# Alloy
# Fields and Types

Curt Clifton
Rose-Hulman Institute of Technology

# Where we've been

- Logic of Alloy
  - We looked at (and practiced) writing logical statements in Alloy
  - These statements are boolean valued constraints about relations

HW3, due Thursday is another chance to practice this

# Where we are

- Looking at how to create models in Alloy
  - We saw an example of all the pieces last time
  - We started looking at the details of signatures

# Goals for Today

- Understand how to declare fields inside signatures
- Learn enough about Alloy types to be able to fix type errors

# Fields

# Fields Declare Relations

- **sig** A {f: e} declares
  - A signature A
  - And a relation f: A -> e, such that **all this**: A | **this**.f **in** e
- **sig** A {f: m e} says **all this**: A | **this**.f **in** m e
- Examples...

# Field Examples

- **abstract sig** Object { }
  **sig** Directory **extends** Object { contents: **set** Object }
  **one sig** Root **extends** Directory { }
  **sig** File **extends** Object { }
  **sig** Alias **extends** File { to: Object }

- **sig** Forecast { weather: City -> **one** Weather }
  **sig** City, Weather { }
  **one sig** Rainy, Sunny, Cloudy **extends** Weather { }

- **sig** TrafficLight { color: Color **some** -> Time }
  **abstract sig** Color { }
  **one sig** Red, Green, Yellow **extends** Color { }
  **sig** Time { }

Q1,2

# Grouping Fields

- A cat has three names:
  - **sig** Cat { **disj** daily, peculiar, ineffable: Name }
    **sig** Name { }

# Dependent Declarations

- A cat still has three names:
  - **sig** Cat {
    daily: Name,
    peculiar: Name - daily,
    ineffable: Name - (daily + peculiar)
    }


- Radio station with translators:
  - **sig** RadioStation {
    owns: **set** Freq,
    freq: Location -> **one** owns
    }
    **sig** Freq, Location {}

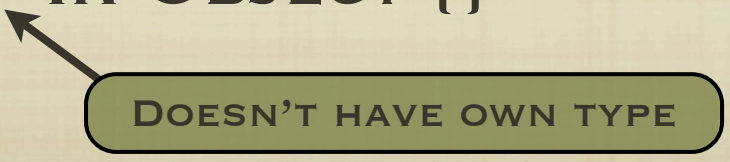
# Types

# Types in Alloy

- Two purposes:
  - Detect redundancy – usually indicates a misconception
  - Resolve overloading
- Different than most programming languages:
  - No false alarms and no guarantees

# Basic Types

- Introduced by signatures
- Can **overlap:**
  - Two basic types **overlap** if one is a subtype of the other
  - Example:
    - **sig** Object {}
      **sig** File, Directory **extends** Object {}
      **sig** Temp **in** Object {}




Q6

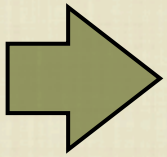
# Relational Types

- Every expression gets a **relational type**
  - A union of products:
    - $A_1$ -> $B_1$ -> ... +
      $A_2$ -> $B_2$ -> ... +
      $A_3$ -> $B_3$ -> ... +
      ...
    - where $A_1$, $B_1$, ..., are basic types
- Inferred automatically

# Type Inference

## 1. Gets basic types from signatures

- **abstract sig** Object {}
  **sig** Directory **extends** Object {...}
  **one sig** Root **extends** Directory {}
  **sig** File **extends** Object {}
  **sig** Alias **extends** File {...}
  **sig** Temp **in** Object {}



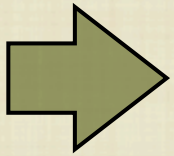
**Basic Types:**
Object,
Directory, Root,
File, Alias

# Type Inference

1. Gets basic types from signatures

2. Each field is given a type

```
...
sig Directory extends Object {
  contents: set Object
}
...
sig Alias extends File {
  to: Object
}
...
```



