-preserving with transform $f^{\mathcal{Z}} = f \circ f^{\mathcal{Y}}$, and thus $\langle \alpha^{\mathcal{Z}}, \sigma^{\mathcal{Z}}, \pi^{\mathcal{Z}} \rangle$ is weak AC-preserving.

$\therefore$ Given a correctness-preserving reduction with weak AC-preserving query mapping from $\mathcal{Y}$ to $\mathcal{Z}$, $\mathcal{Y} \leq^{Ca} \mathcal{Z}$. $\square$

3.3 Pseudo-Injective State Mappings

Definition 2 Given access control systems $\mathcal{Y}_1 = \langle \mathcal{M}_1, \mathcal{L}_1, next_1 \rangle$ and $\mathcal{Y}_2 = \langle \mathcal{M}_2, \mathcal{L}_2, next_2 \rangle$ and reduction $\langle \sigma, \pi \rangle$ from $\mathcal{Y}_1$ to $\mathcal{Y}_2$, σ is *pseudo-injective* if, for all states x_1 and x_2 in $\mathcal{Y}_1$, if $\sigma(x_1) = \sigma(x_2)$, then $\forall \ell \in \mathcal{L}.(\sigma(next(x_1, \ell)) = \sigma(next(x_2, \ell)))$. $\diamondsuit$

Corollary 4 *If there is a reduction $\langle \sigma, \pi \rangle$ from $\mathcal{Y}_1$ to $\mathcal{Y}_2$ where σ is pseudo-injective and preserves reachability, then $\mathcal{Y}_1 \leq^C \mathcal{Y}_2$.*

PROOF This proof proceeds exactly as in the proof of Theorem 8 in [1], up until the "potential problem" of two workload states, w_1 and w_2, which map to the same $\mathcal{Y}_2$ state y. Since we have generalized the one-to-one property required in Theorem 8 [1], we cannot use the same argument to solve this problem in this case.

However, knowing that the state mapping from $\mathcal{Y}_1$ to $\mathcal{Y}_2$ is pseudo-injective, we thus know that one of the following is true.

- Workload states w_1 and w_2 map to the same $\mathcal{Y}_1$ state x, in which case the argument from the proof of Theorem 8 [1] holds (these states need not be implemented differently).
- Workload states w_1 and w_2 map to different $\mathcal{Y}_1$ states, x_1 and x_2 respectively, which both map to y in $\mathcal{Y}_2$. Since σ is pseudo-injective, $\forall \ell \in Labels(\mathcal{Y}_1).(\sigma(next(x_1, \ell)) = \sigma(next(x_2, \ell)))$. Thus, x_1 and x_2 need not be implemented differently (and therefore $\alpha^{\mathcal{Y}_2}(y, l)$ has a well-defined value).

The proof then proceeds again as in Theorem 8 [1], and we have shown that it is sufficient for a correctness-preserving reduction that σ is pseudo-injective, even if it is not one-to-one. $\square$

3.4 Revisiting Reduction Transitivity

Proposition 5 *Suppose ρ_1 is a reduction from $\mathcal{Y}_1$ to $\mathcal{Y}_2$ and ρ_2 is a reduction from $\mathcal{Y}_2$ to $\mathcal{Y}_3$, where ρ_1 and ρ_2 are weak AC-preserving. Then there is a reduction from $\mathcal{Y}_1$ to $\mathcal{Y}_3$ that is weak AC-preserving.*

PROOF Suppose $\rho_1 = \langle \sigma^1, \pi^1 \rangle$ and $\rho_2 = \langle \sigma^2, \pi^2 \rangle$. Then define the reduction ρ_3 from $\mathcal{Y}_1$ to $\mathcal{Y}_2$ as follows (it is shown in [1] that this construction results in a valid reduction).

$$\pi^3(x) = \pi^1(\pi^2(x))$$
$$\sigma^3(x) = \sigma^2(\sigma^1(x))$$

Say π^1 is weak AC-preserving with request tranform f^1, and π^2 with f^2. By Lemma 2, $\pi^3 = \pi^1 \circ \pi^2$ is weak AC-preserving with transform $f^2 \circ f^1$.

Thus, ρ_3 is weak AC-preserving. $\square$

Proposition 6 *Suppose ρ_1 is a reduction from $\mathcal{Y}_1$ to $\mathcal{Y}_2$ and ρ_2 is a reduction from $\mathcal{Y}_2$ to $\mathcal{Y}_3$, where ρ_1 is pseudo-injective and ρ_2 is injective (one-to-one). Then there is a reduction from $\mathcal{Y}_1$ to $\mathcal{Y}_3$ that is pseudo-injective.*

PROOF Suppose $\rho_1 = \langle \sigma_1, \pi_1 \rangle$ and $\rho_2 = \langle \sigma_2, \pi_2 \rangle$. As in the previous proof, define the reduction ρ_3 from $\mathcal{Y}_1$ to $\mathcal{Y}_2$ as follows.

$$\pi_3(x) = \pi_1(\pi_2(x))$$
$$\sigma_3(x) = \sigma_2(\sigma_1(x))$$

Assume ρ_1 is pseudo-injective and ρ_2 is injective. Then, for two states x_1 and x_2 in $\mathcal{Y}_1$, if $\sigma_3(x_1) = \sigma_3(x_2)$, then $\sigma_2(\sigma_1(x_1)) = \sigma_2(\sigma_1(x_2))$.

Since ρ_2 is injective, $\sigma_1(x_1) = \sigma_1(x_2)$. Then, by the pseudo-injectiveness of ρ_1, for any label ℓ_1 in $\mathcal{Y}_1$, $\sigma_1(next(x_1, \ell_1)) = \sigma_1(next(x_2, \ell_1))$. Of course, now $\sigma_2(\sigma_1(next(x_1, \ell))) = \sigma_2(\sigma_1(next(x_2, \ell)))$, and $\sigma_3(next(x_1, \ell)) = \sigma_3(next(x_2, \ell))$.

Thus, ρ_3 is pseudo-injective. □

3.5 Reduction-Implied Implementations

Here we prove that a reduction $\mathcal{Y} \leq^{\mathcal{G}} \mathcal{Z}$ implies there exists an implementation of $\langle \mathcal{Y}, \mathcal{T} \rangle$ in $\mathcal{Z}$ with guarantees $\mathcal{G}$ for any set of traces $\mathcal{T}$.

Lemma 7 *Given access control systems $\mathcal{Y}$ and $\mathcal{Z}$, a set of security guarantees $\mathcal{G}$, and any set $\mathcal{T}$ of traces over $\mathcal{Y}$; if there exists a reduction from $\mathcal{Y}$ to $\mathcal{Z}$ that preserves $\mathcal{G}$ ($\mathcal{Y} \leq^{\mathcal{G}} \mathcal{Z}$) then there exists an implementation of workload $\langle \mathcal{Y}, \mathcal{T} \rangle$ in $\mathcal{Z}$ with guarantees $\mathcal{G}$.*

Proof It is clear that $\mathcal{Y}$ can trivially implement workload $\langle \mathcal{Y}, \mathcal{T} \rangle$ with guarantees $\mathcal{G}$. By definition of parameterized expressiveness, $\mathcal{Y} \leq^{\mathcal{G}} \mathcal{Z}$ says that any workload that can be implemented in $\mathcal{Y}$ with guarantees $\mathcal{G}$ can be implemented in $\mathcal{Z}$ with guarantees $\mathcal{G}$. Thus, there exists an implementation of $\langle \mathcal{Y}, \mathcal{T} \rangle$ in $\mathcal{Z}$. □

Corollary 8 *Given access control systems $\mathcal{Y}$ and $\mathcal{Z}$, a set of security guarantees $\mathcal{G}$, and any set $\mathcal{T}$ of traces over $\mathcal{Y}$; if there does not exist an implementation of workload $\langle \mathcal{Y}, \mathcal{T} \rangle$ in $\mathcal{Z}$ with guarantees $\mathcal{G}$, then there does not exist a reduction from $\mathcal{Y}$ to $\mathcal{Z}$ that preserves $\mathcal{G}$ ($\mathcal{Y} \not\leq^{\mathcal{G}} \mathcal{Z}$).*

Proof Follows immediately from Lemma 7. □

4 Reductions

4.1 Role-like g-SIS and $RBAC_0$

Theorem 9 *There exists a reduction $\langle \sigma, \pi \rangle$ from role-like g-SIS to $RBAC_0$ where:*

- *σ preserves π, is pseudo-injective, preserves reachability, and is homomorphic*
- *π is weak AC-preserving and homomorphic*

Thus, $rgSIS \leq^{CaH} RBAC_0$ ($RBAC_0$ is at least as expressive as role-like g-SIS with respect to correctness, weak AC-preservation, and homomorphism).

Proof We present the reduction, $\langle \sigma, \pi \rangle$. First, σ maps the g-SIS state $\langle S, O, G, T, Time, LiberalJoin, StrictLeave, LiberalAdd, StrictRemove \rangle$ to an $RBAC_0$ state of the form $\langle U, R, P, UR, PA \rangle$. This mapping is described by the following HPL method (and is thus homomorphic).

```
for each (S(s) ∈ M)
   output(U(s))
for each (G(g) ∈ M)
   output(R(g))
for each (O(o) ∈ M)
   output(P(o))

for each (LiberalJoin(s, g, t) ∈ M)
   Old = {}
   for each (StrictLeave(s, g, x) ∈ M)
      If >_T(x, t) ∈ M then
         Old = {<s, g>}
```

```
        endif
    If <s, g> ∉ Old
        output(UR(s, g))
    endif

for each (LiberalAdd(o, g, t) ∈ M)
    Old = {}
    for each (StrictRemove(o, g, x) ∈ M)
        If >_T(x, t) ∈ M then
            Old = {<o, g>}
        endif
    If <o, g> ∉ Old
        output(PA(g, o))
    endif
```

π is defined as follows.

$$\pi_{Member(s,g)}(T) = UR(s,g) \in T$$
$$\pi_{Assoc(o,g)}(T) = PA(g,o) \in T$$
$$\pi_{auth(s,o,g)}(T) = UR(s,g), PA(g,o) \in T$$

This query mapping clearly contains no string manipulation and is thus homomorphic.

Let x be an arbitrary $rgSIS$ state and $\lambda = (s,o,g)$ an arbitrary $rgSIS$ request, and let $f(s,o,g) = (s,o)$ be a request transform. Assume $\pi_{auth(\lambda)}(Th(\sigma(x))) = \text{TRUE}$. Then, $UR(s,g) \in Th(\sigma(x)) \wedge PA(g,o) \in Th(\sigma(x))$. Thus, it is clear that $\exists r.(UR(s,r) \in Th(\sigma(x)) \wedge PA(r,o) \in Th(\sigma(x)))$, and therefore $\sigma(x) \vdash auth(f(\lambda))$.

Now let x be an arbitrary $rgSIS$ state, $\lambda' = (u,p)$ an arbitrary $RBAC_0$ request, and f the request transform defined above. Assume $\sigma(x) \vdash auth(\lambda')$. Then, $\exists r.(UR(s,r) \in Th(\sigma(x)) \wedge PA(r,o) \in Th(\sigma(x)))$. Finally, $f(u,p,r) = (u,p)$, and $\pi_{auth(u,p,r)}(Th(\sigma(x))) = \text{TRUE}$. Thus, π is weak AC-preserving with transform $f(s,o,g) = (s,o)$.

We show that σ preserves π (for all $rgSIS$ states x, $Th(x) = \pi(Th(\sigma(x))))$ by contradiction. Assume that there is some $rgSIS$ state x and query q such that the value of q in x is the opposite of the value of $\pi(q)$ in $\sigma(x)$. We show that, for each of the query forms of $rgSIS$, this assumption leads to contradiction.

- **Member** Assume $x \vdash Member(s,g)$ and $\sigma(x) \nvdash \pi(Member(s,g))$. Then, $\exists t_1.(LiberalJoin(s,g,t_1) \in Th(x) \wedge \forall t_2.(LiberalLeave(s,g,t_2) \in Th(x) \Rightarrow t_1 > t_2))$ (s has joined g and not left). By σ, $UR(s,g) \in Th(\sigma(x))$. Thus, by π, $\pi_{Member(s,g)}(Th(\sigma(x))) = \text{TRUE}$, which is a contradiction on the assumption that $\sigma(x) \nvdash \pi(Member(s,g))$.

  Assume instead that $x \nvdash Member(s,g)$ and $\sigma(x) \vdash \pi(Member(s,g))$. Then, either $\exists t_1.(LiberalLeave(s,g,t_1) \in Th(x) \wedge \forall t_2.(LiberalJoin(s,g,t_2) \in Th(x) \Rightarrow t_1 > t_2))$ (s has left g and not returned), or $\forall t_1.(LiberalJoin(s,g,t_1) \notin Th(x))$ (s has not joined g). By σ, in either case, $UR(s,g) \notin Th(\sigma(x))$. Thus, by π, $\pi_{Member(o,g)}(Th(\sigma(x))) = \text{FALSE}$, which is a contradiction on the assumption that $\sigma(x) \vdash \pi(Member(s,g))$.

- **Assoc** Assume $x \vdash Assoc(o,g)$ and $\sigma(x) \nvdash \pi(Assoc(o,g))$. Then, $\exists t_1.(LiberalAdd(o,g,t_1) \in Th(x) \wedge \forall t_2.(LiberalRemove(o,g,t_2) \in Th(x) \Rightarrow t_1 > t_2))$ (o was added to g and not removed). Thus, by σ, $PA(g,o) \in Th(\sigma(x)))$. By π, $\pi_{Assoc(o,g)}(Th(\sigma(x))) = \text{TRUE}$, which is a contradiction on the assumption that $\sigma(x) \nvdash \pi(Assoc(o,g))$.

  Assume instead that $x \nvdash Assoc(o,g)$ and $\sigma(x) \vdash \pi(Assoc(o,g))$. Then, either $\exists t_1.(LiberalRemove(o,g,t_1) \in Th(x) \wedge \forall t_2.(LiberalAdd(o,g,t_2) \in Th(x) \Rightarrow t_1 > t_2))$ (s was removed from g and not re-added), or $\forall t_1.(LiberalAdd(o,g,t_1) \notin Th(x))$ (o has not added to

g). By σ, in either case, $PA(g,o) \notin Th(\sigma(x))$). Thus, by π, $\pi_{Assoc(o,g)}(Th(\sigma(x))) = \text{FALSE}$, which is a contradiction on the assumption that $\sigma(x) \vdash \pi(Assoc(o,g))$.

- **auth** Assume $x \vdash auth(s,o,g)$ and $\sigma(x) \nvdash \pi(auth(s,o,g))$. Then, $\exists t_1.(LiberalJoin(s,g,t_1) \in Th(x) \wedge \forall t_2.(LiberalLeave(s,g,t_2) \in Th(x) \Rightarrow t_1 > t_2))$ (s has joined g and not left), and $\exists t_1.(LiberalAdd(o,g,t_1) \in Th(x) \wedge \forall t_2.(LiberalRemove(o,g,t_2) \in Th(x) \Rightarrow t_1 > t_2))$ (o was added to g and not removed). By σ, $UR(s,g) \in Th(\sigma(x)) \wedge PA(g,o) \in Th(\sigma(x))$. Thus, by π, $\pi_{auth(s,o,g)}(Th(\sigma(x))) = \text{TRUE}$, which is a contradiction on the assumption that $\sigma(x) \nvdash \pi(auth(s,o,g))$

  Assume instead that $x \nvdash auth(s,o,g)$ and $\sigma(x) \vdash \pi(auth(s,o,g))$. Then, there are four possibilities which we consider in pairs. If $\exists t_1.(LiberalLeave(s,g,t_1) \in Th(x) \wedge \forall t_2.(LiberalJoin(s,g,t_2) \in Th(x) \Rightarrow t_1 > t_2))$ (s has left g and not returned), or $\forall t_1.(LiberalJoin(s,g,t_1) \notin Th(x))$ (s has not joined g), then by σ, $UR(s,g) \notin Th(\sigma(x))$, and thus by π, $\pi_{auth(s,o,g)}(Th(\sigma(x))) = \text{FALSE}$ (a contradiction). If instead $\exists t_1.(LiberalRemove(o,g,t_1) \in Th(x) \wedge \forall t_2.(LiberalAdd(o,g,t_2) \in Th(x) \Rightarrow t_1 > t_2))$ (s was removed from g and not re-added), or $\forall t_1.(LiberalAdd(o,g,t_1) \notin Th(x))$ (o has not added to g), then by σ, $PA(g,o) \notin Th(\sigma(x)))$, and thus by π, $\pi_{auth(s,o,g)}(Th(\sigma(x))) = \text{FALSE}$.

Thus, by contradiction, σ preserves π.

For all $rgSIS$ states s, s', if s' is reachable from s, then there exists a sequence of labels $\langle \ell_1, \ell_2, \ldots, \ell_n \rangle$ such that $terminal(s, \ell_1 \circ \ell_2 \circ \cdots \circ \ell_n) = s'$. We will show that, for any $rgSIS$ state x and label ℓ, $\sigma(next(x,\ell))$ is reachable from $\sigma(x)$ via $RBAC_0$ labels. By induction, this will show that for each intermediate $rgSIS$ state s_i between s and s', $\sigma(s_i)$ is reachable from $\sigma(s)$ and ultimately that $\sigma(s')$ is reachable from $\sigma(s)$ (i.e., that σ preserves reachability).

Given $rgSIS$ state x and label ℓ, $x' = next(x,\ell)$ is the state resulting from executing label ℓ in state x.

- If ℓ is an instance of $addS(s)$, then $x' = next(x,\ell) = x \cup S(s)$. By σ, this maps in $RBAC_0$ to state $\sigma(x') = \sigma(x \cup S(s)) = \sigma(x) \cup U(s)$. By $RBAC_0$'s $next$ relation, $next(\sigma(x), addU(s)) = \sigma(x) \cup U(s)$. Thus, if ℓ is an instance of $addS(s)$, $\sigma(x')$ is reachable from $\sigma(x)$ via execution of $addU(s)$. A similar argument holds for instances of $addG(g)$ and $addO(o)$ (with reachability in $RBAC_0$ via $addR(g)$ and $addP(o)$, respectively).

- If ℓ is an instance of $delS(s)$, then $x' = x \setminus (S(s) \cup Entries(x,s))$, where $Entries(x,s)$ denotes the set of all state tuples in x involving s[1]. By σ, this maps in $RBAC_0$ to state $\sigma(x') = \sigma(x \setminus (S(s) \cup Entries(s))) = \sigma(x) \setminus (U(s) \cup Entries(\sigma(x), s))$. By $RBAC_0$'s $next$ relation, $next(\sigma(x), delU(s)) = \sigma(x) \setminus (U(s) \cup Entries(s))$. Thus, if ℓ is an instance of $delS(s)$, $\sigma(x')$ is reachable from $\sigma(x)$ via execution of $delU(s)$. A similar argument holds for instances of $delG(g)$ and $delO(o)$ (with reachability in $RBAC_0$ via $delR(g)$ and $delP(o)$, respectively).

- If ℓ is an instance of $liberalJoin(s,g)$, then $x' = x \cup LiberalJoin(s,g,t) \cup Time(t+1) \setminus Time(t)$. By σ, this maps in $RBAC_0$ to state $\sigma(x') = \sigma(x) \cup UR(s,g)$. By $RBAC_0$'s $next$ relation, $next(\sigma(x), assignUser(s,g)) = \sigma(x) \cup UR(s,g)$. Thus, if ℓ is an instance of $liberalJoin$, $\sigma(x')$ is reachable from $\sigma(x)$ via execution of $assignUser(s,g)$. A similar argument holds for instances of $liberalAdd(o,g)$ with reachability via $assignPermission(g,o)$.

- If ℓ is an instance of $strictLeave(s,g)$, then $x' = x \cup StrictLeave(s,g,t) \cup Time(t+1) \setminus Time(t)$. By σ, this maps in $RBAC_0$ to $\sigma(x') = \sigma(x) \setminus UR(s,g)$. By $RBAC_0$'s $next$ relation, $next(\sigma(x), revokeUser(s,g)) = \sigma(x) \setminus UR(s,g)$. Thus, if ℓ is an instance of $strictLeave(s,g)$, $\sigma(x')$ is reachable from $\sigma(x)$ via execution of $revokeUser(s,g)$. A similar argument holds for instances of $strictRemove(o,g)$ with reachability via $revokePermission(g,o)$.

[1] In the case of $rgSIS$, $Entries(x,s)$ for subject s is $\{LiberalJoin(s,g,t) \mid LiberalJoin(s,g,t) \in x\} \cup \{StrictLeave(s,g,t) \mid StrictLeave(s,g,t) \in x\}$.

Thus, for any $rgSIS$ state u and label ℓ, $\sigma(next(u, \ell))$ is reachable from $\sigma(u)$ via $RBAC_0$ labels. By induction, for any $rgSIS$ states s and s', if s' is reachable from s, then $\sigma(s')$ is reachable from $\sigma(s)$. Thus, we have shown that σ preserves reachability.

Finally, we show that σ is pseudo-injective. We first inspect σ to identify what set of differences may exist between x and y and still allow $\sigma(x) = \sigma(y)$. We then show that, if two $rgSIS$ states x and y are identical modulo this set of differences, then for any $rgSIS$ label, ℓ, $next(x, \ell)$ and $next(y, \ell)$ are also identical modulo this set.

Assume $\sigma(x) = \sigma(y)$. The state mapping, σ, stores S, G, and O directly in U, R, and P, respectively. The set of times, T, and current time, $Time$, are not stored in $RBAC_0$. However, T is immutable. Thus, states x and y must not differ in S, G, and O, and are guaranteed not to differ in T, but may differ in current time $Time$.

Regarding the handling of $LiberalJoin$ and $StrictLeave$ by σ: These relations of $rgSIS$ are considered in combination. Rather than consider all joins and leaves a particular subject s has performed for a particular group g, σ only considers the most recent join or leave event. If in x, s has joined g and has not since left, by σ, $UR(s, g) \in Th(\sigma(x))$. If in x, s has left g and has not since re-joined, or if s never joined g, $UR(s, g) \notin Th(\sigma(x))$. Since past entries in $LiberalJoin$ and $StrictLeave$ are not considered by σ, x and y can have any contents in these relations, so long as they agree on the most recent event for each s and g (i.e., whether each s is currently a member of each g).

For identical reasons regarding the storage of $\langle g, o \rangle \in PA$ by σ, x and y can have any contents in $LiberalAdd$ and $StrictRemove$ as long as they agree, for each o and g, whether object o is currently in group g.

Finally, $StrictJoin$, $LiberalLeave$, $StrictAdd$, and $LiberalRemove$ are empty and immutable over the labels of $rgSIS$. Thus, x and y are guaranteed to be identical in these relations.

Now, we examine each of the differences that may exist between x and y to ensure that, after execution of any command in both, the resulting states will also differ only in these ways.

- **Current time** Consider two $rgSIS$ states x and y that differ in the current time $Time(t)$. Since the current time is always greater than all times in the join, leave, add, and remove logs, the difference in current time between these states will propogate only to a difference in the absolute time of future events in the logs; relative order of future events will be preserved. Thus, for any $tgSIS$ label ℓ, the differences between $next(x, \ell)$ and $next(y, \ell)$ will only be in current time and absolute time of events. Thus, $next(x, \ell)$ and $next(y, \ell)$ will differ only in ways that ensure $\sigma(next(x, \ell)) = \sigma(next(y, \ell))$.
- **Absolute event times** Consider $rgSIS$ states x and y in which some event occured at different absolute times but in the same order relative to other events. Since $rgSIS$ labels can only add events with the most recent timestamp, and do not consider the absolute times of past events, for any $rgSIS$ label ℓ, $next(x, \ell)$ and $next(y, \ell)$ will differ only in this altered timestamp. Thus, $\forall \ell.(\sigma(next(x, \ell)) = \sigma(next(y, \ell)))$.
- **Relative inter-group event times** Consider $rgSIS$ states x and y in which two adjacent events (operating on different groups) swap times. Since x and y are adjacent, this swap does not affect the relative times of events within any particular group, and more importantly does not alter the most recent event for a particular s, g or o, g pair. Thus, for any $tgSIS$ label ℓ, $next(x, \ell)$ and $next(y, \ell)$ also differ only in these events' relative times. Therefore, $\forall \ell.(\sigma(next(x, \ell)) = \sigma(next(y, \ell)))$.
- **Past events** Consider two $rgSIS$ states x and y which have different sets of $LiberalJoin$, $StrictLeave$, $LiberalAdd$, and $StrictRemove$, but agree on the most recent event for each s, g (join or leave) and o, g (add or remove). New events can only be added to the state with the most recent time, and thus the different records between y and x will never impact a future decision—once events become irrelevant, they cannot become relevant again. Thus, for any ℓ, $next(x, \ell)$ and $next(y, \ell)$ will continue to differ only in these previous events, and thus $\sigma(next(x, \ell)) = \sigma(next(y, \ell))$.

We have enumerated the ways in which two distinct $rgSIS$ states x and y can map to the same $RBAC_0$ state (i.e., $x \neq y$, $\sigma(x) = \sigma(y)$). In each case, we show that $\forall \ell.(\sigma(next(x, \ell)) = \sigma(next(y, \ell)))$, that is, that x and y are functionally equivalent with respect to σ. Thus, we have shown that σ is pseudo-injective.

