# An Electronic Phone Book Example in PVS (Sets and Invariants)

**Néstor Cataño**

ncatano@puj.edu.co

**Faculty of Engineering**

**Pontificia Universidad Javeriana**

# A Phone Book Example

```
phone: theory
 begin
  N: type  % names
  P: type+ % phone numbers
  B: type = [N -> P] % books

  p0: P
  GP: type = {p:P | p/=p0}

  known?(b:B, n:N): bool = b(n)/=p0

  findPhone(b:B, n:N): P = b(n)

  addphone(b:B, n:N, gp:GP): B =
   if known?(b,n)then b else b with [(n):= gp] endif

  changephone(b:B, n:N, gp:GP): B =
   if known?(b,n) then b with [(n):= gp] else b endif

  delphone(b:B, n:N): B =  b with [(n):= p0]

  del_add: theorem
   forall(b: B, n:N, gp:GP):
    delphone(addphone(b,n,gp),n) = delphone(b,n)
end phone
```

# Deficiencies

- The specification does not allow multiple phone numbers per person
- We can fix this by changing the range of type phone book from number to a set of numbers
- The fact that a person has no phone number can be modelled as the empty set

# A Phone Book Example Using Sets

```
B: type = [N -> P] % phone books
```

# A Phone Book Example Using Sets

```
B: type = [N -> P] % phone books

B: type = [N -> setof[P]] % phone books
```

# A Phone Book Example Using Sets

```
emptybook(n:  N): P = p0
```

# A Phone Book Example Using Sets

```
emptybook(n:  N): P = p0

emptybook(n:  N): setof[P] = emptyset[P]
```

# A Phone Book Example Using Sets

```
addphone(b:B, n:N, gp:GP): B =
 if known?(b,n) then b else b with [(n):= gp] endif
```

# A Phone Book Example Using Sets

```
addphone(b:B, n:N, gp:GP): B =
 if known?(b,n) then b else b with [(n):= gp] endif


addphone(b:B, n:N, p:P): B =
 b with [(n):= add(p,b(n))]
```

# A Phone Book Example Using Sets

```
delphone(b:B, n:N): B =  b with [(n):= p0]
```

# A Phone Book Example Using Sets

```
delphone(b:B, n:N): B =  b with [(n):= p0]

delphone(b:B, n:N): B =  b with [(n):= emptyset[P]]

delphoneNum(b:B, n:N, p:P): B = b with [(n):= remove(p,b(n))]
```

# A Phone Book Example Using Sets

```
phone_3: theory
 begin
  N: type  % names
  P: type+ % phone numbers
  B: type = [N -> setof[P]] % books

  findphone(b:B, n:N): setof[P] = b(n)

  addphone(b:B, n:N, p:P): B =
   b with [(n):= add(p,b(n))]

  delphone(b:B, n:N): B =  b with [(n):= emptyset[P]]

  delphoneNum(b:B, n:N, p:P): B = b with [(n):= remove(p,b(n))]

  del_add: conjecture
   forall(b: B, n:N, p:P):
    delphone(addphone(b,n,p),n) = delphone(b,n)

  delNum_add: conjecture
   forall(b: B, n:N, p:P):
    delphoneNum(addphone(b,n,p),n,p) = delphoneNum(b,n,p)
end phone_3
```

# Deficiencies

- The same phone number can be assigned simultaneously to two different names
- The phone numbers for all names must be disjoint
- We want to establish this as an invariant of the specification
  - We can ensure invariants with the use of predicate subtypes
  - PVS automatically generate proof obligations for the use of subtypes
  - TCCs (Type Correctness Conditions)

# Version that Maintains an Invariant

```
M-x show-tccs  ...
```

```
phone_4: theory
  begin
  N: type  % names
  P: type+  % phone numbers
  B: type = [N -> setof[P]] % books

  unused_phone(b:B, p:P): bool =
   forall(n:N): not member(p, b(n))

  VB: type = {b:B | forall(x,y:N): x /= y implies disjoint?(b(x),b(y))}

  addphone(b:VB, n:N, p:P): VB =
   if unused_phone(b,p) then b with [(n):= add(p,b(n))] else b endif
end phone_4
```

# Type Correctness Conditions (TCCs)

```
addphone_TCC1: OBLIGATION
  forall(b: VB, n: N, p: P):
    unused_phone(b, p) implies
     forall(x, y: N):
        x /= y implies
         disjoint?[P]
             ((b with [(n) := add[P](p, b(n))])(x),
              (b with [(n) := add[P](p, b(n))])(y));
```