CIS 771: Software Specifications

Lecture: Alloy Whirlwind Tour (part B)

Outline

- Simple email address book scenario
- A simple Alloy address book model
- Using the Alloy analyzer to automatically create model instances (examples)
- Directing the analyzer toward certain instances via…
  - scope settings
  - constraints

Address Book Application

Requirements (version 1)

- Name
- Address
- Book
  - entries that associate each name with at most one address

Address Book Application

Requirements (version 1)

- Name
- Address
- Book
  - entries that associate each name with at most one address

Modeling

- How do we represent each of the basic entities?
  - book, name, address
- How do we represent relationships between entities?
  - book is related to entries that relate names to addresses
  - how do we enforce the constraint “at most one”

Three Views of Alloy Specs

```
module tour/addressBook1

    sig Name, Addr {}
    sig Book {
          addr: Name -> lone Addr
    }
```

There are three different ways to view Alloy specifications

- Object-oriented view, useful for drawing connections to UML and OO languages
- Set-theoretic view, our primary view for modeling and reasoning about Alloy
- Atoms and relations view, the true semantics of the language

See Alloy Seater-Dennis tutorial sidenote-levels-of-understanding.html

Object-Oriented View

```
module tour/addressBook1

    sig Name, Addr {}
    sig Book {
          addr: Name -> lone Addr
    }
```

In the object-oriented view of Alloy specs…

- Signatures (`Name`, `Addr`, and `Book`) are analogous to classes
- Fields (`addr`) of type T holds references to objects of type T.
- Relations (`Name -> lone Addr`) play a role similar to Java hash tables, or Python dictionaries -- they map elements in the domain type (`Name`) to elements in the range type (`Addr`).

See Alloy Seater-Dennis tutorial sidenote-levels-of-understanding.html

Set-Theoretic View

```
module tour/addressBook1

    sig Name, Addr {}
    sig Book {
        addr: Name -> lone Addr
    }
```

Each Alloy signature represents a set of entities (objects)

In the set-theoretic view of Alloy specs…

- Signatures (`Name`, `Addr`, and `Book`) are sets
- Fields (`addr`) of type T is a relation mapping elements elements of the signature set to elements of T…
- …thus a relation that appears as the type of a field (`Name -> lone Addr`) actually is part of the definition of a 3-ary relation relating elements of `Book`, `Name`, and `Addr`.

See Alloy Seater-Dennis tutorial sidenote-levels-of-understanding.html

Set-Theoretic View

```
module tour/addressBook1

    sig Name, Addr {}
    sig Book {
        addr: Name -> lone Addr
    }
```


Fields form relations between signature elements and elements of field type

`lone` multiplicity states that at most one `Addr` is associated with a `Name`.

In the set-theoretic view of Alloy specs…

- Signatures (`Name`, `Addr`, and `Book`) are sets
- Fields (`addr`) of type T is a relation mapping elements elements of the signature set to elements of T…
- …thus a relation that appears as the type of a field (`Name -> lone Addr`) actually is part of the definition of a 3-ary relation relating elements of `Book`, `Name`, and `Addr`.

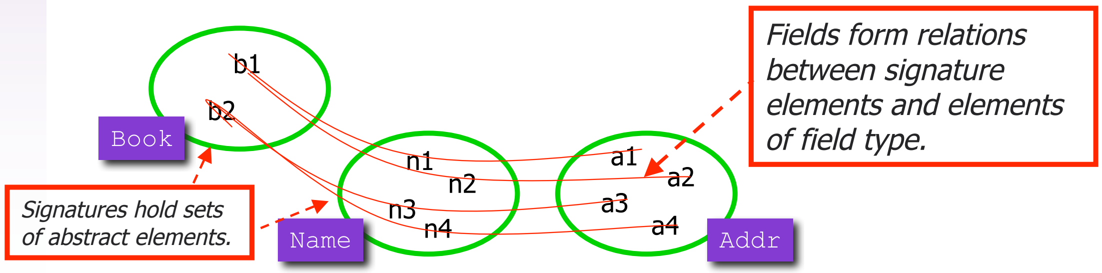
See Alloy Seater-Dennis tutorial sidenote-levels-of-understanding.html

