CS:5810 Formal Methods in Software Engineering

Modeling in Alloy: Academia Model

“Academia” Modeling Example

- We will model an academic enterprise expressing relationships between
  - People
    - Faculty
    - Students
      - Graduate
      - Undergraduate
    - Instructors – which can be grad students or faculty
  - Courses
  - Academic departments
  - Personal ID numbers

How should we model these basic domains in Alloy?

Strategy

1. Build and validate your model incrementally
   - Start with basic signatures and fields
   - Add basic constraints
   - Instantiate the model and study the results
   - Probe the model with assertions

Strategy

2. Add groups of features at a time
   - Add new signatures and fields
   - Add new constraints
   - Confirm previous assertions
   - Probe new features with assertions

Basic Components

- People
  - Students: Undergrads and Grads
  - Instructors: Faculty and Grads
- Courses
- Relationships
  - One *instructor* *teaches* a *course*
  - One or more *students* are *taking* a *course*
  - *Students* can be *waiting for* for *course*

Academia Signatures

```
abstract sig Person {}

sig Faculty extends Person {}

abstract sig Student extends Person {}

sig Graduate, Undergrad extends Student {}

sig Instructor in Person {}


sig Course {}

…
```

We are not specifying here that instructors can only be graduate students or faculty.

We will do that later with a **`fact`** *constraint.*

Academia Fields

- Only one *instructor* per *course*

2 choices:

```
sig Instructor in Person {
  teaches: set Course }

fact oneInstrucPerCourse {
  all c: Course | one teaches.c }
```

We cannot specify that there is exactly one instructor per course

We have to add a fact specifying this constraint

```
sig Course {
  taughtby: one Instructor }
```

Course Fields

- Only one *instructor per course*
- One or more *students taking* a *course*
- *Students* can be *waiting for* a *course*

Course Fields

- Only one *instructor per course*
- One or more *students taking* a *course*
- *Students* can be *waiting for* a *course*

```
sig Course {
  taughtby: one Instructor,
  enrolled: some Student,
  waitlist: set Student
}
```

Exactly one instructor per course

One or more students per course

Zero or more students per course

Dependent Relations

- We may choose to define dependent fields as auxiliary relations instead:

`teaches` (transpose of `taughtby`)
`taking` (transpose of `enrolled`)
`waitingfor` (transpose of `waitlist`)

```
fun teaches []: Instructor -> Course { ~taughtby }
fun taking []: Student -> Course { ~enrolled }
fun waitingfor []: Student -> Course { ~waitlist }
```

- Or we may choose not to have them at all:

if `i` is an instructor,

```
i.teaches = taughtby.i
```

Note

- Let `i` be an Instructor
- Let `taughtby` be the following binary relation
  - `taughtby: Course -> one Instructor`
- The following expressions denote the same set of courses
  - `taugthby.i`
  - `i.~taugthby`
  - `i[taugthby]`

Academia Constraints

- All instructors are either faculty or graduate students
  - Was not expressed in signature definition — although it could have:

```
sig Instructor in Graduate + Faculty
```

- No one is waiting for a course unless someone is enrolled
- No graduate students teach a course they are enrolled in

Academia Constraints

```
fact {
   -- All instructors are either Faculty or Graduate Students
   all i: Instructor | i in Faculty + Graduate

   Instructor !in Faculty    -- not all instructors are faculty

}
```

Academia Constraints

```
fact {
  -- no student is waiting for a course unless someone is enrolled
  all c: Course | some c.waitlist implies some c.enrolled
  -- alternative
  all c: Course | no c.enrolled implies no c.waitlist


}
```

Academia Constraints

```
fact {

   -- graduate students do not teach courses they are enrolled in or waiting to enroll in
    all c: Course | c.taughtby in Graduate implies
                       c.taughtby !in c.enrolled + c.waitlist

   -- Alternative
    no s: Graduate | (some c: s.teaches | s in c.enrolled + c.waitlist)

}
```

