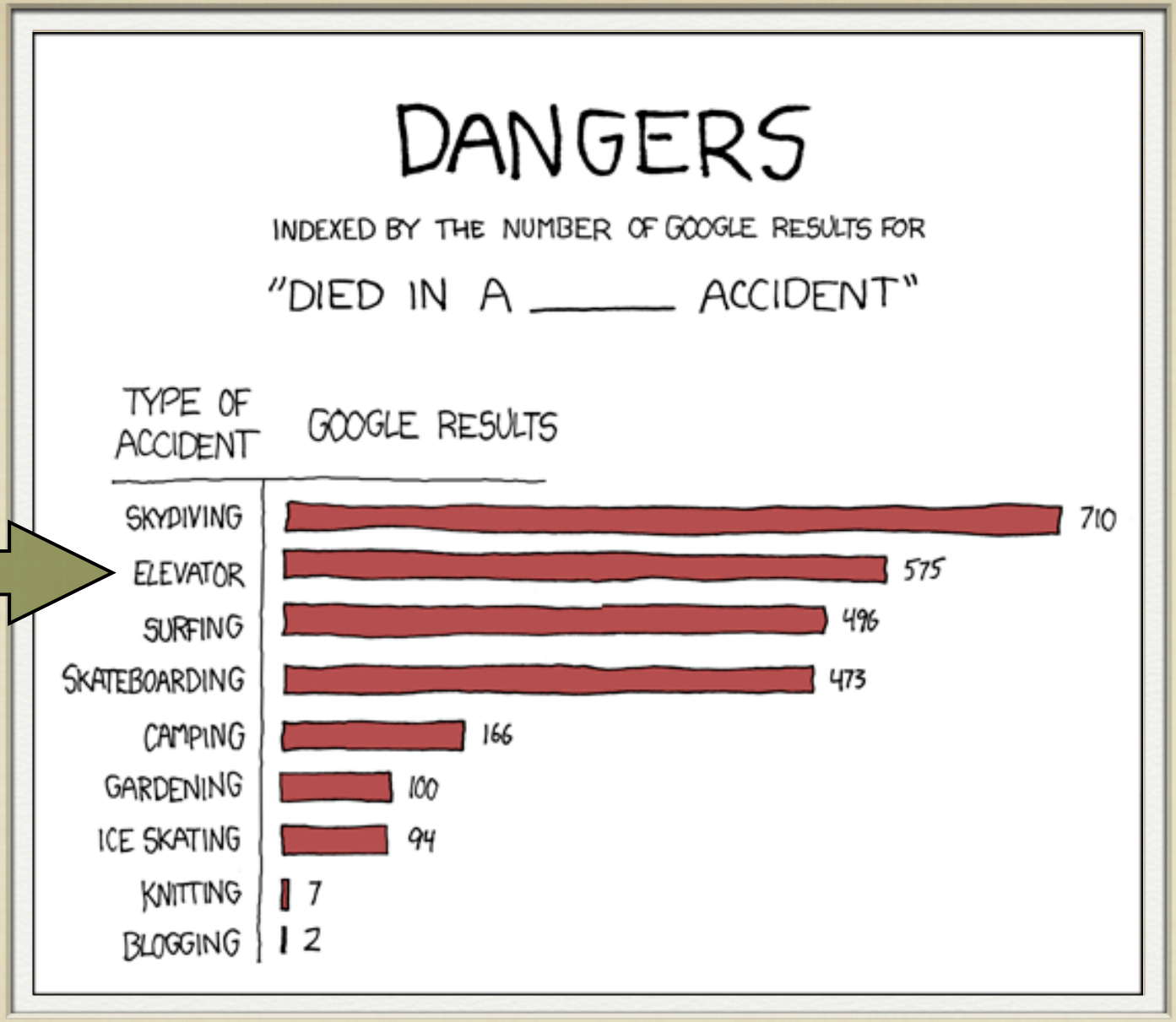
Alloy Commands and Modules

Curt Clifton
Rose-Hulman Institute of Technology

Goals for Today

- Discuss project details
- Understand details of "running" models
- See how to reuse Alloy models
- Understand why we generally avoid integers and booleans in Alloy



Zero results:

'snake charming' and 'haberdashery'

