SWE-4805: Software Verification and Validation

Introduction

Md. Nazmul Haque
Lecturer, IUT

Department of Computer Science and Engineering
Islamic University of Technology

December 02, 2021

Requirement Specification

Permission management system

There are three kinds of things in it such as **Accounts**, **Resources** and **Users**.

- **Resources** and **Users** belong to **Accounts**.
- **Users** can have direct access to **Resources**.
- **Users** can have a parent **Resource**.
- If a **User** can access a parent **Resource**, then s/he gets access to any child **Resource**.

Permission Management System: Signatures

```
sig Account {

}

sig User {

}

sig Resource {

}
```

```
run {} for 2 but exactly 2 Account
```

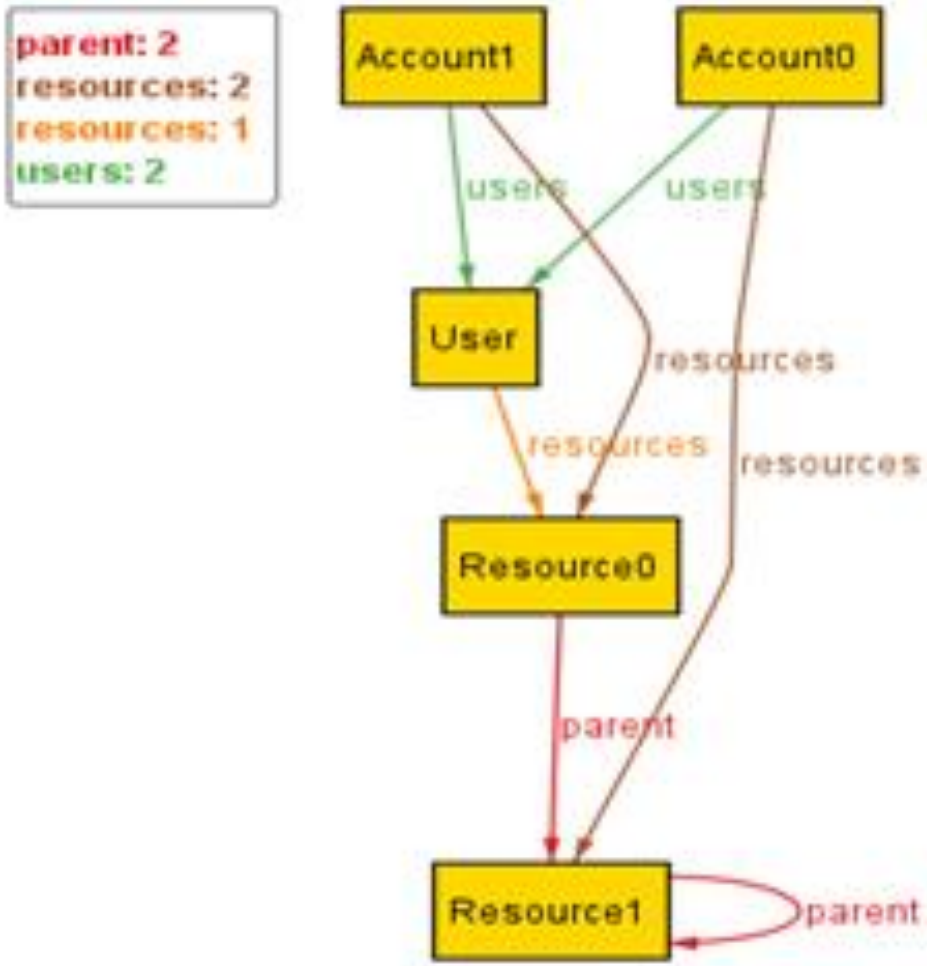
Permission Management System: Fields (Relations)

```
sig Account {
        // Every Account has 0 or more Resources
        resources: set Resource,
        // Every Account has 0 or more Users
        users: set User
}

sig User {
        // Every User has direct access to 0 or more Resources
        resources: set Resource
}

sig Resource {
        // Every Resource has 0 or 1 parent Resource
        parent: lone Resource
}
```

run {} for 2 but exactly 2 Account

Permission Management System

Instance



For every user, there is exactly one account.

