EECS498-003
Formal Verification of Systems Software

Material and slides created by

Jon Howell and Manos Kapritsos

A state machine definition

```
datatype Card = Shelf | Patron(name: string)
datatype Book = Book(title: string)
type Variables= map<Book, Card>
```

```
ghost predicate Init(v: Variables) {
  forall book | book in v :: v[book] == She
}
ghost predicate CheckOut(v : Variables, v’ : Variables, book: Book, name: string) {
  && book in v
  && v[book] == Shelf
  && (forall book | book in v :: v[book] != Patron(name))
  && v’ == v[book := Patron(name)]
}
ghost predicate CheckIn(v : Variables, v’ : Variables, book: Book, name: string) {
  && book in v
  && v[book] == Patron(name)
  && v’ == v[book := Shelf]
}
ghost predicate Next(v: Variables, v’: Variables) {
  || (exists book, name :: CheckOut(v, v’, book, name))
  || (exists book, name :: CheckIn(v, v’, book, name))
}
```

Administrivia etc.

- Problem Set 1 is due **in two days**, September 19
- Problem Set 2 will be released on September 20
  - Chapters 3 and 4
    - Due October 3

- Be careful about spoilers when posting on Piazza
  - If in doubt, make it private

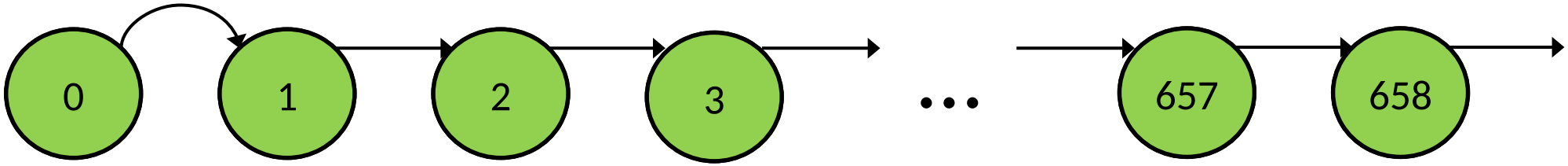
- `assert multiset({1,1}) == multiset({1});  // Does this prove?`
- `assert multiset([1,1]) == multiset([1]);  // Does this prove?`

Chapter 4: Proving properties

Expressing a system as a state machine allows us to prove that it has certain properties

- We will focus on safety properties
  - i.e. properties that hold throughout the execution

Basic tool: induction



- Show that the property holds on state 0
- Show that if the property holds on state k, it must hold on state k+1

Let's prove a safety invariant!

```
predicate Safety(v:Variables) {
  true // TBD
}

lemma SafetyProof()
  ensures forall v :: Init(v) ==> Safety(v)
  ensures forall v, v' :: Safety(v) && Next(v, v') ==> Safety(v')
{
}
```

Base case

VSCode transition

Inductive Step

Jay Normal Form

As you begin writing more interesting specs, proofs will be nontrivial.

Pull all the nondeterminism into one place, and get a receipt.

image: flickr/afagen CC-by-nc-sa

Jay Normal Form

```
datatype Step =
  | Action1Step( <parameters> )
  | Action2Step( <parameters> )
  ...

ghost predicate NextStep(v: Variables, v': Variables, step:Step)
{
  match step
    case Action1Step(<parameters>) => Action1(v, v', <parameters>)
    case Action2Step(<parameters>) => Action2(v, v', <parameters>)
    ...
}
predicate Next(v: Variables, v': Variables)
{
    exists step :: NextStep(v, v', step)
}
```