# OCL WORKSHOP 2016

## A Comparison of Textual Modeling Languages: OCL, Alloy, FOML

Mira Balaban[1], Phillipa Bennett[2], Khanh Hoang Doan[3], Geri Georg[2], Martin Gogolla[3], Igal Khitron[1], Michael Kifer[4]

1. *Computer Science Department, Ben-Gurion University of the Negev*
2. *Computer Science Department, Colorado State University*
3. *Department for Mathematics and Computer Science, University of Bremen*
4. *Department of Computer Science, Stony Brook University*

# Introduction

- Textual languages are used in model-driven engineering for wide range of purposes.
- OCL, Alloy, and FOML are three popular textual languages.
- Our objectives?
  - Showing a comparison between three languages on major modeling criteria.
  - Discussing the similarities and differences among the languages.
  - Helping one in choosing a suitable textual language for modeling.

# Criteria for comparison

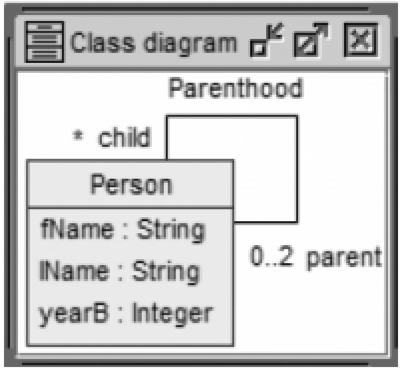
- Mode of usage and problems being solved
  - Constraining a model.
  - Querying and analysis.
  - Checking satisfiability of constraints.
  - Multiple levels of modeling.
- Representation aspects
  - Navigation through the elements of the models.
  - Supporting for collections.
  - Recursion.
  - Subtyping/instantiation.

# Modeling with OCL

- Navigation
  - Using role names from associations or object-valued attributes

    *context p: Person*

    *p.parent*




- Collections
  - Support four collection kinds*: sets, bags, sequences and ordered sets.*
  - Number of collection operations: *isEmpty, size, select, collect, union, intersection, . . .*
- Recursion: use transitive closure functionality

    *p.parent ->closure(parent)*

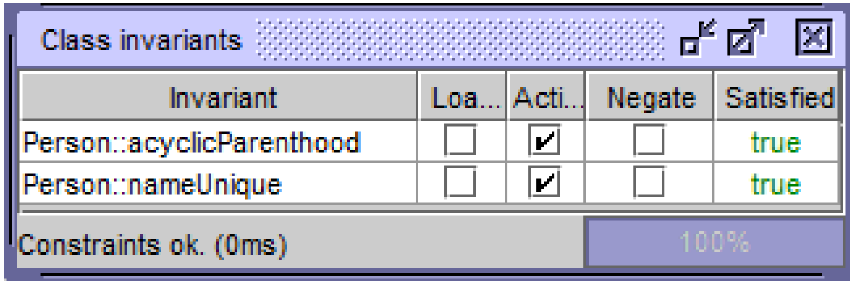
# Modeling with OCL (con)

- Formulating constraint with OCL
  - Formulate at class level
  - Its semantics is applied on the level of objects.
  - Three types of constraints: invariant, postcondition and precondition.

*context p:Person  inv acyclicParenthood:*

*p.parent->closure(parent)->excludes(p)*

- Checking satisfiability of constraints
  - Tool support (e.g., tool USE)



Class invariants

| Invariant | Loa... | Acti... | Negate | Satisfied |
|---|---|---|---|---|
| Person::acyclicParenthood | ☐ | ☑ | ☐ | true |
| Person::nameUnique | ☐ | ☑ | ☐ | true |

Constraints ok. (0ms) 100%

# OCL vs Alloy

- Similarities
  - The center of both languages is set and collection.
  - Using transitive closure functionality for recursion.
  - Formulating constraint quite similar → not much effort for translate constraints between.
- Differences
  - Alloy navigates through relation names, OCL navigates through association end names.
  - OCL supports n-ary associations and navigation through them, which cannot be done in Alloy.
  - One can define and use **predicate** in Alloy, which is not directly support in OCL.

