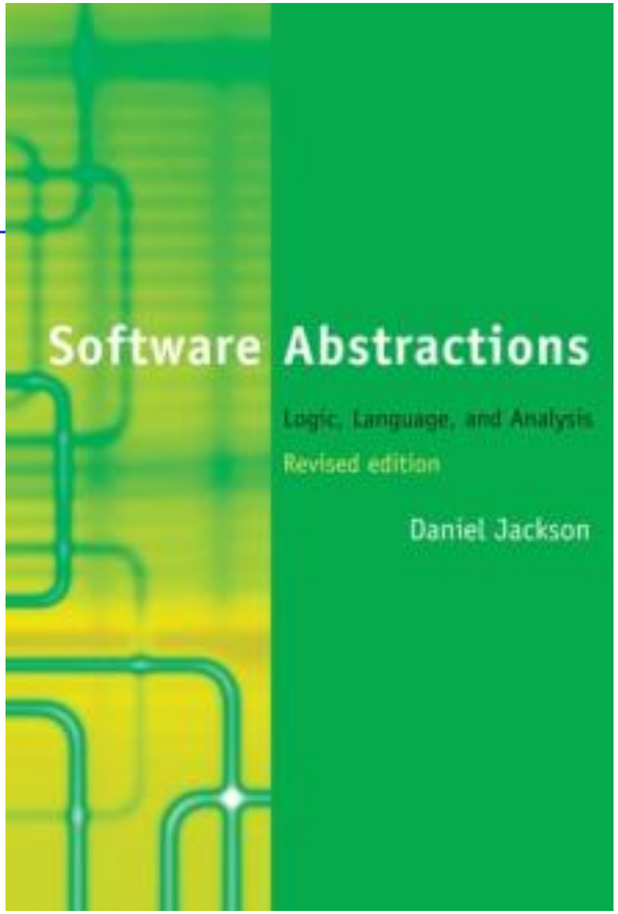
Alloy

Alloy's Resources

- The book is the main resource
- Many examples, and online tutorial
- Everything (code, documentation, tool) on:

http://alloy.mit.edu/



"Some slides are adapted from
Greg Dennis and Rob Seater
Software Design Group, MIT"

Outline

- Alloy
  - ✓ Introduction
  - ✓ Logic
  - ✓ Language

- Alloy Tool
  - All comes through the examples using the tool

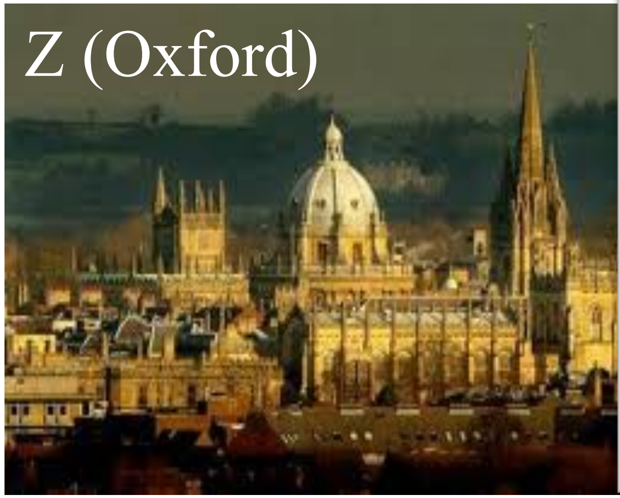
Alloy

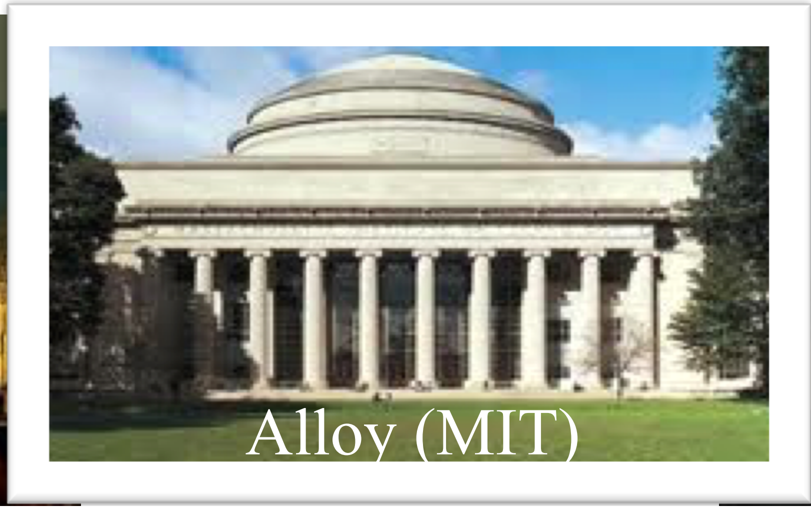
- Alloy is a formal notation for specifying models of systems and software
  - ✓ Looks like a declarative OO language
  - ✓ But also has a strong Mathematical foundation

- Alloy comes with a tool support to simulate specifications and perform property verification

Why Alloy is Important?

