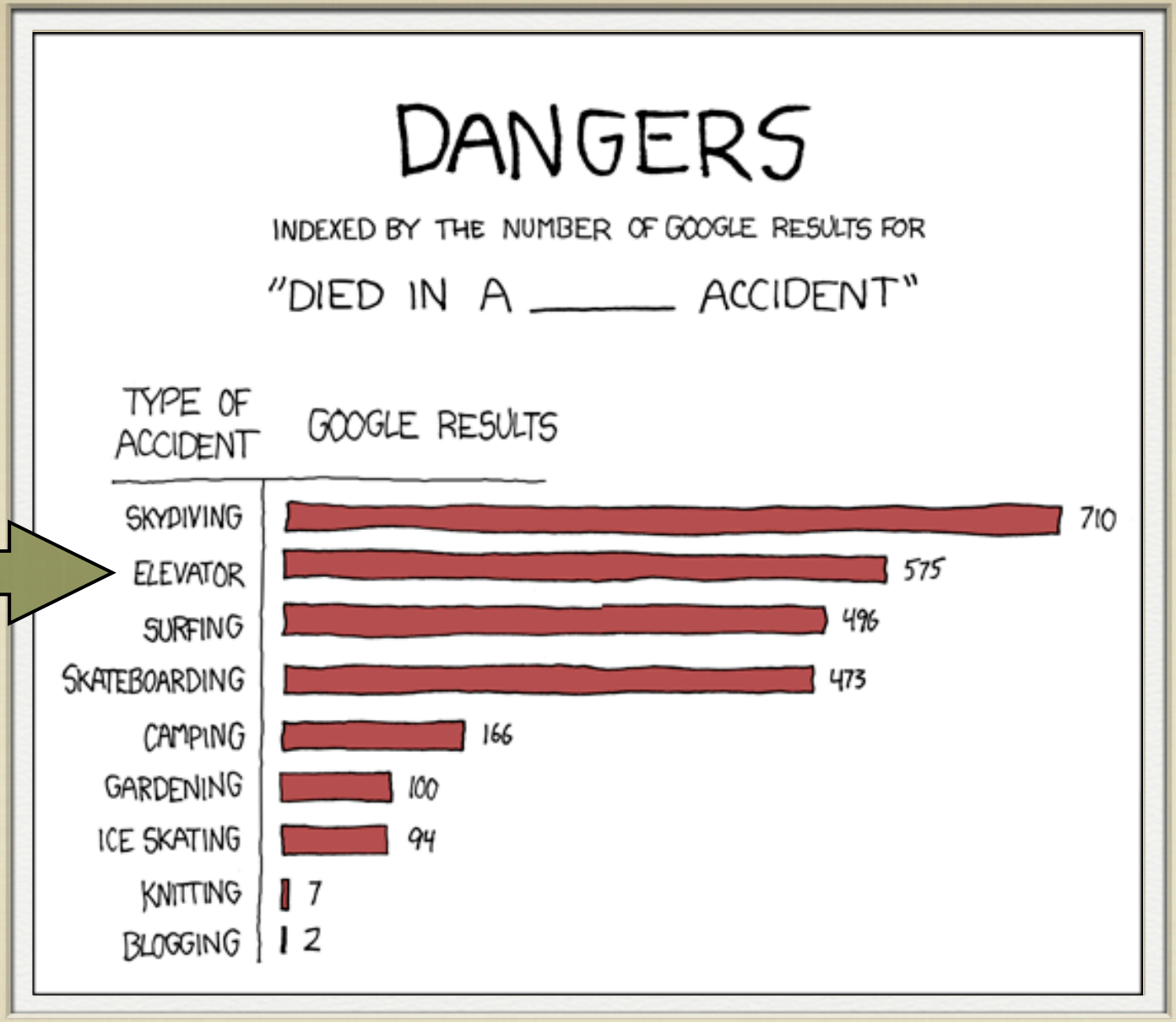
Alloy Modules, Integers

Curt Clifton
Rose-Hulman Institute of Technology

Goals for Today

- Discuss project details
- 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.

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++
- 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(); }

?!?

Q1

Integer Example

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

Why wait so long?

- Not actually very useful
- "If you think you need them, think again."
- What properties 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

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