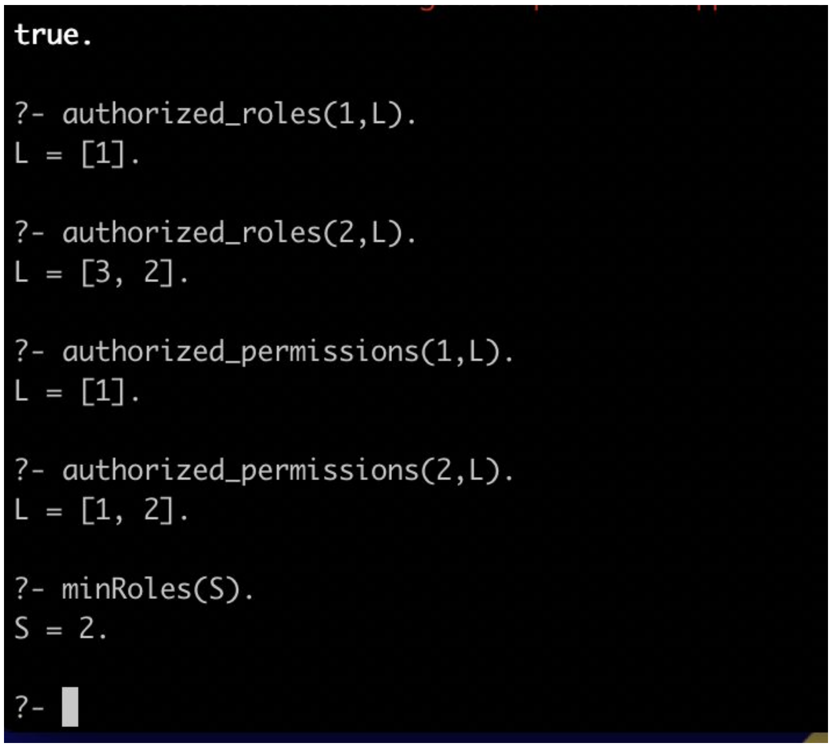
CSE 537
ARTIFICIAL INTELLIGENCE

Project 3
Role Based Access Control
Prolog

REPORT

Submitted on 31st September 2018



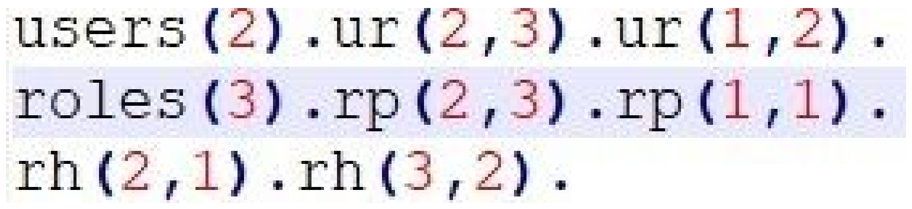
```
true.

?- authorized_roles(1,L).
L = [1].

?- authorized_roles(2,L).
L = [3, 2].

?- authorized_permissions(1,L).
L = [1].

?- authorized_permissions(2,L).
L = [1, 2].

?- minRoles(S).
S = 2.

?- 
```

Presented By:

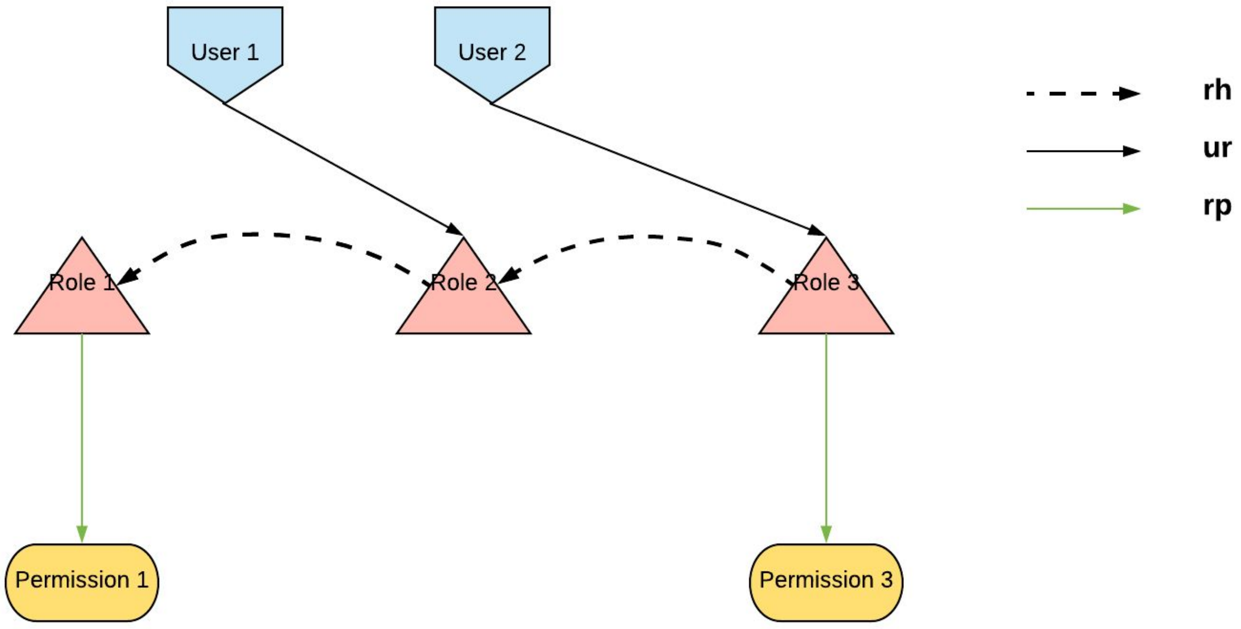
Debjyoti Roy
112070373

Siddarth Harinarayanan
112026390

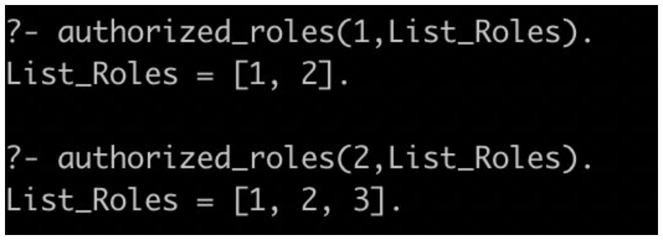
Example 1:



```
users(2).ur(2,3).ur(1,2).
roles(3).rp(2,3).rp(1,1).
rh(2,1).rh(3,2).
```



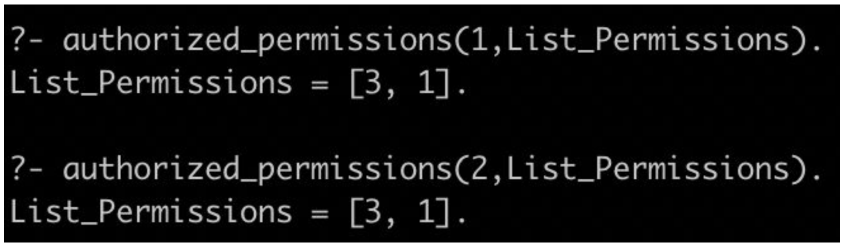
Output 1: authorized_roles(User,List_Roles) :-



```
?- authorized_roles(1,List_Roles).
List_Roles = [1, 2].

?- authorized_roles(2,List_Roles).
List_Roles = [1, 2, 3].
```

From the above architecture, user(1) has direct roles as 2 and 3. Role 2 has a hierarchy of Role 1. Thus overall roles of the user(1) are roles 1,2. Similarly we get the output for User 2 as shown in the figure above.

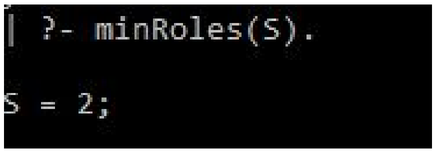
Output 2: authorized_permissions(User, List_Permissions) :-



```
?- authorized_permissions(1,List_Permissions).
List_Permissions = [3, 1].

?- authorized_permissions(2,List_Permissions).
List_Permissions = [3, 1].
```

From authorized_roles predicate we get the list of roles for the user considering the role hierarchy. For instance, user 1 has the roles 1,2. With this data we get the permissions as 3 (rp(2,3)) and 1. Similarly we get the permissions for user 2 as shown in the figure above.