Thus, we have shown that σ preserves π, is pseudo-injective, preserves reachability, and is homomorphic; and that π is weak AC-preserving and homomorphic.

$\therefore$ $\langle \sigma, \pi \rangle$ is a reduction from $rgSIS$ to $RBAC_0$ which shows $rgSIS \leq^{CaH} RBAC_0$. □

4.2 Top g-SIS and $RBAC_1$

Theorem 10 *There exists a reduction $\langle \sigma, \pi \rangle$ from top g-SIS (tgSIS) to $RBAC_1$ where:*

- *σ preserves π, is pseudo-injective, preserves reachability, and is homomorphic*
- *π is weak AC-preserving and homomorphic*

Thus, $tgSIS \leq^{CaH} RBAC_1$ ($RBAC_1$ is at least as expressive as top g-SIS with respect to correctness, weak AC-preservation, and homomorphism).

Proof We present the reduction, $\langle \sigma, \pi \rangle$. First, σ maps the g-SIS state $\langle S, O, G, T, Time, StrictJoin, StrictLeave, StrictAdd, StrictRemove \rangle$ to the $RBAC_1$ state $\langle U, R, P, UR, PA, RH \rangle$. This mapping is described as follows.

```
sigma(M)
   for each (S(s) ∈ M)
      output(U(s))
   for each (G(g) ∈ M)
      output(R(g))
   for each (O(o) ∈ M)
      output(P(o))

   Let Records = sortByTime(StrictJoin ∪ StrictLeave ∪
                            StrictAdd ∪ StrictRemove)
   Let WildRoles = {}
   Let UR = {}
   Let PA = {}
   Let RH = {}
   for each (Record ∈ Records)
      If ∃ s, g, t.(Record = <s, g, t> ∧
                    StrictJoin(s, g, t) ∈ M)
         ProcessJoin(M, s, g, UR, RH, WildRoles)
      else If ∃ s, g, t.(Record = <s, g, t> ∧
                         StrictLeave(s, g, t) ∈ M)
         ProcessLeave(M, s, g, UR, RH)
      else If ∃ o, g, t.(Record = <o, g, t> ∧
                         StrictAdd(o, g, t) ∈ M)
         ProcessAdd(M, o, g, PA, RH)
      else If ∃ o, g, t.(Record = <o, g, t> ∧
                         StrictRemove(o, g, t) ∈ M)
         ProcessRemove(M, o, g, PA, RH)
      endif
   outputSet(UR ∪ PA ∪ RH)
```

```
ProcessJoin(M, s, g, UR, RH, WildRoles)
   NewRole = nFreshConst(1, Consts(M) ∪ WildRoles, Univ)
   WildRoles = WildRoles ∪ {NewRole}
   OldBottom = FindBottom(g, RH)
   output(R(NewRole))
   RH = RH ∪ {<OldBottom, NewRole>}
   UR = UR ∪ {<s, NewRole>}

FindBottom(r, RH)
   If ∃ q.(<r, q> ∈ RH)
      return FindBottom(q, RH)
   else
      return r
   endif

ProcessLeave(M, s, g, UR, RH)
   AssignedRoles = FindUser(s, g, UR, RH, {})
   for each (AssignedRole ∈ AssignedRoles)
      UR = UR \ {<s, AssignedRole>}

FindUser(u, r, PA, RH, AssignedRoles)
   If <u, r> ∈ UR
      AssignedRoles = AssignedRoles ∪ {r}
   endif
   If ∃ q.(<r, q> ∈ RH)
      return FindUser(u, q, PA, RH, AssignedRoles)
   else
      return AssignedRoles
   endif

ProcessAdd(M, o, g, PA, RH)
   Bottom = FindBottom(g, RH)
   PA = PA ∪ {<Bottom, o>}

ProcessRemove(M, o, g, PA, RH)
   AssignedRoles = FindPerm(o, g, PA, RH, {})
   for each (AssignedRole ∈ AssignedRoles)
      PA = PA \ {<AssignedRole, o>}

FindPerm(p, r, PA, RH, AssignedRoles)
   If <r, p> ∈ PA
      AssignedRoles = AssignedRoles ∪ {r}
   endif
   If ∃ q.(<r, q> ∈ RH)
      return FindPerm(p, q, PA, RH, AssignedRoles)
   else
      return AssignedRoles
   endif
```

As the mapping is described in HPL, it is homomorphic.

The query mapping, π, is defined as follows.

$$\begin{aligned}
\pi_{Member(s,g)}(T) &= \exists r.(UR(s,r) \in T \wedge Senior(g,r) \in T) \\
\pi_{Assoc(o,g)}(T) &= \exists r.(PA(r,o) \in T \wedge Senior(g,r) \in T) \\
\pi_{auth(s,o,g)}(T) &= \exists r_1, r_2.(UR(s,r_1) \in T \wedge PA(r_2,o) \in T \wedge \\
&\qquad\qquad (r_1 = r_2 \vee Senior(r_1,r_2) \in T) \wedge \\
&\qquad\qquad Senior(g,r_1) \in T)
\end{aligned}$$

This query mapping clearly contains no string manipulation and is thus homomorphic.

Let x be an arbitrary $tgSIS$ state and $\lambda = (s,o,g)$ an arbitrary $tgSIS$ request, and let $f(s,o,g) = (s,o)$ be a request transform. Assume $\pi_{auth(\lambda)}(Th(\sigma(x))) =$ TRUE. Then, by π, $\exists r_1, r_2.(r_1 \geq r_2 \wedge UR(s,r_1) \in Th(\sigma(x)) \wedge PA(r_2,o) \in Th(\sigma(x)))$. Thus, by $RBAC_1$'s $\vdash$ relation, $\sigma(x) \vdash auth(s,o)$.

Now let x be an arbitrary $tgSIS$ state, $\lambda' = (u,p)$ an arbitrary $RBAC_1$ request, and f the request transform defined above. Assume $\sigma(x) \vdash auth(\lambda')$. Then, $\exists r_1, r_2.(r_1 \geq r_2 \wedge UR(u,r_1) \in Th(\sigma(x)) \wedge PA(r_2,p) \in Th(\sigma(x)))$. Furthermore, since σ only assigns roles to users which correspond to some group, r_1 must exist in the hierarchy below a role corresponding to a group: $Senior(g,r_1) \in Th(\sigma(x))$. Finally, $f(u,p,g) = (u,p)$, and $\pi_{auth(u,p,g)}(Th(\sigma(x))) =$ TRUE. Thus, π is weak AC-preserving with transform $f(s,o,g) = (s,o)$.

We show that σ preserves π (for all $tgSIS$ states x, $Th(x) = \pi(Th(\sigma(x))))$ by contradiction. Assume that there is some $tgSIS$ state x and query q such that the value of q in x is the opposite of the value of $\pi(q)$ in $\sigma(x)$. We show that, for each of the query forms of $tgSIS$, this assumption leads to contradiction.

- **Member** Assume $x \vdash Member(s,g)$ and $\sigma(x) \nvdash \pi(Member(s,g))$. Then, $\exists t_1.(StrictJoin(s,g,t_1) \in Th(x) \wedge \forall t_2.(StrictLeave(s,g,t_2) \in Th(x) \Rightarrow t_1 > t_2))$ (s has joined g and not left). By σ, $\exists r_1.(Senior(g,r_1) \in Th(\sigma(x)) \wedge UR(s,r_1) \in Th(\sigma(x)))$. By π, $\pi_{Member(s,g)}(Th(\sigma(x))) =$ TRUE, which is a contradiction on the assumption that $\sigma(x) \nvdash \pi(Member(s,g))$.

  Assume instead that $x \nvdash Member(s,g)$ and $\sigma(x) \vdash \pi(Member(s,g))$. Then, either $\exists t_1.(StrictLeave(s,g,t_1) \in Th(x) \wedge \forall t_2.(StrictJoin(s,g,t_2) \in Th(x) \Rightarrow t_1 > t_2))$ (s has left g and not returned), or $\forall t_1.(StrictJoin(s,g,t_1) \notin Th(x))$ (s has not joined g). By σ, in either case, $\forall r_1.(Senior(g,r_1) \notin Th(\sigma(x)) \vee UR(s,r_1) \notin Th(\sigma(x)))$. By π, $\pi_{Member(o,g)}(Th(\sigma(x))) =$ FALSE, which is a contradiction on the assumption that $\sigma(x) \vdash \pi(Member(s,g))$.

- **Assoc** Assume $x \vdash Assoc(o,g)$ and $\sigma(x) \nvdash \pi(Assoc(o,g))$. Then, $\exists t_1.(StrictAdd(o,g,t_1) \in Th(x) \wedge \forall t_2.(StrictRemove(o,g,t_2) \in Th(x) \Rightarrow t_1 > t_2))$ (o was added to g and not removed). By σ, $\exists r_1.(Senior(g,r_1) \in Th(\sigma(x)) \wedge PA(r_1,o) \in Th(\sigma(x)))$. By π, $\pi_{Assoc(o,g)}(Th(\sigma(x))) =$ TRUE, which is a contradiction on the assumption that $\sigma(x) \nvdash \pi(Assoc(o,g))$.

  Assume instead that $x \nvdash Assoc(o,g)$ and $\sigma(x) \vdash \pi(Assoc(o,g))$. Then, either $\exists t_1.(StrictRemove(o,g,t_1) \in Th(x) \wedge \forall t_2.(StrictAdd(o,g,t_2) \in Th(x) \Rightarrow t_1 > t_2))$ (s was removed from g and not re-added), or $\forall t_1.(StrictAdd(o,g,t_1) \notin Th(x))$ (o has not added to g). By σ, in either case, $\forall r_1.(Senior(g,r_1) \notin Th(\sigma(x)) \vee PA(r_1,o) \notin Th(\sigma(x)))$. By π, $\pi_{Assoc(o,g)}(Th(\sigma(x))) =$ FALSE, which is a contradiction on the assumption that $\sigma(x) \vdash \pi(Assoc(o,g))$.

- **auth** Assume $x \vdash auth(s,o,g)$ and $\sigma(x) \nvdash \pi(auth(s,o,g))$. Then, $\exists t_1, t_2.(StrictJoin(s,g,t_1) \in Th(x) \wedge StrictAdd(o,g,t_2) \in Th(x) \wedge t_2 > t_1)$ (s joined g, and o was later added to g). Furthermore, $\forall t_3.(StrictLeave(s,g,t_3) \in Th(x) \Rightarrow t_1 > t_3)$ (s did not leave g), and $\forall t_3.(StrictRemove(o,g,t_3) \in Th(x) \Rightarrow t_2 > t_3)$ (o was not removed from g). By σ, $\exists r_1, r_2.(UR(s,r_1) \in Th(\sigma(x)) \wedge PA(r_2,o) \in Th(\sigma(x)) \wedge (r_1 = r_2 \vee Senior(r_1,r_2) \in Th(\sigma(x))))$ (s belongs to a role authorized to o or senior to a role authorized to o). Also by σ, $Senior(g,r_1) \in Th(\sigma(x))$ (s and o are in the hierarchy below

g). Thus, by π, $\pi_{auth(s,o,g)}(Th(\sigma(x))) = \text{TRUE}$, which is a contradiction on the assumption that $\sigma(x) \nvdash \pi(auth(s,o,g))$

Assume instead that $x \nvdash auth(s,o,g)$ and $\sigma(x) \vdash \pi(auth(s,o,g))$. Then, either $\forall t_1, t_2.(StrictJoin(s,g,t_1) \notin Th(x) \vee StrictAdd(o,g,t_2) \notin Th(x) \vee t_1 > t_2)$ (o was not added to g after s joined g) or $\exists t_3.((t_3 > t_1 \wedge StrictLeave(s,g,t_3)) \vee (t_3 > t_2 \wedge StrictRemove(o,g,t_3)))$ (one of s and o has since left/been removed from group g). Thus, by σ, if $\exists r_1, r_2.(UR(s,r_1) \in Th(\sigma(x)) \wedge PA(r_2,o) \in Th(\sigma(x)) \wedge (r_1 = r_2 \vee Senior(r_1,r_2) \in Th(\sigma(x))))$ (s belongs to a role authorized to o or senior to a role authorized to o), then it must be in conjunction with a group other than g: $Senior(g,r_1) \notin Th(\sigma(x))$ (s and o are not in the hierarchy below g). Thus, by π, $\pi_{auth(s,o,g)}(Th(\sigma(x))) = \text{FALSE}$, which is a contradiction on the assumption that $\sigma(x) \vdash \pi(auth(s,o,g))$.

Thus, by contradiction, σ preserves π.

As in Theorem 9, we prove that σ preserves reachability by induction by showing the following: for any $tgSIS$ state x and label ℓ, $\sigma(next(x,\ell))$ is reachable from $\sigma(x)$ via $RBAC_1$ labels.

Given $tgSIS$ state x and label ℓ, $x' = next(x,\ell)$ is the state resulting from executing label ℓ in state x.

- If ℓ is an instance of $addS(s)$, then $x' = next(x,\ell) = x \cup S(s)$. By σ, this maps in $RBAC_1$ to state $\sigma(x') = \sigma(x \cup S(s)) = \sigma(x) \cup U(s)$. By $RBAC_1$'s *next* relation, $next(\sigma(x), addU(s)) = \sigma(x) \cup U(s)$. Thus, if ℓ is an instance of $addS(s)$, $\sigma(x')$ is reachable from $\sigma(x)$ via execution of $addU(s)$. A similar argument holds for instances of $addG(g)$ and $addO(o)$ (with reachability in $RBAC_1$ via $addR(g)$ and $addP(o)$, respectively).
- If ℓ is an instance of $delS(s)$, then $x' = x \setminus (S(s) \cup Entries(x,s))$, where $Entries(x,s)$ denotes the set of all state tuples in x involving s. By σ, this maps in $RBAC_1$ to state $\sigma(x') = \sigma(x \setminus (S(s) \cup Entries(x,s))) = \sigma(x) \setminus (U(s) \cup Entries(\sigma(x),s))$. By $RBAC_1$'s *next* relation, $next(\sigma(x), delU(s)) = \sigma(x) \setminus (U(s) \cup Entries(\sigma(x),s))$. Thus, if ℓ is an instance of $delS(s)$, $\sigma(x')$ is reachable from $\sigma(x)$ via execution of $delU(s)$. A similar argument holds for instances of $delO(o)$, with reachability via $delP(o)$.
- If ℓ is an instance of $delG(g)$, then $x' = x \setminus (G(g) \cup Entries(x,g))$. By σ, $\sigma(x') = \sigma(x) \setminus (R(g) \cup ConnectedEntries(\sigma(x),g))$, where $ConnectedEntries(\sigma(x),g)$ denotes the set of state tuples in $\sigma(x)$ involving either g or any role connected to g in the role hierarchy of $\sigma(x)$ (i.e., $ConnectedEntries(x,r) \triangleq r \cup Entries(x,r) \cup \{ConnectedEntries(x,q) \mid RH(r,q) \in Th(x) \vee RH(q,r) \in Th(x)\}$). By $RBAC_1$'s *next* relation, $terminal(\sigma(x), delR(g) \circ delR(r_1) \circ \cdots \circ delR(r_k)) = \sigma(x) \setminus (R(g) \cup ConnectedEntries(\sigma(x),g))$, where $r_1, \ldots, r_k$ is the (finite) set of roles connected to g in the role hierarchy. Thus, if ℓ is an instance of $delG(g)$, $\sigma(x')$ is reachable from $\sigma(x)$ via execution of $delR(g)$, $delR(r_1)$, $\ldots$, $delR(r_k)$.
- If ℓ is an instance of $strictJoin(s,g)$, then $x' = x \cup StrictJoin(s,g,t) \cup Time(t+1) \setminus Time(t)$. By σ, this maps in $RBAC_1$ to state $\sigma(x') = \sigma(x) \cup R(r_1) \cup UR(s,r_1) \cup RH(r_2,r_1)$, where r_1 is a newly-created wildcard role and r_2 is the bottom role of the role hierarchy chain of which g is the top. By $RBAC_1$'s *next* relation, $terminal(\sigma(x), addR(r_1) \circ assignUser(s,r_1) \circ addHierarchy(r_2,r_1)) = \sigma(x) \cup R(r_1) \cup UR(s,r_1) \cup RH(r_2,r_1)$. Thus, if ℓ is an instance of $strictJoin$, $\sigma(x')$ is reachable from $\sigma(x)$ via execution of $addR(r_1)$, $assignUser(s,r_1)$, and $addHierarchy(r_2,r_1)$.
- If ℓ is an instance of $strictAdd(o,g)$, then $x' = x \cup StrictAdd(o,g,t) \cup Time(t+1) \setminus Time(t)$. By σ, $\sigma(x') = \sigma(x) \cup PA(r,o)$, where r is the bottom role of the role hierarchy chain of which g is the top. By $RBAC_1$'s *next* relation, $next(\sigma(x), assignPermission(r,o)) = \sigma(x) \cup PA(r,o)$. Thus, if ℓ is an instance of $strictAdd(o,g)$, $\sigma(x')$ is reachable from $\sigma(x)$ via execution of $assignPermission(r,o)$ where r is the junior-most role below role g.
- If ℓ is an instance of $strictLeave(s,g)$, then $x' = x \cup StrictLeave(s,g,t) \cup Time(t+1) \setminus Time(t)$. By σ, $\sigma(x') = \sigma(x) \setminus UR(s,r)$, where r is the role of which s is a member, among the roles

in the role hierarchy chain below g (i.e., $UR(s,r) \wedge Senior(g,r)$). By $RBAC_1$'s *next* relation, $next(\sigma(x), revokeUser(s,r)) = \sigma(x) \setminus UR(s,r)$. Thus, if ℓ is an instance of $strictLeave(s,g)$, $\sigma(x')$ is reachable from $\sigma(x)$ via execution of $revokeUser(s,r)$ for r such that $UR(s,r) \wedge Senior(g,r)$. A similar argument holds for instances of $strictRemove(o,g)$ with reachability via $revokePermission(r,o)$ for r such that $PA(r,o) \wedge Senior(g,r)$.

Thus, for any $tgSIS$ state x and label ℓ, $\sigma(next(x,\ell))$ is reachable from $\sigma(x)$ via $RBAC_1$ labels. By induction, for any $tgSIS$ states s and s', if s' is reachable from s, then $\sigma(s')$ is reachable from $\sigma(s)$. Thus, we have shown that σ preserves reachability.

Finally, we show that σ is pseudo-injective. We first inspect σ to identify what set of differences may exist between x and y and still allow $\sigma(x) = \sigma(y)$. We then show that, if two $tgSIS$ states x and y are identical modulo this set of differences, then for any $tgSIS$ label, ℓ, $next(x,\ell)$ and $next(y,\ell)$ are also identical modulo this set.

Assume $\sigma(x) = \sigma(y)$. The state mapping, σ, stores S, G, and O directly in U, R, and P, respectively. The set of times, T, and current time, $Time$, are not stored in $RBAC_1$. However, T is immutable. Thus, states x and y must not differ in S, G, and O, and are guaranteed not to differ in T, but may differ in current time $Time$.

All entries in *StrictJoin*, *StrictLeave*, *StrictAdd*, and *StrictRemove* are considered in t order by σ, and each s and g for joins and leaves, and each o and g for adds and removes, are utilized in building the $RBAC_1$ state. Since these entries are also considered in t order, for any particular group g, these entries must remain in order. However, the absolute times are not stored in the $RBAC_1$ state, and thus x and y can differ in absolute time for any events, and can differ in relative time for events in different groups.

Finally, *LiberalJoin*, *LiberalLeave*, *LiberalAdd*, and *LiberalRemove* are empty and immutable over the labels of $tgSIS$. Thus, x and y are guaranteed to be identical in these relations.

Now, we examine each of the differences that may exist between x and y to ensure that, after execution of any command in both, the resulting states will also differ only in these ways.

- **Current time** Consider two $tgSIS$ states x and y that differ in the current time $Time(t)$. Since the current time is always greater than all times in the join, leave, add, and remove logs, the difference in current time between these states will propogate only to a difference in the absolute time of future events in the logs; relative order of future events will be preserved. Thus, for any $tgSIS$ label ℓ, the differences between $next(x,\ell)$ and $next(y,\ell)$ will only be in current time and absolute time of events. Thus, $next(x,\ell)$ and $next(y,\ell)$ will differ only in ways that ensure $\sigma(next(x,\ell)) = \sigma(next(y,\ell))$.
- **Absolute event times** Consider $tgSIS$ states x and y in which some event occured at different absolute times but in the same order relative to other events. Since $tgSIS$ labels can only add events with the most recent timestamp, and do not consider the absolute times of past events, for any $tgSIS$ label ℓ, $next(x,\ell)$ and $next(y,\ell)$ will differ only in this altered timestamp. Thus, $\forall \ell.(\sigma(next(x,\ell)) = \sigma(next(y,\ell)))$.
- **Relative inter-group event times** Consider $tgSIS$ states x and y in which two adjacent events (operating on different groups) swap times. Since x and y are adjacent, this swap does not affect the relative times of events within any particular group. Since $tgSIS$ is an *isolated group* model (i.e., one group's contents do not affect other groups), and since each group's events are ordered equivalently in x and y, then for any $tgSIS$ label ℓ, $next(x,\ell)$ and $next(y,\ell)$ also differ only in these (inter-group) events' relative times. Thus, $\forall \ell.(\sigma(next(x,\ell)) = \sigma(next(y,\ell)))$.

We have enumerated the ways in which two distinct $rgSIS$ states x and y can map to the same $RBAC_1$ state (i.e., $x \neq y$, $\sigma(x) = \sigma(y)$). In each case, we show that $\forall \ell.(\sigma(next(x,\ell)) = \sigma(next(y,\ell)))$, that is, that x and y are functionally equivalent with respect to σ. Thus, we have shown that σ is pseudo-injective.

Thus, we have shown that σ preserves π, is pseudo-injective, preserves reachability, and is homomorphic; and that π is weak AC-preserving and homomorphic.

$\therefore$ $\langle\sigma, \pi\rangle$ is a reduction from $tgSIS$ to $RBAC_1$ which shows $tgSIS \leq^{CaH} RBAC_1$. □

4.3 Bottom g-SIS and $RBAC_1$

Theorem 11 *There exists a reduction $\langle\sigma, \pi\rangle$ from bottom g-SIS ($bgSIS$) to $RBAC_1$ where:*

- *σ preserves π, is pseudo-injective, preserves reachability, and is homomorphic*
- *π is weak AC-preserving and homomorphic*

Thus, $bgSIS \leq^{CaH} RBAC_1$ ($RBAC_1$ is at least as expressive as bottom g-SIS with respect to correctness, weak AC-preservation, and homomorphism).

Proof We present the reduction, $\langle\sigma, \pi\rangle$. First, σ maps the g-SIS state $\langle S, O, G, T, Time, LiberalJoin, LiberalLeave, LiberalAdd, LiberalRemove\rangle$ to the $RBAC_1$ state $\langle U, R, P, UR, PA, RH\rangle$. This mapping is described as follows.

```
sigma(M)
    Let WildRoles = {}
    Let UR = {}
    Let PA = {}
    Let RH = {}

    for each (S(s) ∈ M)
        output(U(s))
    for each (G(g) ∈ M)
        InitGroup(M, g, RH, WildRoles)
    for each (O(o) ∈ M)
        output(P(o))

    Let Records = sortByTime(LiberalJoin ∪ LiberalLeave ∪
                                LiberalAdd ∪ LiberalRemove)
    for each (Record ∈ Records)
        If ∃ s, g, t.(Record = <s, g, t> ∧
                      LiberalJoin(s, g, t) ∈ M)
            ProcessJoin(M, s, g, UR, RH)
        else If ∃ s, g, t.(Record = <s, g, t> ∧
                           LiberalLeave(s, g, t) ∈ M)
            ProcessLeave(M, s, g, UR, RH, WildRoles)
        else If ∃ o, g, t.(Record = <o, g, t> ∧
                           LiberalAdd(o, g, t) ∈ M)
            ProcessAdd(M, o, g, PA, RH)
        else If ∃ o, g, t.(Record = <o, g, t> ∧
                           LiberalRemove(o, g, t) ∈ M)
            ProcessRemove(M, o, g, UR, PA, RH, WildRoles)
        endif
    outputSet(UR ∪ PA ∪ RH)

InitGroup(M, g, RH, WildRoles)
    output(R(g))
    <Top, Bottom> = nFreshConst(2, Consts(M) ∪ WildRoles,
```

```
                                    Univ)
   WildRoles = WildRoles ∪ {Top, Bottom}
   output(R(Top))
   output(R(Bottom))
   RH = RH ∪ {<Top, g>, <g, Bottom>}

ProcessJoin(M, s, g, RH)
   Top = FindTop(g, RH)
   UR = UR ∪ {<s, Top>}

FindTop(r, RH)
   If ∃ q.(<q, r> ∈ RH)
      return FindTop(q)
   else
      return r
   endif

ProcessLeave(M, s, g, UR, RH, WildRoles)
   OldTop = FindTop(g, RH)
   If <s, OldTop> ∈ UR
      NewTop = nFreshConst(1, Consts(M) ∪ WildRoles,
                              Univ)
      WildRoles = WildRoles ∪ {NewTop}
      output(R(NewTop))
      RH = RH ∪ {<NewTop, OldTop>}
      for each (<x, OldTop> ∈ UR)
         If x ≠ s then
            Movers = Movers ∪ {x}
            UR = UR \ {<x, OldTop>}
         endif
      for each Mover ∈ Movers
         UR = UR ∪ {<Mover, NewTop>}
   endif

ProcessAdd(M, o, g, PA, RH)
   Top = FindTop(g, RH)
   PA = PA ∪ {<Top, o>}

ProcessRemove(M, o, g, UR, PA, RH, WildRoles)
   Orphan = nFreshConst(1, Consts(M) ∪ WildRoles, Univ)
   WildRoles = WildRoles ∪ {Orphan}
   output(R(Orphan))
   FirstPermRole = FindPermOnce(o, g, PA, RH)
   If FirstPermRole ≠ {}
      Accessors = UsersBetweenRoles(FindTop(g, RH),
                                    FirstPermRole, PA,
                                    RH, {})
      for each (User ∈ Accessors)
         UR = UR ∪ {<User, Orphan>}
      PA = PA ∪ {<Orphan, o>}
      RH = RH ∪ {<Orphan, FindBottom(g, RH)>}
```

```
      PermRoles = FindPerm(o, g, PA, RH, {})
      for each (PermRole ∈ PermRoles)
         PA = PA \ {<PermRole, o>}
   endif

FindBottom(r, RH)
   If ∃ q.(<r, q> ∈ RH)
      return FindBottom(q, RH)
   else
      return r
   endif

FindPermOnce(p, r, PA, RH)
   If <r, p> ∈ PA
      return r
   else If ∃ q.(<q, r> ∈ RH)
      return FindPermOnce(p, q, PA, RH)
   else
      return {}
   endif

FindPerm(p, r, PA, RH, AssignedRoles)
   If <r, p> ∈ PA
      AssignedRoles = AssignedRoles ∪ {r}
   endif
   If ∃ q.(<q, r> ∈ RH)
      return FindPerm(p, q, PA, RH, AssignedRoles)
   else
      return AssignedRoles
   endif

UsersBetweenRoles(Top, Bottom, UR, RH, Users)
   for each (<u, Top> ∈ UR)
      Users = Users ∪ {u}
   If Top = Bottom then
      return Users
   else If ∃ r.(<Top, r> ∈ RH)
      return UsersBetweenRoles(r, Bottom, UR, RH, Users)
   else
      return {}
   endif
```

As the mapping is described in HPL, it is homomorphic.