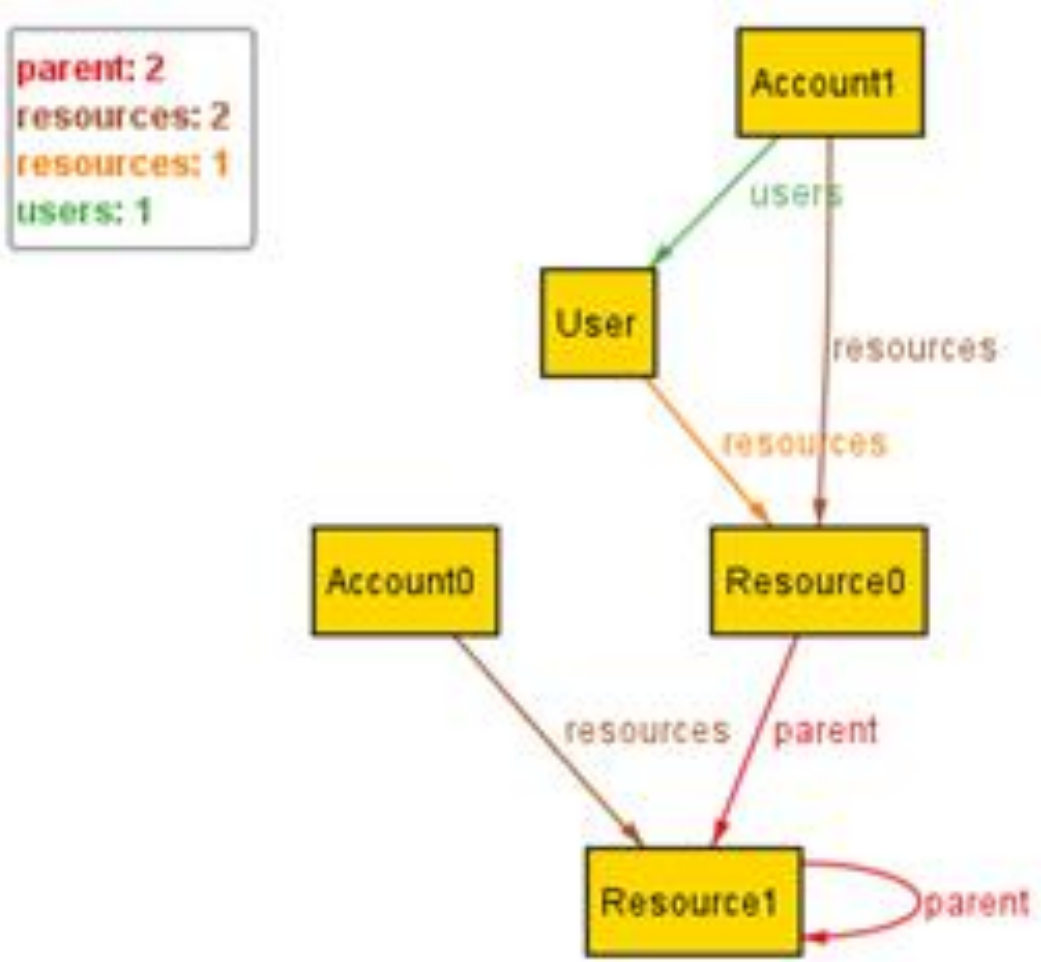
Permission Management System

User belong to one account

```
fact "no shared users" {
        // For each User 'u'
        all u: User |
                // there is exactly one Account 'a'
                one a:Account |
                        // for which 'u' belongs to 'a'
                        u in a.users
}

run {} for 2 but exactly 2 Account
```