Set-Theoretic View

```
module tour/addressBook1

    sig Name, Addr {}
    sig Book {
          addr: Name -> lone Addr
    }
```

In the set-theoretic view of Alloy specs…



Assessment

- The Alloy modeling language includes data constructs that roughly correspond to the OO paradigm
- Alloy is not meant to represent the concrete details of a system data model *nor* concrete details of operations on data.
- Instead, the elements of an Alloy model are very abstract
  - …no strings for name or addresses, no integer fields for phone number
  - such details are added *later* in the development process
  - the goal at this point is to model basic entities, relationships between them, and constraints on relationships
- Alloy has a very clean semantic interpretation based on sets
- Alloy's clean mathematical semantics enables automated reasoning techniques that allow us to…
  - automatically generate and visualize instances of the model
  - reason about the implications of our declaring structures and constraints
- The above automated capabilities are very important for helping up determine the appropriateness of our design

Querying the Model

- We can use Alloy's constraint solver to automatically generate different instances of our declared data structures
- We use Alloy's notion of *scope* to bound the size of the data structures created (bound the number of elements in each signature).
- We use Alloy's *constraint language* to put conditions on the particular instances created.
- Thus, we use scope settings and constraints to guide a search through the space of possible model instances
  - we can "hone in" on particularly interesting/troublesome instances
- Instead of trying think of (non) examples just using our brains and instead of drawing a bunch of possibilities on a whiteboard for the purpose of evaluating our design, Alloy produces instances and answers queries that help us evaluate our design much more rapidly and effectively.

Generating Instances

Typical steps for generating model instances

```
pred show0 () {
 // constraints
}
```

Define a Alloy predicate construct that contains constraints which instances to be generated must satisfy (for now, we have no constraints).

Default is maximum of 3 elements for each signature

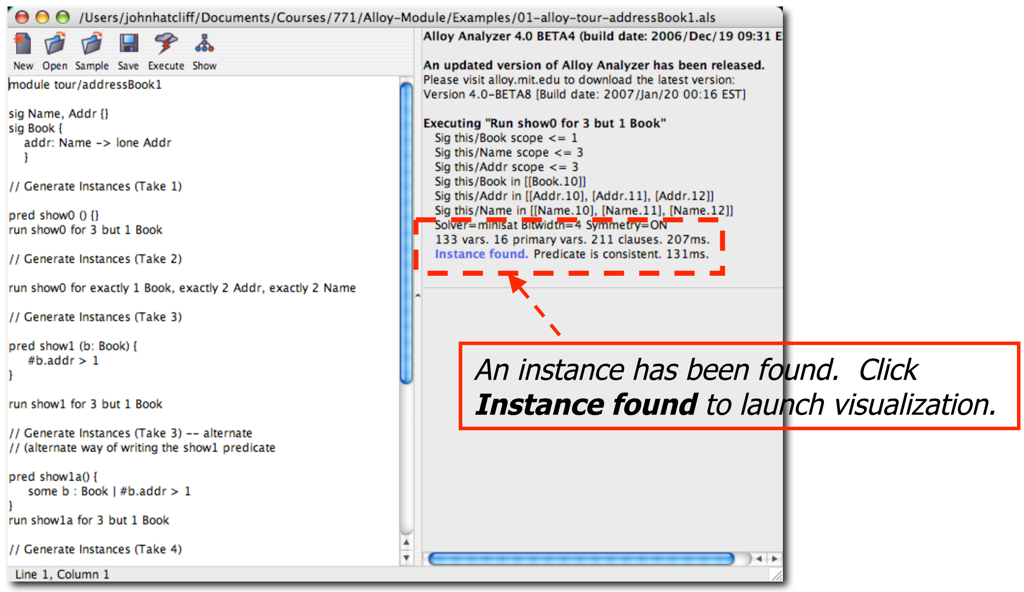
Exception: maximum elements is 1 for Book signature

```
run show0 for 3 but 1 Book
```

Use the run command to tell the Alloy analyze to find an instance/example of the model that satisfies the constraints in the given predicate.

