

FM 2006 Alloy Tutorial

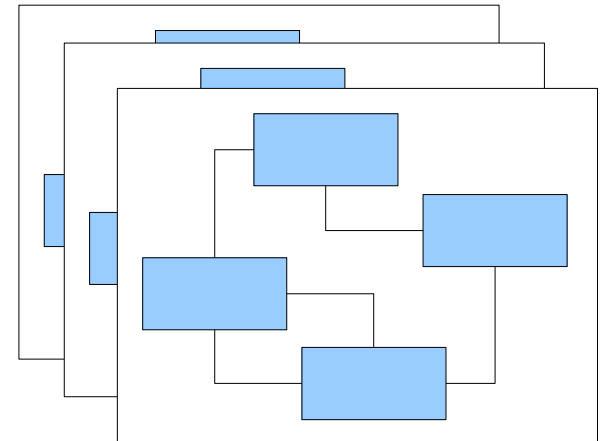
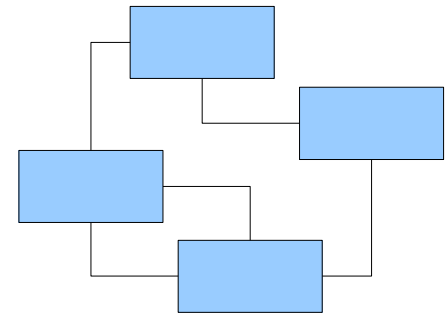
Session 3: Static Modeling

Greg Dennis and Rob Seater
Software Design Group, MIT



static vs. dynamic models

- static models
 - describes states, not behaviors
 - properties are *invariants*
 - e.g. that a list is sorted
- dynamic models
 - describe transitions between states
 - properties are *operations*
 - e.g. how a sorting algorithm works



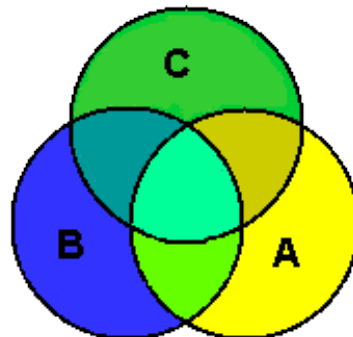
modeling academic records

- course catalog and graduation requirements
- create a new file in the Alloy Analyzer
- save it as *examples/tutorial/courses.als*
- write the appropriate module header



set declarations

- declare signatures for the following
 - our system has *courses*, *students*, and *departments*
 - all courses are either *introductory* or *advanced*
 - courses of either type can be *electives*
 - students are *freshmen*, *sophomores*, *juniors*, *seniors*



classification

- first step of building a model
 - consider what things are relevant
 - structure them hierarchically
 - subsets for orthogonal classification
- why *not* include in your classification . . . ?
 - the registrar *meaning unclear*
 - course prerequisites *relationship, not entity*
 - rooms where courses meet *irrelevant*

modeling the relationships

- create fields for the following
 - course belongs to a single department
 - department has courses required to graduate
 - advanced course has one or more prerequisites
 - student has at most one major department
 - student has courses they have taken

pattern: definition

- define a new term using existing terms
 - declare new relation and constrain to existing relations
 - constraint often written as equality, e.g.

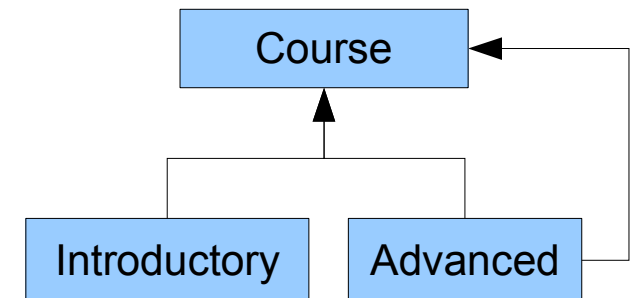
```
sig Person {  
    spouse: lone Person,  
    parents: set Person,  
    inlaws: set Person  
}  
fact { inlaws = spouse.parents }
```

- define a term for all the courses in a department
 - differs from courses required by a department

pattern: composite

- prerequisites establish *composite* hierarchy
 - advanced courses are composites
 - introductory courses are leafs
 - another example: file system directories and files

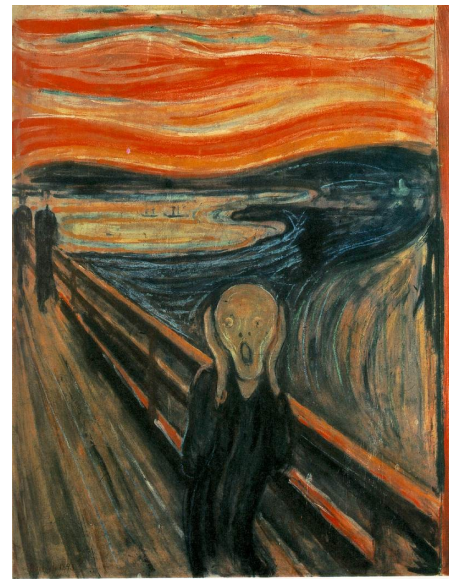
- composites typically must be acyclic
 - e.g. directory cannot contain itself