The query mapping, π, is defined as follows.

$$\begin{aligned}
\pi_{Member(s,g)}(T) &= \exists r_1.(UR(s,r_1) \in T \wedge Senior(r_1,g) \in T \wedge \\
&\qquad \forall r_2.(Senior(r_2,r_1) \notin T)) \\
\pi_{Assoc(o,g)}(T) &= \exists r.(PA(r,o) \in T \wedge Senior(r,g) \in T) \\
\pi_{auth(s,o,g)}(T) &= \exists r_1, r_2.(UR(s,r_1) \in T \wedge PA(r_2,o) \in T \wedge \\
&\qquad (r_1 = r_2 \vee Senior(r_1,r_2) \in T) \wedge \\
&\qquad \exists r_3.(Senior(g,r_3) \in T \wedge Senior(r_2,r_3) \in T))
\end{aligned}$$

This query mapping clearly contains no string manipulation and is thus homomorphic.

Let x be an arbitrary $bgSIS$ state and $\lambda = (s, o, g)$ an arbitrary $bgSIS$ request, and let $f(s, o, g) = (s, o)$ be a request transform. Assume $\pi_{auth(\lambda)}(Th(\sigma(x))) = \text{TRUE}$. Then, by π, $\exists r_1, r_2.(r_1 \geq r_2 \wedge UR(s, r_1) \in Th(\sigma(x)) \wedge PA(r_2, o) \in Th(\sigma(x)))$. Thus, by $RBAC_1$'s $\vdash$ relation, $\sigma(x) \vdash auth(s, o)$.

Now let x be an arbitrary $bgSIS$ state, $\lambda' = (u, p)$ an arbitrary $RBAC_1$ request, and f the request transform defined above. Assume $\sigma(x) \vdash auth(\lambda')$. Then, $\exists r_1, r_2.(r_1 \geq r_2 \wedge UR(u, r_1) \in Th(\sigma(x)) \wedge PA(r_2, p) \in Th(\sigma(x)))$. Furthermore, since σ only assigns permissions to roles which correspond to some group, r_2 must exist either in the hierarchy above a role corresponding to a group, or as an orphan node attached to such a role: $\exists r_3.(Senior(g, r_3) \in Th(\sigma(x)) \wedge Senior(r_2, r_3) \in Th(\sigma(x)))$. Finally, $f(u, p, g) = (u, p)$, and $\pi_{auth(u,p,g)}(Th(\sigma(x))) = \text{TRUE}$. Thus, π is weak AC-preserving with transform $f(s, o, g) = (s, o)$.

We show that σ preserves π (for all $bgSIS$ states x, $Th(x) = \pi(Th(\sigma(x))))$ by contradiction. Assume that there is some $bgSIS$ state x and query q such that the value of q in x is the opposite of the value of $\pi(q)$ in $\sigma(x)$. We show that, for each of the query forms of $bgSIS$, this assumption leads to contradiction.

- **Member** Assume $x \vdash Member(s, g)$ and $\sigma(x) \nvdash \pi(Member(s, g))$. Then, $\exists t_1.(LiberalJoin(s, g, t_1) \in Th(x) \wedge \forall t_2.(LiberalLeave(s, g, t_2) \in Th(x) \Rightarrow t_1 > t_2))$ (s has joined g and not left). By σ, $\exists r_1.(Senior(r_1, g) \in Th(\sigma(x)) \wedge UR(s, r_1) \in Th(\sigma(x)))$. By π, $\pi_{Member(s,g)}(Th(\sigma(x))) = \text{TRUE}$, which is a contradiction on the assumption that $\sigma(x) \nvdash \pi(Member(s, g))$.

  Assume instead that $x \nvdash Member(s, g)$ and $\sigma(x) \vdash \pi(Member(s, g))$. Then, either $\exists t_1.(LiberalLeave(s, g, t_1) \in Th(x) \wedge \forall t_2.(LiberalJoin(s, g, t_2) \in Th(x) \Rightarrow t_1 > t_2))$ (s has left g and not returned), or $\forall t_1.(LiberalJoin(s, g, t_1) \notin Th(x))$ (s has not joined g). By σ, in either case, $\forall r_1.(Senior(r_1, g) \notin Th(\sigma(x)) \vee UR(s, r) \notin Th(\sigma(x)))$. By π, $\pi_{Member(o,g)}(Th(\sigma(x))) = \text{FALSE}$, which is a contradiction on the assumption that $\sigma(x) \vdash \pi(Member(s, g))$.

- **Assoc** Assume $x \vdash Assoc(o, g)$ and $\sigma(x) \nvdash \pi(Assoc(o, g))$. Then, $\exists t_1.(LiberalAdd(o, g, t_1) \in Th(x) \wedge \forall t_2.(LiberalRemove(o, g, t_2) \in Th(x) \Rightarrow t_1 > t_2))$ (o was added to g and not removed). By σ, $\exists r_1.(Senior(r_1, g) \in Th(\sigma(x)) \wedge PA(r_1, o) \in Th(\sigma(x)))$. By π, $\pi_{Assoc(o,g)}(Th(\sigma(x))) = \text{TRUE}$, which is a contradiction on the assumption that $\sigma(x) \nvdash \pi(Assoc(o, g))$.

  Assume instead that $x \nvdash Assoc(o, g)$ and $\sigma(x) \vdash \pi(Assoc(o, g))$. Then, either $\exists t_1.(LiberalRemove(o, g, t_1) \in Th(x) \wedge \forall t_2.(LiberalAdd(o, g, t_2) \in Th(x) \Rightarrow t_1 > t_2))$ (s was removed from g and not re-added), or $\forall t_1.(LiberalAdd(o, g, t_1) \notin Th(x))$ (o has not added to g). By σ, in either case, $\forall r_1.(Senior(r_1, g) \notin Th(\sigma(x)) \vee PA(r_1, o) \notin Th(\sigma(x)))$. By π, $\pi_{Assoc(o,g)}(Th(\sigma(x))) = \text{FALSE}$, which is a contradiction on the assumption that $\sigma(x) \vdash \pi(Assoc(o, g))$.

- **auth** Assume $x \vdash auth(s, o, g)$ and $\sigma(x) \nvdash \pi(auth(s, o, g))$. Then, $\exists t_1, t_2.(LiberalJoin(s, g, t_1) \in Th(x) \wedge LiberalAdd(o, g, t_2) \in Th(x))$ (s has joined g and o has been added to g). If $t_2 > t_1$ (the join occurred first), then $\forall t_3.(LiberalLeave(s, g, t_3) \in Th(x) \Rightarrow t_1 > t_3 \vee t_3 > t_2)$ (s did not leave g between joining and o being added). If $t_1 > t_2$ (the add occurred first), then $\forall t_3.(LiberalRemove(o, g, t_3) \in Th(x) \Rightarrow t_2 > t_3 \vee t_3 > t_1)$ (o was not removed from g between being added and s joining). By σ, $\exists r_1, r_2.(UR(s, r_1) \in Th(\sigma(x)) \wedge PA(r_2, o) \in Th(\sigma(x)) \wedge (r_1 = r_2 \vee Senior(r_1, r_2) \in Th(\sigma(x))))$ (s belongs to a role authorized to o or senior to a role authorized to o). Connection to g is preserved by σ, so $\exists r_3.(Senior(g, r_3) \in Th(\sigma(x)) \wedge Senior(r_2, r_3) \in Th(\sigma(x)))$, either because s and o are in the hierarchy above g or because s and o are in an "orphaned node" due to o's removal from g. Thus, by π, $\pi_{auth(s,o,g)}(Th(\sigma(x))) = \text{TRUE}$, which is a contradiction on the assumption that $\sigma(x) \nvdash \pi(auth(s, o, g))$.

  Assume instead that $x \nvdash auth(s, o, g)$ and $\sigma(x) \vdash \pi(auth(s, o, g))$. Then, either $\forall t_1, t_2.(LiberalJoin(s, g, t_1) \notin Th(x) \vee LiberalAdd(o, g, t_2) \notin Th(x))$ (s has not joined g or o has not been added to g), or s and o's times in g did not overlap. If $t_2 > t_1$ (the join occurred first), then $\exists t_3.(LiberalLeave(s, g, t_3) \in Th(x) \wedge t_2 > t_3 \wedge t_3 > t_1)$ (s left g before o was added). If $t_1 > t_2$ (the add occurred first), then $\exists t_3.(LiberalRemove(o, g, t_3) \in Th(x) \wedge t_1 > t_3 \wedge t_3 > t_2)$ (o was removed from g before s joined). Thus, by σ, if $\exists r_1, r_2.(UR(s, r_1) \in Th(\sigma(x)) \wedge PA(r_2, o) \in Th(\sigma(x)) \wedge (r_1 = r_2 \vee Senior(r_1, r_2) \in Th(\sigma(x))))$ (s belongs to a role authorized to o or senior to a role authorized to o), then it must be in conjunction with a group other than g: $\forall r_3.(Senior(g, r_3) \notin Th(\sigma(x)) \vee Senior(r_2, r_3) \notin Th(\sigma(x)))$. Thus, by π, $\pi_{auth(s,o,g)}(Th(\sigma(x))) =$ FALSE, which is a contradiction on the assumption that $\sigma(x) \vdash \pi(auth(s, o, g)$.

Thus, by contradiction, σ preserves π.

As before (Theorems 9 and 10), we prove that σ preserves reachability by induction by showing that for any $bgSIS$ state x and label ℓ, $\sigma(next(x, \ell))$ is reachable from $\sigma(x)$ via $RBAC_1$ labels.

Given $bgSIS$ state x and label ℓ, $x' = next(x, \ell)$ is the state resulting from executing label ℓ in state x.

- If ℓ is an instance of $addS(s)$, then $x' = next(x, \ell) = x \cup S(s)$. By σ, this maps in $RBAC_1$ to state $\sigma(x') = \sigma(x \cup S(s)) = \sigma(x) \cup U(s)$. By $RBAC_1$'s *next* relation, $next(\sigma(x), addU(s)) = \sigma(x) \cup U(s)$. Thus, if ℓ is an instance of $addS(s)$, $\sigma(x')$ is reachable from $\sigma(x)$ via execution of $addU(s)$. A similar argument holds for instances of $addO(o)$, with reachability in $RBAC_1$ via $addP(o)$.
- If ℓ is an instance of $delS(s)$, then $x' = x \setminus (S(s) \cup Entries(x, s))$, where $Entries(x, s)$ denotes the set of all state tuples in x involving s. By σ, $\sigma(x') = \sigma(x \setminus (S(s) \cup Entries(x, s))) = \sigma(x) \setminus (U(s) \cup Entries(\sigma(x), s))$. By $RBAC_1$'s *next* relation, $next(\sigma(x), delU(s)) = \sigma(x) \setminus (U(s) \cup Entries(\sigma(x), s))$. Thus, if ℓ is an instance of $delS(s)$, $\sigma(x')$ is reachable from $\sigma(x)$ via execution of $delU(s)$. A similar argument holds for instances of $delO(o)$, with reachability via $delP(o)$.
- If ℓ is an instance of $addG(g)$, then $x' = x \cup G(g)$. By σ, $\sigma(x') = \sigma(x \cup G(g)) = \sigma(x) \cup R(g) \cup R(r_{top}) \cup R(r_{bottom}) \cup RH(r_{top}, g) \cup RH(g, r_{bottom})$, where r_{top} and r_{bottom} are newly-created roles. By $RBAC_1$'s *next* relation, $terminal(\sigma(x), addR(g) \circ addR(r_{top}) \circ addR(r_{bottom}) \circ addHierarchy(r_{top}, g) \circ addHierarchy(g, r_{bottom})) = \sigma(x) \cup R(g) \cup R(r_{top}) \cup R(r_{bottom}) \cup RH(r_{top}, g) \cup RH(g, r_{bottom})$. Thus, if ℓ is an instance of $addG(g)$, $\sigma(x')$ is reachable from $\sigma(x)$ via execution of $addR(g)$, $addR(r_{top})$, $addR(r_{bottom})$, $addHierarchy(r_{top}, g)$, and $addHierarchy(g, r_{bottom})$.
- If ℓ is an instance of $delG(g)$, then $x' = x \setminus (G(g) \cup Entries(x, g))$. By σ, $\sigma(x') = \sigma(x) \setminus (R(g) \cup ConnectedEntries(\sigma(x), g))$. By $RBAC_1$'s *next* relation, $terminal(\sigma(x), delR(g) \circ delR(r_1) \circ \cdots \circ delR(r_k)) = \sigma(x) \setminus (R(g) \cup ConnectedEntries(\sigma(x), g))$, where $r_1, \ldots, r_k$ is the (finite) set of roles connected to g in the role hierarchy. Thus, if ℓ is an instance of $delG(g)$, $\sigma(x')$ is reachable from $\sigma(x)$ via execution of $delR(g)$, $delR(r_1)$, ..., $delR(r_k)$.
- If ℓ is an instance of $liberalJoin(s, g)$, then $x' = x \cup LiberalJoin(s, g, t) \cup Time(t+1) \setminus Time(t)$. By σ, $\sigma(x') = \sigma(x) \cup UR(s, r)$, where r is the top role of the role hierarchy chain of which g is the second-bottom. By $RBAC_1$'s *next* relation, $next(\sigma(x), assignUser(s, r)) = \sigma(x) \cup UR(s, r)$. Thus, if ℓ is an instance of $liberalJoin$, $\sigma(x')$ is reachable from $\sigma(x)$ via execution of $assignUser(s, r)$.
- If ℓ is an instance of $liberalAdd(o, g)$, then $x' = x \cup LiberalAdd(o, g, t) \cup Time(t+1) \setminus Time(t)$. By σ, $\sigma(x') = \sigma(x) \cup PA(r, o)$, where r is the top role of the role hierarchy chain of which g is the second-bottom. By $RBAC_1$'s *next* relation, $next(\sigma(x), assignPermission(r, o)) = \sigma(x) \cup PA(r, o)$. Thus, if ℓ is an instance of $liberalAdd(o, g)$, $\sigma(x')$ is reachable from $\sigma(x)$ via execution of $assignPermission(r, o)$.
- If ℓ is an instance of $liberalLeave(s, g)$, then $x' = x \cup liberalLeave(s, g, t) \cup Time(t+1) \setminus Time(t)$. By σ, $\sigma(x') = \sigma(x) \cup R(r_2) \cup RH(r_2, r_1) \cup \{UR(u, r_2) \mid UR(u, r_1) \in Th(\sigma(x)) \wedge u \neq s\} \setminus \{UR(u, r_1) \mid UR(u, r_1) \in Th(\sigma(x)) \wedge u \neq s\}$, where r_1 is the current top of the role hierarchy chain of which g is the second-bottom, and r_2 is the newly-created wildcard role and new top of the hierarchy chain. By $RBAC_1$'s *next* relation, $terminal(\sigma(x), addR(r_2) \circ addHierarchy(r_2, r_1) \circ assignUser(u_1, r_2) \circ$

$\cdots \circ assignUser(u_k, r_2) \circ revokeUser(u_1, r_1) \circ \cdots \circ revokeUser(u_k, r_1))$ is $\sigma(x)$ as above. Thus, if ℓ is an instance of $liberalLeave(s, g)$, $\sigma(x')$ is reachable from $\sigma(x)$ via execution of $addR(r_2)$, $addHierarchy(r_2, r_1)$, and the following for each user u where $UR(u, r_1) \in Th(\sigma(x)) \wedge u \neq s$: $assignUser(u, r_2)$ and $revokeUser(u, r_1)$.

- If ℓ is an instance of $liberalRemove(s, g)$, then $x' = x \cup liberalRemove(o, g, t) \cup Time(t+1) \setminus Time(t)$. By σ, $\sigma(x') = \sigma(x) \cup R(r_2) \cup RH(r_2, r_1) \cup PA(r_2, o) \cup \{UR(u, r_2) \mid \exists r_4.(UR(u, r_4) \in Th(\sigma(x)) \wedge Senior(r_4, r_3) \in Th(\sigma(x)))\} \setminus \{PA(r_5, o) \mid PA(r_5, o) \in Th(\sigma(x)) \wedge Senior(r_5, g) \in Th(\sigma(x))\}$, where r_1 is the current bottom of the role hierarchy chain of which g is the second-bottom, r_2 is the wildcard role newly created above r_1, and r_3 is the lowest role in the hierarchy chain above g such that r_3 is authorized to o. By $RBAC_1$'s $next$ relation, $terminal(\sigma(x), addR(r_2) \circ addHierarchy(r_2, r_1) \circ assignPermission(r_2, o) \circ assignUser(u_1, r_2) \circ \cdots \circ assignUser(u_k, r_2) \circ revokePermission(q_1, o) \circ \cdots \circ revokePermission(q_k, o))$ is $\sigma(x)$ as above. Thus, if ℓ is an instance of $liberalRemove(s, g)$, $\sigma(x')$ is reachable from $\sigma(x)$ via execution of $addR(r_2)$, $addHierarchy(r_2, r_1)$, $assignPermission(r_2, o)$, $assignUser(u, r_2)$ for each user u with access to o (those assigned to roles above r_3 in the hierarchy chain), and $revokePermission(q, o)$ for each role q such that q is authorized to o and q is above g in the hierarchy chain.

Thus, for any $bgSIS$ state x and label ℓ, $\sigma(next(x, \ell))$ is reachable from $\sigma(x)$ via $RBAC_1$ labels. By induction, for any $bgSIS$ states s and s', if s' is reachable from s, then $\sigma(s')$ is reachable from $\sigma(s)$. Thus, we have shown that σ preserves reachability.

Finally, σ is shown to be pseudo-injective using the same arguments as used in Theorem 10. If $x \neq y$ but $\sigma(x) = \sigma(y)$, then x and y can differ in the same ways as in that proof, and for any $bgSIS$ label ℓ, $next(x, \ell)$ and $next(y, \ell)$ will differ only in ways from this list. Thus, $\forall \ell.(\sigma(next(x, \ell)) = \sigma(next(y, \ell)))$, and so σ is pseudo-injective.

Thus, we have shown that σ preserves π, is pseudo-injective, preserves reachability, and is homomorphic; and that π is weak AC-preserving and homomorphic.

$\therefore \langle \sigma, \pi \rangle$ is a reduction from $bgSIS$ to $RBAC_1$ which shows $tgSIS \leq^{CaH} RBAC_1$. □

4.4 Helper Reductions

Proposition 12 *There exists a reduction $\langle \sigma, \pi \rangle$ from $RBAC_0$ to $RBAC_1$ where:*

- *σ preserves π, is injective, preserves reachability, and is homomorphic*
- *π is AC-preserving and homomorphic*

Furthermore, both $RBAC_0$ and $RBAC_1$ have homomorphic transition functions and entailment relations.

Thus, $RBAC_0 \leq^{CAH} RBAC_1$ ($RBAC_1$ is at least as expressive as $RBAC_0$ with respect to correctness, AC-preservation, and homomorphism).

PROOF We present the reduction, $\langle \sigma, \pi \rangle$. First, σ maps the $RBAC_0$ state $\langle U, R, P, UR, PA \rangle$ to the $RBAC_1$ state $\langle U, R, P, UR, PA, RH \rangle$. This mapping is described as follows.

$$\begin{aligned} U_1(u) &\triangleq U_0(u) \\ R_1(r) &\triangleq R_0(r) \\ P_1(p) &\triangleq P_0(p) \\ UR_1(u, r) &\triangleq UR_0(u, r) \\ PA_1(r, p) &\triangleq PA_0(r, p) \\ RH_1(r_1, r_2) &\triangleq \{\} \end{aligned}$$

The query mapping is defined as follows.

$$\pi_{UR(u,r)}(T) = UR(u,r) \in T$$
$$\pi_{PA(r,p)}(T) = PA(r,p) \in T$$
$$\pi_{R(r)}(T) = R(r) \in T$$
$$\pi_{auth(u,p)}(T) = auth(u,p) \in T$$

These degenerate mappings are trivially homomorphic: the state mapping stores all elements of the $RBAC_0$ state unchanged in the $RBAC_1$ state, and the query mapping asks all $RBAC_0$ queries unchanged in $RBAC_1$. σ is injective, since no two $RBAC_0$ states will map to the same $RBAC_1$ state. π is also AC-preserving, since it maps authorization query $auth(r)$ to TRUE for theory T exactly when T contains $auth(r)$.

For all $RBAC_0$ states x, $Th(x) = \pi(Th(\sigma(x)))$, since:

- All UR, PA, and R queries (and their entailment relations) are identical between $RBAC_0$ and $RBAC_1$, and the relevant state elements are mapped identically between them by σ.
- The definitions for *auth* in $RBAC_0$ and $RBAC_1$ are identical, given that σ sets RH in $RBAC_1$ to be empty and guarantees the *Senior* condition in $RBAC_1$'s *auth* can not be satisfied.

Thus, σ preserves π.

Since $RBAC_0$ contains a subset of the labels included in $RBAC_1$, it is easy to show that σ preserves reachability. For all $RBAC_0$ states x, x', if x' is reachable from x, then there exists a sequence of labels $\langle \ell_1, \ldots, \ell_n \rangle$ such that $terminal(x, \ell_1 \circ \cdots \circ \ell_n) = x'$. Since $RBAC_1$ has a superset of the labels of $RBAC_0$ (and the equivalent *next* relation), this sequence of labels also exists in $RBAC_1$ and guarantees that $terminal(\sigma(x), \ell_1 \circ \ldots \circ \ell_n) = \sigma(x')$. Thus, σ preserves reachability.

Thus, we have shown that σ preserves π, is injective, preserves reachability, and is homomorphic; and that π is AC-preserving and homomorphic.

$\therefore \langle \sigma, \pi \rangle$ is a reduction from $RBAC_0$ to $RBAC_1$ which shows $RBAC_0 \leq^{CAH} RBAC_1$. □

Theorem 13 *There exists a reduction $\langle \sigma, \pi \rangle$ from $RBAC_1$ to $RBAC_0$ where:*

- *σ preserves π, is injective, and preserves reachability*
- *π is AC-preserving*

Thus, $RBAC_1 \leq^{CA} RBAC_0$ ($RBAC_0$ is at least as expressive as $RBAC_1$ with respect to correctness and AC-preservation).