# OCL vs FOML

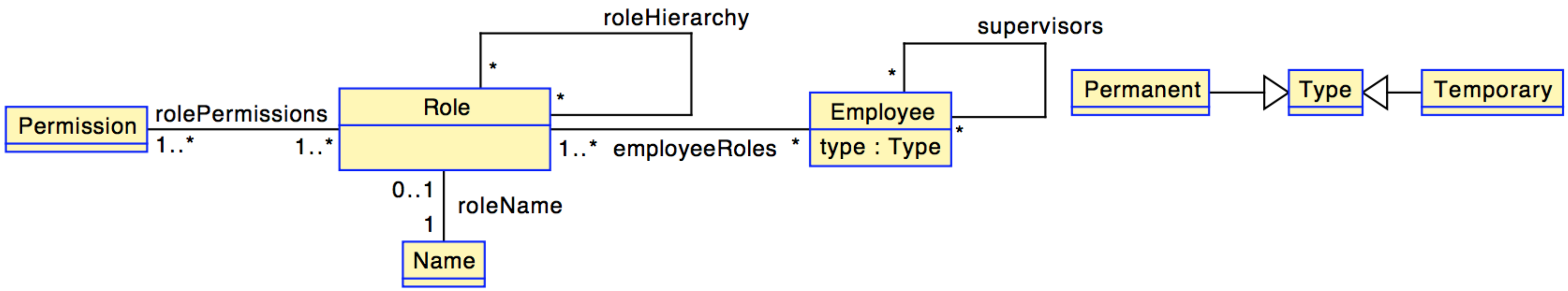
- Similarities
  - Most of the language features of FOML are supported in OCL.
  - Navigate through association-end names (role names).
  - Support composite associations (n-ary associations)
  - Support closure functionality.
- Differences
  - Main difference between the two modeling languages is the multilevel modeling support.
  - FOML supports three-layer specification: *data, model,* and *meta-model*. Current OCL version only supports two level

# Modeling with Alloy

# Alloy and the Alloy Analyzer

- Alloy is a Declarative Modeling Language.
- Alloy is supported by the Alloy Analyzer.
  - Classified as a Model Finder that searches for valid instances and counterexamples within a specified scope.
- Uses signatures, relations, facts, and predicates for model specifications.
- Uses predicates and assertions to query a model.
- Navigation occurs via relations, using the dot operator (which also serves as a relational join operator).