Academia Constraints

```
fact {
   -- All instructors are either Faculty or Graduate Students
   all i: Instructor | i in Faculty + Graduate

   -- no student is waiting for a course unless someone is enrolled
   all c: Course | some c.waitlist implies some c.enrolled

   -- graduate students do not teach courses they are enrolled in or waiting to enroll in
   all c: Course | c.taughtby in Graduate implies
                   c.taughtby !in c.enrolled + c.waitlist
}
```

Academia Constraints

```
fact {
   -- All instructors are either Faculty or Graduate Students
   Instructor in Faculty + Graduate

   -- no student is waiting for a course unless someone is enrolled
   no s: Student | some c: s.waitingfor | no c.enrolled

   -- graduate students do not teach courses they are enrolled in or waiting to enroll in
   no s: Graduate | (some c: s.teaches | s in c.enrolled + c.waitlist)
}
```

An alternative formulation

Academia *Realism* Constraints

- There is a graduate student who is an instructor
- There are at least:
  - Two courses and
  - Three undergraduates

Academia *Realism* Constraints

```
pred RealismConstraints [] {
  -- there is a graduate student who is an instructor
   some Graduate & Instructor

  -- there are at least two courses
   #Course > 1

  -- there are at least three undergraduates
   #Undergrad > 2
}
```

Can be added to the model as a fact or put in a **`run`** command
(to instruct the Alloy Analyzer to ignore unrealistic instances)

Academia Assertions

Does the current model have these properties?

- No *instructors* are on the waitlist for a *course* they teach
- No *student* is enrolled and on the waitlist for the same *course*

We can check that with assertions

Academia Assertions

```
-- no instructors are on the waitlist for a course they teach
assert NoWaitingTeacher
{

}

-- no student is enrolled and on the waitlist for the same course
assert NoEnrolledAndWaiting
{

}
```

Academia Assertions

```
-- no instructors are on the waitlist for a course they teach
assert NoWaitingTeacher
{
  all c: Course | no (c.taughtby & c.waitlist)
}

-- no student is enrolled and on the waitlist for the same course
assert NoEnrolledAndWaiting
{
  all c: Course | no (c.enrolled & c.waitlist)
}
```

Exercises

- Load `academia-1.als`
- With realism conditions enabled, do any instances exist in the default scopes?
  - Manipulate the scopes as necessary to obtain an instance under the realism conditions
- By looking at various sample instances, do you consider the model to be underconstrained in any way?
- Check assertions

Realism Constraints

- No instances exist in the default scope
- Why ?
  - default scope:
    at most 3 tuples in each top-level signature

    allows: at most 3 Students

  - `some Graduate & Instructor`
    `#Undergrad > 2`

    needs at least 4 Students

Realism Constraints

```
pred [] RealismConstraints
{
  -- there is a graduate student who's an instructor
   some Graduate & Instructor

  -- there are at least two courses
   #Course > 1

  -- there are at least three undergraduates
   #Undergrad > 2
}

run RealismConstraints for 4
```

Instance

#Undergrad > 2 ~~#Undergrad > 1~~

```
Instance found:

Signatures:
  Course = {C0,C1}
  Person = {U0,U1,G}
  Faculty = {}
  Student = {U0,U1,G}
  Undergrad = {U0,U1}
  Graduate = {G}
  Instructor = {G}

Relations:
  taughtby = {(C0,G),(C1,G)}
  enrolled = {(C0,U1),(C1,U0)}
  waitlist = {(C1,U1),(C1,U0)}
```

Counter-example to assertion

```
Analyzing NoEnrolledAndWaiting ...

Counterexample found:

Signatures:
  Course = {C}
  Person = {G0,G1,F}
  Faculty = {F}
  Student = {G0,G1}
  Undergrad = {}
  Graduate = {G0,G1}
  Instructor = {G0,G1}

Relations:
  taughtby = {(C,G0)}
  enrolled = {(C,G1)}
  waitlist = {(C,G1)}
```

Need to relate
enrollment and waiting lists

Academia Assertions

- No *student* is enrolled and on the waitlist for the same *course*
  - A counterexample has been found, hence

    we transform this assertion into a fact

- No *instructors* are on the waitlist for a *course* they teach
  - No counterexample

Academia Assertions

`NoWaitingTeacher` assertion

- No counterexample within the default scope
- No counterexample within the scope 4, 5, 6, 10

Can we conclude that the assertion is valid?

