Report on Project 3: XSB Prolog Role Based Access Control

Date: 10/25/2018

Project Members:

Dhruva Gaidhani (111971181)

Sumedh Arani (112078010)

MS CS Fall ’18

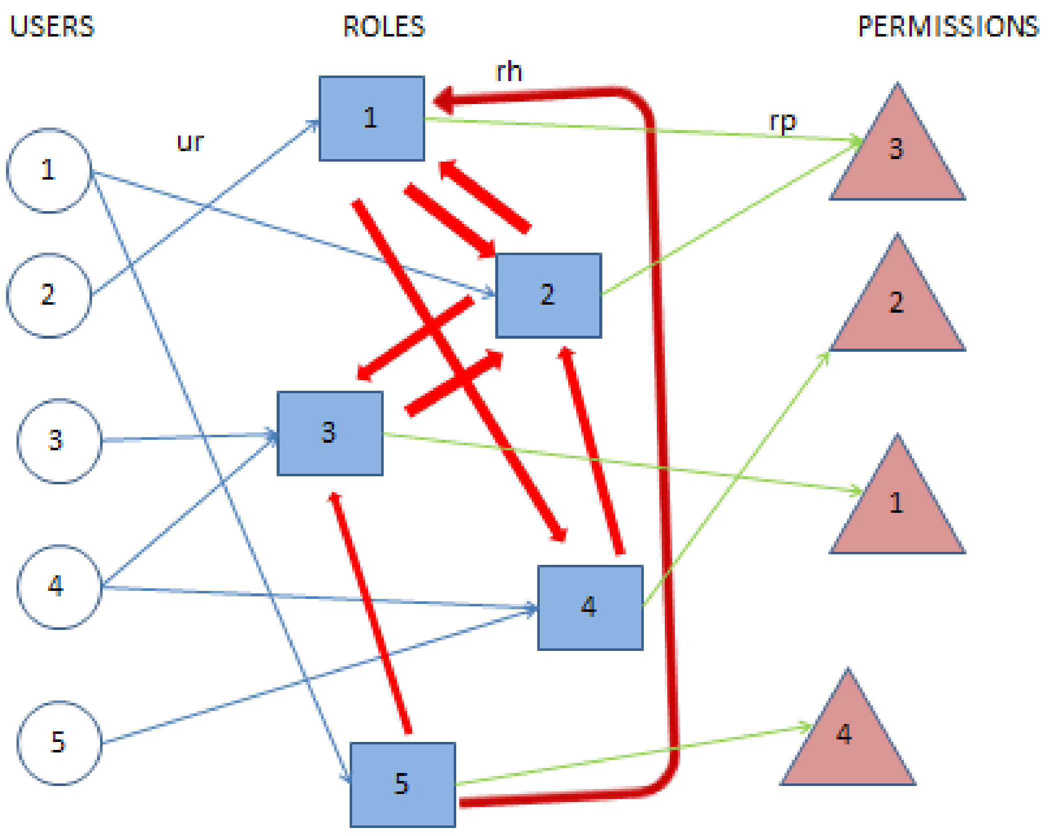
Input:-

users(5).
roles(5).
perms(4).

Ur	Rp	Rh
ur(1,2).	rp(1,3).	rh(1,2).
ur(1,5).	rp(2,3).	rh(1,4).
ur(2,1).	rp(3,1).	rh(2,1).
ur(3,3).	rp(4,2).	rh(2,3).
ur(4,3).	rp(5,4).	rh(3,2).
ur(4,4).		rh(4,2).
ur(5,4).		rh(5,3).
		rh(5,1).