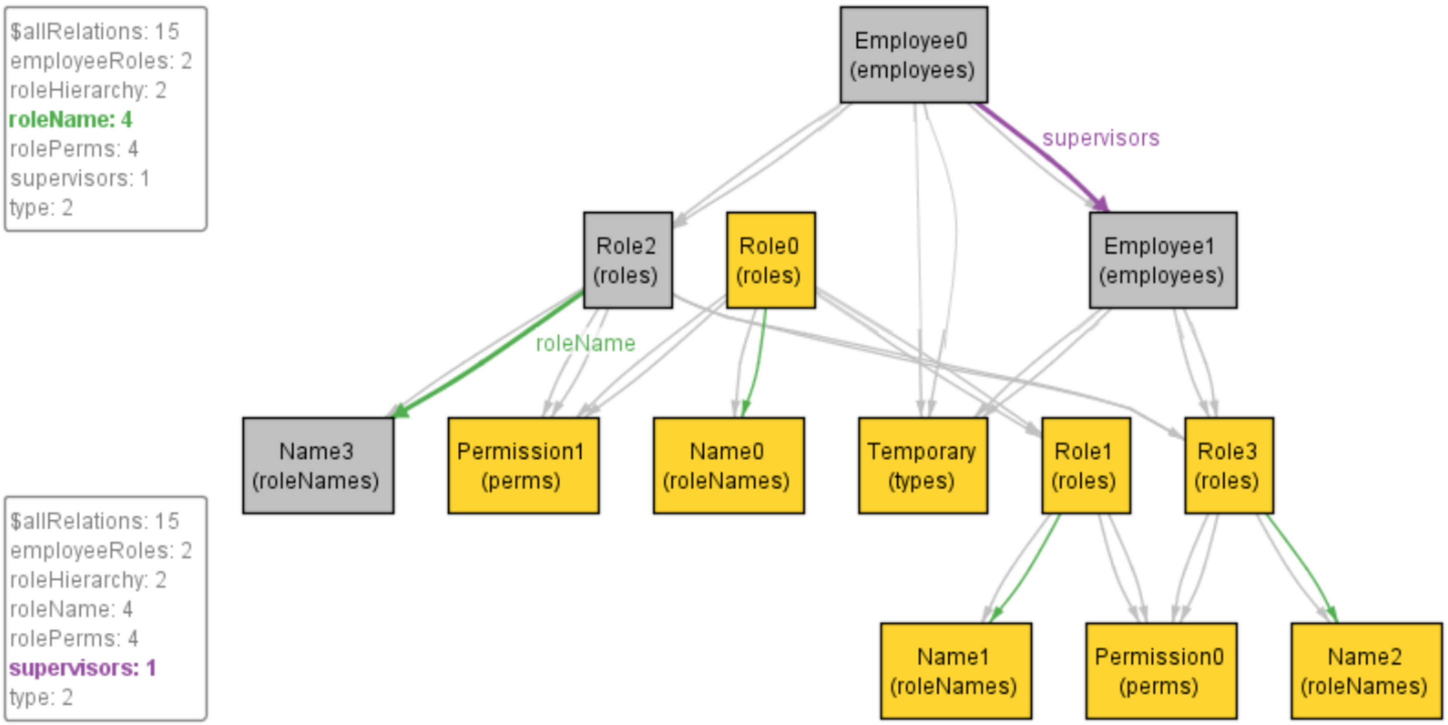
# The RolePermissionEmployee Alloy Model




```
module RolePermissionEmployee
open util/graph[Role] as g_r

sig Employee, Name, Permission {}
sig Role {roleName: Name }

sig Sys {
    roles: set Role,
    roleNames: set Name,
    perms: set Permission,
    roleHierarchy: roles -> roles,
    rolePermissions: roles some -> some perms
}{
    /* constrain the set of role names */
    roleNames = roles.roleName

    /* role names are unique */
    all n: roleNames | lte[#roleName.n, 1]
```

```
    /* supervisors relation is a tree */
    tree[supervisors]

    /* a supervisor also has the roles of
    those supervised through an ancestor
    role in the Role Hierarchy */
    all
        e1, e2: employees |
    some e1->e2 & ^supervisors implies
        e1.employeeRoles->e2.employeeRoles in
                ^roleHierarchy }

assert noLoopsInSupervisors {
    all
        sys: Sys |
    no iden & ^(sys.supervisors) }
check noLoopsInSupervisors for 7expect 0
```

# Alloy Instance




```
pred show (sys: Sys) {
    let
        n = 1 |
    gt[#sys.employees, n] and
        gt[#roots[sys.roleHierarchy],
              n] }
run show for 4but 1Sys expect 1
```

