Alloy Constraints

Curt Clifton
Rose-Hulman Institute of Technology

Homework Teams

Team **01**: mcginnda,wellska1

Team **02**: theisje,wattsbn

Team **03**: gaot,siegleal,freemacc

Team **04**: glowskst,dashjb,spiegekr

Team **05**: kehmbv,manndj,fullerra

Team **06**: stamptd,shermabj,hollintl

Team **07**: mosttw,anderswc,orlowsap

Team **08**: mendelnt,covertcj,mlynarcs

Team **09**: banksda,hinesen,mankeap

Team **10**: purviatt,frankmp,lint

Team **11**: haffnedm,scherepn,jonesjg

http://svn.csse.rose-hulman.edu/repos/csse373-201130-hwteamXX

Project Teams

- Project 1 is posted
  - Due Tuesday, March 29, 11:55pm
  - Teams of 2–3 people
  - Can form your own teams
  - Project teams disjoint from Homework ones
- Complete team survey on ANGEL by 6am, Monday, March 21.

Goals for Today

- Learn some more Alloy operators that we'll use to describe designs
- Learn how to write basic constraints about what "must be" true about a design

Relational Operators

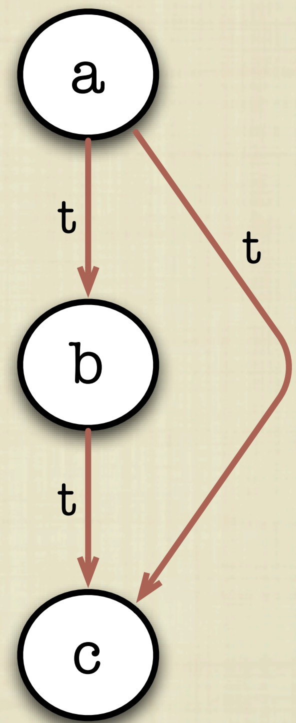
- Combining relations
  - -> – arrow product
  - . – dot join
  - [] – box join
- Reachability
  - ^ – transitive closure
  - * – reflexive-transitive closure
- "Modifying" relations
  - ~ – transpose
  - <: – domain restriction
  - :> – range restriction
  - ++ – override

"Modifying" Relations: Override

- p ++ q – override
  - Like union, but tuples in q replace "matching" tuples from p
  - "Matching" means first elements are the same
  - Yields a new relation
  - p and q must have same arity
  - Useful for modeling mutation

Transitive Relation

- **Transitive relation:**
  - Binary relation, t, such that if a→b and b→c are in t, then a→c also is



Q1,2

Transitive Closure

- **Transitive closure** of binary relation r:
  - Smallest transitive relation containing r
- In Alloy: ^r
  - Equivalent to r + r.r + r.r.r + ...

Q3

Reachability

- Transitive closure is used to express "reachability"
    - bacon.^appearedWith
- Related:
    - Reflexive transitive closure, *r
    - *r = ^r + iden

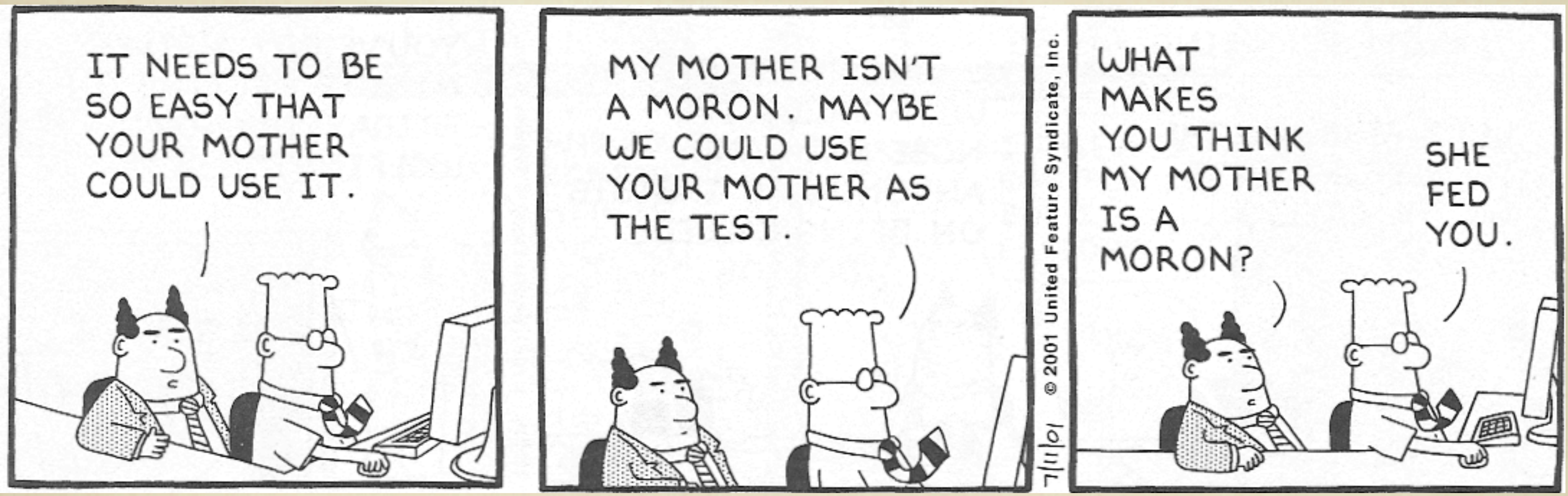
Kevin Bacon is zero degrees from Kevin Bacon

```
sig Actor {
  appearedWith: set Actor
}

pred degrees[bacon: Actor] {
  all a: Actor |
    a in bacon.^appearedWith
}

run degrees for 6
```