PROOF We present the reduction, $\langle \sigma, \pi \rangle$. First, σ maps the $RBAC_1$ state $\langle U, R, P, UR, PA, RH \rangle$ to the $RBAC_0$ state $\langle U, R, P, UR, PA \rangle$. This mapping is described as follows, where concat() is a non-homomorphic string concatenation procedure; contains(), StartsWith(), and EndsWith() are self-explanatory string testing procedures; and Sentinel() is a non-homomorphic procedure which returns a sentinel string, a string which is not contained in any real role name[2].

```
sigma(M)
    Let U = {u | U(u) ∈ M}
    Let R = {r | R(r) ∈ M}
    Let P = {p | P(p) ∈ M}

    for each (<Senior, Junior> ∈ {<r, q> | RH(r, q) ∈ M})
```

[2]Note that [1] shows that, if a role is added later which invalidates Sentinel(), we can then select a new string and adjust all existing instances of Sentinel() before adding the new role.

```
    ProcessLink(Senior, Junior, R)

  Let UR = BuildUr(M, R)
  Let PA = BuildPa(M, R)

  outputSet(U ∪ R ∪ P ∪ UR ∪ PA)

ProcessLink(Senior, Junior, R)
  Let Prefixes = {Senior}
  Let Suffixes = {Junior}
  for each (Role ∈ R)
    If StartsWith(Role, concat(Junior, Sentinel()))
      Suffixes = Suffixes ∪ {Role}
    else If EndsWith(Role, concat(Sentinel(), Senior))
      Prefixes = Prefixes ∪ {Role}
    endif
  for each (Prefix in Prefixes)
    for each (Suffix in Suffixes)
      R = R ∪ {concat(Prefix, Sentinel(), Suffix)}

BuildUr(M, R)
  Let UR = {}
  for each (<u, r> in {<u, r> | UR(u, r) ∈ M})
    UR = UR ∪ {<u, r>}
    for each (q ∈ R)
      If StartsWith(q, concat(r, Sentinel()))
        UR = UR ∪ {<u, q>}
      endif

BuildPa(M, R)
  Let PA = {}
  for each (<r, p> in {<r, p> | PA(r, p) ∈ M})
    PA = PA ∪ {<r, p>}
    for each (q ∈ R)
      If EndsWith(q, concat(Sentinel(), r))
        PA = PA ∪ {<q, p>}
      endif
```

Since this mapping contains string manipulating functions like *concat*, it is not homomorphic. It is injective, since no two $RBAC_1$ states will map to the same $RBAC_0$ state: all $RBAC_1$ state is represented in the $RBAC_0$ state.

The query mapping, π, is defined as follows.

$$
\begin{aligned}
\pi_{UR(u,r)}(T) &= UR(u,r) \in T \wedge \neg\text{contains}(r, \text{Sentinel}()) \\
\pi_{PA(r,p)}(T) &= PA(r,p) \in T \wedge \neg\text{contains}(r, \text{Sentinel}()) \\
\pi_{R(r)}(T) &= R(r) \in T \wedge \neg\text{contains}(r, \text{Sentinel}()) \\
\pi_{RH(r_1,r_2)}(T) &= R(\text{concat}(r_1, \text{Sentinel}(), r_2)) \in T \\
\pi_{Senior(r_1,r_2)}(T) &= \exists r.(R(r) \in T \wedge \text{StartsWith}(r, \text{concat}(r_1, \text{Sentinel}()))\wedge \\
&\qquad \text{EndsWith}(r, \text{concat}(\text{Sentinel}(), r_2))) \\
\pi_{auth(u,p)}(T) &= auth(u,p) \in T
\end{aligned}
$$

This query mapping is obviously AC-preserving since it maps authorization requests $auth(r)$ to TRUE for a theory T exactly when T contains $auth(r)$. However, due to use of non-homomorphic routines such as contains(), it is not homomorphic.

We show that σ preserves π (for any $RBAC_1$ state x, $Th(x) = \pi(Th(\sigma(x))))$ by contradiction. Assume that there is some $RBAC_1$ state x and query q such that the value of q in x is the opposite of the value of $\pi(q)$ in $\sigma(x)$. We show that, for each of the query forms of $RBAC_1$, this assumption leads to contradiction.

- **UR** Assume $x \vdash UR(u, r)$ and $\sigma(x) \nvdash \pi(UR(u, r))$. Then, $UR(u, r) \in Th(x)$, and by σ, $UR(u, r) \in Th(\sigma(x))$. By π, $\pi_{UR(u,r)}(Th(\sigma(x))) =$ TRUE, since if r is a valid role it does not contain Sentinel(), and thus we have a contradiction that $\sigma(x) \nvdash \pi(UR(u, r))$.

  Assume instead that $x \nvdash UR(u, r)$ and $\sigma(x) \vdash \pi(UR(u, r))$. Then, $UR(u, r) \notin Th(x)$, and by σ, either $UR(u, r) \notin Th(\sigma(x))$, or r is a role that encodes a hierarchy chain, in which case it must contain Sentinel(). In either case, by π, $\pi_{UR(u,r)}(Th(\sigma(x))) =$ FALSE, which is a contradiction on the assumption that $\sigma(x) \vdash \pi(UR(u, r))$.

- **PA** Assume $x \vdash PA(r, p)$ and $\sigma(x) \nvdash \pi(PA(r, p))$. Then, $PA(r, p) \in Th(x)$, and by σ, $PA(r, p) \in Th(\sigma(x))$. By π, $\pi_{PA(r,p)}(Th(\sigma(x))) =$ TRUE, since if r is a valid role then it does not contain Sentinel(), and thus we have a contradiction that $\sigma(x) \nvdash \pi(PA(r, p))$.

  Assume instead that $x \nvdash PA(r, p)$ and $\sigma(x) \vdash \pi(PA(r, p))$. Then, $PA(r, p) \notin Th(x)$, and by σ, either $PA(r, p) \notin Th(\sigma(x))$, or r is a chain-encoding role, and thus contains Sentinel(). In either case, by π, $\pi_{PA(r,p)}(Th(\sigma(x))) =$ FALSE, which is a contradiction on the assumption that $\sigma(x) \vdash \pi(PA(r, p))$.

- **R** Assume $x \vdash R(r)$ and $\sigma(x) \nvdash \pi(R(r))$. Then, $R(r) \in Th(x)$, and by σ, $R(r) \in Th(\sigma(x))$. By π, $\pi_{R(r)}(Th(\sigma(x))) =$ TRUE, since if r is a valid role then it does not contain Sentinel(), and thus we have a contradiction that $\sigma(x) \nvdash \pi(R(r))$.

  Assume instead that $x \nvdash R(r)$ and $\sigma(x) \vdash \pi(R(r))$. Then, $R(r) \notin Th(x)$, and by σ, either $R(r) \notin Th(\sigma(x))$, or r is a chain-encoding role, and thus contains Sentinel(). In either case, by π, $\pi_{R(r)}(Th(\sigma(x))) =$ FALSE, which is a contradiction on the assumption that $\sigma(x) \vdash \pi(R(r))$.

- **RH** Assume $x \vdash RH(s, j)$ and $\sigma(x) \nvdash \pi(RH(s, j))$. Then, $RH(s, j) \in Th(x)$, and by σ, $R(r) \in Th(\sigma(x))$, where $r = \text{concat}(s, \text{Sentinel}(), j)$. By π, $\pi_{RH(s,j)}(Th(\sigma(x))) =$ TRUE, which is a contradiction on the assumption that $\sigma(x) \nvdash \pi(RH(s, j))$.

  Assume instead that $x \nvdash RH(s, j)$ and $\sigma(x) \vdash \pi(RH(s, j))$. Then, $RH(s, j) \notin Th(x)$, and by σ, $R(r) \notin Th(\sigma(x))$, where $r = \text{concat}(s, \text{Sentinel}(), j)$. By π, $\pi_{RH(s,j)}(Th(\sigma(x))) =$ FALSE, which is a contradiction on the assumption that $\sigma(x) \vdash \pi(RH(s, j))$.

- **Senior** Assume $x \vdash Senior(s, j)$ and $\sigma(x) \nvdash \pi(Senior(s, j))$. Then, there is some sequence of roles r_i such that $RH(s, r_1) \in Th(x) \wedge RH(r_1, r_2) \in Th(x) \wedge \cdots \wedge RH(r_k, j) \in Th(x)$. By σ, $R(r) \in Th(\sigma(x))$, where $r = \text{concat}(s, \text{Sentinel}(), r_1, \text{Sentinel}(), \ldots, \text{Sentinel}(), r_k, \text{Sentinel}(), j)$. By π, $\pi_{Senior(s,j)}(Th(\sigma(x))) =$ TRUE, which is a contradiction on the assumption that $\sigma(x) \nvdash \pi(Senior(s, j))$.

  Assume instead that $x \nvdash Senior(s, j)$ and $\sigma(x) \vdash \pi(Senior(s, j))$. Then, there is no sequence of roles r_i such that $RH(s, r_1) \in Th(x) \wedge RH(r_1, r_2) \in Th(x) \wedge \cdots \wedge RH(r_k, j) \in Th(x)$. By σ, there is thus no role r such that $R(r) \in Th(\sigma(x))$ and r begins with $\text{concat}(s, \text{Sentinel})$ and ends with $\text{concat}(\text{Sentinel}, s)$. By π, $\pi_{Senior(s,j)}(Th(\sigma(x))) =$ FALSE, which is a contradiction on the assumption that $\sigma(x) \vdash \pi(Senior(s, j))$.

- **auth** Assume $x \vdash auth(u, p)$ and $\sigma(x) \nvdash \pi(auth(u, p))$. Then, either $\exists r.(UR(u, r) \in Th(x) \wedge PA(r, p) \in Th(x))$ (u belongs to a role which is authorized to p) or $\exists s, j.(UR(u, s) \in Th(x) \wedge PA(j, p) \in Th(x) \wedge Senior(s, j) \in Th(x))$ (u belongs to a role senior to a role that is authorized to p). In the former case, by σ, then trivially $\exists r.(UR(u, r) \in Th(\sigma(x)) \wedge PA(r, p) \in Th(\sigma(x)))$.

For the latter case, note that σ ensures users authorized to role s are also authorized to roles that begin with concat(s, Sentinel()). In addition, if role j is authorized to p, then so are roles ending in concat(Sentinel(), j). Thus, in the latter case above, it is also true that $\exists r.(UR(u,r) \in Th(\sigma(x)) \wedge PA(r,p) \in Th(\sigma(x)))$. Thus, by π, $\pi_{auth(u,p)}(Th(\sigma(x))) =$ TRUE, which is a contradiction on the assumption that $\sigma(x) \nvdash \pi(auth(u,p))$.

Assume instead that $x \nvdash auth(u,p)$ and $\sigma(x) \vdash \pi(auth(u,p))$. Then, there is no sequence of roles r_i such that $RH(r_1,r_2) \in Th(x) \wedge RH(r_2,r_3) \in Th(x) \wedge \cdots \wedge RH(r_{k-1},r_k) \in Th(x)$ and $UR(u,r_1) \in Th(x) \wedge PA(r_k,p) \in Th(x)$. Thus, by σ, there is no r such that $UR(u,r) \in Th(\sigma(x)) \wedge PA(r,p) \in Th(\sigma(x))$. By π, $\pi_{auth(u,p)}(Th(\sigma(x))) =$ FALSE, which is a contradiction on the assumption that $\sigma(x) \vdash \pi(auth(u,p))$.

Thus, by contradiction, σ preserves π.

We prove that σ preserves reachability by induction by showing that, for any $RBAC_1$ state x and label ℓ, $\sigma(next(x,\ell))$ is reachable from $\sigma(x)$ via $RBAC_0$ labels.

Given $RBAC_1$ state x and label ℓ, let $x' = next(x,\ell)$ by the state resulting from executing label ℓ in state x.

- If ℓ is an instance of $addU(u)$, then $x' = next(x,\ell) = x \cup U(u)$. By σ, this maps in $RBAC_0$ to state $\sigma(x') = \sigma(x) \cup U(u)$. By $RBAC_0$'s *next* relation, $next(\sigma(x), addU(u)) = \sigma(x) \cup U(u)$. Thus, if ℓ is an instance of $addU(u)$, $\sigma(x')$ is reachable from $\sigma(x)$ via execution of $addU(u)$. A similar argument holds for instances of $addR(r)$ and $addP(p)$, with reachability in $RBAC_0$ via $addR(r)$ and $addP(p)$, respectively.
- If ℓ is an instance of $delU(u)$, then $x' = x \setminus (U(u) \cup Entries(x,u))$, where $Entries(x,u)$ denotes the set of all state tuples in x involving u. By σ, $\sigma(x') = \sigma(x \setminus (U(u) \cup Entries(x,u))) = \sigma(x) \setminus (U(u) \cup Entries(\sigma(x),u))$. By $RBAC_0$'s *next* relation, $next(\sigma(x), delU(u)) = \sigma(x) \setminus (U(u) \cup Entries(\sigma(x),u))$. Thus, if ℓ is an instance of $delU(u)$, $\sigma(x')$ is reachable from $\sigma(x)$ via execution of $delU(u)$. A similar argument holds for instances of $delP(p)$, with reachability via $delP(p)$.
- If ℓ is an instance of $delR(r)$, then $x' = x \setminus (R(r) \cup Entries(x,r))$. By σ, $\sigma(x') = \sigma(x) \setminus (R(r) \cup NHEntries(\sigma(x),r))$, where $NHEntries(\sigma(x),r)$ denotes the set of state tuples in $\sigma(x)$ involving either r or any role non-homomorphically encoding r: roles that start with concat(r, Sentinel()), end with concat(Sentinel(), r), or contain concat(Sentinel(), r, Sentinel()). By $RBAC_0$'s *next* relation, $terminal(\sigma(x), delR(r) \circ delR(r_1) \circ \cdots \circ delR(r_k)) = \sigma(x) \setminus (R(r) \cup NHEntries(\sigma(x),r))$, where $r_1,\ldots,r_k$ is the (finite) set of roles non-homomorphically encoding r. Thus, if ℓ is an instance of $delR(r)$, $\sigma(x')$ is reachable from $\sigma(x)$ via execution of $delR(r)$, $delR(r_1)$, …, $delR(r_k)$.
- If ℓ is an instance of $assignUser(u,r)$, then $x' = x \cup UR(u,r)$. By σ, $\sigma(x') = \sigma(x) \cup (UR(u,r) \cup UR(u,r_1) \cup \cdots \cup UR(u,r_k))$ where $r_1,\ldots,r_k$ is the set of roles which encode hierarchy chains of which r is the head (i.e., the set of roles that begin with concat(r, Sentinel())). By $RBAC_0$'s *next* relation, $terminal(\sigma(x), assignUser(u,r) \circ assignUser(u,r_1) \circ \cdots \circ assignUser(u,r_k)) = \sigma(x) \cup (UR(u,r) \cup UR(u,r_1) \cup \cdots \cup UR(u,r_k))$. Thus, if ℓ is an instance of $assignUser(u,r)$, $\sigma(x')$ is reachable from $\sigma(x)$ via execution of $assignUser(u,r)$, $assignUser(u,r_1)$, …, $assignUser(u,r_k)$. A similar argument holds for instances of $revokeUser(u,r)$, with reachability via $revokeUser(u,r)$, $revokeUser(u,r_1)$, …, $revokeUser(u,r_k)$ (for the same set of roles, $r_1,\ldots,r_k$).
- If ℓ is an instance of $assignPermission(r,p)$, then $x' = x \cup PA(r,p)$. By σ, $\sigma(x') = \sigma(x) \cup (PA(r,p) \cup PA(r_1,p) \cup \cdots \cup PA(r_k,p))$ where $r_1,\ldots,r_k$ is the set of roles which encode hierarchy chains of which r is the tail (i.e., the set of roles that end with concat(Sentinel(), r)). By $RBAC_0$'s *next* relation, $terminal(\sigma(x), assignPermission(r,p) \circ assignPermission(r_1,p) \circ \cdots \circ assignPermission(r_k,p)) = \sigma(x) \cup (PA(r,p) \cup PA(r_1,p) \cup \cdots \cup PA(r_k,p))$. Thus, if ℓ is an instance of $assignPermission(r,p)$, $\sigma(x')$ is reachable from $\sigma(x)$ via execution of $assignPermission(r,p)$, $assignPermission(r_1,p)$, …, $assignPermission(r_k,p)$. A similar argument holds for instances of $revokePermission(r,p)$, with reachability via $revokePermission(r,p)$, $revokePermission(r_1,p)$, …, $revokePermission(r_k,p)$ (for the same set of roles, $r_1,\ldots,r_k$).

- If ℓ is an instance of $addHierarchy(r_1, r_2)$, then $x' = x \cup RH(r_1, r_2)$. By σ, $\sigma(x') = \sigma(x) \cup ChainRoles(\sigma(x), r_1, r_2) \cup RoleAssigns(\sigma(x), r_1, r_2) \cup PermAssigns(\sigma(x), r_1, r_2)$. Here, $ChainRoles(\sigma(x), r_1, r_2)$ denotes the set of roles encoding hierarchy chains that are newly formed in $\sigma(x)$ by adding a link from r_1 to r_2—every role that ends with concat(Sentinel(), r_1) is joined with every role that begins with concat(r_2, Sentinel()). $RoleAssigns(\sigma(x), r_1, r_2)$ denotes the set of new assignments of users to chain-encoding roles formed in $\sigma(x)$ by adding a link from r_1 to r_2—for each new chain-encoding role r_i, if u is assigned to the head of the chain (i.e., r_i begins with concat(r_j, Sentinel()) and $UR(u, r_j) \in Th(\sigma(x))$), then $RoleAssigns$ contains $UR(u, r_i)$. $PermAssigns(\sigma(x), r_1, r_2)$ denotes the set of new assignments of permissions to chain-encoding roles—for each new chain-encoding role r_i, if p is assigned to the tail of the chain (i.e., r_i ends with concat(Sentinel(), r_j) and $PA(r_j, p) \in Th(\sigma(x))$), then $PermAssigns$ contains $PA(r_i, p)$.

  By $RBAC_0$'s *next* relation, $terminal(\sigma(x), addR(r_1^c) \circ \cdots \circ addR(r_j^c) \circ assignUser(u_1, r_1^u) \circ \cdots \circ assignUser(u_k, r_k^u) \circ assignPermission(r_1^p, p_1) \circ \cdots \circ assignPermission(r_l^p, p_l)) = \sigma(x')$, where each r_i^c is a role encoding newly-formed hierarchy chains, each u_i, r_i^u is a new assignment of a user to a chain-encoding role, and each r_i^p, p_i is a new assignment of a permission to a chain-encoding role. Thus, if ℓ is an instance of $addHierarchy(r_1, r_2)$, $\sigma(x')$ is reachable from $\sigma(x)$ via execution of this sequence of $RBAC_0$ labels.

- If ℓ is an instance of $removeHierarchy(r_1, r_2)$, then $x' = x \setminus RH(r_1, r_2)$. By σ, $\sigma(x') = \sigma(x) \setminus (R(r_1^c) \cup \cdots \cup R(r_k^c))$ where $r_1^c, \ldots, r_k^c$ is the set of roles which encode hierarchy chains relying on the connection of r_1 to r_2 (i.e., the set of roles that begin with concat(r_1, Sentinel(), r_2, Sentinel()), end with concat(Sentinel(), r_1, Sentinel(), r_2), or contain concat(Sentinel(), r_1, Sentinel(), r_2, Sentinel())). By $RBAC_0$'s *next* relation, $terminal(\sigma(x), delR(r_1^c) \circ \cdots \circ delR(r_k^c)) = \sigma(x) \setminus (R(r_1^c) \cup \cdots \cup R(r_k^c))$. Thus, if ℓ is an instance of $removeHierarchy(r_1, r_2)$, $\sigma(x')$ is reachable from $\sigma(x)$ via execution of $delR(r_1^c)$, ..., $delR(r_k^c)$.

Thus, for any $RBAC_1$ state x and label ℓ, $\sigma(next(x, \ell))$ is reachable from $\sigma(x)$ via $RBAC_0$ labels. By induction, for any $RBAC_1$ states s and s', if s' is reachable from s, then $\sigma(s')$ is reachable from $\sigma(s)$. Thus, we have shown that σ preserves reachability.

Thus, we have shown that σ preserves π, is injective, and preserves reachability; and that π is AC-preserving.

$\therefore \langle \sigma, \pi \rangle$ is a reduction from $RBAC_1$ to $RBAC_0$ which shows $RBAC_1 \leq^{CA} RBAC_0$. □

Theorem 14 *There exists a reduction $\langle \sigma, \pi \rangle$ from $RBAC_0$ to ugo where:*

- *σ preserves π, is injective, and preserves reachability*
- *π is weak AC-preserving*

Thus, $RBAC_0 \leq^{Ca}$ ugo (ugo is at least as expressive as $RBAC_0$ with respect to correctness and weak AC-preservation).

Proof We present the reduction, $\langle \sigma, \pi \rangle$. First, σ maps the $RBAC_0$ state $\langle U, R, P, UR, PA \rangle$ to the *ugo* state $\langle S, O, G, Member, Owner, Group, OwnerRight, GroupRight, OtherRight \rangle$. This mapping is described as follows.

```
sigma(M)
    Let WildGroups = {}

    Let S = {x | U(x) ∈ M ∨ R(x) ∈ M}
    Let O = {o | P(o) ∈ M}
    Let G = {x | R(x) ∈ M ∨ P(x) ∈ M}

    Let Member = {<x, y> | UR(x, y) ∈ M ∨ PA(x, y) ∈ M}
```

```
Let Group = {}
Let GroupRight = {}

for each (o ∈ O)
   Grp = nFreshConst(1, Consts(M) ∪ WildGroups, Univ)
   WildGroups = WildGroups ∪ {Grp}
   G = G ∪ {Grp}
   Group = Group ∪ {<o, Grp>}
   GroupRight = GroupRight ∪ {<o, read>}
   for each (s ∈ {s | ∃ r.(r ∈ S ∧ r ∈ G ∧
                          <s, r> ∈ Member ∧
                          <r, o> ∈ Member)})
      Member = Member ∪ {<s, Grp>}

 outputSet(S ∪ O ∪ G ∪ Member ∪ Group ∪ GroupRight)
```

We note that this mapping is not strict HPL due to its use of the constant right *read*. In addition, if the universe of possible users and roles (or roles and permissions) in $RBAC_0$ intersect (i.e., it is possible that $U(x) \wedge R(x)$), some non-homomorphic encoding is necessary to store elements as described.

Since the entire $RBAC_0$ state is stored uniquely in the *ugo* state, no two $RBAC_0$ states will map to the same *ugo* state. Thus, σ is injective.

The query mapping, π, is defined as follows.

$$\begin{aligned}
\pi_{UR(u,r)}(T) &= Member(u,r) \in T \wedge \forall x.(Group(r,x) \notin T \wedge Group(x,r) \notin T)\\
\pi_{PA(r,p)}(T) &= Member(r,p) \in T \wedge \exists x.(Group(p,x) \in T)\\
\pi_{R(r)}(T) &= G(r) \in T \wedge \forall x.(Group(r,x) \notin T \wedge Group(x,r) \notin T)\\
\pi_{auth(u,p)}(T) &= auth(u,p,read) \in T
\end{aligned}$$

Let x be an arbitrary $RBAC_0$ state and $\lambda = (u,p)$ an arbitrary $RBAC_0$ request, and let $f(u,p) = (u,p,read)$ be a request transform. Assume $\pi_{auth(\lambda)}(Th(\sigma(x))) = \text{TRUE}$. Then, by π, $auth(u,p,read) \in Th(\sigma(x))$, or $\sigma(x) \vdash auth(u,p,read)$.

Now let x be an arbitrary $RBAC_0$ state, $\lambda' = (s,o,r)$ an arbitrary *ugo* request, and f the request transform defined above. Assume $\sigma(x) \vdash auth(s,o,r)$. Since σ never constructs a *ugo* state that grants any rights besides *read*, $r = read$, and $\lambda' = (s,o,r)$. Finally, $f(s,o) = (s,o,read)$, and $\pi_{auth(s,o)}(Th(\sigma(x))) = \text{TRUE}$. Thus, π is weak AC-preserving with transform $f(u,p) = (u,p,read)$.

We show that σ preserves π (for any $RBAC_0$ state x, $Th(x) = \pi(Th(\sigma(x))))$ by contradiction. Assume that there is some $RBAC_0$ state x and query q such that the value of q in x is the opposite of the value of $\pi(q)$ in $\sigma(x)$. We show that, for each of the query forms of $RBAC_0$, this assumption leads to contradiction.

- **UR** Assume $x \vdash UR(u,r)$ and $\sigma(x) \nvdash \pi(UR(u,r))$. Then, $UR(u,r) \in Th(x)$, and by σ, $Member(u,r) \in Th(\sigma(x))$. Since r is a role, $\forall x.(Group(r,x) \notin Th(\sigma(x)) \wedge Group(x,r) \notin Th(\sigma(x)))$. By π, $\pi_{UR(u,r)}(Th(\sigma(x))) = \text{TRUE}$, which is a contradiction on the assumption that $\sigma(x) \nvdash \pi(UR(u,r))$.

  Assume instead that $x \nvdash UR(u,r)$ and $\sigma(x) \vdash \pi(UR(u,r))$. Then, $UR(u,r) \notin Th(x)$, and by σ, if $Member(u,r) \in Th(\sigma(x))$ then it is not to represent $UR(u,r)$, which is only possible if r does not represent a role, and thus it must represent a permission or a generated group used to grant a permission. Thus, $\exists x.(Group(r,x) \in Th(\sigma(x)) \vee Group(x,r) \in Th(\sigma(x)))$. By π, $\pi_{UR(u,r)}(Th(\sigma(x))) = \text{FALSE}$, which is a contradiction on the assumption that $\sigma(x) \vdash \pi(UR(u,r))$.