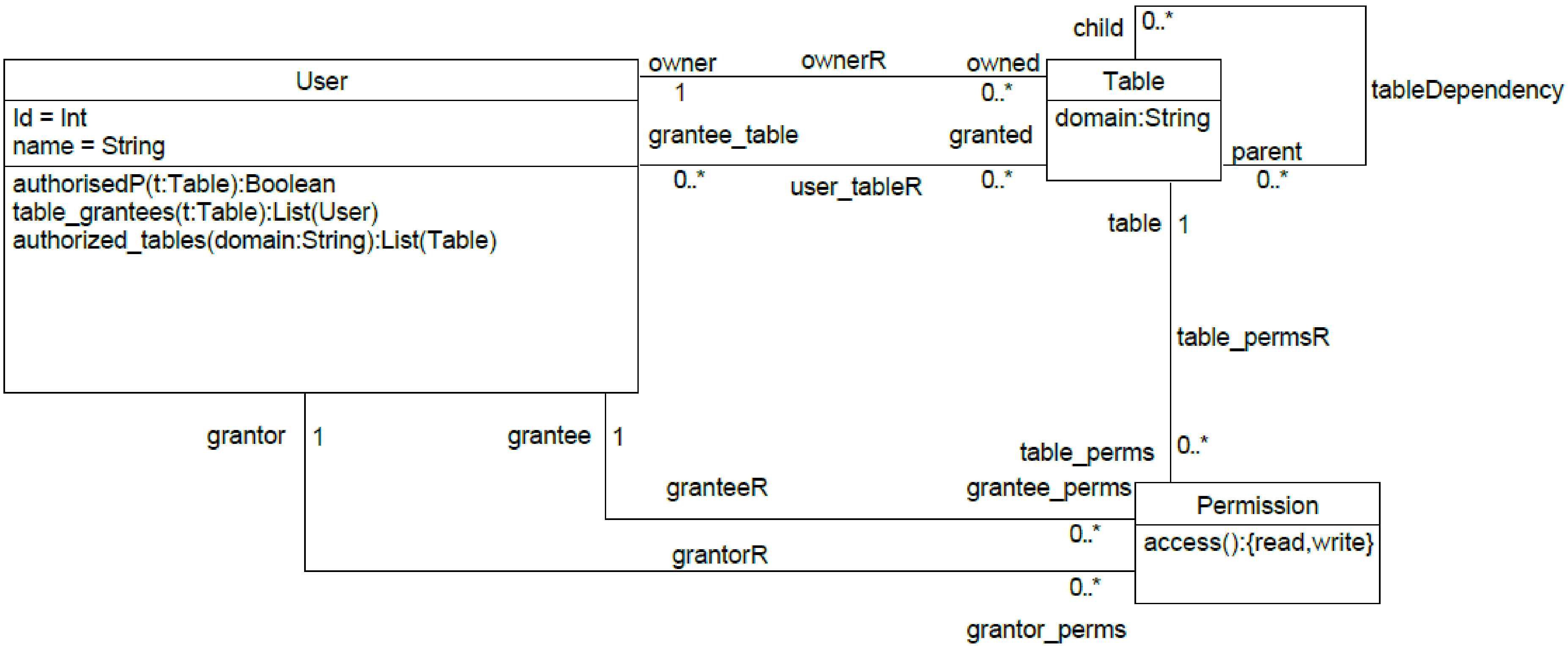
- *Role0* and *Role2* are the roots of *roleHierarchy*
- *Employee0* supervises *Employee1*
- *Employee0* gains the role of Employee1 through the *roleHierarchy*.
- No cycles in *roleHierarchy* or *supervisors*
- Complete model online, see reference in paper