Cartoon of the Day



Constraints

- Like usual boolean operators
- else goes with implies:
  - C1 => F1 **else** F2
  - C1 => F1
    else C2 => F2
    else C3 => F3

Long form	Shorthand
not	!
and	&&
or	||
implies	=>
else	
iff	<=>

Q4

Quantification

Can declare multiple vars here

all x: e | F	F holds for every x in e
some x: e | F	F holds for some x in e
no x: e | F	F holds for no x in e
lone x: e | F	F holds for at most one x in e
one x: e | F	F holds for exactly one x in e

Think Less than or equal to one

Examples

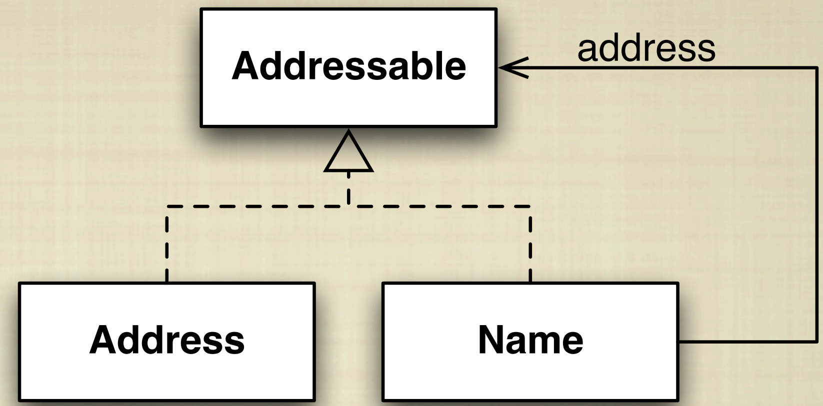
- **some** n: Name, a: Address |
  a **in** n.address

- **no** n: Name |
  n **in** n.^address

- **all** n: Name |
  lone d: Address |
  d **in** n.address

- **all** n: Name |
  no disj d, d’: Address |
  d + d’ **in** n.address

“DISJOINT”




```
sig Address { }
sig Name {
    // multi-level address book
    address: set (Name + Address)
}
```

Q5

Quantified Expressions

- **some** Name
- **some** address
- **no** (address.Addr - Name)
- **all** n: Name | **lone** n.address

```
sig Address { }
sig Name {
    // multi-level address book
    address: set (Name + Address)
}
```

Q6

Let expressions

- **let** x = e | A
- Just a shorthand to avoid writing out e multiple times
- **all** a: Alias |
  let w = a.workAddress |
  a.address = (**some** w => w **else** a.homeAddress)

Variable and Formal Declarations

- name: expression
  - name is a subset of the relation given by expression
- Examples:
  - address: Name->Addr
  - addr: Book->Name->Addr
  - address: Name->(Name + Addr)
  - workAddress, homeAddress: Alias->Addr
    prefAddress: workAddress + homeAddress

Set Multiplicities

- Used to constrain the possible subsets that a variable can be
  - x: **set** e – x can be any subset of e
  - x: **one** e – x is a singleton subset of e (i.e., an Alloy scalar)
  - x: **lone** e – x is an option, either empty set or a scalar
  - x: **some** e – x is a non-empty subset of e

Careful: if e is a unary relation (i.e., a set), then x: e is equivalent to x: **one** e

Relation Multiplicities

- Too bizarre for words
- Almost
- r: A m -> n B means:
  - **Each** member of A maps to **n** members of B
  - and for **each** member of B, **m** members of A map to it

Examples:

r: A -> **one** B
r: A **one** -> B
r: A -> **lone** B
r: A **one** -> **one** B
r: A **some** -> **some** B

```
sig Thing, OtherThing {}

pred relMult[r: Thing some -> some OtherThing]  {}
run relMult for 3
```

Q8

Cardinality Constraints

- #e gives the size (number of tuples) in the relation given by e
- Can use all regular integer operations on the result
- Can use 1, 2, 3, ... as constants
- **sum** x: e | ie means $\sum_{x \in e} ie$

Next Time

- More examples
- Building our own Alloy models