- No! (It might have conterexamples but out of scope)

But we take comfort in the small scope hypothesis:

- if an assertion is not valid, it usually has a small counterexample

Why *NoWaitingTeacher* holds

Assertion

```
-- no instructors are on the waitlist for a course they teach
assert NoWaitingTeacher {
  all c: Course | no (c.taughtby & c.waitlist)
}
```

Signatures + Facts

```
-- (i) faculty are not students
-- (ii) graduate students do not teach courses they are enrolled in or waiting to enroll in
all c: Course | c.taughtby !in c.enrolled + c.waitlist
```

Extension 1

- Add an attribute for students
  - Unique ID numbers
  - This requires a new signature
- Add student transcripts
- Add prerequisite structure for courses

New Relations

```
sig Id {}
abstract sig Student extends Person {
  id: one Id,
  transcript: set Course
}
sig Graduate, Undergrad extends Student {}
sig Instructor in Person {}
sig Course {
  taughtby: one Instructor,
  enrolled: some Student,
  waitlist: set Student,
  prerequisites: set Course
}
```

New Constraints

- Each Student is identified by a unique ID
  - Exactly one ID per Student

    already enforced by multiplicities
  - No two distinct students have the same ID

    has to be specified as a fact
- A student's transcript contains a course *only if* it contains the course's prerequisites
- A course does not have itself as a prerequisite
- **Realism constraint:**
  there is a course with prerequisites and with students enrolled

Academia Constraints

```
fact {
  ...
  -- A student's transcript contains a course only if it contains the course's prerequisites
   all s: Student |
        s.transcript.prerequisites in s.transcript

  -- A course does not have itself as a prerequisite
   all c: Course | c !in c.prerequisites
}

run {
   ...
  -- there is a course with prerequisites and enrolled students
   some c: Course |
        some c.prerequisites and some c.enrolled
}
```

not sufficient!

Academia Constraints

```
fact {
 ...
 -- A student's transcript contains a course only if it contains the course's prerequisites
  all s: Student |
       s.transcript.prerequisites in s.transcript

 -- There are no cycles in the prerequisite dependencies
  all c: Course | c !in c.^prerequisites
}

run {
  ...
 -- there is a course with prerequisites and enrolled students
  some c: Course |
       some c.prerequisites and some c.enrolled
}
```

Academia Assertions

Is it this case in our model?

Students can only wait for courses they already have the prerequisites for

```
assert AllWaitsHavePrereqs {
  all s: Student | (waitlist.s).prerequisites in s.transcript
}
```

Exercises

- Load `academia-2.als`
- With realism conditions enabled, do any instances exist in the default scopes?
  - Manipulate the scopes as necessary to obtain an instance under the realism conditions
- By looking at various sample instances, do you consider the model to be underconstrained in any way?

Counter-example

```
Analyzing AllWaitsHavePrereqs ...

Counterexample found:

Signatures:
  Id = {Id0,Id1,Id2}
  Course = {C0,C1}
  Person = {U,G0,G1}
  Faculty = {}
  Student = {U,G0,G1}
  Undergrad = {U}
  Graduate = {G0,G1}
  Instructor = {G0,G1}

Relations:
  taughtby = {(C0,G0),(C1,G0)}
  enrolled = {(C0,U),(C1,G1)}
  waitlist = {(C1,U)}
  prerequisites = {(C1,C0)}
  transcript = {(G1,C0)}
  id = {(U,Id0),(G0,Id2),(G1,Id1)}
```

U waits for the course C1

and

C0 is a prerequisite for C1

but

U does not have C0

Where is (U,C0)?