Project 1

- Due Tuesday at midnight
- Initial Alloy model of elevator system

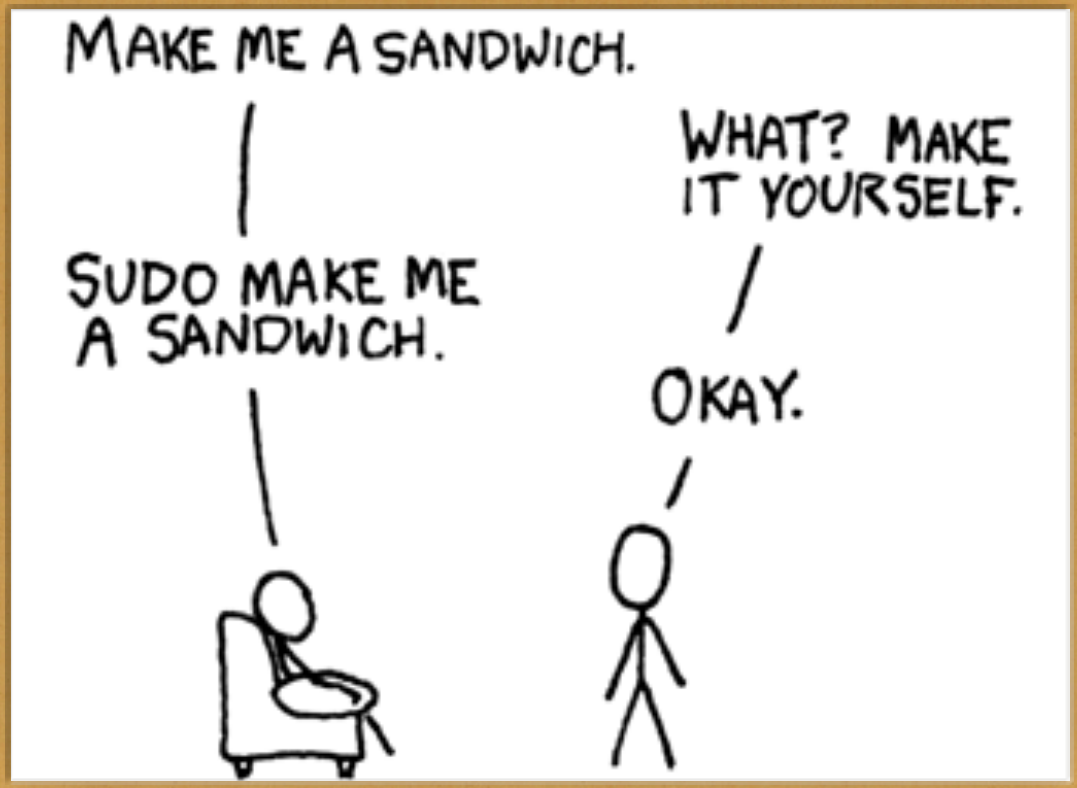
Project 1 Specification

- one lift, five floors, no floor call buttons
- The lift has a set of buttons, one for each floor. These illuminate when pressed and cause the lift to visit the corresponding floor. The illumination is cancelled when the corresponding floor is visited by the lift.
- The lift has two arrows indicating future direction. One arrow is illuminated as the doors open. The illumination is cancelled as the doors close.

Project Teams

- 01: anderswc,mcginnda,stamptd
- 02: covertcj,frankmp,shermabj
- 03: freemacc,scherepn
- 04: gaot,kehmbv
- 05: glowskst,wattsbn
- 06: hollintl,jonesjg
- 07: theisje,wellska1
- 08: fullerra,manndj,siegleal
- 09: banksda,haffnedm,purviatt
- 10: dashjb,mankeap,mendelnt
- 11: hinesen,mosttw,orlowsap
- 12: lint,mlynarcs,spiegekr

http://svn.csse.rose-hulman.edu/repos/csse373-201130-projteam**NN**



http://xkcd.com/149/

Commands & Scope

Two Commands

- **run** – search for an instance of a predicate
- **check** – search for a counterexample for an assertion

Scope

- Not specified?
  - Command uses default:
    Up to 3 of each **top-level** signature
- Can bound mix of top-level and sub-signatures
  - As long as bounds are complete...

Choosing the Scope

- Small system? Scope might match real world
  - E.g., exactly 3 elevators, exactly 5 floors
- Larger systems
  - Scope is set large enough for confidence
  - Small enough for speed

http://www.bigelowaerospace.com

Modules

Module System

- Declare module: **module** pathname
- Use module: **open** pathname
- Alloy uses actual location of main module to guess location of opened modules
  - Suppose this file is in C:\Desktop\main.als:
    - **module** filesystem/main
      open filesystem/debug
      open filesystem/library/dirmodel
      ...

Avoiding Name Clashes

- Use aliases:
  - **open** library/graphs **as** G
    ...
    pred Acyclic[] { G/Acyclic[parents] }

Parametric Modules

- Like generic types in C++ and C#
- Example declaration:
  - **module** library/tree[nodeType]

    pred isTree[r: nodeType -> nodeType] { ... }
    ...
- Example use:
  - **open** library/tree[Object]

Cryptophilus integer Heer

http://www.zin.ru/animalia/Coleoptera/eng/cryintkm.htm

Alloy Integers

Alloy Integers

- Has Int atoms and int values
- Like Java's Integer wrapper class vs. int type
- int → Int
  - Alloy: Int[4]
  - Java: new Integer(4)
- Int → int
  - Alloy: int[e]
  - Java: int s = 0; for(Integer i : e) { s += i.intValue(); }

?!?

Integer Example

```
// Weighted Graph
sig Node {
    adj: Int lone -> Node
}
fact {
    all n: Node |
        let selfEdges = n.adj.n | int[selfEdges] = 0
}
run {} for exactly 3 Node
```

Alloy Integers are not very useful.

- What do you really need:
  - Unique IDs? Just use atoms and a fact to make them disjoint
  - Ordering? Use util/ordering to impose order on atoms

"If you think you need them, think again."

Booleans?

- Avoid them! Conflict with set logic and make constraints too "wordy".

- Use sub-signatures:

```
// BAD!
sig Lift { ...
      doorOpen: Time -> Boolean
}
```

```
// Good!
abstract sig DoorState{}
one sig Open, Closed extends DoorState{}
sig Lift { ...
      door: Time -> DoorState
}
```

- Use options:

```
// BAD!
sig Lift { ...
      upArrowLit: Time -> Boolean
      downArrowLit: Time -> Boolean
}
```

```
// Good!
abstract sig Direction{}
one sig Up, Down extends Direction{}
sig Lift { ...
      arrowLit: Time -> lone Direction
}
```

Q2