CIS 771:
Software Specifications

Lecture: Alloy Whirlwind Tour
(part D)

Outline

- Extending the Address Book model with a stronger notion of hierarchy
- Adding constraints via the Alloy *fact* construct

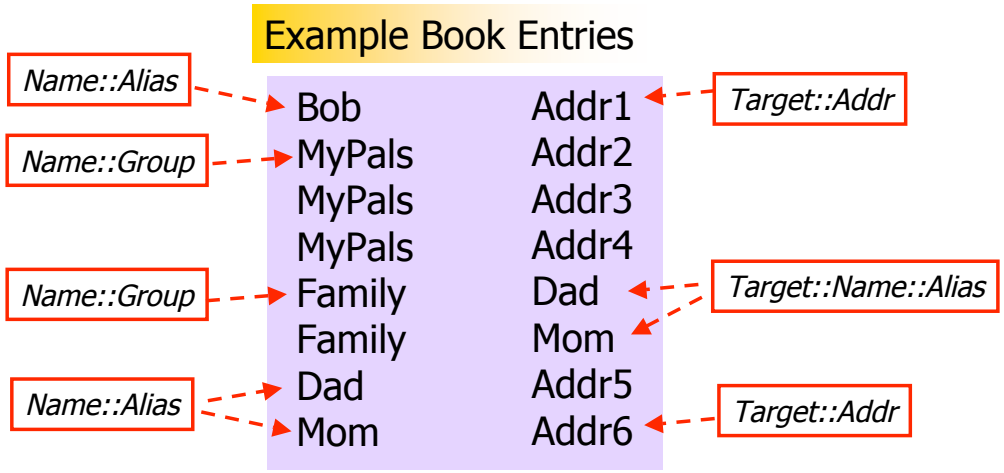
Adding the notion of *Alias*

- The *Alias* feature should enable one to…
  - create an alias (second name) for an address
  - use that alias as the target for another alias (create an alias for the alias)
  - name multiple targets with one alias, so that a group of addresses can be referred to with a single name

New Address Book Concepts

- Previously, Names (which were keys in the Book lookup table), were associated with Addresses
- Now, we want to have two categories of Name keys with keys binding to *targets.*
  - Group
    - Intuition: label that refers to a collection of targets
  - Alias
    - Intuition: label that refers to a single target
- A target can be one of the following
  - Address
  - Name (which ends up being a Group label or an Alias label)

Examples



...illustrating intended usage (some cases are omitted)

Enhanced Address Book

```
module tour/addressBookd

abstract sig Target {}
sig Addr extends Target {}
abstract sig Name extends Target {}

sig Alias, Group extends Name {}
sig Book {addr: Name -> Target}
```

A Target is either a Addr or Name

A Name is either an Alias label or Group label

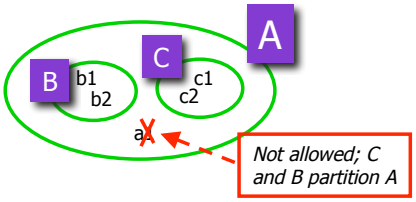
Lookup tables map Names to Targets

Issues with `abstract` *are explained on next slide.*

Abstract Construct

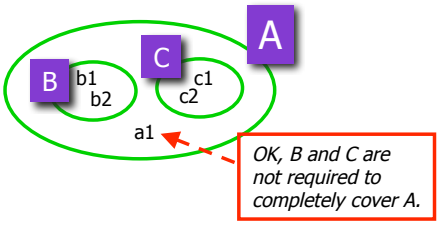
Using "abstract" forces all extending signatures to form a partition of A

```
abstract sig A {}
sig B,C extends A {}
```

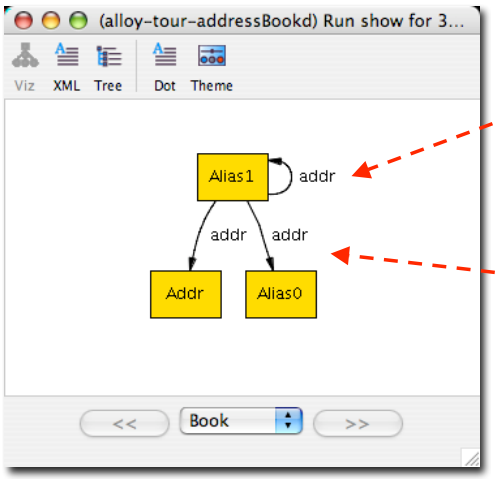


Without "abstract", all extending signatures simply form disjoint subsets of A

```
sig A {}
sig B,C extends A {}
```



Visualize Instance



Several problems...