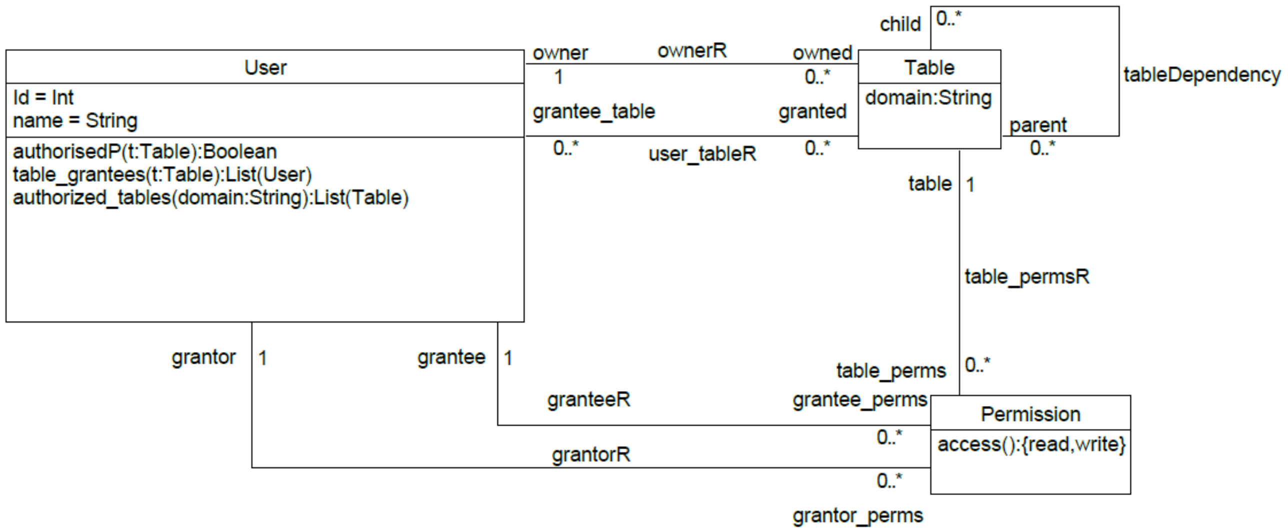
# Modeling with FOML

# *FOML – Feature Summary*

- Expressive rule logic language
  - **Extensional** (data-based) & **intensional** (inference-based)
  - **Executable**
  - **Extendable**
- Services:
  - **Modeling**: Textual model specification
  - **Constraints** (model extension)
  - Ad-hoc (on the fly) **querying** & **inference**
  - **Validation**, testing
  - **Metamodeling**, model analysis
  - **Multilevel** modeling

# Modeling – Industry Motivation

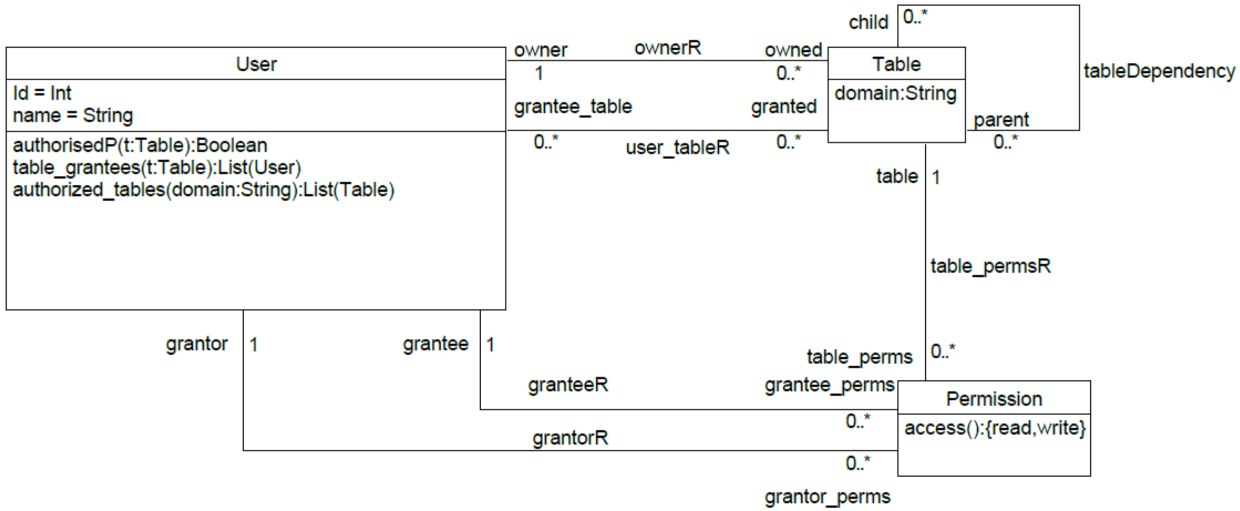





- **Metamodeling:**
  - *User:**Class**;*
  - *grantorR.**prop**(grantor,1,1)[User];*
- **Data:** *mary.granted[t1].table_perms[p1].grantee[mary];*
- **Query** (on the fly):

*Find grantor-grantee-permission triplets (**?u, ?v, ?p**) to tables whose domain is “teaching“:*

*?- **?u**:User, **?u**.grantor_perms[**?p**].grantee[**?v**], **?p**.table.domain["teaching"];*