- Alloy is declarative
- It has been created to offer an expressive power similar to Z language as well as the strong automated analysis of SMV model checker





Alloy for Declarative Modeling

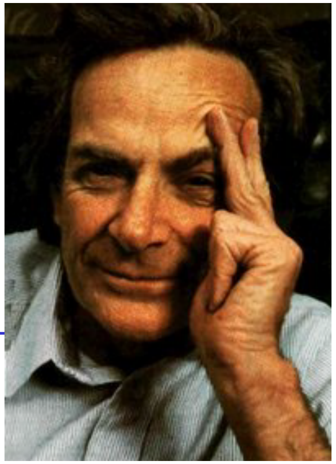
- Alloy is used for abstractions and conceptual modeling in a declarative manner
- ***Declarative approach to programming and modeling****:*
  - "declarative programming is a programming paradigm that expresses the logic of a computation without describing its control flow"
  - "describing *what* the program should accomplish, rather than describing *how* to go about accomplishing it"

Alloy's Applications for SE

- In RE Alloy can be used to
  - formally describe the domain and its properties, or
  - operations that the machine has to provide
- In software design to formally model components and their interactions

Alloy for Automated Analysis

- Any real-life system has a set of properties and constraints
- Specification analysis can help check whether or not the properties will be satisfied by the systems and the constraints will never be violated
- "*The first principle is that you must not fool yourself, and you are the easiest person to fool.*"
  ~ Richard P. Feynman



What is Alloy?

- First order predicate logic + relational calculus
- Carefully chosen subset of relational algebra
  - uniform model for individuals, sets and relations
  - no higher-order relations
- Almost no arithmetic
- Modules and hierarchies
- Suitable for small, exploratory specifications
- Powerful and fast analysis tool
  - is this specification satisfiable?
  - is this predicate true?

Let's Get Started!

- Imagine that we are asked to model a very simple address book
- The books that contain a bunch of addresses linked to the corresponding names

What to model?

How to model?

AddressBook in Alloy

- Name and Addr are two entities
- Book is an entity
- addr is linking Name to Addr within the context of Book (it is a ternary relation b->n->a)
- The keyword “lone”: each Name can correspond to at most one Addr

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

Logic: Everything is a Relation

- Sets are unary (1 column) relations

```
Name = {(N0),        Addr = {(A0),        Book = {(B0),
        (N1),                (A1),                (B1)}
        (N2)}                (A2)}
```

- Scalars are singleton sets

```
myName   = {(N1)}
yourName = {(N2)}
myBook   = {(B0)}
```

- Ternary relation

```
addr = {(B0, N0, A0),
        (B0, N1, A1),
        (B1, N1, A2),
        (B1, N2, A2)}
```

what is myName.(Book.addr)?

myName.(Book.addr)

```
Book = {          addr = {                Book.addr = {
    (B0),             (B0, N0, A0),           (N0, A0),
    (B1)}             (B0, N1, A1),           (N1, A1),
                      (B1, N1, A2),           (N1, A2),
                      (B1, N2, A2)}           (N2, A2)}
```

```
myName = {        Book.addr = {           myName.(Book.addr) = {
    (N1)}             (N0, A0),                (A1),
                      (N1, A1),                (A2),}
                      (N1, A2),
                      (N2, A2)}
```

Static Analysis

- Let’s open the Alloy tool and play a bit!
- We add an empty constraint “show” by using keyword “pred” to find the instances of the entities involved in the modeling
- In this case, the state exploration is limited to 3 for each entity except Book that is set to 1

```
pred show {}
run show  for 3  but 1 Book
```

Static Analysis

- Adding a constraint on the number of (Name,Address) relations in a given book

```
pred show [b: Book] {
  #b.addr > 1
  }
```

- Constraining the number of names

```
pred show [b: Book] {
  #b.addr > 1
  #Name.(b.addr) > 1 }
```

- Can we fulfill the constraint below?

