Logic and Computation – CS 2800
Fall 2019

Lecture 25

Induction

Stavros Tripakis

Northeastern University
Khoury College of
Computer Sciences

Induction

The limitations of equational reasoning

- Consider:

```
(definec aapp (x :tl y :tl) :tl
  (if (endp x) y
    (cons (car x) (aapp (cdr x) y))))


(defthm nil-aapp
  (implies (tlp y)
           (equal (aapp nil y) y)))

(defthm aapp-nil
  (implies (tlp x)
           (equal (aapp x nil) ???)))
```

The limitations of equational reasoning

- Consider:

```
(definec aapp (x :tl y :tl) :tl
  (if (endp x) y
    (cons (car x) (aapp (cdr x) y))))


(defthm nil-aapp
  (implies (tlp y)
           (equal (aapp nil y) y)))

(defthm aapp-nil
  (implies (tlp x)
           (equal (aapp x nil) x)))
```

What happens when we try to prove the two theorems above?

Induction

- Recall the standard mathematical induction that you learned in discrete math:
  - In order to prove that something holds for all $n \in Nat$, it suffices to prove:
    1. **Base case**: that it holds for $n = 0$
    2. **Induction step**: that it holds for $n + 1$, assuming that it holds for n
       - The "assuming that it holds for n" part is called **induction hypothesis**
  - Example: prove that $\sum_{k=0}^{n} n = \frac{n(n+1)}{2}$

- In this course we will see a much more powerful kind of induction:
  - Every admissible (so, terminating) function generates an induction scheme!

Example:

- The following function is terminating and admissible:

```
(definec nind (n :nat) :nat
  (if (equal n 0)
     0
     (nind (- n 1))))
```

- Induction scheme generated by `nind` :
  - In order to prove a property `Phi`, it suffices to prove:

```
PO1: (not (natp n)) => Phi
PO2: (natp n) & (equal n 0) => Phi
PO3: (natp n) & (not (equal n 0)) & Phi|((n (- n 1))) => Phi
```

  - Proof obligations `PO1` and `PO2` are base cases.
  - Proof obligation `PO3` is the induction step.
  - `Phi|((n (- n 1)))` is the induction hypothesis.

Example:

- We use the induction scheme generated by `nind` to prove the following:

```
(definec sumn (n :nat) :nat
  (if (equal n 0)
    0
    (+ n (sumn (- n 1)))))

Phi:
(thm (implies (natp n)
              (equal (sumn n)
                     (/ (* n (+ n 1)) 2))))

What are our proof obligations?
```

Example:

```
(definec sumn (n :nat) :nat
  (if (equal n 0)
    0
    (+ n (sumn (- n 1)))))

Phi:
(thm (implies (natp n)
              (equal (sumn n)
                     (/ (* n (+ n 1)) 2))))


PO1: (not (natp n)) => Phi
PO1: (not (natp n))
             => ((natp n)
                    => (sumn n) = (/ (* n (+ n 1)) 2))
```

Example:

```
(definec sumn (n :nat) :nat
  (if (equal n 0)
    0
    (+ n (sumn (- n 1)))))

Phi:
(thm (implies (natp n)
              (equal (sumn n)
                     (/ (* n (+ n 1)) 2))))


PO2: (natp n) & (equal n 0) => Phi
PO2: (natp n) & (equal n 0)
             => ((natp n)
                   => (sumn n) = (/ (* n (+ n 1)) 2))
```

Example:

```
(definec sumn (n :nat) :nat
  (if (equal n 0)
    0
    (+ n (sumn (- n 1)))))

Phi:
(thm (implies (natp n)
              (equal (sumn n)
                     (/ (* n (+ n 1)) 2))))


PO3: (natp n) & (not (equal n 0)) & Phi|((n (- n 1))) => Phi
PO3: (natp n) & (not (equal n 0)) &
   ((natp (n-1)) => (sumn (n-1)) = (/ (* (n-1) (n-1+1)) 2))
             => ((natp n)
                   => (sumn n) = (/ (* n (+ n 1)) 2))
```

Next time

- Induction continued