The Logic of Alloy

Curt Clifton
Rose-Hulman Institute of Technology

Labels in the Visualizer

- Look at example model addressBook1g.als
- What is the counterexample for delUndoesAdd?

See Task 25 in the Alloy Lab

From HW1: Hope to Learn

- Practical techniques
- Why disco matters

Formal methods—the BFG of development techniques

http://www.wickedcoolstuff.com/

From HW1: Concerns

- Difficulty of course
- Remembering Disco
- Doing proofs
- Course load (at least 3 taking PLC now)
- Notation
- Relevance

Goals for Today

- Remember some set theory ideas and see how they're used in Alloy
- See how we can use relations to represent design models

The Logic of Alloy

- **Everything** is either an atom or a relation
  - Relations are sets of tuples
  - Like tables in relational databases

Atoms

- Indivisible
- Immutable
- Uninterpreted
- Examples:
  - Book$1, Name$0, Addr$2

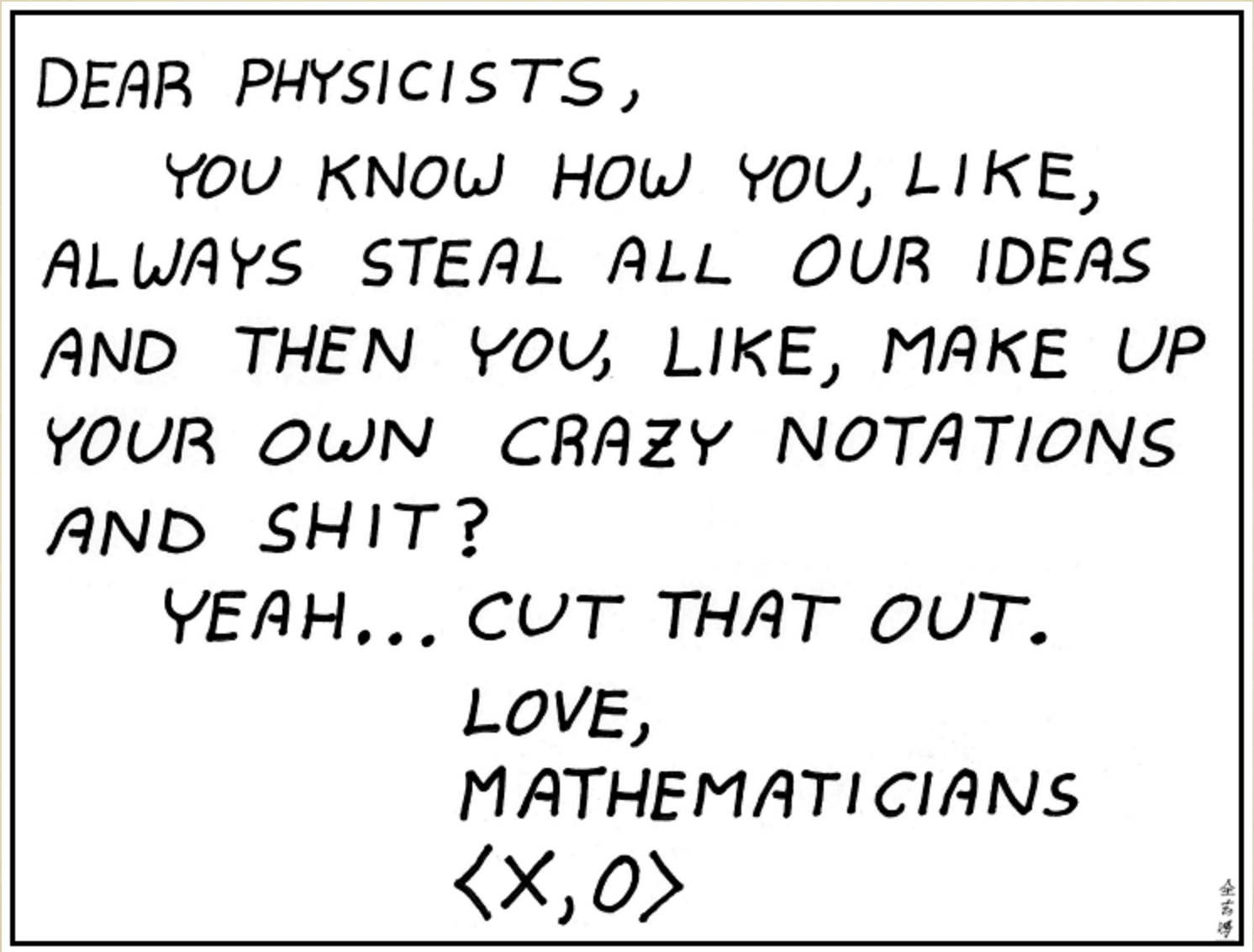
To say **anything** interesting about atoms in Alloy we use relations