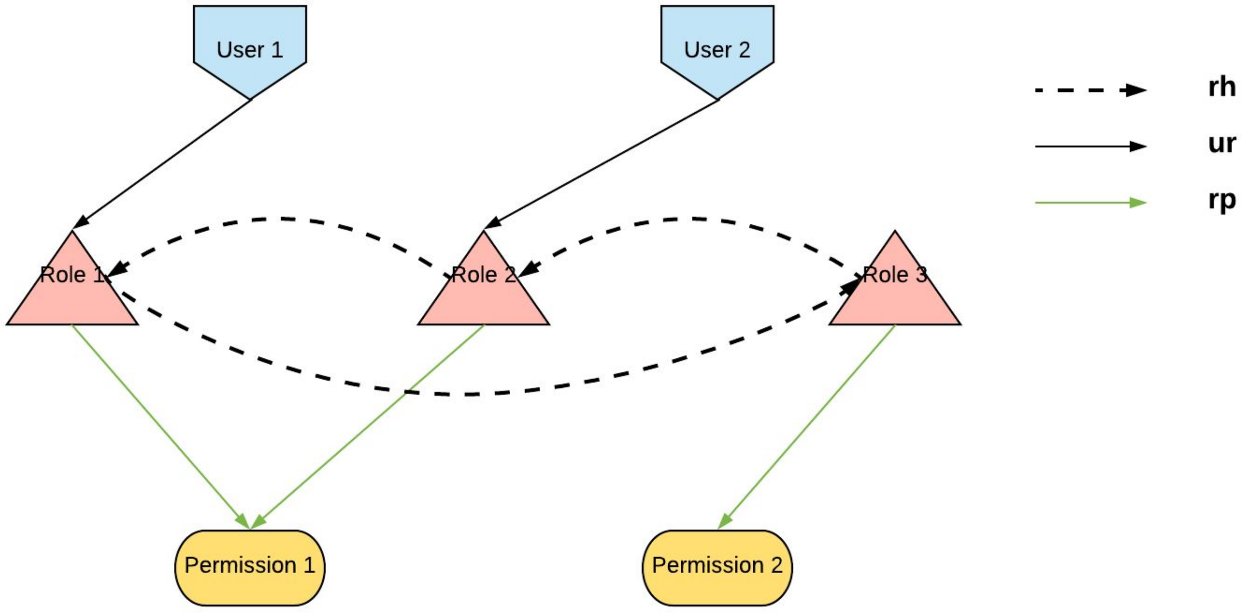
Output 3: minRoles(S) :-

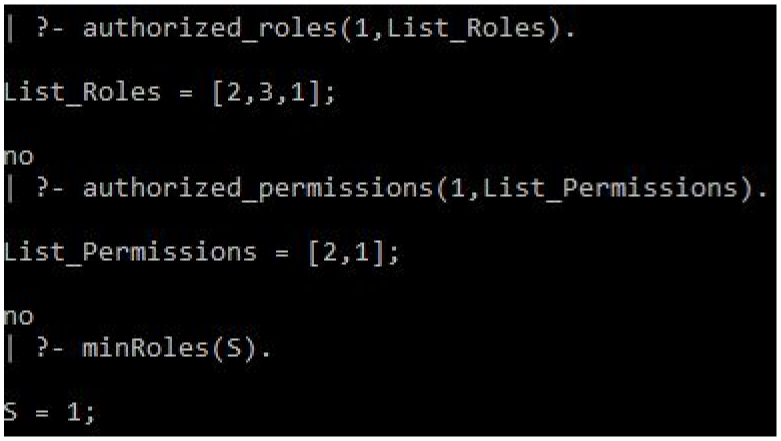


```
| ?- minRoles(S).

S = 2;
```

Permission 1 has roles [1,2]. Permission 3 has roles [1,2,3]. So we can create a new role having permissions 1 and 2. We create another role having permissions 1, 2 and 3. Hence there are minimum two roles that we need to have to cover all the users. Hence the output is 2.