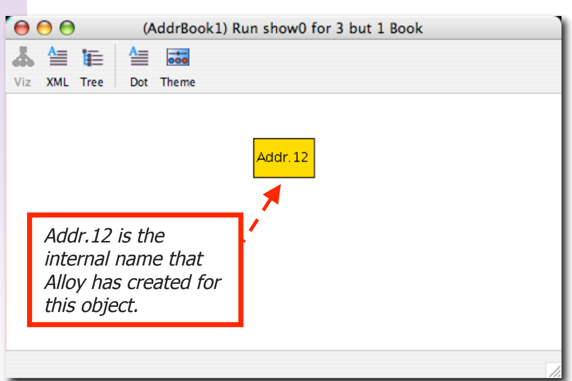
Scope settings determine the number of elements that generated examples will have.

User Interface



Results

Result of execution from previous slide…



Consult Alloy documentation for options for launching and adjusting visualizations.

Warning: output results will differ based on Alloy version, underlying SAT solver, etc.

- A valid, but boring instance (a single Addr object -- and nothing else).
- We told Alloy the *max* size for signature sets, but said nothing about the *minimum* size.
- Often need to add constraints that force the example instance to have at least a certain number of elements, etc.

Generating Instances (Take 2)

Be more precise in scope settings…

```
pred show0 () {
 // constraints
}
```

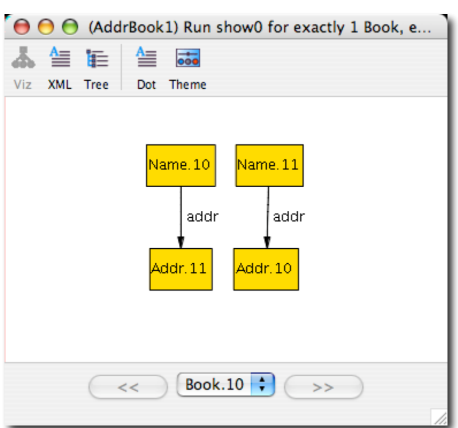
…still no constraints here

```
run show0 for exactly 1 Book,
exactly 2 Addr, exactly 2 Name
```

*Use **exactly** keyword to specify precise number of elements from each signature.*

Results (Take 2)

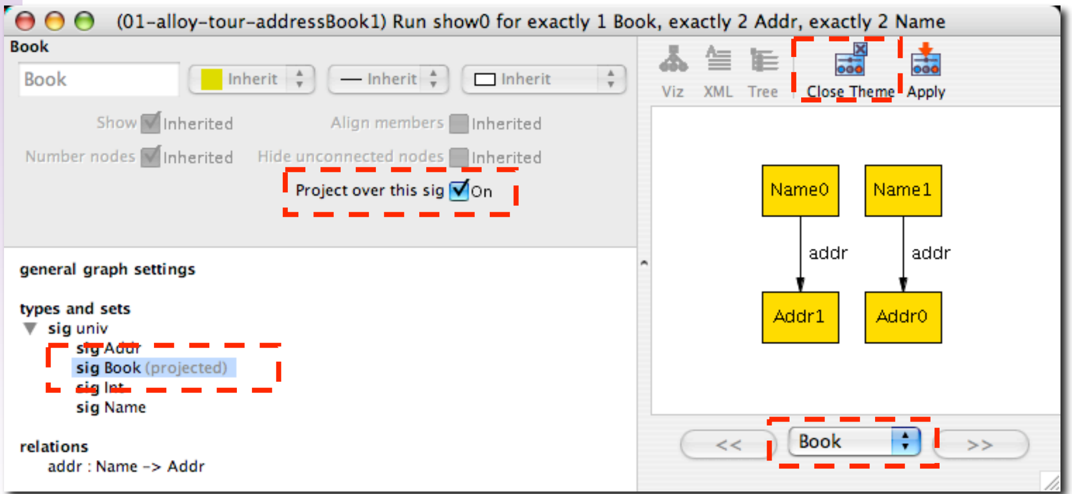
Result of execution from previous slide…