Circular definition (Alias1 maps to itself)

Alias maps to more than one target (only Group should map to more than one target).

```
// show a non-empty address book
pred show (b: Book) {some b.addr}
run show for 3 but 1 Book
```

Disallow Circularity

Add a constraint (via the fact clause) that disallows direct or indirect circularity.

```
fact {
  all b: Book | no n: Name | n in n.^(b.addr)
}
```

Transitive closure of the `b.addr` *relation.*

Intuition:

"for all Books b, there is no Name n such that n is reachable via b's addr map from itself"

...but since this fact applies to every member of the signature Book, it's better written as a signature fact.

Signature Fact

Using `fact` clause

```
fact {
  all b: Book | no n: Name | n in n.^(b.addr)
}
```

`addr` *field now implicitly refers to* `this.addr`

Signature fact

```
sig Book {addr: Name -> Target}
  {no n: Name | n in n.^addr}
```

...in a signature fact, the fact automatically applies to all elements of that signature and no need to refer to signature element.

For You To Do

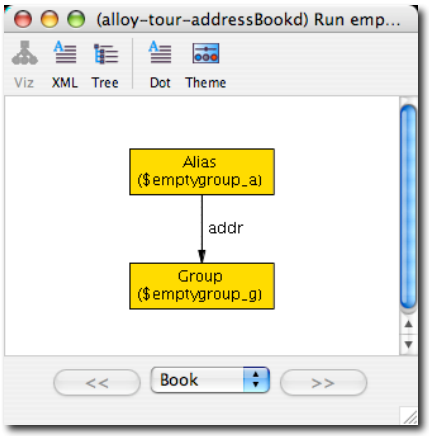
- Write a predicate that requires the existence of an alias that maps to more than one target.
- Use the analyzer on the model that we have built so far (including the last constraint) to see if there exists an instance in which an alias maps to more than one target.
- How might you add a constraint that forces an alias to map to no more than one target?

Issues

- Can we have a group referenced by an alias in the address book, but the group itself does not reference any addresses?
  - Let's write a predicate to test is such an instance exists…

```
pred emptygroup (b: Book) {
   some a: Alias |
     some g: Group |
        a.(b.addr) = g  && no g.(b.addr)
}
run emptygroup for 3 but 1 Book
```

Result



- Visualize using "project over book" option
- Shows a group present, but not linked to any targets.

Address Book Modification

Make the set of names defined by the address book explicit…

```
sig Book {
  names: set Name,
  addr: names -> some Target}
 {no n: Name | n in n.^addr
  all a: Alias | lone a.addr
 }
```

`names` *gives explicitly the set of names in the book*

Each name in `names` *must map to at least one target*

New Version of Operations

Now, a name is mapped to a target

```
pred add (b,b': Book, n: Name, t: Target)
  {b'.addr = b.addr + n->t}

pred del (b,b': Book, n: Name, t: Target)
  {b'.addr = b.addr - n->t}

fun lookup (b : Book, n: Name): set Addr
  {n.^(b.addr) & Addr}
```

Find all addresses reachable from name `n` *using transitive closure of* `addr` *relation.*

For You To Do

- Modify (if necessary) and recheck the following assertions from the previous lecture…
  - delUndoesAdd (with condition that name to be added does not already exist in the book)
  - addIdempotent
  - addLocal
- Do each of these assertions still hold, or can you find a counterexample.
- If a counterexample exists, explain the intuition behind it and indicate whether or not you believe that it is acceptable to allow that behavior in the model.

Acknowledgements

- The material in this lecture is based on Section 2.3 from…
  - *Software Abstractions: Logic, Language, and Analysis,* Daniel Jackson, MIT Press, 2006.