**Intensional:**

Intensional (defined) property

– *mary.**compose**(granted, table_perms, grantee)[mary];*

– ***compose**(granted, table_perms, grantee).**circular**[true];*

An inference rule

– ***?p.circular**[true] :- **?o.closure**(**?p**)[**?o**];*

A constraint definition of circular

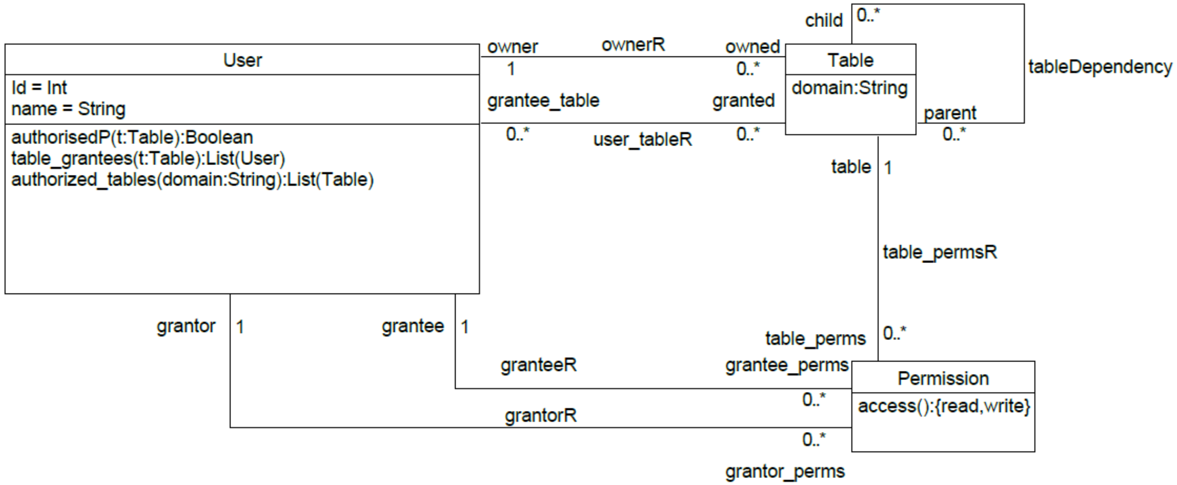
– !- ***?p.circular**[true], **not ?o. closure**(**?p**)[**?o**];*

– *For a table **?t**, the composition of grantor_perms and grantee is not circular*

**?u**.grantor_grantee(**?t**)[**?v**] :- **?u.compose_via_obj**(grantor_perms, **?p**, grantee)[**?v**], **?p**.table[**?t**];

!- ***?t**:Table, grantor_grantee(**?t**).circular[true];*

Intensional parameterized property




- **Association class constraint on** *Permission, user_tableR, grantee, table:*
  - *A user ?u that is a **grantee** in a permission to a **table** ?t , is **granted** access to ?t*

    *?u.granted[?t] :- ?u.grantee_perms.table[?t];* rule (9) in paper

  - *Every pair of a granted user ?u to a table ?t has a corresponding permission:*

    !- *?u.granted[?t], not ?u.grantee_perms.table[?t];* constraint (13) in paper

  - *For every grantee user ?u to a table, there is a single corresponding permission:*

    constraint (14) !- *?u.grantee_perms[?p1].table[?u.grantee_perms[?p2].table], ?p1!=?p2;*

- **Challenge:**

  ***Express the* association class *constraint in the other languages!***

# Comparison Summary

- Representation

| | Navigation | Recursion | Subtyping | Instance creation & completion |
|---|---|---|---|---|
| **OCL** | Individual & Collection; intermediate filtering; follows associations and derived associations | Transitive closure | Yes | Yes |
| **Alloy** | Individual; follows associations and virtual relations | Transitive closure | Yes | Yes |
| **FOML** | Individual; intermediate filtering; follows associations and virtual relations; wildcard navigation | User-defined recursion (includes transitive closure) | Yes | No |

# Comparison Summary

- Usage

| | Textual modeling | Querying | Inference | Validation | Multilevel Modeling |
|---|---|---|---|---|---|
| **OCL** | Yes | Yes | Via tools | Yes | No |
| **Alloy** | Yes | Yes | No | Yes | No |
| **FOML** | Yes | Yes | Yes | via constraints | Yes |

# Conclusion

- We present a comparison between modeling languages on the basis of their mode of usage and representation aspects.
- The similarities, differences, strengths and weaknesses are showed.
- The representation aspects of the languages have a lot of similarities.
- The mode of use of Alloy and OCL is closely related, whereas  FOML is quite different (e.g. multi level modeling)

# Thanks for your attention!