Case 1: When there is a cycle in the role hierarchy

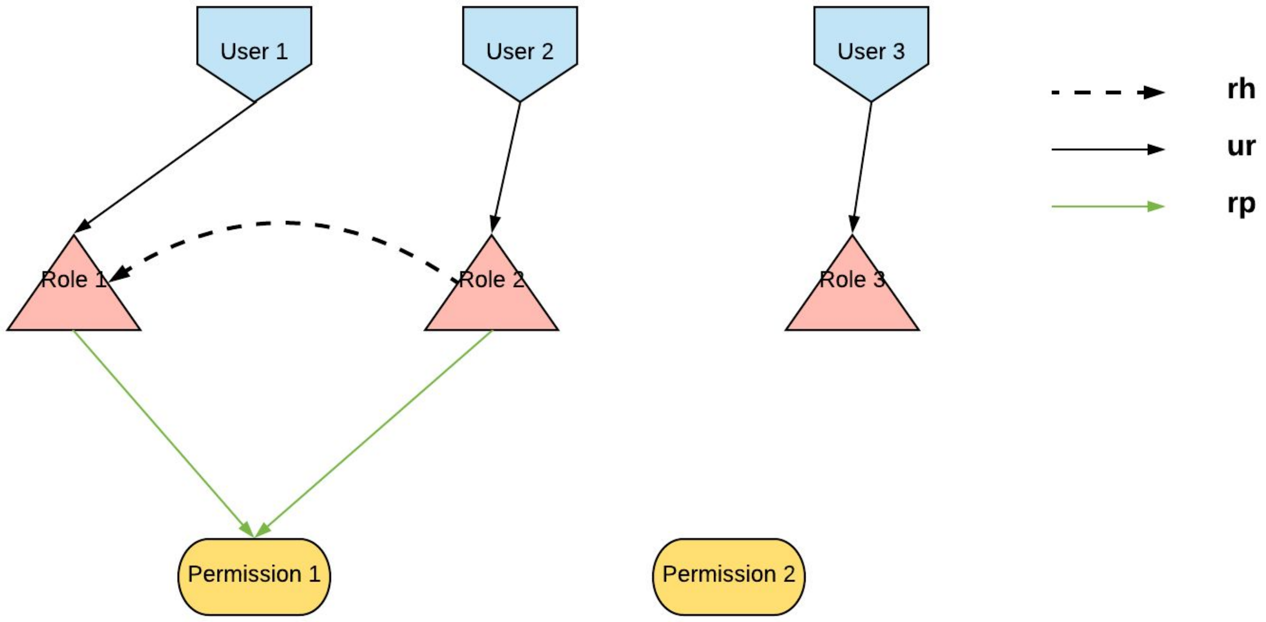


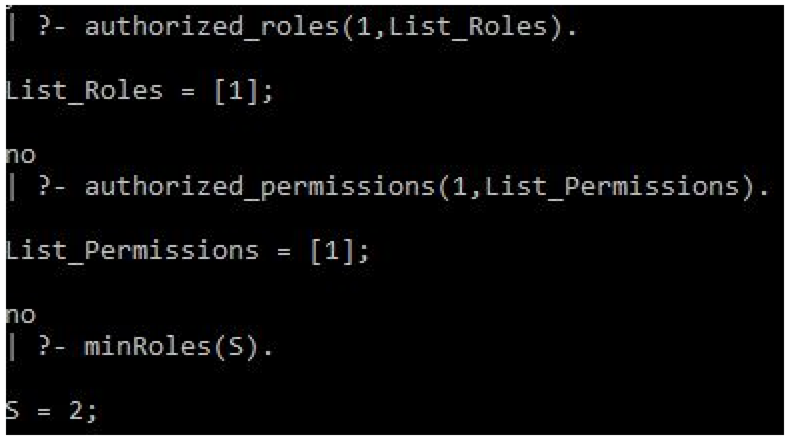


```
| ?- authorized_roles(1,List_Roles).

List_Roles = [2,3,1];

no
| ?- authorized_permissions(1,List_Permissions).

List_Permissions = [2,1];

no
| ?- minRoles(S).

S = 1;
```

In this case, there is a cycle of role hierarchies that is formed (3->2, 2->1, 1->3). We are handling this case of cycles in our hierarchy by using the concept analogous to Visited nodes in a graph while traversing. We check if the descendant of the role in the hierarchy has already been visited and its descendants already checked, then we do not consider that role again. This helps us to avoid being stuck in a loop.

Case 2: When there is no permission for a role assigned to a user.





```
| ?- authorized_roles(1,List_Roles).

List_Roles = [1];

no
| ?- authorized_permissions(1,List_Permissions).

List_Permissions = [1];

no
| ?- minRoles(S).

S = 2;
```

In this case, there is a User who has a role assigned to it. But that role does not have any permissions attached to it. These type of users fall under the same class of users who do not have any permissions to be assigned under our new hierarchy of single layered role structure. Hence, if there is such a User in the test data, we should add a Role to be assigned to all such users who will not have any permissions. And we will add a role under our new hierarchy of roles for this class of users.