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

- 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
- Add groups of features at a time
  - New signatures and fields
  - 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 a course.
Academia Signatures

abstract sig Person {}
sig Faculty extends Person {}
abstract sig Student extends Person {}
sig Graduate, Undergrad extends Student {}
sig Instructors in Person {}

sig Course {}
...

Note that 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

- One instructor *teaches* a course.
- 2 choices:

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

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

sig Course {
  taughtby: one Instructor
}
```

We cannot specify that there is exactly one instructor per course

We have to add a fact specifying this constraint
Course Fields

- One instructor teaches a course.
- One or more students are taking a course.
- Students can be waiting for a course.

\[\text{sig Course } \{ \text{taughtby: one Instructor,} \]
\[\text{enrolled: some Student,} \]
\[\text{waitlist: set Student} \} \]

Exactly one instructor per course.

One or more students per course.

Zero or more students per course.
More relations

- We may choose to define auxiliary relations:

  - **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 not:

  ```
  if i is an instructor, then
  i.teaches <=> taughtby.i
  ```
Note

- Let \( i \) be an Instructor
- Let \( \text{taughtby} \) be the following binary relation
  - \( \text{taughtby}: \text{Course} \rightarrow \text{one Instructor} \)
- The following expressions are equivalents and give a set of courses as result
  - \( \text{taughtby}.i \)
  - \( i.\sim\text{taughtby} \)
  - \( i[\text{taughtby}] \)
Academia Constraints

- All *instructors* are either *faculty* or *graduate* students.
  - Was not expressed in set definition--although it could have, with
    $$\text{sig Instructor in Graduate+Faculty}$$

- No one is waiting for a *course* unless someone is enrolled.

- No *graduate* student teaches a *course* that they are enrolled in.
Academia Constraints

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

    -- no one is waiting for a course unless someone is enrolled
    -- (This is actually superfluous. Why?)
    all c: Course |
        some c.waitlist => some c.enrolled

    -- 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
}
There is a graduate student who is an instructor.

There are at least...

- Two courses and
- Three undergraduates
Academia “Realism” Constraints

Can be added to the model as facts, or just put in a `run` command to instruct the Alloy Analyzer to ignore unrealistic instances

```
run {
    -- 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
}
```
Academia Assertions

Let’s check if our model has these properties:

- No *instructor* is on the waitlist for a *course* that he/she teaches.
- No *student* is enrolled and on the waitlist for the same *course*.
Academia Assertions

-- no instructor is on the waitlist for a course that he/she teaches

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 |
    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
  - $\Rightarrow$ at most 3 Students
  - some Graduate & Instructor
    #Undergrad $> 2$
  - $\Rightarrow$ at least 4 Students
“Realism” Constraints

pred [] RealismConstraints {
  -- there is a graduate student whom is 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)\}

Need to relate enrollment and waiting lists
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)}
Academia Assertions

- No *student* is enrolled and on the waitlist for the same *course*.
  - A counterexample has been found
  - => we transform this assertion into a fact

- No *instructor* is on the waitlist for a *course* that he/she teaches.
  - 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 counterexamples but out of scope)
- But we take comfort in the
  - small scope hypothesis: if an assertion is not valid, it probably has a small counter-example.
Why NoWaitingTeacher holds

- **Assertion**
  -- no instructor is on the waitlist for a course that he/she teaches
  
  ```
  assert NoWaitingTeacher {
    all c: Course | no (c.taughtby & c.waitlist)
  }
  ```

- **Facts**

  -- (i) faculty are not students and (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 one 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: there exists a course with prerequisites and with students enrolled
Academia Constraints

\[
\text{fact} \{ \\
... \\
\quad -- \text{A student's transcript contains course only if it contains} \\
\quad -- \text{the course's prerequisites} \\
\quad \text{all } s: \text{Student} | \\
\quad \quad \text{s.transcript.prerequisites in s.transcript} \\
\quad -- \text{A course does not have itself as a prerequisite} \quad \text{not sufficient!} \\
\quad \text{all } c: \text{Course} | \quad c \notin \text{in c.prerequisites} \\
\} \\
\text{run} \{ \\
\quad ... \\
\quad -- \text{some course has prerequisites and students enrolled} \\
\quad \text{some c: Course} | \\
\quad \quad \text{some c.prerequisites and some c.enrolled} \\
\}
Academia Constraints

fact {
...

-- A student’s transcript contains 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 dependences
all c: Course | c !in c.^prerequisites
}

run {
...

-- some course has prerequisites and students enrolled
some c: Course |
    some c.prerequisites and some c.enrolled
}
Academia Assertions

- Students can only wait to be in a course for which they already have the prerequisites

```plaintext
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 didn’t validate 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: A student can only have, wait for or take a course if they already have the prerequisites

\[
\forall s: \text{Student} | \\
(\text{waitlist.s.prerequisites} + \\
\text{enrolled.s.prerequisites} + \\
\text{s.transcript.prerequisites}) \\
in s.\text{transcript}
\]

\[
\forall s: \text{Student} | \\
(\text{waitlist.s} + \text{enrolled.s} + s.\text{transcript} ).\text{prerequisites} \text{ in } s.\text{transcript}
\]
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
}

------------------------- Fact --------------------------

-- 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 }
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
}
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.
- Assertion: 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.