- **PA** Assume $x \vdash PA(r,p)$ and $\sigma(x) \nvdash \pi(PA(r,p))$. Then, $PA(r,p) \in Th(x)$, and by σ, $Member(r,p) \in Th(\sigma(x)) \wedge \exists x.(Group(p,x) \in Th(\sigma(x)))$. By π, $\pi_{PA(r,p)}(Th(\sigma(x))) = \text{TRUE}$, which is a contradiction on the assumption that $\sigma(x) \nvdash \pi(PA(r,p))$.

  Assume instead that $x \nvdash PA(r,p)$ and $\sigma(x) \vdash \pi(PA(r,p))$. Then, $PA(r,p) \notin Th(x)$, and by σ, if $Member(r,p) \in Th(\sigma(x))$ then it is not to represent $PA(r,p)$, which is only possible if p does not represent a permission, and thus $\forall x.(Group(p,x) \notin Th(\sigma(x)))$. By π, $\pi_{PA(r,p)}(Th(\sigma(x))) = \text{FALSE}$, which is a contradiction on the assumption that $\sigma(x) \vdash \pi(PA(r,p))$.

- **R** Assume $x \vdash R(r)$ and $\sigma(x) \nvdash \pi(R(r))$. Then, $R(r) \in Th(x)$, and by σ, $G(r) \in Th(\sigma(x))$. Since r is a role, $\forall x.(Group(r,x) \notin Th(\sigma(x)) \wedge Group(x,r) \notin Th(\sigma(x)))$. By π, $\pi_{R(r)}(Th(\sigma(x))) = \text{TRUE}$, which is a contradiction on the assumption that $\sigma(x) \nvdash \pi(R(r))$.

  Assume instead that $x \nvdash R(r)$ and $\sigma(x) \vdash \pi(R(r))$. Then, $R(r) \notin Th(x)$, and by σ, if $G(r) \in Th(\sigma(x))$ then r does not represent a role, and thus it must represent a permission or a generated group used to grant a permission. Thus, $\exists x.(Group(r,x) \in Th(\sigma(x)) \vee Group(x,r) \in Th(\sigma(x)))$. By π, $\pi_{R(r)}(Th(\sigma(x))) = \text{FALSE}$, which is a contradiction on the assumption that $\sigma(x) \vdash \pi(R(r))$.

- **auth** Assume $x \vdash auth(u,p)$ and $\sigma(x) \nvdash \pi(auth(u,p))$. Then, $\exists r.(UR(u,r) \in Th(x) \wedge PA(r,p) \in Th(x))$. By σ, $GroupRight(p, read) \in Th(\sigma(x)) \wedge \exists g.(Group(p,g) \in Th(\sigma(x)) \wedge Member(u,g) \in Th(\sigma(x)))$. By π, $\pi_{auth(u,p)}(Th(\sigma(x))) = \text{TRUE}$, which is a contradiction on the assumption that $\sigma(x) \nvdash \pi(auth(u,p))$.

  Assume instead that $x \nvdash auth(u,p)$ and $\sigma(x) \vdash \pi(auth(u,p))$. Then, there is no role r such that $UR(u,r) \in Th(x) \wedge PA(r,p) \in Th(x)$. Thus, by σ, if $Group(p,g) \in Th(\sigma(x))$, then $Member(u,g) \notin Th(\sigma(x))$. Furthermore, $OwnerRight$ and $OtherRight$ are empty via σ. Thus, by π, $\pi_{auth(u,p)}(Th(\sigma(x))) = \text{FALSE}$, which is a contradiction on the assumption that $\sigma(x) \vdash \pi(auth(u,p))$.

Thus, by contradiction, σ preserves π.

We prove that σ preserves reachability by induction by showing that, for any $RBAC_0$ state x and label ℓ, $\sigma(next(x,\ell))$ in reachable from $\sigma(x)$ via $RBAC_0$ labels.

Given $RBAC_0$ state x and label ℓ, let $x' = next(x,\ell)$ by the state resulting from executing label ℓ in state x.

- If ℓ is an instance of $addU(u)$, then $x' = next(x,\ell) = x \cup U(u)$. By σ, this maps in ugo to state $\sigma(x') = \sigma(x) \cup S(u)$. By ugo's $next$ relation, $next(\sigma(x), addS(u)) = \sigma(x) \cup S(u)$. Thus, if ℓ is an instance of $addU(u)$, $\sigma(x')$ is reachable from $\sigma(x)$ via execution of $addS(u)$.

- If ℓ is an instance of $delU(u)$, then $x' = x \setminus (U(u) \cup Entries(x,u))$, where $Entries(x,u)$ denotes the set of all state tuples in x involving u. By σ, $\sigma(x') = \sigma(x) \setminus (S(u) \cup Entries(\sigma(x),u))$ (or simply $\sigma(x)$ if $G(u) \in Th(\sigma(x))$, u represents a role). By ugo's $next$ relation, $next(\sigma(x), delS(u)) = \sigma(x) \setminus (S(u) \cup Entries(\sigma(x),u))$. Thus, if ℓ is an instance of $delU(u)$, $\sigma(x')$ is reachable from $\sigma(x)$ via execution of $delS(u)$ (or via no action if $G(u) \in Th(\sigma(x))$).

- If ℓ is an instance of $addR(r)$, then $x' = next(x,\ell) = x \cup R(r)$. By σ, this maps in ugo to state $\sigma(x') = \sigma(x) \cup S(r) \cup G(r)$. By ugo's $next$ relation, $terminal(\sigma(x), addS(r) \circ addG(r)) = \sigma(x) \cup S(r) \cup G(r)$. Thus, if ℓ is an instance of $addR(r)$, $\sigma(x')$ is reachable from $\sigma(x)$ via execution of $addS(r)$, $addG(r)$.

- If ℓ is an instance of $delR(r)$, then $x' = x \setminus (R(r) \cup Entries(x,r))$. By σ, $\sigma(x') = \sigma(x) \setminus (S(r) \cup G(r) \cup Entries(\sigma(x),r))$ (or simply $\sigma(x)$ if $S(r) \notin Th(\sigma(x))$, r does not represent a role). By ugo's $next$ relation, $terminal(\sigma(x), delS(r) \circ delG(r)) = \sigma(x) \setminus (S(r) \cup G(r) \cup Entries(\sigma(x),r))$. Thus, if ℓ is an instance of $delR(r)$, $\sigma(x')$ is reachable from $\sigma(x)$ via execution of $delS(r)$, $delG(r)$ (or via no action if $S(r) \notin Th(\sigma(x))$).

- If ℓ is an instance of $addP(p)$, then $x' = x \cup P(p)$. By σ, $\sigma(x') = \sigma(x) \cup O(p) \cup G(p) \cup G(g) \cup Group(p,g) \cup GroupRight(p, read)$, where g is any fresh constant to represent a new group. By *ugo*'s *next* relation, $terminal(\sigma(x), addO(p) \circ addG(p) \circ addG(g) \circ changeGroup(p, g) \circ grantGroup(p, read)) = \sigma(x')$. Thus, if ℓ is an instance of $addP(p)$, $\sigma(x')$ is reachable from $\sigma(x)$ via execution of $addO(p)$, $addG(p)$, $addG(g)$, $changeGroup(p, g)$, and $grantGroup(p, read)$.

- If ℓ is an instance of $delP(p)$, then $x' = x \setminus (P(p) \cup Entries(x, p))$. By σ, $\sigma(x') = \sigma(x) \setminus (O(p) \cup G(p) \cup G(g))$, where $Group(p, g) \in Th(\sigma(x))$. By *ugo*'s *next* relation, $terminal(\sigma(x), delO(p) \circ delG(p) \circ delG(g) = \sigma(x')$. Thus, if ℓ is an instance of $delP(p)$, $\sigma(x')$ is reachable from $\sigma(x)$ via execution of $delO(p)$, $delG(p)$, and $delG(g)$.
- If ℓ is an instance of $assignUser(u, r)$, then $x' = x \cup UR(u, r)$. By σ, $\sigma(x') = \sigma(x) \cup Member(u, r) \cup Member(u, g_1) \cup \circ \cup Member(u, g_k)$, where $g_1, \ldots, g_k$ is the set of groups which grant accesses to objects u should gain by joining r, i.e., all g_i such that $\exists o.(Member(r, o) \in Th(\sigma(x)) \wedge O(o) \in Th(\sigma(x)) \wedge Group(o, g_i) \in Th(\sigma(x)))$. By *ugo*'s *next* relation, $terminal(\sigma(x), addMember(u, r) \circ addMember(u, g_1) \circ \cdots \circ addMember(u, g_k)) = \sigma(x')$. Thus, if ℓ is an instance of $assignUser(u, r)$, $\sigma(x')$ is reachable from $\sigma(x)$ via execution of $addMember(u, r)$, $addMember(u, g_1)$, $\ldots$, $addMember(u, g_k)$.
- If ℓ is an instance of $revokeUser(u, r)$, then $x' = x \setminus UR(u, r)$. By σ, $\sigma(x') = \sigma(x) \setminus (Member(u, r) \cup Member(u, g_1) \cup \circ \cup Member(u, g_k))$, where $g_1, \ldots, g_k$ is the set of groups which grant accesses to objects u should lose by leaving r. ($\sigma(x')$ is simply $\sigma(x)$ if $S(r) \notin Th(\sigma(x))$, r does not represent a role). By *ugo*'s *next* relation, $terminal(\sigma(x), removeMember(u, r) \circ removeMember(u, g_1) \circ \cdots \circ removeMember(u, g_k)) = \sigma(x')$. Thus, if ℓ is an instance of $revokeUser(u, r)$, $\sigma(x')$ is reachable from $\sigma(x)$ via execution of $removeMember(u, r)$, $removeMember(u, g_1)$, $\ldots$, $removeMember(u, g_k)$ (or via no action if $S(r) \notin Th(\sigma(x))$).
- If ℓ is an instance of $assignPermission(r, p)$, then $x' = x \cup PA(r, p)$. By σ, $\sigma(x') = \sigma(x) \cup Member(r, p) \cup Member(u_1, g) \cup \circ \cup Member(u_k, g)$, where $Group(p, g) \in Th(\sigma(x))$ and $u_1, \ldots, u_k$ is the set of users which should gain access to object p by its being added to r, i.e., all u_i such that $Member(u_i, r) \in Th(\sigma(x))) \wedge G(u_i) \in Th(\sigma(x))$. By *ugo*'s *next* relation, $terminal(\sigma(x), addMember(r, p) \circ addMember(u_1, g) \circ \cdots \circ addMember(u_k, g)) = \sigma(x')$. Thus, if ℓ is an instance of $assignPermission(r, p)$, $\sigma(x')$ is reachable from $\sigma(x)$ via execution of $addMember(r, p)$, $addMember(u_1, g)$, $\ldots$, $addMember(u_k, g)$.
- If ℓ is an instance of $revokePermission(r, p)$, then $x' = x \setminus PA(r, p)$. By σ, $\sigma(x') = \sigma(x) \setminus (Member(r, p) \cup Member(u_1, g) \cup \circ \cup Member(u_k, g))$, $Group(p, g) \in Th(\sigma(x))$ and $u_1, \ldots, u_k$ is the set of users in r, as in the previous point. ($\sigma(x')$ is simply $\sigma(x)$ if $O(p) \notin Th(\sigma(x))$, p does not represent a permission). By *ugo*'s *next* relation, $terminal(\sigma(x), removeMember(r, p) \circ removeMember(u_1, g) \circ \cdots \circ removeMember(u_k, g)) = \sigma(x')$. Thus, if ℓ is an instance of $revokePermission(r, p)$, $\sigma(x')$ is reachable from $\sigma(x)$ via execution of $removeMember(r, p)$, $removeMember(u_1, g)$, $\ldots$, $removeMember(u_k, g)$ (or via no action if $S(r) \notin Th(\sigma(x))$).

Thus, for any $RBAC_0$ state x and label ℓ, $\sigma(next(x, \ell))$ is reachable from $\sigma(x)$ via *ugo* labels. By induction, for any $RBAC_0$ states s and s', if s' is reachable from s, then $\sigma(s')$ is reachable from $\sigma(s)$. Thus, we have shown that σ preserves reachability.

Thus, we have shown that σ preserves π, is injective, and preserves reachability; and that π is weak AC-preserving.

$\therefore \langle \sigma, \pi \rangle$ is a reduction from $RBAC_0$ to *ugo* which shows $RBAC_0 \leq^{Ca} ugo$. □

4.5 Reductions by Transitivity

Corollary 15 $rgSIS \leq^{CaH} RBAC_1$

PROOF Follows directly from Theorem 9 and Propositions 5, 6 and 12. □

Corollary 16 $rgSIS \leq^{Ca} ugo$

PROOF Follows directly from Theorems 9 and 14 and Propositions 5 and 6. □

Corollary 17 $tgSIS \leq^{Ca} RBAC_0$

PROOF Follows directly from Theorems 10 and 13 and Propositions 5 and 6. □

Corollary 18 $tgSIS \leq^{Ca} ugo$

PROOF Follows directly from Corollary 17, Theorem 14, and Propositions 5 and 6. □

Corollary 19 $bgSIS \leq^{Ca} RBAC_0$

PROOF Follows directly from Theorems 11 and 13 and Propositions 5 and 6. □

Corollary 20 $bgSIS \leq^{Ca} ugo$

PROOF Follows directly from Corollary 19, Theorem 14, and Propositions 5 and 6. □

5 Implementations

5.1 Program Committee in $RBAC_1$

Theorem 21 *There exists an implementation $\langle \alpha, \sigma, \pi \rangle$ of PC in $RBAC_1$ where:*

- *α preserves σ, is homomorphic, and preserves safety*
- *σ preserves π and is homomorphic*
- *π is weak AC-preserving and homomorphic*

Thus, $RBAC_1$ admits a correct, weak AC-preserving, homomorphic, safe implementation of PC.

PROOF We present the implementation, $\langle \alpha, \sigma, \pi \rangle$. First, σ maps the g-SIS state $\langle S, O, G, T, Time, StrictJoin, LiberalJoin, StrictLeave, LiberalLeave, LiberalAdd \rangle$ to the $RBAC_1$ state $\langle U, R, P, UR, PA, RH \rangle$. This mapping is described as follows.

```
sigma(M)
   Let WildRoles = {}
   Let UR = {}
   Let PA = {}
   Let RH = {}

   for each (S(s) ∈ M)
      output(U(s))
   for each (G(g) ∈ M)
      InitGroup(M, g, RH, WildRoles)
   for each (O(o) ∈ M)
      output(P(o))

   Let Records = sortByTime(StrictJoin ∪ LiberalJoin ∪
                            StrictLeave ∪ LiberalLeave ∪
                            LiberalAdd)
   for each (Record ∈ Records)
      If ∃ s, g, t.(Record = <s, g, t> ∧
                 StrictJoin(s, g, t) ∈ M)
```

```
            ProcessSJoin(M, s, g, UR, RH, WildRoles)
        else If ∃ s, g, t.(Record = <s, g, t> ∧
                          LiberalJoin(s, g, t) ∈ M)
            ProcessLJoin(M, s, g)
        else If ∃ s, g, t.(Record = <s, g, t> ∧
                          StrictLeave(s, g, t) ∈ M)
            ProcessSLeave(M, s, g, UR, RH)
        else If ∃ s, g, t.(Record = <s, g, t> ∧
                          LiberalLeave(s, g, t) ∈ M)
            ProcessLLeave(M, s, g, UR, PA, RH, WildRoles)
        else If ∃ o, g, t.(Record = <o, g, t> ∧
                          LiberalAdd(o, g, t) ∈ M)
            ProcessLAdd(M, o, g, PA, RH)
        endif
    outputSet(UR ∪ PA ∪ RH)

InitGroup(M, g, RH, WildRoles)
    output(R(g))
    <Top, Bottom> = nFreshConst(2, Consts(M) ∪ WildRoles,
                                Univ)
    WildRoles = WildRoles ∪ {Top, Bottom}
    output(R(Top))
    output(R(Bottom))
    RH = RH ∪ {<Top, g>, <g, Bottom>}

ProcessSJoin(M, s, g, UR, RH, WildRoles)
    NewBottom = nFreshConst(1, Consts(M) ∪ WildRoles,
                            Univ)
    WildRoles = WildRoles ∪ {NewBottom}
    OldBottom = FindBottom(g, RH)
    output(R(NewBottom))
    RH = RH ∪ {<OldBottom, NewBottom>}
    UR = UR ∪ {<s, NewBottom>}

FindBottom(r, RH)
    If ∃ q.(<r, q> ∈ RH)
        return FindBottom(q, RH)
    else
        return r
    endif

ProcessLJoin(M, s, g)
    UR = UR ∪ {<s, g>}

ProcessSLeave(M, s, g, UR, RH)
    LeaveDown(s, g, UR, RH)
    LeaveOrphans(s, g, UR, RH)

LeaveDown(u, r, UR, RH)
    UR = UR \ {<u, r>}
    If ∃ q.(<r, q> ∈ RH)
```

```
      LeaveDown(u, q, UR, RH)
   endif

LeaveOrphans(u, g, UR, RH)
   for each (Orphan in {r | ∃ Head.(<Head, g> ∈ RH ∧
                                      <Head, r> ∈ RH)})
      UR = UR \ {<u, Orphan>}

ProcessLLeave(M, s, g, UR, PA, RH, WildRoles)
   Orphan = nFreshConst(1, Consts(M) ∪ WildRoles, Univ)
   WildRoles = WildRoles ∪ {Orphan}
   output(R(Orphan))
   Permissions = PermsInChain(s, g, UR, PA, RH)
   Top = FindTop(g, RH)
   for each (Permission ∈ Permissions)
      PA = PA ∪ {<Orphan, Permission>}
   UR = UR ∪ {<s, Orpan>}
   RH = RH ∪ {<Top, Orphan>}
   LeaveDown(s, g, UR, RH)

PermsInChain(u, r, UR, PA, RH)
   If <u, r> ∈ UR
      return PermsBelow(r, PA, RH, {})
   else If ∃ q.(<r, q> ∈ RH)
      return PermsInChain(u, q, UR, PA, RH)
   else
      return {}
   endif

PermsBelow(r, PA, RH, Perms)
   Perms = Perms ∪ {p | <r, p> ∈ PA}
   If ∃ q.(<r, q> ∈ RH)
      return PermsBelow(q, PA, RH, Perms)
   else
      return Perms
   endif

FindTop(r, RH)
   If ∃ q.(<q, r> ∈ RH)
      return FindTop(q)
   else
      return r
   endif

ProcessLAdd(M, o, g, PA, RH)
   Bottom = FindBottom(g, RH)
   PA = PA ∪ {<Bottom, o>}
```

As the mapping is described in HPL, it is homomorphic.

The query mapping, π, is defined as follows.

$$\pi_{Member(s,g)}(T) = UR(s,g) \in T \vee \exists r.(UR(s,r) \in T \wedge Senior(g,r) \in T)$$
$$\pi_{Assoc(o,g)}(T) = \exists r.(PA(r,o) \in T \wedge Senior(g,r) \in T)$$
$$\pi_{auth(s,o,g)}(T) = \exists r_1, r_2.(UR(s,r_1) \in T \wedge PA(r_2,o) \in T \wedge (r_1 = r_2 \vee Senior(r_1,r_2) \in T) \wedge \exists r_3.(Senior(r_3,g) \in T \wedge Senior(r_3,r_2) \in T))$$

This query mapping clearly contains no string manipulation and is thus homomorphic.

Let x be an arbitrary PC state and $\lambda = (s,o,g)$ an arbitrary PC request, and let $f(s,o,g) = (s,o)$ be a request transform. Assume $\pi_{auth(\lambda)}(Th(\sigma(x))) = \text{TRUE}$. Then, by π, $\exists r_1, r_2.(r_1 \geq r_2 \wedge UR(s,r_1) \in Th(\sigma(x)) \wedge PA(r_2,o) \in Th(\sigma(x)))$. Thus, by $RBAC_1$'s $\vdash$ relation, $\sigma(x) \vdash auth(s,o)$.

Now let x be an arbitrary PC state, $\lambda' = (u,p)$ an arbitrary $RBAC_1$ request, and f the request transform defined above. Assume $\sigma(x) \vdash auth(\lambda')$. Then, $\exists r_1, r_2.(r_1 \geq r_2 \wedge UR(u,r_1) \in Th(\sigma(x)) \wedge PA(r_2,p) \in Th(\sigma(x)))$. Furthermore, since σ only assigns permissions to role which correspond to some group, r_2 must exist either in the hierarchy below a role corresponding to a group, or as an orphan node attached to such a role: $\exists r_3.(Senior(r_3,g) \in Th(\sigma(x)) \wedge Senior(r_3,r_2) \in Th(\sigma(x)))$. Finally, $f(u,p,g) = (u,p)$, and $\pi_{auth(u,p,g)}(Th(\sigma(x))) = \text{TRUE}$. Thus, π is weak AC-preserving with transform $f(s,o,g) = (s,o)$.

We show that σ preserves π (for all PC states x, $Th(x) = \pi(Th(\sigma(x))))$ by contradiction. Assume that there is some PC state x and query q such that the value of q in x is the opposite of the value of $\pi(q)$ in $\sigma(x)$. We show that, for each of the query forms of PC, this assumption leads to contradiction.

- **Member** Assume $x \vdash Member(s,g)$ and $\sigma(x) \nvdash \pi(Member(s,g))$. Then, $\exists t_1.(Join(s,g,t_1) \in Th(x) \wedge \forall t_2.(Leave(s,g,t_2) \in Th(x) \Rightarrow t_1 > t_2))$ (s has joined g and not left). By σ, if the *Join* is a *LiberalJoin*, then $UR(s,g) \in Th(\sigma(x))$. If the *Join* is a *StrictJoin*, then $\exists r_1.(UR(s,r_1) \in Th(\sigma(x)) \wedge Senior(g,r_1) \in Th(\sigma(x)))$. By π, in either case, $\pi_{Member(s,g)}(Th(\sigma(x))) = \text{TRUE}$, which is a contradiction on the assumption that $\sigma(x) \nvdash \pi(Member(s,g))$.

  Assume instead that $x \nvdash Member(s,g)$ and $\sigma(x) \vdash \pi(Member(s,g))$. Then, either $\exists t_1.(Leave(s,g,t_1) \in Th(x) \wedge \forall t_2.(Join(s,g,t_2) \in Th(x) \Rightarrow t_1 > t_2))$ (s has left g and not returned), or $\forall t_1.(Join(s,g,t_1) \notin Th(x))$ (s has not joined g). By σ, in either case, $UR(s,g) \notin Th(\sigma(x)) \wedge \forall r_1.(Senior(g,r_1) \notin Th(\sigma(x)) \vee UR(s,r_1) \notin Th(\sigma(x)))$. By π, $\pi_{Member(o,g)}(Th(\sigma(x))) = \text{FALSE}$, which is a contradiction on the assumption that $\sigma(x) \vdash \pi(Member(s,g))$.

- **Assoc** Assume $x \vdash Assoc(o,g)$ and $\sigma(x) \nvdash \pi(Assoc(o,g))$. Then, $\exists t_1.(LiberalAdd(o,g,t_1) \in Th(x))$ (o was added to g). By σ, $\exists r_1.(Senior(g,r_1) \in Th(\sigma(x)) \wedge PA(r_1,o) \in Th(\sigma(x)))$. By π, $\pi_{Assoc(o,g)}(Th(\sigma(x))) = \text{TRUE}$, which is a contradiction on the assumption that $\sigma(x) \nvdash \pi(Assoc(o,g))$.

  Assume instead that $x \nvdash Assoc(o,g)$ and $\sigma(x) \vdash \pi(Assoc(o,g))$. Then, $\forall t_1.(LiberalAdd(o,g,t_1) \notin Th(x))$ (o has not added to g). By σ, $\forall r_1.(Senior(g,r_1) \notin Th(\sigma(x)) \vee PA(r_1,o) \notin Th(\sigma(x)))$. By π, $\pi_{Assoc(o,g)}(Th(\sigma(x))) = \text{FALSE}$, which is a contradiction on the assumption that $\sigma(x) \vdash \pi(Assoc(o,g))$.

- **auth** Assume $x \vdash auth(s,o,g)$ and $\sigma(x) \nvdash \pi(auth(s,o,g))$. Then, $\exists t_1, t_2.(Join(s,g,t_1) \in Th(x) \wedge LiberalAdd(o,g,t_2) \in Th(x) \wedge \forall t_3.(StrictLeave(s,g,t_3) \in Th(x) \Rightarrow t_1 > t_3))$ (s has joined g and not strict left; o has been added to g). If $t_2 > t_1$ (the join occurred first), then $\forall t_4.(Leave(s,g,t_4) \in Th(x) \Rightarrow t_1 > t_4 \vee t_4 > t_2)$ (s did not leave g between joining and o being added). If $t_1 > t_2$ (the add occurred first), then s's join must be a liberal join. In either case, by σ, $\exists r_1, r_2.(UR(s,r_1) \in Th(\sigma(x)) \wedge PA(r_2,o) \in Th(\sigma(x)) \wedge r_1 \geq r_2 \in Th(\sigma(x)))$ (s belongs to a role authorized to o or senior to a role authorized to o). Connection to g is preserved by σ, so $\exists r_3.(Senior(r_3,g) \in Th(\sigma(x)) \wedge Senior(r_3,r_2) \in Th(\sigma(x)))$, either because s and o are in the hierarchy below g or because s and o

are in an "orphaned node" due to o's removal from g. Thus, by π, $\pi_{auth(s,o,g)}(\mathit{Th}(\sigma(x))) = \text{TRUE}$, which is a contradiction on the assumption that $\sigma(x) \nvdash \pi(auth(s,o,g))$.