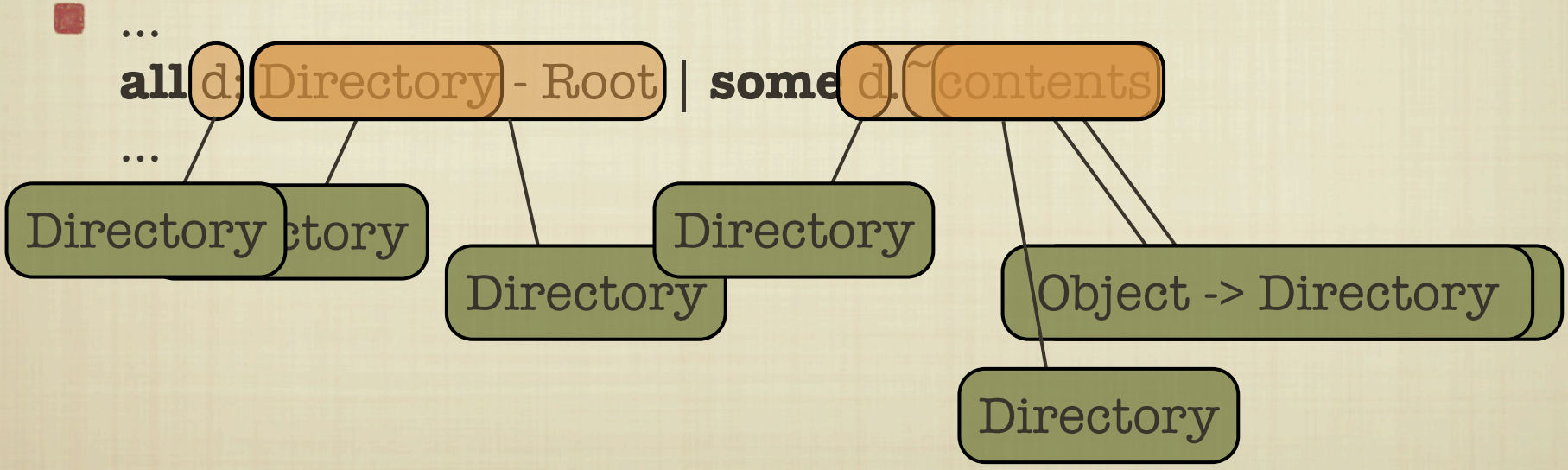
**Field Types:**

contents: Directory -> Object

to: Alias -> Object

# Type Inference

1. Gets basic types from signatures
2. Each field is given a type
3. Each expression gets a type



# Two Kinds of Type Errors

- Mixed Arity:
  - contents + Directory
  - Can't union set of pairs with unary set
- Redundancy...

# Redundancy

- **abstract sig** Object {}
  **sig** Directory **extends** Object { contents: **set** Object}
  **one sig** Root **extends** Directory {}
  **sig** File **extends** Object {}
  **sig** Alias **extends** File {to: Object}