New Constraint

Old Assertion: `AllWaitsHavePrereqs`

Students can wait only for those courses for which they already have the prerequisites

Old Fact:

Students can have a course only if they already have the prerequisites

New Fact:

Students can have, wait for or take a course only if they already have the prerequisites

New Constraint

New Fact:

Students can have, wait for or take a course only if they already have the prerequisites

```
all s: Student |
  (s.waitingfor.prerequisites +
   s.taking.prerequisites +
   s.transcript.prerequisites) in s.transcript
```

or

```
all s: Student |
  (s.waitingfor + s.taking + s.transcript).prerequisites
  in s.transcript
```

Extension 2

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

Department Relations

- Each *instructor* is in a single *department*
  - Each *department* has at least one *instructor*
- Each *department* has some *courses*
  - *Courses* are in a single *department*
- Each *student* has a single *department* as his/her *major*

Faculty-Student Relations

- A *graduate* student has exactly one *faculty* member as an *advisor*

- *Faculty* members serve on *graduate* students' *committees*

New Relations

```
sig Faculty extends Person {
  incommittee: set Graduate
}

abstract sig Student extends
Person {
  major: one Department
}

sig Graduate extends Student {
  advisor: one Faculty
}
```

```
sig Instructor in Person {
  department: one Department
}

sig Department {
  course: some Course,
  required: some Course
}
```

```
------------------------ Facts ------------------------

  -- Each department has at least one instructor
  all d: Department | some department.d

  -- Each course is in a single department
  all c: Course | one course.c
```

New Constraints

- Advisors are on their advisees' committees
- Students are advised by faculty in their major
- Only faculty can teach required courses
- Faculty members only teach courses in their department
- Required courses for a major are a subset of the courses in that major
- Students must be enrolled in at least one course from their major

Exercise

- Express as an Alloy fact each of the new constraints in the previous slide

Advisors are on their advisees' committees

```
------------------- Signatures and Fields -----------------
abstract sig Person {}

sig Faculty extends Person {
  incommittee: set Graduate
}

abstract sig Student extends
Person {
  id: one Id,
  transcript: set Course,
  major: one Department
}

sig Undergrad extends Student {}

sig Graduate extends Student {
  advisor: one Faculty
}

sig Instructor in Person {
  department: one Department
}

sig Course {
  taughtby: one Instructor,
  enrolled: some Student,
  waitlist: set Student,
  prerequisites: set Course
}

sig Id {}

sig Department {
  courses: some Course,
  required: some Course
}
```

Students are advised by faculty in their major

```
------------------- Signatures and Fields -----------------
abstract sig Person {}

sig Faculty extends Person {
  incommittee: set Graduate
}

abstract sig Student extends
Person {
  id: one Id,
  transcript: set Course,
  major: one Department
}

sig Undergrad extends Student {}

sig Graduate extends Student {
  advisor: one Faculty
}

sig Instructor in Person {
  department: one Department
}

sig Course {
  taughtby: one Instructor,
  enrolled: some Student,
  waitlist: set Student,
  prerequisites: set Course
}

sig Id {}

sig Department {
  courses: some Course,
  required: some Course
}
```

Required courses for a major are a subset of the courses in that major

```
------------------- Signatures and Fields -----------------
abstract sig Person {}

sig Faculty extends Person {
  incommittee: set Graduate
}

abstract sig Student extends
Person {
  id: one Id,
  transcript: set Course,
  major: one Department
}

sig Undergrad extends Student {}

sig Graduate extends Student {
  advisor: one Faculty
}

sig Instructor in Person {
  department: one Department
}

sig Course {
  taughtby: one Instructor,
  enrolled: some Student,
  waitlist: set Student,
  prerequisites: set Course
}

sig Id {}

sig Department {
  courses: some Course,
  required: some Course
}
```