Permission Management System

Instance



Resource sharing: CONFLICT

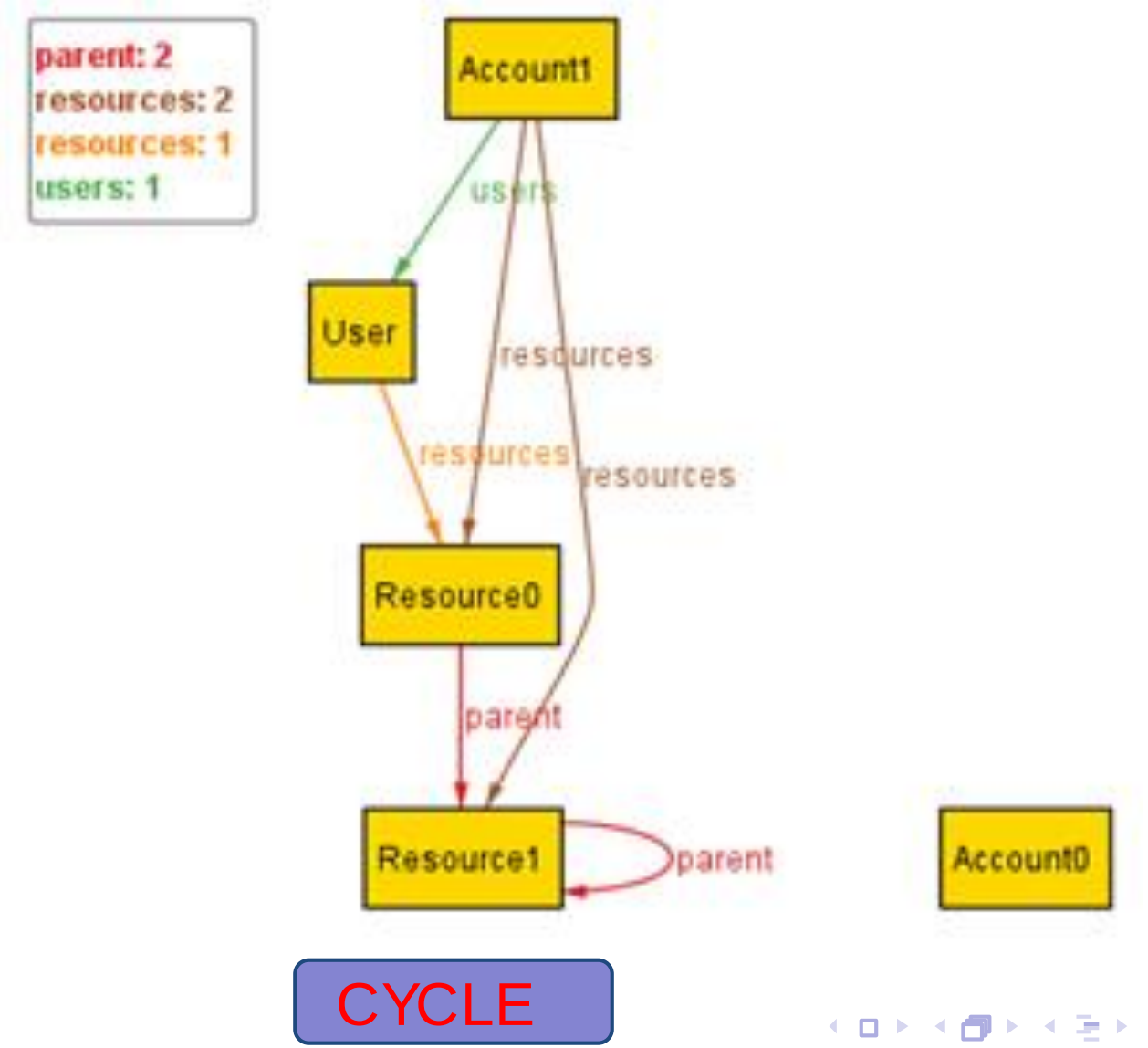
Permission Management System

```
fact "parent resource in same account" {
        // For each Resource 'r'
        all r:Resource |
                // if 'r' has a parent it implies that
                some r.parent implies
                        // there is exactly one account 'a'
                        (one a:Account |
                                // for which 'r'  and 'r.parent' both belong to 'a'
                                r in a.resources and r.parent in a.resources)
}
```

```
run {} for 2 but exactly 2 Account
```