Relations

- A set of tuples
- Terms:
  - Size—number of tuples (or rows)
  - Arity—length of tuples (or no. of cols.)
    - Unary means arity of 1
    - Binary means arity of 2
    - Ternary means arity of 3

Q2,3

Cartoon of the Day



http://abstrusegoose.com/128

Abuse of Notation–1

- Sets in alloy?
  - Just unary relations
  - Examples:
    - Name = {(Name$0), (Name$1), (Name$2)}
    - Addr = {(Addr$0), (Addr$1)}

Alloy

math

Q4

Abuse of Notation–2

- Scalars in alloy?
  - Just singleton, unary relations
  - Examples:
    - myName = {(Name$0)}
    - yourAddr = {(Addr$1)}

Abuse of Notation–3

- **Options in Alloy**
  - Represented as a relation that is a scalar or is empty
  - Example:

```
sig Addr {}
sig Message {
        replyTo: set Addr
        ...
}
fact {
        all m: Message | #m.replyTo = 0 or #m.replyTo = 1
}
```

Abuse of Notation–4

- **Tuples in Alloy**
  - **Just singleton, non-unary relations**
  - **Example:**
    - myName = {(Name$0)}
    - myAddr = {(Addr$0)}
    - myAssociation = {(Name$0, Addr$0)}

Why all the abuse?

- Allows uniform use of almost all operators
  - Very few special cases
  - Very few type errors
  - No type casts/conversions
- OK, because no "higher-order" relations

Q6,7

Three Styles in Alloy

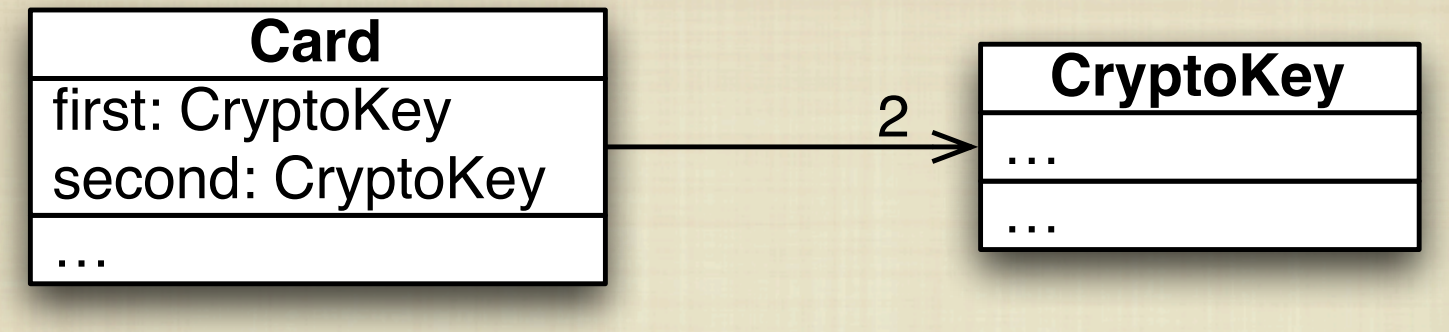
Don't be scared by the $5 words!

- Predicate calculus style:
  - **all** n: Name, d, d': Addr |
    n->d **in** addr **and** n->d' **in** addr **implies** d=d'
- Navigation expression style:
  - **all** n: Name | **lone** n.addr
- Relational calculus style:
  - **no** ~addr.addr - **iden**

Relations for Describing Structure

Just a sketch so you can see where we're headed.

