Outline

- OCL type hierarchy
- Collection operations
- Using collection operations in invariants

...with the Academia model as the running example.
Subtyping

- If $T_1, T_2$ are model types then $T_1 < T_2$ holds exactly when $T_1$ is a subclass of $T_2$ in a class diagram
- Integer < Real
- For all type expressions $T$, not denoting a collection type,
  - $\text{Set}(T) < \text{Collection}(T)$
  - $\text{Sequence}(T) < \text{Collection}(T)$
  - $\text{Bag}(T) < \text{Collection}(T)$
- If $T$ is a model, basic, or enumeration type then $T < \text{OclAny}$
- If $T_1 < T_2$ and $C$ is any of the type constructors Collection, Set, Bag, Sequence, then $C(T_1) < C(T_2)$
- The relation $<$ is transitive
- For any OCL type $T$, $T : \text{OclType}$

Examples

- Grad < Student < Person, so Grad < Person
  - ... because $<$ is transitive

- Collection(Grad) < Collection(Person)
  - ... because Grad < Person

- Set(Grad) < Collection(Grad)
  - ... because Set is a Collection type

- Set(Grad) < Collection(Person)
  - ... because Set is a Collection type and Grad < Person
Examples

- Grad < OclAny
  - ... because OclAny is a supertype of any model type

- Integer < OclAny
  - ... because OclAny is the supertype of any basic type

- Set(Grad) !< OclAny
  - ... because OclAny is NOT the supertype of a collection type

- Grad: OclType, Set(Grad): OclType, OclAny: OclType
  - ... because OclType is the type of every OCL type

For You To Do...

- Is Bag(Grad) < Set(Person)?
- Is Grad: OclAny?
- Is 5: OclAny?
- Is Sequence(Student) < Set(Person)?
- Is 5 < Integer?
- Is Bag(Grad) < Collection(OclAny)?
- Is Sequence(Grad) < Bag(Grad)?
- Is Set(Integer): OclType?
### Basic Collection Operations

#### Membership tests

- `collection->includes(object: OclAny): Boolean`
- `collection->excludes(object: OclAny): Boolean`

#### Examples

- `cis771.enrolled->includes(robby)` ...
  is true
- `cis771.enrolled->excludes(adam)` ...
  is true
- `robby.taking->includes(cis775)` ...
  is false

#### Inclusion tests

- `collection->includesAll(c2: Collection(T)): Boolean`
- `collection->excludesAll(c2: Collection(T)): Boolean`

#### Examples

- `cis771.enrolled->includesAll(Set {robby,oksana})` ...
  is true
- `cis771.enrolled->excludesAll(Bag {adam, adam})` ...
  is true

(see OMG-UML v1.3 Section 7.6.1 pp. 7.22-7.23)
Select

- collection->select(exp: Boolean)
  - ...yields the sub-collection of components satisfying the condition exp

Example

```
(robby, robbyId)
(oksana, oksanaId)
(william, williamId)
```

```
Id.allInstances->select(number > 150)
...Set{oksanaId,williamId}: Set(Id)
```

 Reject

- collection->reject(exp: Boolean)
  - ...returns a collection with the components satisfying the condition exp removed from the original collection

Example

```
(robby, robbyId)
(oksana, oksanaId)
(william, williamId)
```

```
Id.allInstances->reject(number > 150)
...Set{robbyId}: Set(Id)
```
Alternate Forms

- A variable (called the iterator) can be introduced to refer to the items in the resulting collection directly
  - \( \text{collection} \rightarrow \text{select}(v \mid \text{exp-with-}v: \text{Boolean}) \)
  - \( \text{collection} \rightarrow \text{select}(v: \text{Type} \mid \text{exp-with-}v: \text{Boolean}) \)

- \( v \) iterates over the collection and the \( \text{exp-with-}v \) is evaluated for each \( v \). The result is the collection containing the items for which \( \text{exp-with-}v \) is true.
  - \( \text{Id.all} \text{instances} \rightarrow \text{select}(id \mid id.\text{number} > 150) \)
  - ..similarly for ‘reject’

Collect

- \text{select} and \text{reject} always result in sub-collections of the original collection.
- When we want to specify a collection which is derived from some other collection, but which is not a sub-collection, we can use a \text{collect} operation.
  - \( \text{collection} \rightarrow \text{collect}(\text{exp}) \)
  - \( \text{collection} \rightarrow \text{collect}(v \mid \text{exp-with-}v) \)
  - \( \text{collection} \rightarrow \text{collect}(v: \text{Type} \mid \text{exp-with-}v) \)

- Example
  - \( \text{robb} \text{y.taking} \rightarrow \text{collect}(\text{number}) \)
  - \( \text{robb} \text{y.taking.number} \ldots \text{abbreviation (matches Alloy)} \)

- The result of the collect operation is always a bag!
Example

Consider the Academia constraint...
No one is taking or waiting for a course unless they have already taken all the prerequisites

context s: Student
Inv PrerequisitesRequired:
  s.transcript
  -> includesAll(s.taking.prerequisites
  - > union(s.waitingFor.prerequisites))

Note: the prerequisite bags above may contain duplicate courses, e.g., if the same course is a prerequisite for multiple courses being taken.

For You To Do...

