Logic and Computation – CS 2800 Fall 2019

Lecture 26

Induction

Stavros Tripakis

Northeastern University
Khoury College of
Computer Sciences

Outline

- Induction continued

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))
```

Induction schemes: general case

- Induction scheme generated by `foo`:

```
(defunc foo (x 1 ... x n )
  :input-contract IC
  :output-contract OC
  (cond (t1 c1)
        (t2 c2)
        ...
        (tm cm)
        (t  cm+1 )))
```

Let:

$Case_1 = t_1$
$Case_2 = \neg t_1 \wedge t_2$
$Case_3 = \neg t_1 \wedge \neg t_2 \wedge t_3$
$\ldots$
$Case_m = \neg t_1 \wedge \neg t_2 \wedge \cdots \wedge \neg t_{m-1} \wedge t_m$
$Case_{m+1} = \neg t_1 \wedge \neg t_2 \wedge \cdots \wedge \neg t_{m-1} \wedge \neg t_m$

- We will call case i a **recursive case** if c_i contains a recursive call to `foo`.
  - Let R_i be the number of recursive calls in case i.
  - Let σ_i^j be the substitution of the j-th recursive call in case i.
- Otherwise we will call case i a **base case**.

Induction schemes: general case

- Induction scheme generated by `foo`:

```
(defunc foo (x 1 ... x n )
  :input-contract IC
  :output-contract OC
  (cond (t1 c1)
        (t2 c2)
        ...
        (tm cm)
        (t  cm+1 )))
```

Let:

$Case_1 = t_1$
$Case_2 = \neg t_1 \wedge t_2$
$Case_3 = \neg t_1 \wedge \neg t_2 \wedge t_3$
$\ldots$
$Case_m = \neg t_1 \wedge \neg t_2 \wedge \cdots \wedge \neg t_{m-1} \wedge t_m$
$Case_{m+1} = \neg t_1 \wedge \neg t_2 \wedge \cdots \wedge \neg t_{m-1} \wedge \neg t_m$

- Proof obligations in order to prove ϕ:
  1. We need to prove: $\neg IC \Rightarrow \phi$
  2. For each base case i we need to prove: $(IC \wedge Case_i) \Rightarrow \phi$
  3. For each recursive case i we need to prove:
     $$\left(IC \wedge Case_i \wedge \bigwedge_{1 \le j \le R_i} \phi | \sigma_i^j\right) \Rightarrow \phi$$

Example 1

- What induction scheme does `rrev` give rise to?

```
(definec rrev (x :tl) :tl
  (if (endp x)
    nil
    (aapp (rrev (rest x)) (list (first x)))))
```

Example 1

- What induction scheme does `rrev` give rise to?

```
(definec rrev (x :tl) :tl
  (if (endp x)
    nil
    (aapp (rrev (rest x)) (list (first x)))))
```

```
PO1: (not (tlp x)) => Phi
PO2: (tlp x) & (endp x) => Phi
PO3: (tlp x) & (not (endp x)) & Phi|((x (rest x))) => Phi
```

Example 2

- What induction scheme does `aapp` give rise to?

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

Example 2

- What induction scheme does `aapp` give rise to?

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

```
PO1: (not ((tlp x) & (tlp y))) => Phi
PO2: (tlp x) & (tlp y) & (endp x) => Phi
PO3: (tlp x) & (tlp y) & (not (endp x))
                          & Phi|((x (rest x))) => Phi
```

Our original example

- Can we now prove the second theorem?
- Which induction scheme should we use?

```
(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)))
```

Our original example

- The fact that our theorem talks about a certain function doesn't mean that we should necessarily use the induction scheme of that function to prove the theorem:

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

- Here the easiest is to use the induction scheme of true lists (which is the same as that of `rrev`):

```
PO1: (not (tlp x)) => Phi
PO2: (tlp x) & (endp x) => Phi
PO3: (tlp x) & (not (endp x)) & Phi|((x (cdr x))) => Phi
```

Our original example

- Using the induction scheme of true lists, our proof obligations become:

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

Our original example

- Using the induction scheme of true lists, our proof obligations become:

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

```
PO1: (not (tlp x)) => Phi
     (not (tlp x)) => ((tlp x) => (aapp x nil) = x)

PO2: (tlp x) & (endp x) => Phi
     (tlp x) & (endp x) => ((tlp x) => (aapp x nil) = x)

PO3: (tlp x) & (not (endp x)) & Phi|((x (cdr x))) => Phi
     (tlp x) & (not (endp x)) &
     ((tlp (cdr x)) => (aapp (cdr x) nil) = (cdr x))
                          => ((tlp x) => (aapp x nil) = x)
```

Next time

- Induction continued