Permission Management System

Instance



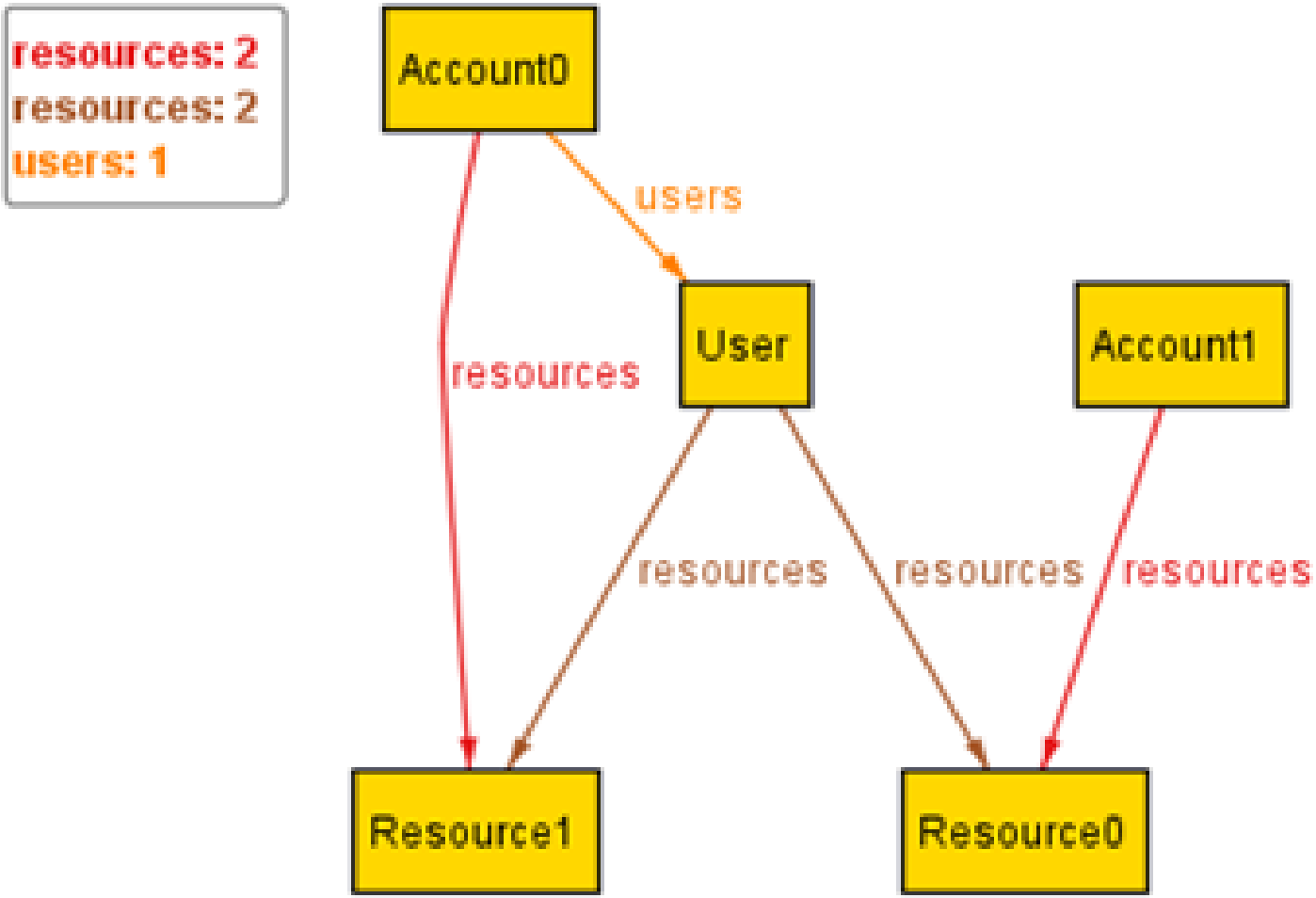
Permission Management System

```
fact "No_cycles" {
        // There is no Resource 'r' for which
        no r:Resource |
                // 'r' is in its own parent chain
                r in r.^parent
}
```

```
run {} for 2 but exactly 2 Account
```

Permission Management System

Instance



User shouldn't have access to another account's resources

Permission Management System

```
fact "only permit owning resources in same account" {
        // For every combination of User 'u' and Account 'a'
        all u: User, a:Account |
                // if 'u' belongs to 'a' it implies that
                // all of the resources that 'u' has access to
                // belong to 'a'
                u in a.users implies u.resources in a.resources
}
```