- Pause the lecture
- Download the academia-4.use file
- Add to this file the class, association and invariant declarations for the extension to the Academia model described on the next two slides
- Load the resulting model into USE
- Create some system snapshots
  - some that satisfy your invariants
  - some that violate your invariants
  - you can build off of the academia-basic-instantiation.cmd script
- Use the USE command line to enter in some expressions for USE to evaluate (type ‘help’ to see syntax), e.g.,
  - use> ? cis771.enrolled->includes(robby)
- In your examples, explore the differences between sets and bags
- You may notice the appearance of the ‘undefined’ value in your results. What implications does that have for us?
I. Academia State

- Add an unique ID attribute for students
  - since we have integers and strings in OCL, there is more than one way to model IDs
- Add student transcripts
  - a transcript gives a set of courses associated with a student (the courses that a student has completed)
- Add prerequisite structure for courses
  - relates a course to courses that are prerequisites for it

I. Academia Constraints

- Enforce the uniqueness constraint for IDs
- A student can only take a course for which they have already taken the prereqs
- A course does not have itself as a prerequisite
  - An even stronger requirement is that there are no cycles in the prerequisite structure (can you do this one?)
- Realism: that there exists a course with prerequisites that someone is enrolled in
ForAll

Select

- collection->forAll(exp: Boolean)
  - ...returns true if exp holds for all objects in the collection

Example

(robby, robbyId)
(oksana, oksanaId)
(william, williamId)

Id.allInstances->forAll(number > 150)
...false

number=104
robbyId: Id
number=156
oksanaId: Id
number=200
williamId: Id

(see OMG -UML v1.3 Section 7.6.3 pp. 7.25-7.26)

 Exists

Select

- collection->exists(exp: Boolean)
  - ...returns true if there is at least one object in the collection for which exp holds

Example

(robby, robbyId)
(oksana, oksanaId)
(william, williamId)

Id.allInstances->exists(number > 150)
...true

number=104
robbyId: Id
number=156
oksanaId: Id
number=200
williamId: Id

(see OMG-UML v1.3 Section 7.6.3 pp. 7.25-7.26)
Example

Consider the Academia constraint...
Only faculty members teach required courses

```
context d: Department
inv FacultyTeachReqCourses:
  d.required.taughtby ->forall(i: Instructor |
    i.instructor.oclIsKindOf(Faculty))
```

Collection operation applied to obtain the bag of all instructors that teach required courses in department d

Alternate Forms

- We have the same sort of alternate forms as before
  - collection->forall(exp: Boolean)
  - collection->forall(v | exp-with-v: Boolean)
  - collection->forall(v: Type | exp-with-v: Boolean)

- In addition, if we need two quantified variables, we can use an abbreviation as in the following example...
  - Id.allinstances
    ->forall(id1, id2 | id1 = id2 implies id1.number = id2.number)

- The above is semantically equivalent to...
  - Id.allinstances
    ->forall(id1 | Id.allinstances
    ->forall(id2 | id1 = id2 implies id1.number = id2.number))

(see OMG-UML v1.3 Section 7.6.3 pp. 7.25-7.26)
Iterate

Generic iteration scheme that can be used to define all of the collection operations that we have seen so far.

- `collection->iterate( v : Type; acc : Type = exp | exp-with-v-and-acc)`

  - `v` iterates over the collection and the `exp-with-v-and-acc` is evaluated for each value of `v`.
  - After each evaluation of `exp-with-v-and-acc`, its value is assigned to `acc`.
  - In this way, the value of `acc` is built up during the iteration of the collection.

(see OMG-UML v1.3 Section 7.6.3 pp. 7.25-7.26)

Iterate

Example

- `collection->collect(x : T | x.property)`

  ...is semantically equivalent to...

- `collection->iterate(x : T; acc : T2 = Bag {} | acc->including(x.property))`
Iterate

As it would be coded in Java-like pseudo-code

iterate(v : T, acc : T2 = value)
{
    acc = value;
    for (Enumeration e = collection.elements();
        e.hasMoreElements(); ) {
        v = e.nextElement();
        acc = <exp-with-v-and-acc>
    }
}

For You To Do...

- Pause the lecture
- Download the academia-5.use file
- Add to this file the class, association and invariant declarations for the extension to the Academia model described on the next four slides
- Load the resulting model into USE
- Create some system snapshots
  - some that satisfy your invariants
  - some that violate your invariants
  - you can build off of the academia-extension1-instantiation.cmd script
- Use the USE command line to enter in some expressions for USE to evaluate (type 'help' to see syntax), e.g.,
  - use> ? cis771.enrolled->includes(robby)
- In your examples, code up some of your collection operations using the iterate operation
II. Academia State

- Add Departments
  - Instructors per
  - Courses per
  - Required courses
  - Student majors
- Add Faculty-Grad student relationships
  - Advisor
  - Thesis committee

II. Department Associations

- Each faculty is in a single department.
  - Each department has at least one faculty.
- Each department offers some courses
  - Courses are offered in exactly one department
- Each department requires some courses
  - Courses are required in at most one department
- Each student has a single department as his/her major
  - i.e., a department
II. Faculty-Student Associations

- A graduate student has exactly one faculty member as an advisor.
- A faculty member serves on five or fewer graduate student committees.

II. Academia Constraints

- Advisors are on their student’s committees
- Students are advised by faculty in their major
- Only faculty teach required courses
- Required courses are a subset of the courses for a major
- Students must take at least one course from their major each semester
- Realism: There are at least two departments and some required courses.
Acknowledgements

- Material for this lecture is based on the following sources
  - Chapter 7 (the OCL chapter) of the OMG-UML specification (version 1.3 - March 2000)