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

Dynamic Analysis

- Let’s model some operations.
  - A predicate that adds an address and name to a book

```
pred add [b, b’: Book, n: Name, a: Addr] {
    b’.addr = b.addr + n -> a
}
```

  - b and b’ model two versions of the book, respectively before and after the operation

- In the tool we can invoke an operation

```
pred showAdd [b, b’: Book, n: Name, a: Addr] {
   add [b, b’, n, a]
   #Name.(b’.addr) > 1
}

run showAdd
```

Dynamic Analysis

- Deleting a name from the address book

```
pred del [b, b': Book, n: Name] {
        b'.addr = b.addr - n->Addr
}
```

Assertions and counterexamples

- What happens if we run a delete after an add predicate? Will this take us to the initial state?

```
assert delUndoesAdd {
        all b, b', b'': Book, n: Name, a: Addr |
                add [b, b', n, a] and del [b', b'', n]
                implies
                b.addr = b''.addr
}
check delUndoesAdd for 3
```

- Counterexample is a scenario in which the assertion is violated
- While checking an assertion, Alloy searches for counterexamples
- Do we find a counterexample in this case?

Resolving the Counterexample

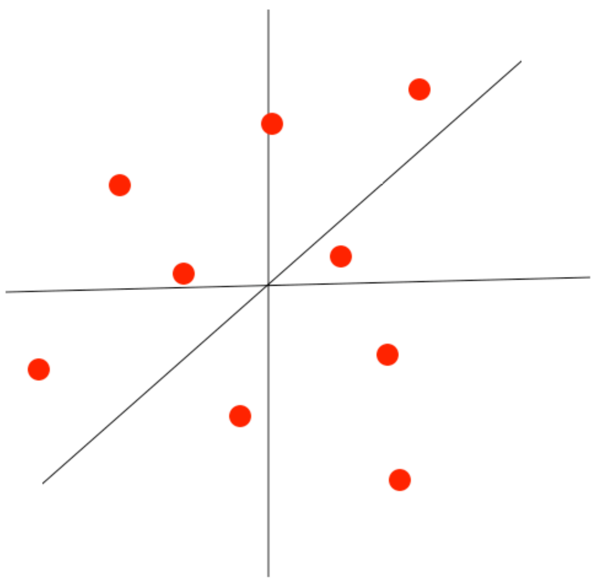
- … Yes! When add tries to add a name that already exists and delete will delete the name and the corresponding address
- Here is how we can solve the problem:

```
assert delUndoesAdd {
        all b, b', b'': Book, n: Name, a: Addr |
                no n.(b.addr) and add [b, b', n, a] and del [b', b'', n]
                implies
                b.addr = b''.addr
}
```

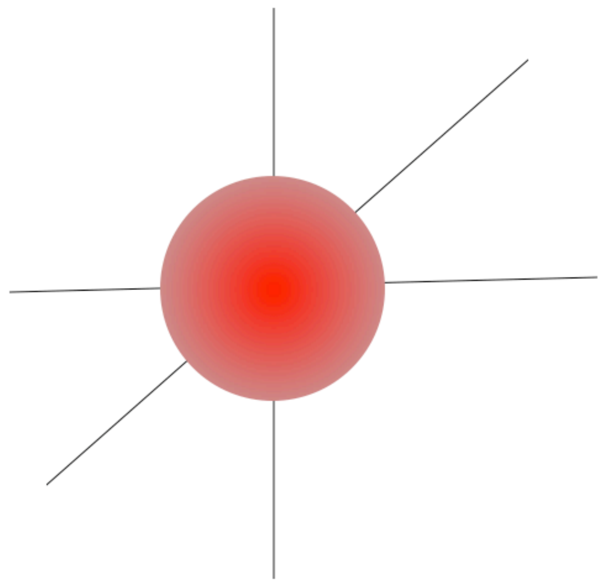
Counterexample in Alloy

- The analysis by Alloy is always bounded to a defined scope
- This means that there is NO guarantee that the assertions always hold if no counterexample is found in a certain scope!

Testing vs Counterexample



testing:
a few cases of arbitrary size



scope-complete:
all cases within a small bound

Functions: An Example

- Can we get the addresses in the book?

```
fun lookup [b: Book, n: Name] : set Addr {
        n.(b.addr)
}
```

- Example of usage of lookup

```
assert addLocal {
        all b, b': Book, n, n': Name, a: Addr |
                add [b, b', n, a] and n != n'
                implies
                lookup [b, n'] = lookup [b', n']
}
check addLocal for 3 but 2 Book
```

Alloy: Syntax and Semantics

- Alloy is a language
  - It has a syntax *how do I write a right specification?*
  - It has a semantics *what does it mean?*
- In a programming language
  - Syntax defines correct programs (i.e allowed programs)
  - Semantics defines the meaning of a program as its possible (many?) computations
- In Alloy
  - Syntax as usual...
  - Semantics defines the meaning of a specification as the collection of its models, i.e., of the worlds that make our Alloy description true