Settings: Goto Theme, then Book, then turn on *Project over this sig*

- Since I have visualization settings on *Project over Book* (see next slide), we see the relations that result for each Book instance and the Book instance is not shown (there is only one in this case)
- We see that, as requested, we have exactly two Names, two Addrs with the addr relation mapping between them.

Visualization Configuration

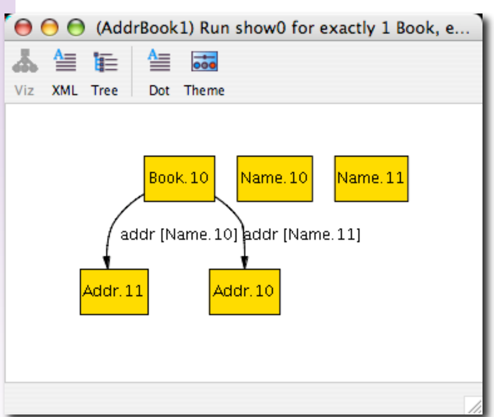


In visualization window, click on theme (drag to expand windown), select Book sig, and turn on "Project over this sig"

This focuses your attention on relationships and properties associated with a single book -- a separate diagram is created for each instance of Book (and suppresses Book objects from the visualization)

Results (Take 2)

Result of execution from previous slide (alternate visualization)



Settings: Goto Theme, then Book, then turn off *Project over this sig*

- I've deactivated *Project over Book,* so you can see the Book object directly
- The two Names *appear to be unconnected…*
- …but in fact, addr[Name.10] shows us that the instance Name.10 is being used as the "key" in the addr relation to lookup Addr.11
- Don't get too hung up over the appearance of these visualizations -- one can imagine other strategies -- these are just what the Alloy developers have chosen to implement

Generating Instances (Take 3)

Add constraints to direct the analysis…

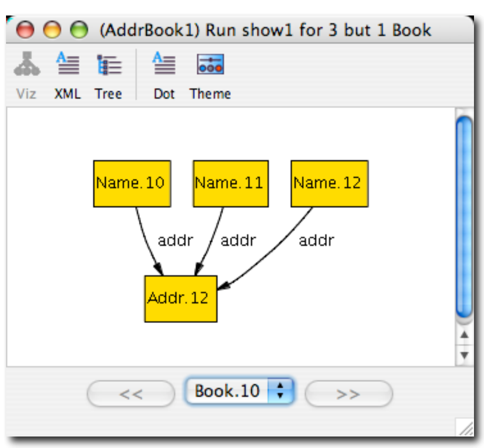
```
pred show1 (b: Book) {
  #b.addr > 1
  }

run show1 for 3 but 1 Book
```

Introduce a constraint that requires the existence of some Book b, such that the number of entries in the addr map for b is > 1.

Results (Take 3)

Result of execution from previous slide…



Settings: Project over Book sig (as explained on previous slides)

- Since I have visualization settings on *Project over Book*, we see the relations that result for each Book instance and the Book instance is not shown (there is only one in this case)
- We requested an instance that, for some book, had more than one addr table entry. If fact, this instance has three addr table entries -- with three different names that all point to the same address.