Hotel Key Cards

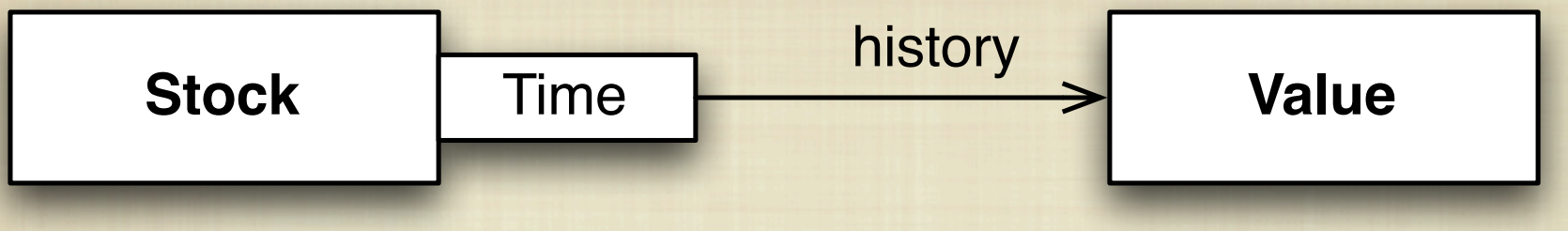



```
sig Card {
    first: CryptoKey,
    second: CryptoKey
}

sig CryptoKey {}

fact {
    all c: Card | c.first != c.second
}
...
```

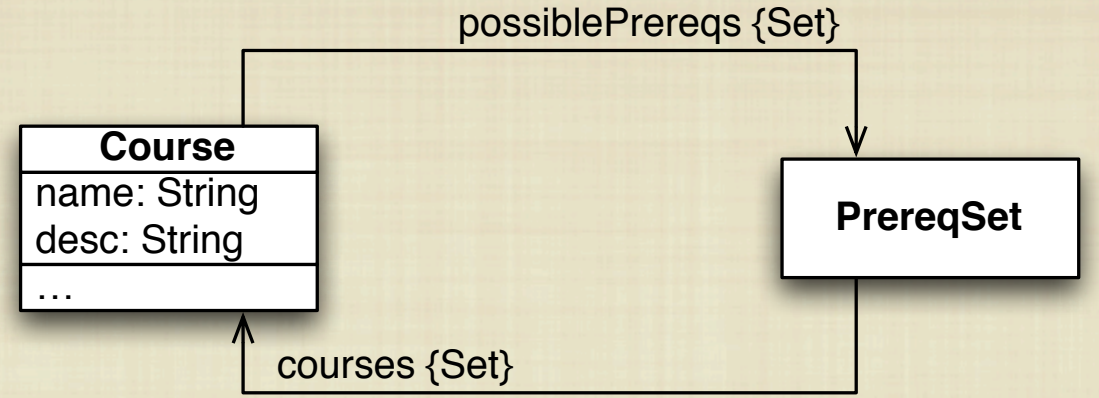
Stock Values over Time




```
sig Stock { history: Time -> Value }
sig Value {}
sig Time {}

fact {
    all s: Stock, t: Time | #s.history[t] <= 1
}
...
```

Course Prerequisites




```
sig Course { prereqs: set PrereqSet }
sig PrereqSet { courses: set Course }

fact {
    // a course may not be its own prereq, directly or transitively
    all c: Course | c not in c.^(prereqs.courses)
    // only allow prereq sets that are non-empty and used by some course
    all p: PrereqSet | some p.courses and p in Course.prereqs
}
...
```

Alloy models of examples for your reference

Hotel Key Cards

```
sig Card {
    first: CryptoKey,
    second: CryptoKey
}

sig CryptoKey {}

fact {
    all c: Card | c.first != c.second
}

pred show {}

run show for 4 but 2 Card
```

Stock Values over Time

```
sig Stock { history: Time -> Value }
sig Value {}
sig Time {}

fact {
    // A stock has just one value for a given point in time:
    all s: Stock, t: Time | #s.history[t] <= 1
}

pred show {
    #Stock = 3
}

run show for 4
```

Course Pre-reqs

```
sig Course { prereqs: set PrereqSet }
sig PrereqSet { courses: set Course }

fact {
    // a course may not be its own prereq, directly or transitively
    all c: Course | c not in c.^(prereqs.courses)
    // only allow prereq sets that are non-empty and used by some course
    all p: PrereqSet | some p.courses and p in Course.prereqs
}

pred show {
    // ensures that some prereq consists of more than 2 course,
    // just to make things interesting
    some p: PrereqSet | #p.courses > 2
}

run show for 6
```