Assume instead that $x \nvdash auth(s,o,g)$ and $\sigma(x) \vdash \pi(auth(s,o,g))$. Then, either: $\forall t_1, t_2.(\mathit{Join}(s,g,t_1) \notin \mathit{Th}(x) \vee \mathit{LiberalAdd}(o,g,t_2) \notin \mathit{Th}(x))$ (s has not joined g or o has not been added to g); $\exists t_3.(\mathit{StrictLeave}(s,g,t_3) \in \mathit{Th}(x) \wedge t_3 > t_1)$ (s has since strict left g); or s's and o's membership in g did not overlap in a way that caused the authorization. In the final case, if $t_2 > t_1$ (the join occurred first), then $\exists t_4.(\mathit{Leave}(s,g,t_4) \in \mathit{Th}(x) \wedge t_2 > t_4 > t_1)$ (s left g before o was added). If $t_1 > t_2$ (the add occurred first), then $\mathit{Join}(s,g,t_1)$ must be $\mathit{StrictJoin}(s,g,t_1)$. Thus, by σ, if $\exists r_1, r_2.(UR(s,r_1) \in \mathit{Th}(\sigma(x)) \wedge PA(r_2,o) \in \mathit{Th}(\sigma(x)) \wedge r_1 \geq r_2 \in \mathit{Th}(\sigma(x)))$ (s belongs to a role authorized to o or senior to a role authorized to o), then it must be in conjunction with a group other than g: $\forall r_3.(\mathit{Senior}(r_3,g) \notin \mathit{Th}(\sigma(x)) \vee \mathit{Senior}(r_3,r_2) \notin \mathit{Th}(\sigma(x)))$. Thus, by π, $\pi_{auth(s,o,g)}(\mathit{Th}(\sigma(x))) = \text{FALSE}$, which is a contradiction on the assumption that $\sigma(x) \vdash \pi(auth(s,o,g)$.

Thus, by contradiction, σ preserves π.

Finally, the label mapping, α, is defined as follows.

```
addS(M, s)
    output(addU(s))

delS(M, s)
    output(delU(s))

addG(M, g)
    output(addR(g))
    <Top, Bottom> = nFreshConst(2, Consts(M), Univ)
    output(addR(Top))
    output(addR(Bottom))
    output(addHierarchy(Top, g))
    output(addHierarchy(g, Bottom))

delG(M, g)
    If ∃ Head.(RH(Head, g) ∈ M)
        DeleteDown(M, g)
        DeleteOrphans(M, Head)
        output(delR(Head))

DeleteDown(M, r)
    If ∃ q.(RH(r, q) ∈ M)
        DeleteDown(M, q)
    endif
    output(delR(r))

DeleteOrphans(M, Head)
    for each (Orphan ∈ {Role | RH(Head, Role) ∈ M}
        output(delR(Orphan))

addO(M, o)
    output(addP(o))

strictJoin(M, s, g)
    NewBottom = nFreshConst(1, Consts(M), Univ)
```

```
  OldBottom = FindBottom(M, g)
  output(addR(NewBottom))
  output(addHierarchy(OldBottom, NewBottom))
  output(assignUser(s, NewBottom))

FindBottom(M, r)
  If ∃ q.(RH(r, q) ∈ M)
    return FindBottom(M, q)
  else
    return r
  endif

liberalJoin(M, s, g)
  output(assignUser(s, g))

strictLeave(M, s, g)
  LeaveDown(M, s, g)
  LeaveOrphans(M, s, g)

LeaveDown u, r)
  If UR(u, r) ∈ M
    output(revokeUser(u, r))
  endif
  If ∃ q.(RH(r, q) ∈ M)
    LeaveDown(M, u, q)
  endif

LeaveOrphans(M, u, g)
  for each (Orphan ∈ {r | ∃ Head.(RH(Head, g) ∈ M ∧
                                  RH(Head, r) ∈ M)})
    If UR(u, Orphan) ∈ M
      output(revokeUser(u, Orphan))
    endif

liberalLeave(M, s, g)
  Orphan = nFreshConst(1, Consts(M), Univ)
  output(addR(Orphan))
  Top = FindTop(M, r)
  for each (Permission ∈ PermsInChain(M, s, g))
    output(assignPermission(Orphan, Permission))
  output(assignUser(s, Orphan))
  output(addHierarchy(Top, Orphan))
  LeaveDown(M, s, g)

FindTop(M, r)
  If ∃ q.(RH(q, r) ∈ M)
    return FindTop(M, q)
  else
    return r
  endif
```

```
PermsInChain(M, u, r)
    If UR(u, r) ∈ M
        return PermsBelow(M, r, {})
    else If ∃ q.(RH(r, q) ∈ M)
        return PermsInChain(M, u, q)
    else
        return {}
    endif

PermsBelow(M, r, Perms)
    Perms = Perms ∪ {p | PA(r, p) ∈ M}
    If ∃ q.(RH(r, q) ∈ M)
        return PermsBelow(M, q, Perms)
    else
        return Perms
    endif

liberalAdd(M, o, g)
    Bottom = FindBottom(M, g)
    output(assignPermission(Bottom, o))
```

This mapping is described in HPL, and is thus homomorphic.

We prove that α preserves σ by showing that, for any PC state x and label ℓ, $\sigma(next(x,\ell)) = terminal(\sigma(x), \alpha(\sigma(x),\ell))$.

Given PC state x and label ℓ, $x' = next(x,\ell)$ is the state resulting from executing label ℓ in state x.

- If ℓ is an instance of $addS(s)$, then $x' = x \cup S(s)$. By σ, this maps in $RBAC_1$ to state $\sigma(x') = \sigma(x) \cup U(s)$. By α, $\alpha(\sigma(x),\ell) = addU(s)$. By $RBAC_1$'s *next* relation, $next(\sigma(x), addU(s)) = \sigma(x) \cup U(s)$. Thus, if ℓ is an instance of $addS(s)$, $\sigma(x') = terminal(\sigma(x), \alpha(\sigma(x),\ell))$.
- If ℓ is an instance of $delS(s)$, then $x' = x \setminus (S(s) \cup Entries(x,s))$, where $Entries(x,s)$ denotes the set of all state tuples in x involving s. By σ, $\sigma(x') = \sigma(x) \setminus (U(s) \cup Entries(\sigma(x),s))$. By α, $\alpha(\sigma(x),\ell) = delU(s)$. By $RBAC_1$'s *next* relation, $next(\sigma(x), delU(s)) = \sigma(x) \setminus (U(s) \cup Entries(\sigma(x),s))$. Thus, if ℓ is an instance of $delS(s)$, $\sigma(x') = terminal(\sigma(x), \alpha(\sigma(x),\ell))$.
- If ℓ is an instance of $addG(g)$, then $x' = x \cup G(g)$. By σ, $\sigma(x') = \sigma(x) \cup R(g) \cup R(r_{top}) \cup R(r_{bottom}) \cup RH(r_{top}, g) \cup RH(g, r_{bottom})$, where r_{top} and r_{bottom} are newly-created roles. By α, $\alpha(\sigma(x),\ell) = addR(g) \circ addR(r_{top}) \circ addR(r_{bottom}) \circ addHierarchy(r_{top}, g) \circ addHierarchy(g, r_{bottom})$. By $RBAC_1$'s *next* relation, $terminal(\sigma(x), addR(g) \circ addR(r_{top}) \circ addR(r_{bottom}) \circ addHierarchy(r_{top}, g) \circ addHierarchy(g, r_{bottom})) = \sigma(x) \cup R(g) \cup R(r_{top}) \cup R(r_{bottom}) \cup RH(r_{top}, g) \cup RH(g, r_{bottom})$. Thus, if ℓ is an instance of $addG(g)$, $\sigma(x') = terminal(\sigma(x), \alpha(\sigma(x),\ell))$.
- If ℓ is an instance of $delG(g)$, then $x' = x \setminus (G(g) \cup Entries(x,g))$. By σ, $\sigma(x') = \sigma(x) \setminus (R(g) \cup ConnectedEntries(\sigma(x),g))$, where $ConnectedEntries(\sigma(x),g)$ denotes the set of state tuples in $\sigma(x)$ involving either g or any role connected to g in the role hierarchy of $\sigma(x)$ (i.e., $ConnectedEntries(x,r) \triangleq r \cup Entries(x,r) \cup \{ConnectedEntries(x,q) \mid RH(r,q) \in Th(x) \vee RH(q,r) \in Th(x)\}$). By α, $\alpha(\sigma(x),\ell) = delR(g) \circ delR(r_1) \circ \cdots \circ delR(r_k)$, where $r_1, \ldots, r_k$ is the (finite) set of roles connected to g in the role hierarchy. By $RBAC_1$'s *next* relation, $terminal(\sigma(x), delR(g) \circ delR(r_1) \circ \cdots \circ delR(r_k)) = \sigma(x) \setminus (R(g) \cup ConnectedEntries(\sigma(x),g))$. Thus, if ℓ is an instance of $delG(g)$, $\sigma(x') = terminal(\sigma(x), \alpha(\sigma(x),\ell))$.
- If ℓ is an instance of $addO(o)$, then $x' = x \cup O(o)$. By σ, $\sigma(x') = \sigma(x) \cup P(o)$. By α, $\alpha(\sigma(x),\ell) = addP(o)$. By $RBAC_1$'s *next* relation, $next(\sigma(x), addP(o)) = \sigma(x) \cup P(o)$. Thus, if ℓ is an instance

of $addP(o)$, $\sigma(x') = terminal(\sigma(x), \alpha(\sigma(x), \ell))$.

- If ℓ is an instance of $strictJoin(s, g)$, then $x' = x \cup StrictJoin(s, g, t) \cup Time(t+1) \setminus Time(t)$. By σ, $\sigma(x') = \sigma(x) \cup R(r_{new}) \cup RH(r_{bottom}, r_{new}) \cup UR(s, r_{new})$, where r_{bottom} is the current bottom of the hierarchy chain below g and r_{new} is a newly-created role. By α, $\alpha(\sigma(x), \ell) = addR(r_{new}) \circ addHierarchy(r_{bottom}, r_{new}) \circ assignUser(s, r_{new})$. By $RBAC_1$'s $next$ relation, $terminal(\sigma(x), addR(r_{new}) \circ addHierarchy(r_{bottom}, r_{new}) \circ assignUser(s, r_{new})) = \sigma(x) \cup R(r_{new}) \cup RH(r_{bottom}, r_{new}) \cup UR(s, r_{new})$. Thus, if ℓ is an instance of $strictJoin(s, g)$, $\sigma(x') = terminal(\sigma(x), \alpha(\sigma(x), \ell))$.
- If ℓ is an instance of $liberalJoin(s, g)$, then $x' = x \cup LiberalJoin(s, g, t) \cup Time(t+1) \setminus Time(t)$. By σ, $\sigma(x') = \sigma(x) \cup UR(s, g)$. By α, $\alpha(\sigma(x), \ell) = assignUser(s, g)$. By $RBAC_1$'s $next$ relation, $next(\sigma(x), assignUser(s, g)) = \sigma(x) \cup UR(s, g)$. Thus, if ℓ is an instance of $liberalJoin(s, g)$, $\sigma(x') = terminal(\sigma(x), \alpha(\sigma(x), \ell))$.
- If ℓ is an instance of $strictLeave(s, g)$, then $x' = x \cup StrictLeave(s, g, t) \cup Time(t+1) \setminus Time(t)$. By σ, $\sigma(x') = \sigma(x) \setminus (UR(s, g) \cup UR(s, r_1) \cup \cdots \cup UR(s, r_k))$, where $r_1, \ldots, r_k$ is the set of roles to which s belongs and which are also connected in the role hierarchy to g. By α, $\alpha(\sigma(x), \ell) = revokeUser(s, g) \circ revokeUser(s, r_1) \circ \cdots \circ revokeUser(s, r_k)$. By $RBAC_1$'s $next$ relation, $terminal(\sigma(x), revokeUser(s, g) \circ revokeUser(s, r_1) \circ \cdots \circ revokeUser(s, r_k)) = \sigma(x) \setminus (UR(s, g) \cup UR(s, r_1) \cup \cdots \cup UR(s, r_k))$. Thus, if ℓ is an instance of $strictLeave(s, g)$, $\sigma(x') = terminal(\sigma(x), \alpha(\sigma(x), \ell))$.
- If ℓ is an instance of $liberalLeave(s, g)$, then $x' = x \cup liberalLeave(s, g, t) \cup Time(t+1) \setminus Time(t)$. By σ, $\sigma(x') = \sigma(x) \cup R(r_{orphan}) \cup PA(r_{orphan}, p_1) \cup \cdots \cup PA(r_{orphan}, p_k) \cup UR(s, r_{orphan}) \cup RH(r_{head}, r_{orphan}) \setminus (UR(s, r_1) \cup \cdots \cup UR(s, r_l))$, where $p_1, \ldots, p_k$ is the set of permissions to which s is authorized in g, and $r_1, \ldots, r_l$ is the set of roles to which s is authorized in the role hierarchy chain below g. By α, $\alpha(\sigma(x), \ell) = addR(r_{orphan}) \circ assignPermission(r_{orphan}, p_1) \circ \cdots \circ assignPermission(r_{orphan}, p_k) \circ assignUser(s, r_{orphan}) \circ addHierarchy(r_{head}, r_{orphan}) \circ revokeUser(s, r_1) \circ \cdots \circ revokeUser(s, r_l)$. By $RBAC_1$'s $next$ relation, $terminal(\sigma(x), addR(r_{orphan}) \circ assignPermission(r_{orphan}, p_1) \circ \cdots \circ assignPermission(r_{orphan}, p_k) \circ assignUser(s, r_{orphan}) \circ addHierarchy(r_{head}, r_{orphan}) \circ revokeUser(s, r_1) \circ \cdots \circ revokeUser(s, r_l)) = \sigma(x) \cup R(r_{orphan}) \cup PA(r_{orphan}, p_1) \cup \cdots \cup PA(r_{orphan}, p_k) \cup UR(s, r_{orphan}) \cup RH(r_{head}, r_{orphan}) \setminus (UR(s, r_1) \cup \cdots \cup UR(s, r_l))$. Thus, if ℓ is an instance of $liberalLeave(s, g)$, $\sigma(x') = terminal(\sigma(x), \alpha(\sigma(x), \ell))$.
- If ℓ is an instance of $liberalAdd(o, g)$, then $x' = x \cup LiberalAdd(o, g, t) \cup Time(t+1) \setminus Time(t)$. By σ, $\sigma(x') = \sigma(x) \cup PA(r_{bottom}, o)$, where r_{bottom} is the bottom role of the role hierarchy chain below g. By α, $\alpha(\sigma(x), \ell) = assignPermission(r_{bottom}, o)$. By $RBAC_1$'s $next$ relation, $next(\sigma(x), assignPermission(r_{bottom}, o)) = \sigma(x) \cup PA(r_{bottom}, o)$. Thus, if ℓ is an instance of $liberalAdd(o, g)$, $\sigma(x') = terminal(\sigma(x), \alpha(\sigma(x), \ell))$.

Thus, for any PC state x and label ℓ, $\sigma(next(x, \ell)) = terminal(\sigma(x), \alpha(\sigma(x), \ell))$. Thus, we have shown that α preserves σ.

Finally, α is safe by inspection—for any PC state x and label ℓ, the sequence of $RBAC_1$ labels $\alpha(\sigma(x), \ell)$ never revokes or grants authorizations except the images of those that are revoked or granted by ℓ.

Thus, we have shown that α preserves σ, is homomorphic, and preserves safety; that σ preserves π and is homomorphic; and that π is weak AC-preserving and homomorphic.

$\therefore$ $\langle \alpha, \sigma, \pi \rangle$ is an implementation of PC in $RBAC_1$ which preserves correctness, weak AC-preservation, homomorphism, and safety. □

5.2 PlayStation Plus in $RBAC_1$

Theorem 22 *There exists an implementation $\langle \alpha, \sigma, \pi \rangle$ of PSP in $RBAC_1$ where:*

- *α preserves σ, is homomorphic, and preserves safety*
- *σ preserves π and is homomorphic*
- *π is weak AC-preserving and homomorphic*

Thus, $RBAC_1$ admits a correct, homomorphic implementation of PSP.

Proof We present the implementation, $\langle \alpha, \sigma, \pi \rangle$. First, σ maps the g-SIS state $\langle S, O, G, T, Time, LiberalJoin, StrictLeave, LiberalAdd, StrictRemove, LiberalRemove \rangle$ to the $RBAC_1$ state $\langle U, R, P, UR, PA, RH \rangle$. This mapping is described as follows.

```
sigma(M)
   Let WildRoles = {}
   Let UR = {}
   Let PA = {}
   Let RH = {}

   for each (S(s) ∈ M)
      output(U(s))
   for each (G(g) ∈ M)
      InitGroup(M, g, RH, WildRoles)
   for each (O(o) ∈ M)
      output(P(o))

   Let Records = sortByTime(LiberalJoin ∪ StrictLeave ∪
                            LiberalAdd ∪ StrictRemove ∪
                            LiberalRemove)
   for each (Record ∈ Records)
      If ∃ s, g, t.(Record = <s, g, t> ∧
                    LiberalJoin(s, g, t) ∈ M)
         ProcessLJoin(M, s, g, UR, RH)
      else If ∃ s, g, t.(Record = <s, g, t> ∧
                         StrictLeave(s, g, t) ∈ M)
         ProcessSLeave(M, s, g, UR, PA, RH, WildRoles)
      else If ∃ o, g, t.(Record = <o, g, t> ∧
                         LiberalAdd(o, g, t) ∈ M)
         ProcessLAdd(M, o, g, PA)
      else If ∃ o, g, t.(Record = <o, g, t> ∧
                         StrictRemove(o, g, t) ∈ M)
         ProcessSRemove(M, o, g, PA, RH)
      else If ∃ o, g, t.(Record = <o, g, t> ∧
                         LiberalRemove(o, g, t) ∈ M)
         ProcessLRemove(M, o, g, UR, PA, RH, WildRoles)
      endif
   outputSet(UR ∪ PA ∪ RH)

InitGroup(M, g, RH, WildRoles)
   output(R(g))
   <SubA, SubB> = nFreshConst(2, Consts(M) ∪ WildRoles,
                              Univ)
   WildRoles = WildRoles ∪ {SubA, SubB}
   output(R(SubA))
```

```
    output(R(SubA))
    RH = RH ∪ {<g, SubA>, <SubA, SubB>}

ProcessLJoin(M, s, g, UR, RH)
    UR = UR ∪ {<s, g>}
    If ∃ x,y,z.({<g, x>, <y, x>, <z, y>} ⊂ RH ∧
                <s, y> ∈ UR)
        UR = UR ∪ {<s, z>} \ {<s, y>}
    endIf

ProcessSLeave(M, s, g, UR, PA, RH, WildRoles)
    If <s, g> ∉ UR
        return
    endif
    If ∃ x,y,z.({<g, x>, <y, x>, <z, y>} ⊂ RH ∧
                <s, z> ∈ UR)
        UR = UR ∪ {<s, y>} \ {<s, z>}
    else
        Let <y, z> = nFreshConst(2, Consts(M) ∪ WildRoles,
                                 Univ)
        WildRoles = WildRoles ∪ {y, z}
        output(R(y))
        output(R(z))
        RH = RH ∪ {<z, y>}
        If ∃ x.(<g, x> ∈ RH)
            RH = RH ∪ {<y, x>}
        endif
        UR = UR ∪ {<s, y>}
    endif
    for each (p ∈ {p | <g, p> ∈ PA})
        PA = PA ∪ {<z, p>}
    UR = UR \ {<s, g>}

ProcessLAdd(M, o, g, PA)
    PA = PA ∪ {<g, o>}

ProcessSRemove(M, o, g, PA, RH)
    PA = PA \ {<g, o>}
    If ∃ SubA, SubB.({<g, SubA>, <SubA, SubB>} ⊂ RH)
        for each (Sup2 ∈ {z | ∃ y.({<y, SubA>,
                                    <z, y>} ⊂ RH)})
            PA = PA \ {<Sup2, o>}
        for each (Orphan ∈ {r | <r, SubB> ∈ RH} \ {SubA})
            PA = PA \ {<Orphan, o>}
    endif

ProcessLRemove(M, o, g, UR, PA, RH, WildRoles)
    If <g, o> ∉ PA
        return
    endif
    If ∃ SubA, SubB.({<g, SubA>, <SubA, SubB>} ⊂ RH)
```

```
        Orphan = nFreshConst(1, Consts(M) ∪ WildRoles,
                             Univ)
        WildRoles = WildRoles ∪ {Orphan}
        output(R(Orphan))
        RH = RH ∪ {<Orphan, SubB>}
        for each (u ∈ {u | <u, g> ∈ UR})
            UR = UR ∪ {<u, Orphan>}
        PA = PA ∪ {<Orphan, o>} \ {<g, o>}
    endif
```

As the mapping is described in HPL, it is homomorphic.

The query mapping, π, is defined as follows.

$$\begin{aligned}
\pi_{Member(s,g)}(T) &= UR(s,g) \in T \\
\pi_{Assoc(o,g)}(T) &= PA(g,o) \in T \\
\pi_{auth(s,o,g)}(T) &= \exists r_1, r_2.(UR(s,r_1) \in T \wedge PA(r_2,o) \in T \wedge \\
&\quad (r_1 = r_2 \vee Senior(r_1,r_2) \in T) \wedge \\
&\quad \exists r_3.(Senior(g,r_3) \in T \wedge Senior(r_2,r_3) \in T))
\end{aligned}$$

This query mapping clearly contains no string manipulation and is thus homomorphic.

Let x be an arbitrary PSP state and $\lambda = (s,o,g)$ an arbitrary PSP request, and let $f(s,o,g) = (s, o)$ be a request transform. Assume $\pi_{auth(\lambda)}(Th(\sigma(x))) =$ TRUE. Then, by π, $\exists r_1, r_2.(r_1 \geq r_2 \wedge UR(s,r_1) \in Th(\sigma(x)) \wedge PA(r_2,o) \in Th(\sigma(x)))$. Thus, by $RBAC_1$'s $\vdash$ relation, $\sigma(x) \vdash auth(s,o)$.

Now let x be an arbitrary PSP state, $\lambda' = (u,p)$ an arbitrary $RBAC_1$ request, and f the request transform defined above. Assume $\sigma(x) \vdash auth(\lambda')$. Then, $\exists r_1, r_2.(r_1 \geq r_2 \wedge UR(u,r_1) \in Th(\sigma(x)) \wedge PA(r_2,p) \in Th(\sigma(x)))$. Furthermore, since σ only assigns permissions to role which correspond to some group, r_2 must exist either in the hierarchy above a role corresponding to a group, or as an orphan node attached to such a role: $\exists r_3.(Senior(g,r_3) \in Th(\sigma(x)) \wedge Senior(r_2,r_3) \in Th(\sigma(x)))$. Finally, $f(u,p,g) = (u,p)$, and $\pi_{auth(u,p,g)}(Th(\sigma(x))) =$ TRUE. Thus, π is weak AC-preserving with transform $f(s,o,g) = (s,o)$.

We show that σ preserves π (for any PSP state x, $Th(x) = \pi(Th(\sigma(x))))$ by contradiction. Assume that there is some PSP state x and query q such that the value of q in x is the opposite of the value of $\pi(q)$ in $\sigma(x)$. We show that, for each of the query forms of PSP, this assumption leads to contradiction.

- **Member** Assume $x \vdash Member(s,g)$ and $\sigma(x) \nvdash \pi(Member(s,g))$. Then, $\exists t_1.(LiberalJoin(s,g,t_1) \in Th(x) \wedge \forall t_2.(StrictLeave(s,g,t_2) \in Th(x) \Rightarrow t_1 > t_2))$ (s has joined g and not left). By σ, $UR(s,g) \in Th(\sigma(x))$. By π, $\pi_{Member(s,g)}(Th(\sigma(x))) =$ TRUE, which is a contradiction on the assumption that $\sigma(x) \nvdash \pi(Member(s,g))$.

  Assume instead that $x \nvdash Member(s,g)$ and $\sigma(x) \vdash \pi(Member(s,g))$. Then, either $\exists t_1.(StrictLeave(s,g,t_1) \in Th(x) \wedge \forall t_2.(LiberalJoin(s,g,t_2) \in Th(x) \Rightarrow t_1 > t_2))$ (s has left g and not returned), or $\forall t_1.(LiberalJoin(s,g,t_1) \notin Th(x))$ (s has not joined g). By σ, in either case, $UR(s,g) \notin Th(\sigma(x))$. By π, $\pi_{Member(o,g)}(Th(\sigma(x))) =$ FALSE, which is a contradiction on the assumption that $\sigma(x) \vdash \pi(Member(s,g))$.