- constrain prerequisite relation to be acyclic
 - course cannot be its own prerequisite

pattern: sanity check

- write simple assertions while building models
- you'll be surprised how many fail
- check that every advanced course has an introductory course that precedes it



functions and predicates

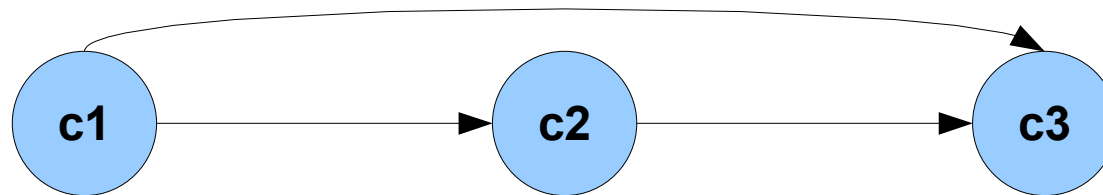
- create predicates or functions for the following
 - condition that a student can take a course
 - student has taken prereqs but not course itself
 - for a set of courses, expression for complete prereqs
 - prereqs of prereqs, prereqs of prereqs of prereqs, etc
 - condition that a student can graduate
 - student is a senior with a major
 - has taken all course's required by dept
 - one or more of student's courses are electives

pattern: guided simulation

- simulates model to check consistency
 - does the model admit any instances?
 - explore typical & interesting configurations
- create predicates with desired configurations
 - run predicates to ensure they exist
- example configuration:
 - every department has at least one advanced course
 - at least one student can graduate

compact prerequisites

- possible redundancy in prerequisite relation
 - transitive prerequisites can be direct prerequisites
 - over-complicates solutions and visualizations



- with constraint, eliminate redundant prereqs
 - try it with and without quantifiers

pattern: multirelation

- use higher-arity relation to model relationship between more than two entities
- address book example:

```
sig Book {  
    addrs: Name -> Addr  
}
```

- create a set of grades
- student has a grade in each course taken

pattern: singleton

- particular elements of set play important roles
- use **one** multiplicity to make a singleton sig

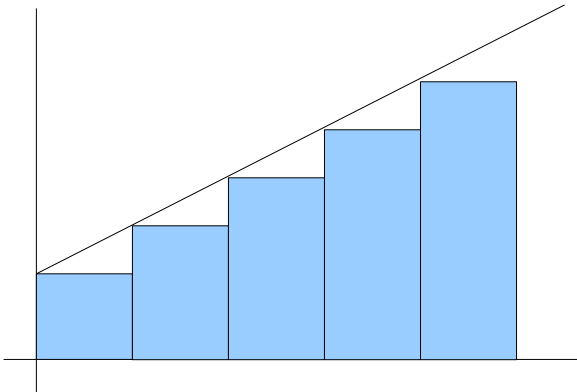
```
one sig Root extends Directory {}
```

- divide grades into exactly A, B, C, D, and F
- change graduation condition so student must pass (C or better) in each required course

A hand-drawn red 'A+' grade with a thick black outline, positioned in the bottom right corner of the slide.

pattern: approximation

- omit/loosen constraints present in reality
 - don't need to model everything!
- looser model often good enough
 - if abstraction, property preservation is sound
- important to keep approximations in mind
- what approximations are in our course model?



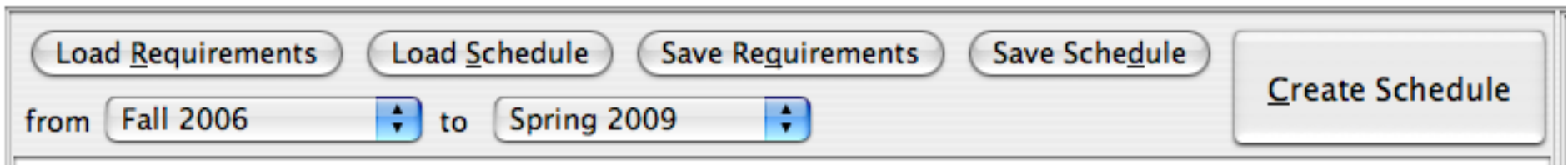
Monet

check and visualize

- write assertion that if a student can graduate, they must have passed all required courses as well as transitive prerequisites of required courses
- check assertion
- create intuitive visualization for counterexample
 - turn on and off sets and relations
 - change colors, shapes, names
 - turn relations into attributes
 - use defined variables
- add sensible constraints to ensure assertion passes

demo: declarative course scheduler

- designed and built by Vincent Yeung
- web application backed by Alloy engine
- generate a course schedule to satisfy MIT degree requirements given past courses
- <http://optima.csail.mit.edu:8080/scheduler/>
- <http://sdg.csail.mit.edu/projects/scheduler.html>



The screenshot shows a web interface for a course scheduler. It features five buttons in a row: 'Load Requirements', 'Load Schedule', 'Save Requirements', 'Save Schedule', and 'Create Schedule'. Below the first four buttons, there is a date range selector. It starts with the word 'from', followed by a text box containing 'Fall 2006', a blue up/down arrow button, the word 'to', another text box containing 'Spring 2009', and a second blue up/down arrow button. The 'Create Schedule' button is larger and positioned to the right of the date range.

pattern: set object

- all relations in Alloy are first order
- *but* some relationships are higher-order
 - relate sets of elements, not individuals
- solution: represent sets themselves as objects
 - single field relating set to its elements
 - often canonicalized: no two sets have same elements
- allow departments multiple sets of required courses
 - student can fulfill anyone of those sets
 - (optional) canonicalize required sets