Permission Management System

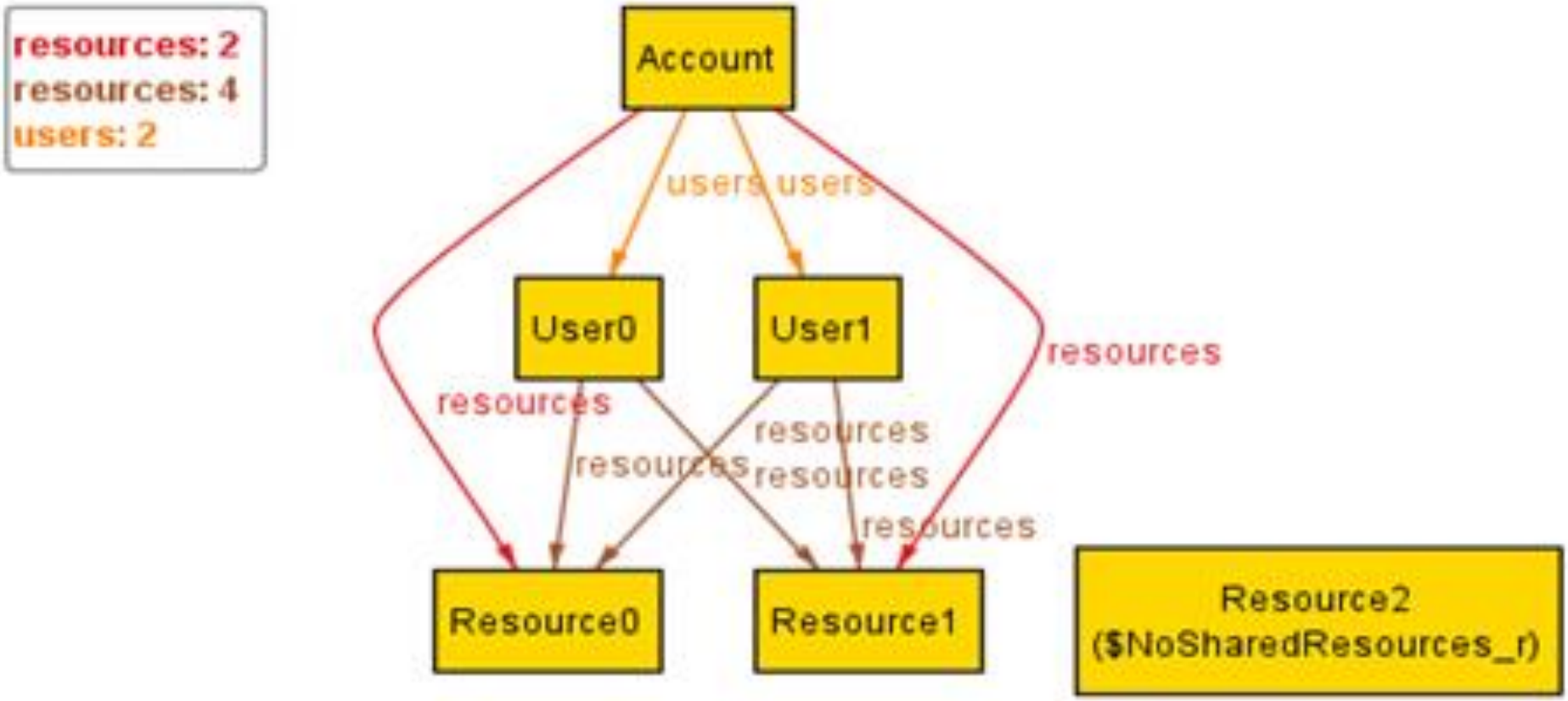
```
assert NoSharedResources {
        // For every Resource 'r'
        all r: Resource |
                // there is exactly one Account 'a' for which
                one a:Account |
                        // 'r' belongs to 'a'
                        r in a.resources
}
```

```
check NoSharedResources
```

Permission Management System

Instance

Believe: Every resource belongs to exactly one account



Every Resource should have an account

Permission Management System

```
fact "every resource has an account" {
        // The set of resources owned by any Account
        // must be equal to the complete set of Resources
        Account.resources = Resource
}
```