- **Assoc** Assume $x \vdash Assoc(o,g)$ and $\sigma(x) \nvdash \pi(Assoc(o,g))$. Then, $\exists t_1.(LiberalAdd(o,g,t_1) \in Th(x) \wedge \forall t_2.(Remove(o,g,t_2) \in Th(x) \Rightarrow t_1 > t_2))$ (o has been added to g and not removed). By σ, $PA(g,o) \in Th(\sigma(x))$. By π, $\pi_{Assoc(o,g)}(Th(\sigma(x))) =$ TRUE, which is a contradiction on the assumption that $\sigma(x) \nvdash \pi(Assoc(o,g))$.

  Assume instead that $x \nvdash Assoc(o,g)$ and $\sigma(x) \vdash \pi(Assoc(o,g))$. Then, either $\exists t_1.(Remove(o,g,t_1) \in Th(x) \wedge \forall t_2.(LiberalAdd(o,g,t_2) \in Th(x) \Rightarrow t_1 > t_2))$ (o has been removed from g and not re-added), or $\forall t_1.(LiberalAdd(o,g,t_1) \notin Th(x))$ (o has not been added to

g). By σ, in either case, $PA(o,g) \notin Th(\sigma(x))$. By π, $\pi_{Assoc(o,g)}(Th(\sigma(x))) = $ FALSE, which is a contradiction on the assumption that $\sigma(x) \vdash \pi(Assoc(o,g))$.

- **auth** Assume $x \vdash auth(s,o,g)$ and $\sigma(x) \nvdash \pi(auth(s,o,g))$. Then, $\exists t_1, t_2.(LiberalJoin(s,g,t_1) \in Th(x) \wedge LiberalAdd(o,g,t_2) \in Th(x) \wedge \forall t_3.(StrictRemove(o,g,t_3) \in Th(x) \Rightarrow t_2 > t_3))$ (s has joined g, o has been added to g and has not been strict removed). If $t_2 > t_1$ (the join occurred first), then $\forall t_3.(StrictLeave(s,g,t_3) \in Th(x) \Rightarrow t_1 > t_3 \vee t_3 > t_2)$ (s did not leave g between joining and o being added). If $t_1 > t_2$ (the add occurred first), then $\forall t_3.(Remove(o,g,t_3) \in Th(x) \Rightarrow t_2 > t_3 \vee t_3 > t_1)$ (o was not removed from g between being added and s joining). In either case, by σ, $\exists r_1, r_2.(UR(s,r_1) \in Th(\sigma(x)) \wedge PA(r_2,o) \in Th(\sigma(x)) \wedge r_1 \geq r_2 \in Th(\sigma(x)))$ (s belongs to a role authorized to o or senior to a role authorized to o). Connection to g is preserved by σ, so $\exists r_3.(Senior(g,r_3) \in Th(\sigma(x)) \wedge Senior(r_2,r_2) \in Th(\sigma(x)))$, either because s and o are both in role g (so $r_1 = r_2 = g$ and r_3 is the automatically-created role below g) or because s and o are in one of two types of "orphaned node." The first results from s strict leaving g and later re-joining (s must have re-joined if $x \vdash auth(s,o,g)$). The second results from o being liberal removed. Thus, in any of these cases, by π, $\pi_{auth(s,o,g)}(Th(\sigma(x))) = $ TRUE, which is a contradiction on the assumption that $\sigma(x) \nvdash \pi(auth(s,o,g))$.

  Assume instead that $x \nvdash auth(s,o,g)$ and $\sigma(x) \vdash \pi(auth(s,o,g))$. Then, either: $\forall t_1, t_2.(LiberalJoin(s,g,t_1) \notin Th(x) \vee LiberalAdd(o,g,t_2) \notin Th(x))$ (s has not joined g or o has not been added to g); $x \nvdash Member(s,g)$ (s is not currently a member of g); $\exists t_3.(StrictRemove(o,g,t_3) \in Th(x) \wedge t_3 > t_1)$ (o has been strict removed from g); or s's and o's membership in g did not overlap in a way that caused the authorization. In the final case, if $t_2 > t_1$ (the join occurred first), then $\exists t_4.(StrictLeave(s,g,t_4) \in Th(x) \wedge t_2 > t_4 > t_1)$ (s left g before o was added). If $t_1 > t_2$ (the add occurred first), then $\exists t_4.(Remove(s,g,t_4) \in Th(x) \wedge t_1 > t_4 > t_2)$ (o was removed from g before s joined). Thus, by σ, if $\exists r_1, r_2.(UR(s,r_1) \in Th(\sigma(x)) \wedge PA(r_2,o) \in Th(\sigma(x)) \wedge r_1 \geq r_2 \in Th(\sigma(x)))$ (s belongs to a role authorized to o or senior to a role authorized to o), then it must be in conjunction with a group other than g: $\forall r_3.(Senior(g,r_3) \notin Th(\sigma(x)) \vee Senior(r_2,r_3) \notin Th(\sigma(x)))$. Thus, by π, $\pi_{auth(s,o,g)}(Th(\sigma(x))) = $ FALSE, which is a contradiction on the assumption that $\sigma(x) \vdash \pi(auth(s,o,g)$.

Thus, by contradiction, σ preserves π.

Finally, the label mapping, α, is defined as follows.

```
addS(M, s)
    output(addU(s))

delS(M, s)
    output(delU(s))

addO(M, o)
    output(addP(o))

delO(M, o)
    output(delP(o))

liberalJoin(M, s, g)
    output(assignUser(s, g))
    If ∃ x,y,z.({RH(g, x), RH(y, x), RH(z, y),
                UR(s, y)} ⊂ M)
        output(assignUser(s, z))
        output(revokeUser(s, y))
```

```
    endIf

strictLeave(M, s, g)
    output(revokeUser(s, g))
    If ∃ x,y,z.({RH(g, x), RH(y, x), RH(z, y),
                UR(s, z)} ⊂ M)
        output(assignUser(s, y))
        output(revokeUser(s, z))
    else
        Let <y, z> = nFreshConst(2, Consts(M), Univ)
        output(addR(y))
        output(addR(z))
        output(assignUser(s, y))
        output(addHierarchy(z, y))
        If ∃ x.(RH(g, x) ∈ M)
            output(addHierarchy(y, x))
        endif
    endif
    for each (p ∈ {p | PA(g, p) ∈ M})
        output(assignPermission(z, p))

liberalAdd(M, o, g)
    output(assignPermission(g, o))

strictRemove(M, o, g)
    output(revokePermission(g, o))
    If ∃ SubA, SubB.({RH(g, SubA), RH(SubA, SubB)} ⊂ M)
        for each (Sup2 ∈ {z | ∃ y.({RH(y, SubA),
                                    RH(z, y)} ⊂ M)})
            output(revokePermission(Sup2, o))
        for each (Orphan ∈ {r | RH(r, SubB) ∈ M} \ {SubA})
            output(revokePermission(Orphan, o))
    endif

liberalRemove(M, o, g)
    If ∃ SubA, SubB.({RH(g, SubA), RH(SubA, SubB)} ⊂ M)
        Orphan = nFreshConst(1, Consts(M), Univ)
        output(addR(Orphan))
        output(addHierarchy(Orphan, SubB))
        for each (u ∈ {u | UR(u, g) ∈ M})
            output(assignUser(u, Orphan))
        output(assignPermission(Orphan, o))
        output(revokePermission(g, o))
    endif
```

This mapping is described in HPL, and is thus homomorphic.

We prove that α preserves σ by showing that, for any PSP state x and label ℓ, $\sigma(next(x,\ell)) = terminal(\sigma(x), \alpha(\sigma(x), \ell))$.

Given PSP state x and label ℓ, $x' = next(x,\ell)$ is the state resulting from executing label ℓ in state x.

- If ℓ is an instance of $addS(s)$, then $x' = x \cup S(s)$. By σ, this maps in $RBAC_1$ to state $\sigma(x') = \sigma(x) \cup$

$U(s)$. By α, $\alpha(\sigma(x), \ell) = addU(s)$. By $RBAC_1$'s $next$ relation, $next(\sigma(x), addU(s)) = \sigma(x) \cup U(s)$. Thus, if ℓ is an instance of $addS(s)$, $\sigma(x') = terminal(\sigma(x), \alpha(\sigma(x), \ell))$.

- If ℓ is an instance of $delS(s)$, then $x' = x \backslash (S(s) \cup Entries(x, s))$, where $Entries(x, s)$ denotes the set of all state tuples in x involving s. By σ, $\sigma(x') = \sigma(x) \backslash (U(s) \cup Entries(\sigma(x), s))$. By α, $\alpha(\sigma(x), \ell) = delU(s)$. By $RBAC_1$'s $next$ relation, $next(\sigma(x), delU(s)) = \sigma(x) \setminus (U(s) \cup Entries(\sigma(x), s))$. Thus, if ℓ is an instance of $delS(s)$, $\sigma(x') = terminal(\sigma(x), \alpha(\sigma(x), \ell))$.
- If ℓ is an instance of $addO(o)$, then $x' = x \cup O(o)$. By σ, $\sigma(x') = \sigma(x) \cup P(o)$. By α, $\alpha(\sigma(x), \ell) = addP(o)$. By $RBAC_1$'s $next$ relation, $next(\sigma(x), addP(o)) = \sigma(x) \cup P(o)$. Thus, if ℓ is an instance of $addP(o)$, $\sigma(x') = terminal(\sigma(x), \alpha(\sigma(x), \ell))$.
- If ℓ is an instance of $delO(o)$, then $x' = x \setminus (O(o) \cup Entries(x, o))$. By σ, $\sigma(x') = \sigma(x) \setminus (P(o) \cup Entries(\sigma(x), o))$. By α, $\alpha(\sigma(x), \ell) = delP(o)$. By $RBAC_1$'s $next$ relation, $next(\sigma(x), delP(o)) = \sigma(x) \setminus (P(o) \cup Entries(\sigma(x), o))$. Thus, if ℓ is an instance of $delO(o)$, $\sigma(x') = terminal(\sigma(x), \alpha(\sigma(x), \ell))$.
- If ℓ is an instance of $liberalJoin(s, g)$, then $x' = x \cup LiberalJoin(s, g, t) \cup Time(t+1) \setminus Time(t)$. If s has not previously strict left g, then by σ, $\sigma(x') = \sigma(x) \cup UR(s, g)$. By α, $\alpha(\sigma(x), \ell) = assignUser(s, g)$. By $RBAC_1$'s $next$ relation, $next(\sigma(x), assignUser(s, g)) = \sigma(x) \cup UR(s, g)$. Thus, $\sigma(x') = terminal(\sigma(x), \alpha(\sigma(x), \ell))$.

  If s has previously strict left g, then by σ, $\sigma(x') = \sigma(x) \cup UR(s, g) \cup UR(s, r_{high}) \backslash UR(s, r_{low})$, where r_{low}, r_{high} is an orphaned rolepair tracking the permissions to be re-granted to s upon re-join. By α, $\alpha(\sigma(x), \ell) = assignUser(s, g) \circ assignUser(s, r_{high}) \circ revokeUser(s, r_{low})$. By $RBAC_1$'s $next$ relation, $terminal(\sigma(x), assignUser(s, g) \circ assignUser(s, r_{high}) \circ revokeUser(s, r_{low}) = \sigma(x) \cup UR(s, g) \cup UR(s, r_{high}) \setminus UR(s, r_{low})$. Thus, $\sigma(x') = terminal(\sigma(x), \alpha(\sigma(x), \ell))$.

  Thus, if ℓ is an instance of $liberalJoin(s, g)$, $\sigma(x') = terminal(\sigma(x), \alpha(\sigma(x), \ell))$.
- If ℓ is an instance of $strictLeave(s, g)$, then $x' = x \cup StrictLeave(s, g, t) \cup Time(t+1) \setminus Time(t)$. If s has not previously strict left g, then by σ, $\sigma(x') = \sigma(x) \setminus UR(s, g) \cup R(r_{low}) \cup R(r_{high}) \cup UR(s, r_{low}) \cup RH(r_{high}, r_{low}) \cup RH(r_{low}, r_{sub}) \cup PA(r_{high}, p_1) \cup \cdots \cup PA(r_{high}, p_k)$, where r_{low}, r_{high} is a newly-created orphaned rolepair tracking the permissions to be re-granted to s upon re-join, r_{sub} is the role below g in the hierarchy, and $p_1, \ldots, p_k$ is the set of permissions currently granted to members of g (the permissions that will be re-granted if s re-joins g). By α, $\alpha(\sigma(x), \ell) = revokeUser(s, g) \circ addR(r_{low}) \circ addR(r_{high}) \circ assignUser(s, r_{low}) \circ addHierarchy(r_{high}, r_{low}) \circ addHierarchy(r_{low}, r_{sub}) \circ assignPermission(r_{high}, p_1) \circ \cdots \circ assignPermission(r_{high}, p_k)$. By $RBAC_1$'s $next$ relation, $terminal(\sigma(x), revokeUser(s, g) \circ addR(r_{low}) \circ addR(r_{high}) \circ assignUser(s, r_{low}) \circ addHierarchy(r_{high}, r_{low}) \circ addHierarchy(r_{low}, r_{sub}) \circ assignPermission(r_{high}, p_1) \circ \cdots \circ assignPermission(r_{high}, p_k) = \sigma(x) \backslash UR(s, g) \cup R(r_{low}) \cup R(r_{high}) \cup UR(s, r_{low}) \cup RH(r_{high}, r_{low}) \cup RH(r_{low}, r_{sub}) \cup PA(r_{high}, p_1) \cup \cdots \cup PA(r_{high}, p_k)$. Thus, $\sigma(x') = terminal(\sigma(x), \alpha(\sigma(x), \ell))$.

  If s has previously strict left g, then by σ, $\sigma(x') = \sigma(x) \setminus UR(s, g) \cup UR(s, r_{low}) \setminus UR(s, r_{high}) \cup PA(r_{high}, p_1) \cup \cdots \cup PA(r_{high}, p_k)$. By α, $\alpha(\sigma(x), \ell) = revokeUser(s, g) \circ assignUser(s, r_{low}) \circ revokeUser(s, r_{high}) \circ assignPermission(r_{high}, p_1) \circ \cdots \circ assignPermission(r_{high}, p_k)$. By $RBAC_1$'s $next$ relation, $terminal(\sigma(x), revokeUser(s, g) \circ assignUser(s, r_{low}) \circ revokeUser(s, r_{high}) \circ assignPermission(r_{high}, p_1) \circ \cdots \circ assignPermission(r_{high}, p_k) = \sigma(x) \setminus UR(s, g) \cup UR(s, r_{low}) \setminus UR(s, r_{high}) \cup PA(r_{high}, p_1) \cup \cdots \cup PA(r_{high}, p_k)$. Thus, $\sigma(x') = terminal(\sigma(x), \alpha(\sigma(x), \ell))$.

  Thus, if ℓ is an instance of $strictLeave(s, g)$, $\sigma(x') = terminal(\sigma(x), \alpha(\sigma(x), \ell))$.
- If ℓ is an instance of $liberalAdd(o, g)$, then $x' = x \cup LiberalAdd(o, g, t) \cup Time(t+1) \setminus Time(t)$. By σ, $\sigma(x') = \sigma(x) \cup PA(g, o)$. By α, $\alpha(\sigma(x), \ell) = assignPermission(g, o)$. By $RBAC_1$'s $next$ relation, $next(\sigma(x), assignPermission(g, o)) = \sigma(x) \cup PA(g, o)$. Thus, if ℓ is an instance of $liberalAdd(o, g)$, $\sigma(x') = terminal(\sigma(x), \alpha(\sigma(x), \ell))$.

- If ℓ is an instance of *strictRemove*(o, g), then $x' = x \cup StrictRemove(o, g, t) \cup Time(t+1) \setminus Time(t)$. By σ, $\sigma(x') = \sigma(x) \setminus (PA(g, o) \cup PA(r_1, o) \cup \cdots \cup PA(r_k, o))$, where $r_1, \ldots, r_k$ is the set of roles r_i connected to g in the role hierarchy such that $PA(r_i, o) \in Th(\sigma(x))$. By α, $\alpha(\sigma(x), \ell) = revokePermission(g, o) \circ revokePermission(r_1, o) \circ \cdots \circ revokePermission(r_k, o)$. By $RBAC_1$'s *next* relation, $next(\sigma(x), revokePermission(g, o) \circ revokePermission(r_1, o) \circ \cdots \circ revokePermission(r_k, o)) = \sigma(x) \setminus (PA(g, o) \cup PA(r_1, o) \cup \cdots \cup PA(r_k, o))$. Thus, if ℓ is an instance of *strictRemove*(o, g), $\sigma(x') = terminal(\sigma(x), \alpha(\sigma(x), \ell))$.
- If ℓ is an instance of *liberalRemove*(o, g), then $x' = x \cup LiberalRemove(o, g, t) \cup Time(t+1) \setminus Time(t)$. By σ, $\sigma(x') = \sigma(x) \cup R(r_{orphan}) \cup RH(r_{orphan}, r_{subb}) \cup UR(u_1, r_{orphan}) \cup \cdots \cup UR(u_k, r_{orphan}) \cup PA(r_{orphan}, o) \setminus PA(g, o)$, where r_{orphan} is a newly-created orphan role to permit users to retain access to o, r_{subb} is the second role below g in the role hierarchy, and $u_1, \ldots, u_k$ is the set of users currently in g (e.g., $\{u_i \mid UR(u_i, g) \in Th(\sigma(x))\}$). By α, $\alpha(\sigma(x), \ell) = addR(r_{orphan}) \circ addHierarchy(r_{orphan}, r_{subb}) \circ assignUser(u_1, r_{orphan}) \circ \cdots \circ assignUser(u_k, r_{orphan}) \circ assignPermission(r_{orphan}, o) \circ revokePermission(g, o)$. By $RBAC_1$'s *next* relation, $next(\sigma(x), addR(r_{orphan}) \circ addHierarchy(r_{orphan}, r_{subb}) \circ assignUser(u_1, r_{orphan}) \circ \cdots \circ assignUser(u_k, r_{orphan}) \circ assignPermission(r_{orphan}, o) \circ revokePermission(g, o)) = \sigma(x) \cup R(r_{orphan}) \cup RH(r_{orphan}, r_{subb}) \cup UR(u_1, r_{orphan}) \cup \cdots \cup UR(u_k, r_{orphan}) \cup PA(r_{orphan}, o) \setminus PA(g, o)$. Thus, if ℓ is an instance of *liberalRemove*(o, g), $\sigma(x') = terminal(\sigma(x), \alpha(\sigma(x), \ell))$.

Thus, for any PSP state x and label ℓ, $\sigma(next(x, \ell)) = terminal(\sigma(x), \alpha(\sigma(x), \ell))$. Thus, we have shown that α preserves σ.

Finally, α is safe by inspection—for any PSP state x and label ℓ, the sequence of $RBAC_1$ labels $\alpha(\sigma(x), \ell)$ never revokes or grants authorizations except the images of those that are revoked or granted by ℓ.

Thus, we have shown that α preserves σ, is homomorphic, and preserves safety; that σ preserves π and is homomorphic; and that π is weak AC-preserving and homomorphic.

$\therefore$ $\langle \alpha, \sigma, \pi \rangle$ is an implementation of PSP in $RBAC_1$ which preserves correctness, weak AC-preservation, homomorphism, and safety. □

5.3 Reduction-Derived Implementations

Corollary 23 *$RBAC_0$ admits a correct, weak AC-preserving implementation of PC.*

Proof Follows directly from Theorems 13 and 21. □

Corollary 24 *ugo admits a correct, weak AC-preserving implementation of PC.*

Proof Follows directly from Theorem 14 and Corollary 23. □

Corollary 25 *$RBAC_0$ admits a correct, weak AC-preserving implementation of PSP.*

Proof Follows directly from Theorems 13 and 22. □

Corollary 26 *ugo admits a correct, weak AC-preserving implementation of PSP.*

Proof Follows directly from Theorem 14 and Corollary 25. □

6 Infeasible Reductions

6.1 $RBAC_1$ and $RBAC_0$

Theorem 27 *There exists a reduction $\langle \sigma, \pi \rangle$ from $RBAC_1$ to $RBAC_0$ where:*

- *σ preserves π, is injective, preserves reachability, and is homomorphic*
- *π is AC-preserving and homomorphic*

Thus, $RBAC_1 \leq^{CAH} RBAC_0$ ($RBAC_0$ is at least as expressive as $RBAC_1$ with respect to correctness, AC-preservation, and homomorphism).

PROOF We present the reduction, $\langle \sigma, \pi \rangle$. First, σ maps the $RBAC_1$ state $\langle U, R, P, UR, PA, RH \rangle$ to the $RBAC_0$ state $\langle U, R, P, UR, PA \rangle$.

```
sigma(M)
    Let U = {u | U(u) ∈ M} ∪ {r | R(r) ∈ M}
    Let R = {r | R(r) ∈ M}
    Let P = {p | P(p) ∈ M} ∪ {r | R(r) ∈ M}
    Let UR = {}
    Let PA = {}
    Let Skolems = {}

    EncodeUr(M, Skolems, U, R, UR)
    EncodePa(M, Skolems, R, P, PA)
    EncodeRh(M, Skolems, R, P, PA)
    EncodeAuth(Consts(M), Skolems, U, R, P, UR, PA)

    outputSet(U ∪ R ∪ P ∪ UR ∪ PA)

EncodeUr(M, Skolems, U, R, UR)
    for each (<u, r> ∈ {<u, r> | UR(u, r) ∈ M})
        StoreUr(Consts(M), Skolems, U, R, UR, u, r)

StoreUr(Constants, Skolems, U, R, UR, u, r)
   <z, y> = nFreshConst(2, Constants ∪ Skolems, Univ)
   Skolems = Skolems ∪ {z, y}
   U = U ∪ {z}
   R = R ∪ {z, y}
   UR = UR ∪ {<z, y>, <u, z>, <r, y>}

EncodePa(M, Skolems, R, P, PA)
    for each (<r, p> ∈ {<r, p> | PA(r, p) ∈ M})
        StorePa(Consts(M), Skolems, R, P, PA, r, p)

StorePa(Constants, Skolems, R, P, PA, r, p)
   <z, y> = nFreshConst(2, Constants ∪ Skolems, Univ)
   Skolems = Skolems ∪ {z, y}
   R = R ∪ {z, y}
   P = P ∪ {y}
   PA = PA ∪ {<z, y>, <z, r>, <y, p>}

EncodeRh(M, Skolems, R, P, PA)
    for each (<s, j> ∈ {<s, j> | RH(s, j) ∈ M})
        StoreRh(Consts(M), Skolems, R, P, PA, s, j)

StoreRh(Constants, Skolems, R, P, PA, s, j)
   <z, y, x> = nFreshConst(3, Constants ∪ Skolems, Univ)
```

```
    Skolems = Skolems ∪ {z, y, x}
    R = R ∪ {z, y, x}
    P = P ∪ {y, x}
    PA = PA ∪ {<z, y>, <y, x>, <z, s>, <x, j>}

EncodeAuth(Constants, Skolems, U, R, P, UR, PA)
    for each (u ∈ {u | u ∈ U \ R})
        for each (p ∈ {p | p ∈ P \ R})
            If Authorized(UR, PA, u, p)
                StoreAuth(Constants, Skolems, R, UR, PA, u,
                          p)
            endif

StoreAuth(Constants, Skolems, R, UR, PA, u, p)
    z = nFreshConst(1, Constants ∪ Skolems, Univ)
    Skolems = Skolems ∪ {z}
    R = R ∪ {z}
    UR = UR ∪ {<u, z>}
    PA = PA ∪ {<z, p>}

Authorized(UR, PA, u, p)
    If ∃ z,y,x,w,r.({<z, y>, <u, z>, <r, y>} ⊆ UR ∧
                    {<x, w>, <x, r>, <w, p>} ⊆ PA)
        return TRUE
    else If ∃ z,y,x,w,s,j.({<z, y>, <u, z>, <s, y>} ⊆ UR ∧
                           {<x, w>, <x, j>, <w, p>} ⊆ PA ∧
                           Senior(PA, s, j))
        return TRUE
    else
        return FALSE
    endif

Senior(PA, s, j)
    If ∃ z,y,x.({<z, y>, <y, x>, <z, s>, <x, j>} ⊆ PA)
        return TRUE
    else If ∃ r.(Senior(PA, s, r) ∧ Senior(PA, r, j))
        return TRUE
    else
        return FALSE
    endif
```

As this mapping is described in HPL, it is homomorphic. It is also injective, since no two $RBAC_1$ states will map to the same $RBAC_0$ state: the entirety of the $RBAC_1$ state is encoded in (and can be uniquely extracted from) the $RBAC_0$ state.

