David Pick
CM 2403

This assignment took me 2 hours.

```
//HW4 by David Pick
module appendixA/addressBook2

sig Addr, Name { }

sig Book {
addr: Name -> (Name + Addr)
}

pred inv [b: Book] {
let addr = b.addr |
all n: Name {
//You can't have cycles in the addr field
n not in n.^addr
//If this addr is a name
//it eventually points to an Addr
some addr.n => some n.^addr & Addr
}
}

pred add [b, b': Book, n: Name, t: Name+Addr] {
//Check to see if the relation is already in the set of addr
//This way no cycles can be created in addr
not n->t in b.addr
b'.addr = b.addr + n->t
}

pred del [b, b': Book, n: Name, t: Name+Addr] {
//make sure the relation is already in addr
//can't remove a relation that isn't there
n->t in b.addr
b'.addr = b.addr - n->t
}

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

//smallest checked was 2
pred validBook[b: Book] { inv[b] }

//smallest checked was 2
pred invalidBook[b: Book] { not inv[b] }

//smallest checked was 2
pred validAdd[b, b': Book, n: Name, t: Name + Addr] {
```

```
add[b, b', n, t]
}

//smallest checked was 2
pred invalidAdd[b, b': Book, n: Name, t: Name + Addr] {
not add[b, b', n, t]
}

//smallest checked was 2
pred validDel[b, b': Book, n: Name, t: Name+Addr] {
del[b, b', n, t]
}

//smallest checked was 2
pred invalidDel[b, b': Book, n: Name, t: Name+Addr] {
not del[b, b', n, t]
}

assert addDoesNotPreserve {
no Name or no Book or no Addr or
//after an add has happened check that b
//and b' are still valid Books
some b, b': Book, n: Name, t: Name + Addr |
add[b, b', n, t] implies inv[b] and inv[b']
}
check addDoesNotPreserve for 8

assert delDoesNotPreserve {
no Name or no Book or no Addr or
//after a del has happened check that b
//and b' are still valid Books
some b, b': Book, n: Name, t: Name + Addr |
del[b, b', n, t] implies inv[b] and inv[b']
}
check delDoesNotPreserve for 8

run invalidAdd for 4
run validAdd for 4
run validBook for 1
run invalidBook for 1
```