User	Direct Roles	Total Roles
1	2,5	[2,5] + [1,3,4]
2	1	[1] + [2,4,3]
3	3	[3] + [2,1,4]
4	3,4	[3,4] + [1,2]
5	4	[4] + [2,1,3]

User	Permissions
1	[3,4,1,2]
2	[3,2,1]
3	[1,3,2]
4	[1,2,3]
5	[2,3,1]


USERS
ROLES
PERMISSIONS
ur
rh
rp
1
2
3
4
5
1
2
3
4
5
3
2
1
4

Q1. Authorized roles

Predicate call:

```
| ?- authorized_roles(1,List_Role).
List_Role = [2,5,1,4]
```

Predicate analysis:

Following is a list and explanation of the supporting predicates used

authorized_roles(User,List_Role)	Takes a user as input and returns corresponding roles
findall(Y,ur(User,Y),List_1)	It first finds all direct roles of given user
role_hierarchy(List_1,[],List_Role)	Then it looks for hierarchical roles
remove_common_from2(F2,N,N2)	This predicate performs a lookahead and avoids appending roles that already exist to the list to avoid infinite cycles
remove_duplicates(List_Prev,List_Role)	This predicate removes any existing duplicates to give distinct results

The query runs in a matter of milliseconds. System time recorded to run it is:

```
| ?- time(authorized_roles(1,A)).
% 0.0 CPU in 0.0 seconds (Inf% CPU)
A = [2,5,1,3,4]
```

Explanation - From ur() we first get [2,5]
Then we go through rh() to get [1,3] from 2 and [1,3] from 5
Recursively we get [2,4] from 1 and then we stop since we are now producing only duplicates.
Thus the function returns [2,5,1,3,4] in the order that it appended non duplicate members.

Q2. Authorized Permissions

```
% | ?- authorized_permissions(1,List_Permissions).
% List_Permissions = [4,3,1,2]
```

authorized_permission(User,List_Perm)	Takes a user as input and returns corresponding permissions
authorized_roles(User,List_Role)	Takes a user as input and returns corresponding roles
permission_hierarchy([F|L],F1,List_Per)	Takes roles recursively and returns the permissions for each role in a list
findall(Y,rp(User,Y),List_1)	It first finds all direct permissions of given user
role_hierarchy(List_1,[],List_Role)	Then it looks for hierarchical roles
remove_duplicates(List_Prev,List_Role)	This predicate removes any existing duplicates to give distinct results

The query runs in a matter of milliseconds. System time recorded to run it is:

```
| ?- time(authorized_permissions(1,A)).
% 0.0 CPU in 0.0 seconds (Inf% CPU)
A = [3,4,1,2]
```

Explanation - We first get all the roles of a user by calling authorized_roles and store result in a list.
We now parse this result and append the corresponding permission of that user to a new list while keeping a check on duplicates. We return this final list.

For 1 we get [2->3, 5 -> 4, 1 -> 3, 3 -> 1, 4 -> 2]
Which is [3, 4, 1, 2]

Q3. Minimum number of roles

```
% | ?- minRoles(S).
% S = 2
```

`minRoles(S)`	This predicate returns the minimum number of roles that sufficiently cover the system.
`list_maker(X,User_list)`	This predicate returns a list of X numbers from 1 to X.
`authorized_rp_recur(User_list,[],All_r)`	This predicate recursively calculates a list of list of permissions for all the users.
`list_sort(All_roles,[],All_sort_roles)`	This predicate sorts all the lists which were present inside the previous list so that the remove_duplicate predicate can correctly remove the lists which are common but its elements are out of order.
`length_list(All_unique_roles,S)`	This predicate returns the length of the list, which actually gives the final answer.

The query runs in a matter of milliseconds. System time recorded to run it is:

```
 | ?- time(minRoles(S)).
% 0.0 CPU in 0.0 seconds (Inf% CPU)
S = 2
```

Explanation - We first build a list of all users. Then we build a list of list of all the permissions of the users. Then we check for common elements in the list.
After common elements have been removed we will be left with distinct elements. The count of this will give us the required minimum roles.

```
All roles: [[2,5,1,3,4],[1,2,4,3],[3,2,1,4],[3,4,2,1],[4,2,1,3]]
All permissions: [[4,3,1,2],[3,2,1],[1,3,2],[1,2,3],[2,3,1]]
Distinct permissions: [[1,2,3,4],[1,2,3]]
```

Number of minimum roles required = length(dist_perm) = 2