The query mapping, π, is defined as follows.

```
UR(T, u, r)
    If ∃ x.(UR(x, u) ∈ T ∨ UR(x, r) ∈ T)
        return FALSE
    else If ∃ z,y.(UR(z, y) ∈ T ∧ UR(u, z) ∈ T ∧
                   UR(r, y) ∈ T)
        return TRUE
```

```
    else
       return FALSE
    endif

PA(T, r, p)
    If ∃ x.(PA(r, x) ∈ T ∨ PA(p, x) ∈ T)
       return FALSE
    else If ∃ z,y.(PA(z, y) ∈ T ∧ PA(z, r) ∈ T ∧
                   PA(y, p) ∈ T)
       return TRUE
    else
       return FALSE
    endif

R(T, r)
    If ∃ x.(UR(x, r) ∈ T ∨ PA(r, x) ∈ T)
       return FALSE
    else If R(r) ∈ T
       return TRUE
    else
       return FALSE
    endif

RH(T, s, j)
    If ∃ x.(PA(s, x) ∈ T ∨ PA(j, x) ∈ T)
       return FALSE
    If ∃ z,y,x.(PA(z, y) ∈ T ∧ PA(y, x) ∈ T ∧
                PA(z, s) ∈ T ∧ PA(x, j) ∈ T)
       return TRUE
    else
       return FALSE
    endif

Senior(T, s, j)
    If RH(T, s, j)
       return TRUE
    else If ∃ r.(Senior(T, s, r) ∧ Senior(T, r, j))
       return TRUE
    else
       return FALSE
    endif

auth(T, u, p)
    If auth(u, p) ∈ T
       return TRUE
    else
       return FALSE
    endif
```

This query mapping is described in HPL and is thus homomorphic. It is also AC-preserving since it maps authorization query $auth(r)$ to TRUE for theory T exactly when T contains $auth(r)$.

We show that σ preserves π (for any $RBAC_1$ state x, $Th(x) = \pi(Th(\sigma(x)))$) by contradiction.

Assume that there is some $RBAC_1$ state x and query q such that the value of q in x is the opposite of the value of $\pi(q)$ in $\sigma(x)$. We show that, for each of the query forms of $RBAC_1$, this assumption leads to contradiction.

- **UR** Assume $x \vdash UR(u,r)$ and $\sigma(x) \nvdash \pi(UR(u,r))$. Then, $UR(u,r) \in Th(x)$, and by σ, $\exists z,y.(UR(z,y) \in Th(\sigma(x)) \wedge UR(u,z) \in Th(\sigma(x)) \wedge UR(r,y) \in Th(\sigma(x)))$. Since σ only stores tuples in UR in which the second element is a skolem, $\forall w.(UR(w,u) \notin Th(\sigma(x)) \wedge UR(w,r) \notin Th(\sigma(x)))$. By π, $\pi_{UR(u,r)}(Th(\sigma(x))) =$ TRUE, which is a contradiction on the assumption that $\sigma(x) \nvdash \pi(UR(u,r))$.

  Assume instead that $x \nvdash UR(u,r)$ and $\sigma(x) \vdash \pi(UR(u,r))$. Then, $UR(u,r) \notin Th(x)$. By σ, either $\forall z,y.(UR(z,y) \notin Th(\sigma(x)) \vee UR(u,z) \notin Th(\sigma(x)) \vee UR(r,y) \notin Th(\sigma(x)))$, or one or both of u and r is a skolem value, in which case $\exists w.(UR(w,u) \in Th(\sigma(x)) \vee UR(w,r) \in Th(\sigma(x)))$. In either case, by π, $\pi_{UR(u,r)}(Th(\sigma(x))) =$ FALSE, which is a contradiction on the assumption that $\sigma(x) \vdash \pi(UR(u,r))$.

- **PA** Assume $x \vdash PA(r,p)$ and $\sigma(x) \nvdash \pi(PA(r,p))$. Then, $PA(r,p) \in Th(x)$, and by σ, $\exists z,y.(PA(z,y) \in Th(\sigma(x)) \wedge PA(z,r) \in Th(\sigma(x)) \wedge PA(y,p) \in Th(\sigma(x)))$. Since σ only stores tuples in PA in which the first element is a skolem, $\forall w.(PA(r,w) \notin Th(\sigma(x)) \wedge PA(p,w) \notin Th(\sigma(x)))$. By π, $\pi_{PA(r,p)}(Th(\sigma(x))) =$ TRUE, which is a contradiction on the assumption that $\sigma(x) \nvdash \pi(PA(r,p))$.

  Assume instead that $x \nvdash PA(r,p)$ and $\sigma(x) \vdash \pi(PA(r,p))$. Then, $PA(r,p) \notin Th(x)$. By σ, either $\forall z,y.(PA(z,y) \notin Th(\sigma(x)) \vee PA(z,r) \notin Th(\sigma(x)) \vee PA(y,p) \notin Th(\sigma(x)))$, or one or both of r and p is a skolem value, in which case $\exists w.(PA(r,w) \in Th(\sigma(x)) \vee PA(p,w) \in Th(\sigma(x)))$. In either case, by π, $\pi_{PA(r,p)}(Th(\sigma(x))) =$ FALSE, which is a contradiction on the assumption that $\sigma(x) \vdash \pi(PA(r,p))$.

- **R** Assume $x \vdash R(r)$ and $\sigma(x) \nvdash \pi(R(r))$. Then, $R(r) \in Th(x)$, and by σ, $R(r) \in Th(\sigma(x))$. By σ's storage of skolems in UR and PA, $\forall w.(UR(w,r) \notin Th(\sigma(x)) \wedge PA(r,w) \notin Th(\sigma(x)))$. By π, $\pi_{R(r)}(Th(\sigma(x))) =$ TRUE, which is a contradiction on the assumption that $\sigma(x) \nvdash \pi(R(r))$.

  Assume instead that $x \nvdash R(r)$ and $\sigma(x) \vdash \pi(R(r))$. Then, $R(r) \notin Th(x)$, and by σ, either $R(r) \notin Th(\sigma(x))$, or r is a skolem, in which case $\exists w.(UR(w,r) \in Th(\sigma(x)) \vee PA(r,w) \in Th(\sigma(x)))$. In either case, by π, $\pi_{R(r)}(Th(\sigma(x))) =$ FALSE, which is a contradiction on the assumption that $\sigma(x) \vdash \pi(R(r))$.

- **RH** Assume $x \vdash RH(s,j)$ and $\sigma(x) \nvdash \pi(RH(s,j))$. Then, $RH(s,j) \in Th(x)$, and by σ, $\exists z,y,w.(PA(z,y) \in Th(\sigma(x)) \wedge PA(y,w) \in Th(\sigma(x)) \wedge PA(z,s) \in Th(\sigma(x)) \wedge PA(w,j) \in Th(\sigma(x)))$. By σ's storage of skolems in PA, $\forall v.(PA(s,v) \notin Th(\sigma(x)) \wedge PA(j,v) \notin Th(\sigma(x)))$. By π, $\pi_{RH(s,j)}(Th(\sigma(x))) =$ TRUE, which is a contradiction on the assumption that $\sigma(x) \nvdash \pi(RH(s,j))$.

  Assume instead that $x \nvdash RH(s,j)$ and $\sigma(x) \vdash \pi(RH(s,j))$. Then, $RH(s,j) \notin Th(x)$. By σ, either $\forall z,y,w.(PA(z,y) \notin Th(\sigma(x)) \vee PA(y,w) \notin Th(\sigma(x)) \vee PA(z,s) \notin Th(\sigma(x)) \vee PA(w,j) \notin Th(\sigma(x)))$, or one or both of s and j is a skolem value, in which case $\exists v.(PA(s,v) \in Th(\sigma(x)) \vee PA(j,v) \in Th(\sigma(x)))$. In either case, by π, $\pi_{RH(s,j)}(Th(\sigma(x))) =$ FALSE, which is a contradiction on the assumption that $\sigma(x) \vdash \pi(RH(s,j))$.

- **Senior** Assume $x \vdash Senior(s,j)$ and $\sigma(x) \nvdash \pi(Senior(s,j))$. Then, there is some sequence of roles r_i such that $RH(s,r_1) \in Th(x) \wedge RH(r_1,r_2) \in Th(x) \wedge \cdots \wedge RH(r_k,j) \in Th(x)$. By the previous point, σ preserves π_{RH}, so $RH(s,r_1) \in Th(\sigma(x)) \wedge RH(r_1,r_2) \in Th(\sigma(x)) \wedge \cdots \wedge RH(r_k,j) \in Th(\sigma(x))$. By π, $\pi_{Senior(s,j)}(Th(\sigma(x))) =$ TRUE, which is a contradiction on the assumption that $\sigma(x) \nvdash \pi(Senior(s,j))$.

  Assume instead that $x \nvdash Senior(s,j)$ and $\sigma(x) \vdash \pi(Senior(s,j))$. Then, there is no sequence of roles r_i such that $RH(s,r_1) \in Th(x) \wedge RH(r_1,r_2) \in Th(x) \wedge \cdots \wedge RH(r_k,j) \in Th(x)$. Since σ preserves π_{RH}, there is also no sequence of roles r_i such that $RH(s,r_1) \in Th(\sigma(x)) \wedge RH(r_1,r_2) \in$

$Th(\sigma(x)) \wedge \cdots \wedge RH(r_k, j) \in Th(\sigma(x))$ By π, $\pi_{Senior(s,j)}(Th(\sigma(x))) = $ FALSE, which is a contradiction on the assumption that $\sigma(x) \vdash \pi(Senior(s, j))$.

- **auth** Assume $x \vdash auth(u, p)$ and $\sigma(x) \nvdash \pi(auth(u, p))$. Then, either $\exists r.(UR(u, r) \in Th(x) \wedge PA(r, p) \in Th(x))$ (u belongs to a role which is authorized to p) or $\exists s, j.(UR(u, s) \in Th(x) \wedge PA(j, p) \in Th(x) \wedge Senior(s, j) \in Th(x))$ (u belongs to a role senior to a role that is authorized to p). In either case, the HPL procedure Authorized() from σ's specification will return TRUE for u, p. Thus, $\exists z.(UR(u, z) \in Th(\sigma(x)) \wedge PA(z, p) \in Th(\sigma(x)))$, and by π, $\pi_{auth(u,p)}(Th(\sigma(x))) = $ TRUE, which is a contradiction on the assumption that $\sigma(x) \nvdash \pi(auth(u, p))$.

  Assume instead that $x \nvdash auth(u, p)$ and $\sigma(x) \vdash \pi(auth(u, p))$. Then, there is no sequence of roles r_i such that $(RH(r_1, r_2), RH(r_2, r_3), \ldots, RH(r_{k-1}, r_k)) \in Th(x)$ and $UR(u, r_1) \in Th(x) \wedge PA(r_k, p) \in Th(x)$. Thus, by σ, Authorized() will return FALSE for u, p, and thus StoreAuth() will not be called for this pair. Since skolem values are not reused, and this is the only procedure that stores the same skolem in both UR and PA, there is no r such that $UR(u, r) \in Th(\sigma(x)) \wedge PA(r, p) \in Th(\sigma(x))$. By π, $\pi_{auth(u,p)}(Th(\sigma(x))) = $ FALSE, which is a contradiction on the assumption that $\sigma(x) \vdash \pi(auth(u, p))$.

Thus, by contradiction, σ preserves π.

We prove that σ preserves reachability by induction by showing that, for any $RBAC_1$ state x and label ℓ, $\sigma(next(x, \ell))$ is reachable from $\sigma(x)$ via $RBAC_0$ labels.

Given $RBAC_1$ state x and label ℓ, let $x' = next(x, \ell)$ by the state resulting from executing label ℓ in state x.

- If ℓ is an instance of $addU(u)$, then $x' = next(x, \ell) = x \cup U(u)$. By σ, this maps in $RBAC_0$ to state $\sigma(x') = \sigma(x) \cup U(u)$. By $RBAC_0$'s $next$ relation, $next(\sigma(x), addU(u)) = \sigma(x) \cup U(u)$. Thus, if ℓ is an instance of $addU(u)$, $\sigma(x')$ is reachable from $\sigma(x)$ via execution of $addU(u)$. A similar argument holds for instances of $addP(p)$, with reachability in $RBAC_0$ via $addP(p)$.
- If ℓ is an instance of $delU(u)$, then $x' = x \setminus (U(u) \cup Entries(x, u))$, where $Entries(x, u)$ denotes the set of all state tuples in x involving u. By σ, $\sigma(x') = \sigma(x) \setminus (U(u) \cup UEntries(\sigma(x), u))$, where $UEntries(\sigma(x), u)$ denotes a connected set of state tuples in $\sigma(x)$ representing u:
  - Where $\exists z, y.(UR(z, y) \wedge UR(u, z) \wedge UR(r, y))$, all three entries are removed, as well as $U(z)$, $R(y)$, and $R(z)$.
  - Where $\exists w, p.(UR(u, w) \wedge PA(w, p))$, both are removed, as well as $R(w)$.

  By $RBAC_0$'s $next$ relation, $terminal(\sigma(x), delU(u) \circ \{delU(z_i) \circ delR(y_i) \circ delR(z_i)\} \circ \{delR(w_i)\}) = \sigma(x) \setminus (U(u) \cup UEntries(\sigma(x), u))$ (where subsequences surrounded in {} are repeated as necessary). Note that explicitly executing $revokeUser$ and $revokePermission$ labels to remove UR and PA relations is not necessary since deleting the skolems will remove any entries containing them (by typing). Thus, if ℓ is an instance of $delU(u)$, $\sigma(x')$ is reachable from $\sigma(x)$.
- If ℓ is an instance of $delP(p)$, then $x' = next(x, \ell) = x \setminus P(p)$. By σ, this maps in $RBAC_0$ to state $\sigma(x') = \sigma(x) \setminus (P(p) \cup PEntries(\sigma(x), p))$, where $PEntries(\sigma(x), u)$ denotes a connected set of state tuples in $\sigma(x)$ representing p:
  - Where $\exists z, y.(PA(z, y) \wedge PA(z, r) \wedge PA(y, p))$, all three entries are removed, as well as $R(z)$, $P(y)$, and $R(y)$.
  - Where $\exists u, w.(UR(u, w) \wedge PA(w, p))$, both are removed, as well as $R(w)$.

  By $RBAC_0$'s $next$ relation, $terminal(\sigma(x), delP(p) \circ \{delR(z_i) \circ delP(y_i) \circ delR(y_i)\} \circ \{delR(w_i)\}) = \sigma(x) \setminus (P(p) \cup PEntries(\sigma(x), p))$ (again repeating subsequences surrounded in {} as necessary). Thus, if ℓ is an instance of $delP(p)$, $\sigma(x')$ is reachable from $\sigma(x)$.

- If ℓ is an instance of $addR(r)$, then $x' = next(x, \ell) = x \cup R(r)$. By σ, this maps in $RBAC_0$ to state $\sigma(x') = \sigma(x) \cup (U(r) \cup R(r) \cup P(r))$. By $RBAC_0$'s *next* relation, $terminal(\sigma(x), addU(r) \circ addR(r) \circ addP(r)) = \sigma(x) \cup (U(r) \cup R(r) \cup P(r))$. Thus, if ℓ is an instance of $addR(r)$, $\sigma(x')$ is reachable from $\sigma(x)$ via execution of $addU(r)$, $addR(r)$, and $addP(r)$.
- If ℓ is an instance of $delR(r)$, then $x' = x \setminus (R(r) \cup Entries(x, r))$. By σ, $\sigma(x') = \sigma(x) \setminus (U(r) \cup R(r) \cup P(r) \cup REntries(\sigma(x), r))$, where $REntries(\sigma(x), r)$ denotes a connected set of state tuples in $\sigma(x)$ representing r:
  - Where $\exists z, y.(UR(z, y) \land UR(u, z) \land UR(r, y))$, all three are removed, as well as $U(z)$, $R(y)$, and $R(z)$.
  - Where $\exists z, y.(PA(z, y) \land PA(z, r) \land PA(y, p))$, all three are removed, as well as $R(z)$, $P(y)$, and $R(y)$.
  - Where $\exists z, y, w, j.(PA(z, y) \land PA(y, w) \land PA(z, r) \land PA(w, j))$, all four are removed, as well as $R(z)$, $P(y)$, $R(y)$, $P(w)$, and $R(w)$.
  - Where $\exists z, y, w, s.(PA(z, y) \land PA(y, w) \land PA(z, s) \land PA(w, r))$, all four are removed, as well as $R(z)$, $P(y)$, $R(y)$, $P(w)$, and $R(w)$.

  By $RBAC_0$'s *next* relation, $terminal(\sigma(x), delU(r) \circ delR(r) \circ delP(r) \circ \{delU(z_i) \circ delR(y_i) \circ delR(z_i)\} \circ \{delR(z_i) \circ delP(y_i) \circ delR(y_i)\} \circ \{delR(z_i) \circ delP(y_i) \circ delR(y_i) \circ delP(w_i) \circ delR(w_i)\}) = \sigma(x) \setminus (U(r) \cup R(r) \cup P(r) \cup REntries(\sigma(x), r))$ (repeating subsequences as needed). Thus, if ℓ is an instance of $delR(r)$, $\sigma(x')$ is reachable from $\sigma(x)$.
- If ℓ is an instance of $assignUser(u, r)$, then $x' = x \cup UR(u, r)$. By σ, $\sigma(x') = \sigma(x) \cup (U(z) \cup R(z) \cup R(y) \cup UR(z, y) \cup UR(u, z) \cup UR(r, y) \cup Accesses(\sigma(x), u, r))$, where z and y are skolem values. Here, $Accesses(\sigma(x), u, r)$ denotes the set of state tuples needed to grant u access to the permissions implied by membership in r (skipping those that are already granted by other role assignments). That is, for each permission p_i gained by u by being assigned to r, the following are added to the state: $(R(w_i) \cup UR(u, w_i) \cup PA(w_i, p_i))$, where each w_i is a skolem value. By $RBAC_0$'s *next* relation, $terminal(\sigma(x), addU(z) \circ addR(z) \circ addR(y) \circ assignUser(z, y) \circ assignUser(u, z) \circ assignUser(r, y) \circ \{addR(w_i) \circ assignUser(u, w_i) \circ assignPermission(w_i, p_i)\}) = \sigma(x) \cup (U(z) \cup R(z) \cup R(y) \cup UR(z, y) \cup UR(u, z) \cup UR(r, y) \cup Accesses(\sigma(x), u, r))$ (repeating subsequences as needed). Thus, if ℓ is an instance of $assignUser(u, r)$, $\sigma(x')$ is reachable from $\sigma(x)$.
- If ℓ is an instance of $revokeUser(u, r)$, then $x' = x \setminus UR(u, r)$. By σ, $\sigma(x') = \sigma(x) \setminus (U(z) \cup R(z) \cup R(y) \cup UR(z, y) \cup UR(u, z) \cup UR(r, y) \cup Accesses(\sigma(x), u, r))$, where $Accesses(\sigma(x), u, r)$ is again the set of tuples representing the granting of permissions to u because of membership in role r, and z and y (and each w_i implicit in *Accesses*) are whatever (skolem) values cause these tuples to exist in the state (e.g., for p_i, whatever w_i such that $\sigma(x) \vdash UR(u, w_i) \land PA(w_i, p_i)$). By $RBAC_0$'s *next* relation, $terminal(\sigma(x), delU(z) \circ delR(z) \circ delR(y) \circ \{delR(w_i)\}) = \sigma(x) \setminus (U(z) \cup R(z) \cup R(y) \cup UR(z, y) \cup UR(u, z) \cup UR(r, y) \cup Accesses(\sigma(x), u, r))$ (repeating subsequences as needed). Thus, if ℓ is an instance of $revokeUser(u, r)$, $\sigma(x')$ is reachable from $\sigma(x)$.
- If ℓ is an instance of $assignPermission(r, p)$, then $x' = x \cup PA(r, p)$. By σ, $\sigma(x') = \sigma(x) \cup (R(z) \cup R(y) \cup P(y) \cup PA(z, y) \cup PA(z, r) \cup PA(y, p) \cup Accessors(\sigma(x), p, r))$, where z and y are skolem values. $Accessors(\sigma(x), p, r)$ denotes the set of state tuples needed to grant access to p to members of r and senior roles (skipping those that are already granted by other role assignments). For each user u_i that gains access to p by it being assigned to r, the following are added to the state: $(R(w_i) \cup UR(u_i, w_i) \cup PA(w_i, p))$, where each w_i is a skolem value. By $RBAC_0$'s *next* relation, $terminal(\sigma(x), addR(z) \circ addR(y) \circ addP(y) \circ assignPermission(z, y) \circ assignPermission(z, r) \circ assignPermission(y, p) \circ \{addR(w_i) \circ assignUser(u_i, w_i) \circ assignPermission(w_i, p)\}) = \sigma(x) \cup (R(z) \cup R(y) \cup P(y) \cup PA(z, y) \cup PA(z, r) \cup PA(y, p) \cup Accessors(\sigma(x), p, r))$ (repeating subsequences as needed). Thus, if ℓ is an instance of $assignPermission(r, p)$, $\sigma(x')$ is reachable from $\sigma(x)$.

- If ℓ is an instance of $revokePermission(r,p)$, then $x' = x \setminus PA(r,p)$. By σ, $\sigma(x') = \sigma(x) \setminus (R(z) \cup R(y) \cup P(y) \cup PA(z,y) \cup PA(z,r) \cup PA(y,p) \cup Accessors(\sigma(x),p,r))$, for whatever values for z and y (and each w_i implicit in $Accessors$) cause these tuples to exist in the state. By $RBAC_0$'s $next$ relation, $terminal(\sigma(x), delR(z) \circ delR(y) \circ delP(y) \circ \{delR(w_i)\}) = \sigma(x) \setminus (R(z) \cup R(y) \cup P(y) \cup PA(z,y) \cup PA(z,r) \cup PA(y,p) \cup Accessors(\sigma(x),p,r))$ (repeating subsequences as needed). Thus, if ℓ is an instance of $revokePermission(r,p)$, $\sigma(x')$ is reachable from $\sigma(x)$.
- If ℓ is an instance of $addHierarchy(s,j)$, then $x' = x \cup RH(s,j)$. By σ, $\sigma(x') = \sigma(x) \cup (R(z) \cup R(y) \cup R(w) \cup P(y) \cup P(w) \cup PA(z,y) \cup PA(y,w) \cup PA(z,s) \cup PA(w,j) \cup CascadePerms(\sigma(x),s,j))$, where z, y, and w are skolem values. $CascadePerms(\sigma(x),s,j)$ denotes the set of state tuples needed to grant authorizations as a result of s now being a senior role of j. For each user u_i that gains access to permission p_i via this action, the following are added to the state: $(R(v_i) \cup UR(u_i,v_i) \cup PA(v_i,p_i))$, where v_i is a skolem value. By $RBAC_0$'s $next$ relation, $terminal(\sigma(x), addR(z) \circ addR(y) \circ addR(w) \circ addP(y) \circ addP(w) \circ assignPermission(z,y) \circ assignPermission(y,w) \circ assignPermission(z,s) \circ assignPermission(w,j) \circ \{addR(v_i) \circ assignUser(u_i,v_i) \circ assignPermission(v_i,p_i)\}) = \sigma(x')$ (repeating subsequences as needed). Thus, if ℓ is an instance of $addHierarchy(s,j)$, $\sigma(x')$ is reachable from $\sigma(x)$.
- If ℓ is an instance of $removeHierarchy(s,j)$, then $x' = x \setminus RH(s,j)$. By σ, $\sigma(x') = \sigma(x) \setminus (R(z) \cup R(y) \cup R(w) \cup P(y) \cup P(w) \cup PA(z,y) \cup PA(y,w) \cup PA(z,s) \cup PA(w,j) \cup CascadePerms(\sigma(x),s,j))$, for whatever values for z, y, and w (and each v_i in $CascadePerms$) cause these tuples to exist in the state. By $RBAC_0$'s $next$ relation, $terminal(\sigma(x), delR(z) \circ delR(y) \circ delR(w) \circ delP(y) \circ delP(y) \circ delP(w) \circ \{delR(v_i)\} = \sigma(x')$ (repeating subsequences as needed). Thus, if ℓ is an instance of $removeHierarchy(s,j)$, $\sigma(x')$ is reachable from $\sigma(x)$.

Thus, for any $RBAC_1$ state x and label ℓ, $\sigma(next(x,\ell))$ is reachable from $\sigma(x)$ via $RBAC_0$ labels. By induction, for any $RBAC_1$ states s and s', if s' is reachable from s, then $\sigma(s')$ is reachable from $\sigma(s)$. Thus, we have shown that σ preserves reachability.

Thus, we have shown that σ preserves π, is injective, preserves reachability, and is homomorphic; and that π is AC-preserving and homomorphic.

$\therefore$ $\langle \sigma, \pi \rangle$ is a reduction from $RBAC_1$ to $RBAC_0$ which shows $RBAC_1 \leq^{CAH} RBAC_0$. □

Corollary 28 *$RBAC_0$ and $RBAC_1$ are equivalent in expressiveness with respect to correctness, AC-preservation, and homomorphism.*

PROOF Follows directly from Proposition 12 and Theorem 27. □

6.2 Reductions by Transitivity

Corollary 29 $tgSIS \leq^{CaH} RBAC_0$

PROOF Follows directly from Theorems 10 and 27 and Propositions 5 and 6. □

Corollary 30 $bgSIS \leq^{CaH} RBAC_0$

PROOF Follows directly from Theorems 11 and 27 and Propositions 5 and 6. □

References

[1] Timothy L. Hinrichs, Diego Martinoia, William C. Garrison III, Adam J. Lee, Alessandro Panebianco, and Lenore Zuck. Application-sensitive access control evaluation using parameterized expressiveness. In *Proceedings of the 26th IEEE Computer Security Foundations Symposium (CSF)*, June 2013.

[2] Ravi S. Sandhu. Rationale for the RBAC96 family of access control models. In *ACM Workshop on Role-Based Access Control*, 1995.