Generating Instances (Take 4)

Question: Does our model allow one name to map to two addresses?

```
pred show2 (b: Book) {
  #b.addr > 1
  some n: Name | #n.(b.addr) > 1
}
```

There exists a Name n…

…such that if we take b's addr map (b.addr) and apply that map with n as the key, we get a result that has more than one element.

```
run show2 for 3 but 1 Book
```


Results (Take 4)

Result of execution from previous slide…

```
Executing "Run show2 for 3 but 1 Book"
   Sig this/Book scope <= 1
   Sig this/Name scope <= 3
   Sig this/Addr scope <= 3
   Sig this/Book in [[Book.10]]
   Sig this/Addr in [[Addr.10], [Addr.11], [Addr.12]]
   Sig this/Name in [[Name.10], [Name.11], [Name.12]]
   Solver=minisat Bitwidth=4 Symmetry=ON
   225 vars. 20 primary vars. 472 clauses. 36ms.
   No instance found. Predicate may be inconsistent. 580ms.
```

- The analyzer tells us that it could not find an instance/example that satisfied the constraints in the show2 predicate.
- This is expected because our declaration of addr stated that the map could only yield 0 or 1 (lone) elements of Addr for each Name key.

```
…
addr: Name -> lone Addr
…
```

At most one Addr output for each Name key

Generating Instances (Take 5)

Time for a sanity check: does our address book even allow for more than one Addr value if we consider the total number of Addr's across all the Name's in the addr table?

```
pred show3 (b: Book) {
  #b.addr > 1
  #Name.(b.addr) > 1
}
```

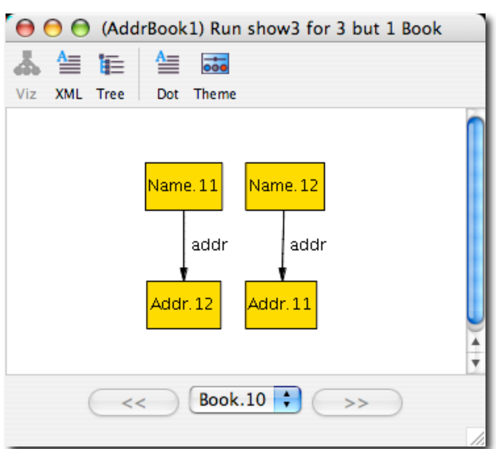
Take the set of all Names…

…apply the addr mapping from Book b to form a set consisting of all Addr that have a key from Names (i.e., the set of all Addr that may result from lookups via b.addr)

```
run show3 for 3 but 1 Book
```


Results (Take 5)

Result of execution from previous slide…



- The analyzer tells us that, indeed, we have not "over-constrained" the model specification: the model is *not* limited to 0 or 1 Addrs
- …Rather lookup of *each name* will yield 0 or 1 Addrs.

Conclusions

- Alloy provides a modeling environment that
  - enables high-level declarative specification of primary system entities and relationships between those entities
  - a data language that follows the OO paradigm
  - a powerful constraint language for imposing conditions on data and relationships
  - an analysis engine that lets of query our models in a variety of ways for the purpose of understanding and refining them
- In this lecture, we saw how to…
  - define the data component of a very simple address book application
  - query and visualize model instances
- Next lecture…
  - dynamic models -- modeling operations that transform modeled data structures

For You To Do…

- Load the file `alloy-tour-addressBook1.als` and run and visualize each of the examples from this lecture.
- Use constraints and/or scope settings to create an instance of the Address Book model with four names and two addresses. Is every Name in your model associated with an Addr?
- Add the following constraint to your show predicate:
  - `all n: Name | some n.(b.addr)`
- What's the intuition behind this predicate? Re-analyze and visualize the example.
- Change the definition of the addr field of Book to create a modified model that reads as follows
  - `addr: Name lone->lone Addr`
- What is the intuition behind the use of the lone multiplicity on both sides of the relation? Analyze the modified model with the additional constraints specified above and with the 1 Book, 4 Name, 2 Addr scope, and visualize. What are the results? Explain why these results are obtained.
- Can you adjust the scope to obtain a different outcome for the scenario above?

Acknowledgements

- The material in this lecture is based on Section 2.1 from…
  - *Software Abstractions: Logic, Language, and Analysis,* Daniel Jackson, MIT Press, 2006.