- Directory & Alias — **Error!**

- Alias.contents — **Error!**

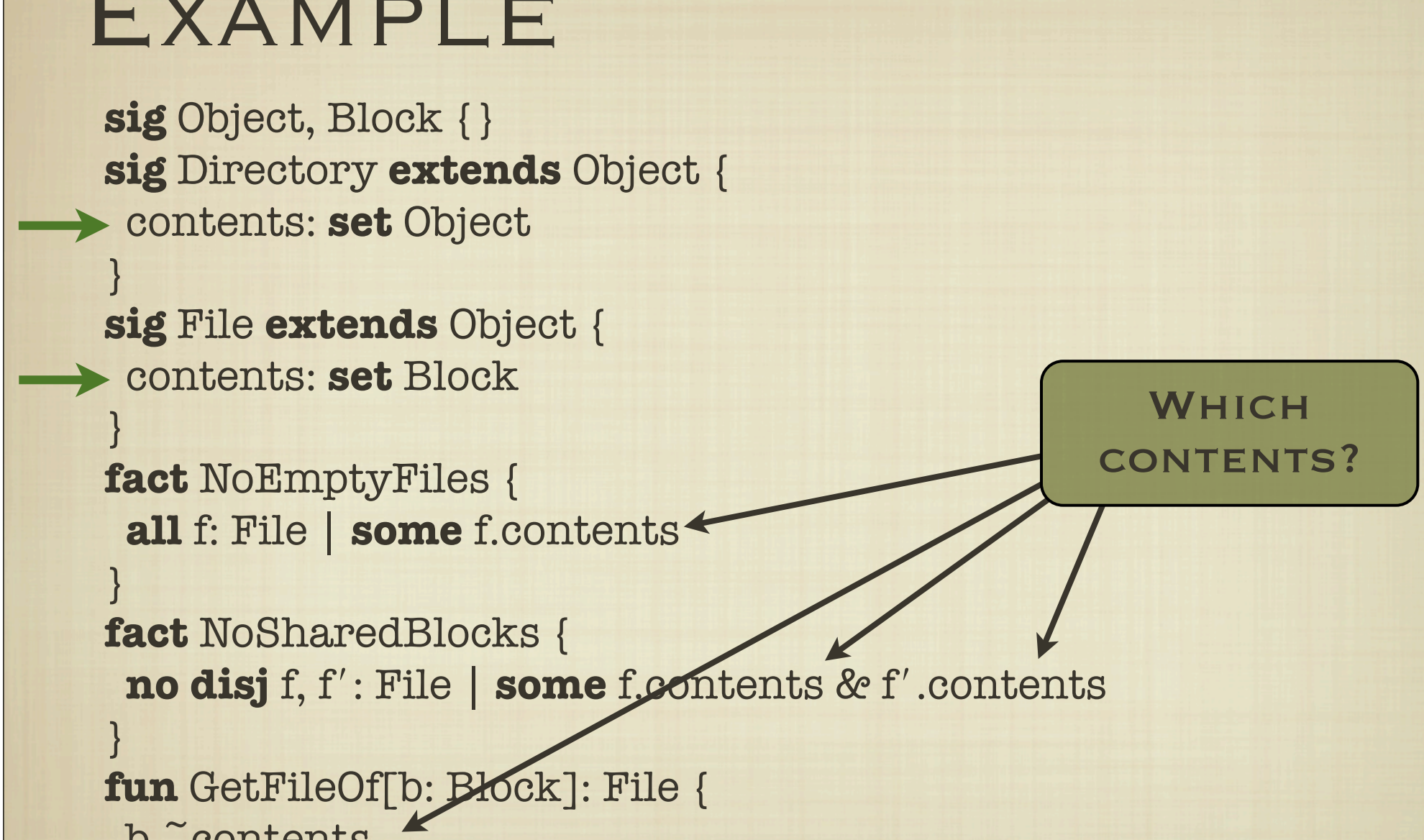
# Field Overloading

# Field Overloading

- Using the same field name in different sigs
- Alloy uses types to guess which field to use

# Example

```
sig Object, Block {}
sig Directory extends Object {
  contents: set Object
}
sig File extends Object {
  contents: set Block
}
fact NoEmptyFiles {
  all f: File | some f.contents
}
fact NoSharedBlocks {
  no disj f, f' : File | some f.contents & f'.contents
}
fun GetFileOf[b: Block]: File {
  b.~contents
}
```



# Subtle Example

```
sig Object, Block { }
sig Directory extends Object {
  contents: set Object
}
sig File extends Object {
  contents: set Block
}

fact NoSelfDirectory {
   no o: Object | o in o.contents
}
```

Which contents?

# Using Restriction

```
sig Node {
 next: Node,
 value: Value one -> Time
}
sig Time {
 next: Time
}
sig Value {}

fact NodesInRing { all n: Node | Node in n.^next }

fact TimeIsAnArrow { no ^next & iden }

fact TimeIsAnArrow { no ^(Time <: next) & iden }
```

AMBIGUOUS

# Goals for Today

- Understand how to declare fields inside signatures
- Learn enough about Alloy types to be able to fix type errors

# Next Time

- Understanding the differences between facts, predicates, assertions, and functions
  - Facts: assumed to always hold
  - Predicates: hold when we want them to
  - Assertions: constraints we hope will hold
  - Functions: helpers that return relations