Permission Management System

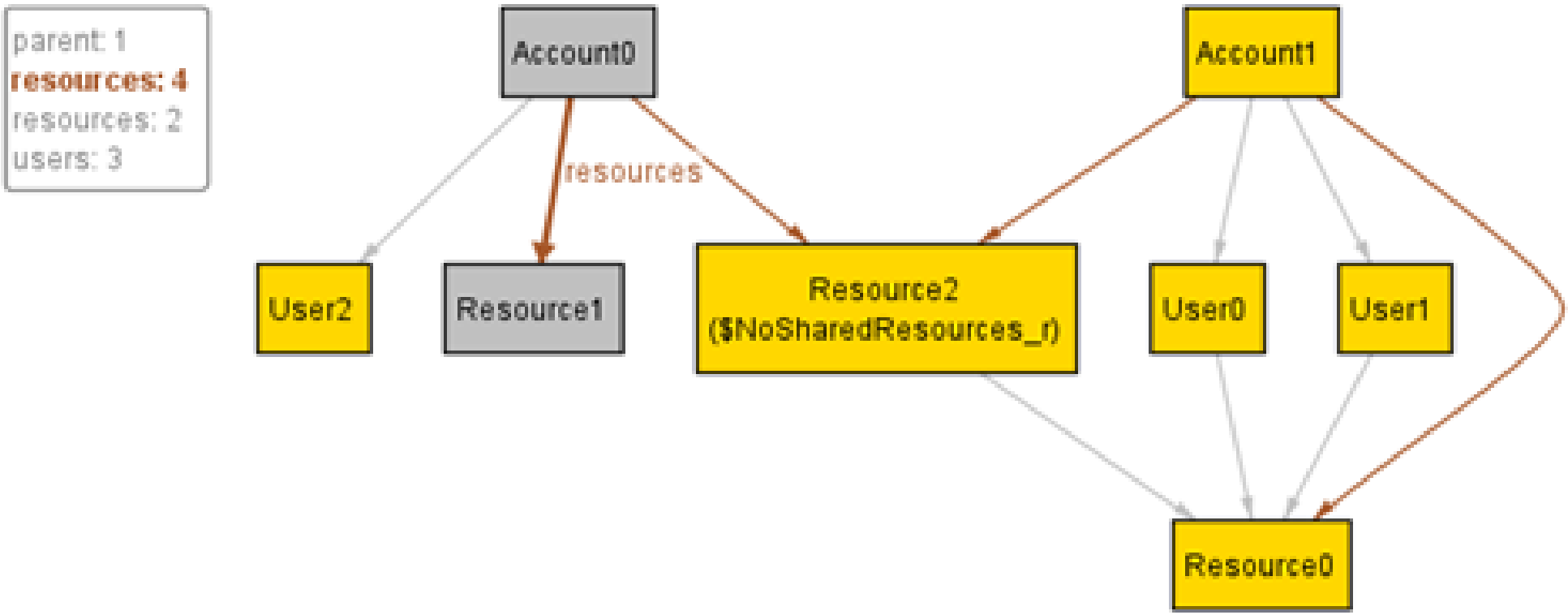
```
assert NoSharedResources {
        // For every Resource 'r'
        all r: Resource |
                // there is exactly one Account 'a' for which
                one a:Account |
                        // 'r' belongs to 'a'
                        r in a.resources
}
```

```
check NoSharedResources
```

Permission Management System

Instance

Believe: Every resource belongs to exactly one account



Resource2 is shared with Account0 and Account1

Permission Management System

```
fact "no inclusive resources of two accounts" {
        // There is no two combination of two Accounts 'a1', 'a2' such that
        no disj a1, a2:Account |
                // for which 'a1' and 'a2' both own
                // at least one of the same Resource
                // (i.e. No intersection between their Resources)
                some a1.resources & a2.resources
}
```

```
check NoSharedResources
```