Only faculty teach required courses

```
------------------- Signatures and Fields -----------------
abstract sig Person {}
sig Faculty extends Person {
  incommittee: set Graduate
}
abstract sig Student extends
Person {
  id: one Id,
  transcript: set Course,
  major: one Department
}
sig Undergrad extends Student {}
sig Graduate extends Student {
  advisor: one Faculty
}
sig Instructor in Person {
  department: one Department
}
sig Course {
  taughtby: one Instructor,
  enrolled: some Student,
  waitlist: set Student,
  prerequisites: set Course
}
sig Id {}
sig Department {
  courses: some Course,
  required: some Course
}
```

Faculty members only teach courses in their department

```
------------------- Signatures and Fields -----------------
abstract sig Person {}

sig Faculty extends Person {
  incommittee: set Graduate
}

abstract sig Student extends
Person {
  id: one Id,
  transcript: set Course,
  major: one Department
}

sig Undergrad extends Student {}

sig Graduate extends Student {
  advisor: one Faculty
}

sig Instructor in Person {
  department: one Department
}

sig Course {
  taughtby: one Instructor,
  enrolled: some Student,
  waitlist: set Student,
  prerequisites: set Course
}

sig Id {}

sig Department {
  courses: some Course,
  required: some Course
}
```

Students must be enrolled in at least one course from their major

```
------------------- Signatures and Fields -----------------
abstract sig Person {}

sig Faculty extends Person {
  incommittee: set Graduate
}

abstract sig Student extends
Person {
  id: one Id,
  transcript: set Course,
  major: one Department
}

sig Undergrad extends Student {}

sig Graduate extends Student {
  advisor: one Faculty
}

sig Instructor in Person {
  department: one Department
}

sig Course {
  taughtby: one Instructor,
  enrolled: some Student,
  waitlist: set Student,
  prerequisites: set Course
}

sig Id {}

sig Department {
  courses: some Course,
  required: some Course
}
```

There are at least two departments and some required courses

```
------------------- Signatures and Fields -----------------
abstract sig Person {}

sig Faculty extends Person {
  incommittee: set Graduate
}

abstract sig Student extends
Person {
  id: one Id,
  transcript: set Course,
  major: one Department
}

sig Undergrad extends Student {}

sig Graduate extends Student {
  advisor: one Faculty
}

sig Instructor in Person {
  department: one Department
}

sig Course {
  taughtby: one Instructor,
  enrolled: some Student,
  waitlist: set Student,
  prerequisites: set Course
}

sig Id {}

sig Department {
  courses: some Course,
  required: some Course
}
```

A student's committee members are faculty in his/her major

```
------------------ Signatures and Fields -----------------
abstract sig Person {}

sig Faculty extends Person {
  incommittee: set Graduate
}

abstract sig Student extends
Person {
  id: one Id,
  transcript: set Course,
  major: one Department
}

sig Undergrad extends Student {}

sig Graduate extends Student {
  advisor: one Faculty
}

sig Instructor in Person {
  department: one Department
}

sig Course {
  taughtby: one Instructor,
  enrolled: some Student,
  waitlist: set Student,
  prerequisites: set Course
}

sig Id {}

sig Department {
  courses: some Course,
  required: some Course
}
```

Assertions

- **Realism constraints:**
  There are at least two departments and some required courses

- **Administrative constraint:**
  A student's committee members are faculty in his/her major

Exercises

- Load `academia-3.als`
- With realism conditions enabled, do any instances exist in the default scopes?
- Manipulate the scopes as necessary to obtain an instance under the realism conditions
  - This requires some thought since constraints may interact in subtle ways
  - For example, adding a department requires at least one faculty member for that department
- Can you think of any more questions about the model?
  - Formulate them as assertions and see if the properties are already enforced by the constraints