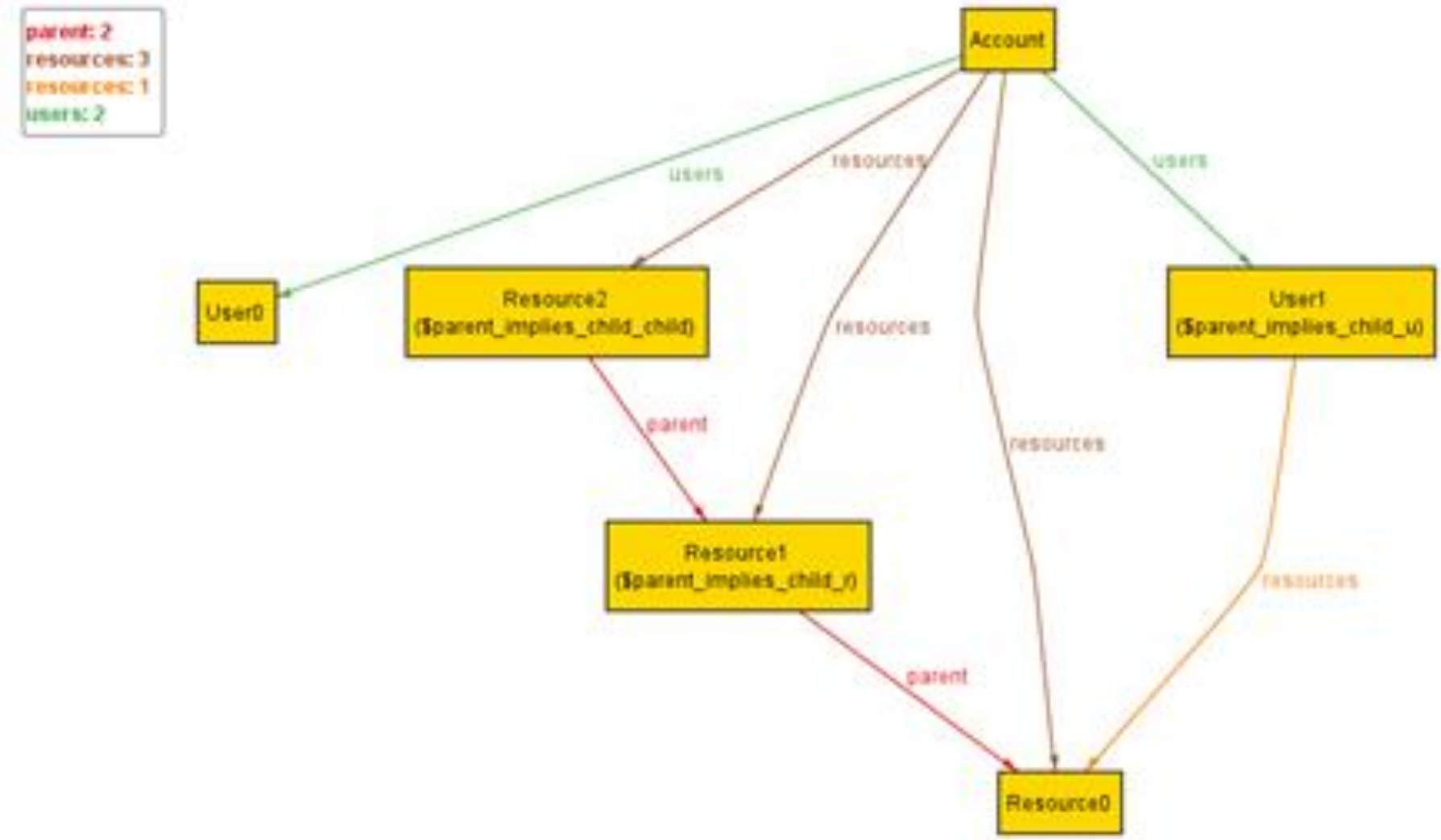
No counterexample found.

Permission Management System

```
// A user can access a Resource if they have direct access to it or
// if they have direct access to the Resource's parent
pred can_access (u: User, r: Resource) {
        r in u.resources or (some r.parent and r.parent in u.resources)
}

assert parent_implies_child {
        // For all combination of User 'u' and Resource 'r'
        all u:User, r:Resource |
                // if 'u' can access 'r' it implies that
                can_access[u,r] implies
                        // 'u' can access every child of 'r'
                        all child: parent.r | can_access[u, child]
}
check parent_implies_child
```

Permission Management System

Counter Example



ANY QUESTION ?
THANK YOU !

Acknowledgements

[1] Marcus S. Fisher, Software Verification and Validation: An Engineering and Scientific Approach, Springer, 2007 edition, 2006.

[2] Chauhan, Naresh. Software Testing: Principles and Practices. Oxford university press, 2010.

[3] Department of Software Engineering, University of Iowa.