1.1.1 Mathematical Induction

Principle: For any $P \subseteq \mathbb{N}$:

$$P(0) \wedge \forall k : \mathbb{N}.[P(k) \rightarrow P(k+1)] \longrightarrow \forall n : \mathbb{N}.P(n)$$

Proof Schema:
Base Case:
  To Show: P(0)
Inductive Step: Take arbituary k
  Inductive Hypothesis: P(k)
  To Show: P(k+1)

1.1.2 Mathematical Induction Technique

Principle: For any $P \subseteq \mathbb{N}$ and any $m : \mathbb{Z}$

$$P(m) \wedge \forall k \geq m.[P(k) \rightarrow P(k+1)] \longrightarrow \forall n \geq m.P(n)$$

Proof Schema:
Base Case:
  To Show: P(m)
Inductive Step: Take arbituary k. Assume that $k \geq m$.
  Inductive Hypothesis: P(k)
  To Show: P(k+1)

1.1.3 Strong Induction

Principle: For any $P \subseteq \mathbb{N}$:

$$P(0) \wedge \forall k : \mathbb{N}.[\forall j \in [0..k].P(j) \rightarrow P(j+1)] \longrightarrow \forall n : \mathbb{N}.P(n)$$

Proof Schema: (for 2 base cases)
Base Case:
  To Show: P(0)
Inductive Step: Take arbituary k
  Inductive Hypothesis: $\forall j \in [0..k].P(j)$
  To Show: P(k+1)
  1st Case: k = 0
    To Show: P(1)
  2nd Case: $k \neq 0$
    (A) $k \geq 1$ because $k : \mathbb{N}$ and $k \neq 0$ by case
    (B) $k, k-1 \in [0..k]$ because $k : \mathbb{N}$ and $k \neq 0$

1.1.4 Strong Induction Technique

Principle: For any $P \subseteq \mathbb{N}$ and any $m : \mathbb{Z}$

$$P(m) \wedge \forall k \geq m.[\forall j \in [m..k].P(j) \rightarrow P(j+1)] \longrightarrow \forall n \geq m.P(n)$$

1.2 Structural Induction

1.2.1 Induction over Lists

Principle: For any type T, and $P \subseteq [T]$:

$$P([]) \wedge \forall vs : [T].\forall v : T.[P(vs) \rightarrow P(v : vs)] \longrightarrow \forall vs : [T].P(xs)$$

Proof Schema:
Base Case:
  To Show: P([])
Inductive Step: Take arbituary v':a, vs':[a]
  Inductive Hypothesis: P(vs')
  To Show: P((v':vs'))

Proof by structural induction on `m`.
Base case:
  To Show: P(`Zero`)
  proof of base case
Inductive Step:
  Take an arbitrary `k:Nat`.
  Inductive Hypothesis: P(`k`)
  To show: P(`Succ k`)

  proof of inductive step
Proof by structural induction on `m`.

List Lemmas:

(A) $us ++ [] = us$

(B) $[] ++ us = us$

(C) $(u : us) ++ vs = u : (us ++ vs)$

(D) $(us ++ vs) ++ ws = us ++ (vs ++ ws)$

1.2.2 Induction over arbituary data structures

- data Nat = Zero | Succ Nat

$$P(Zero) \wedge \forall n : Nat.[P(n) \rightarrow P(Succ\ n)] \longrightarrow \forall n : Nat.P(n)$$

- data Tree a = Empty | Node (Tree a) a (Tree a)

$$P(Empty) \wedge \forall t1, t2 : Tree\ T.\forall x : T.[P(t1) \wedge P(t2) \rightarrow P(Node\ t1\ x\ t2)] \\ \longrightarrow \forall t : Tree\ T.P(t)$$

- data BExp = Tr | Fl | BNt BExp | BAnd BExp BExp

$$P(Tr) \wedge P(Fl) \wedge \forall b : BExp.[P(b) \rightarrow P(BNt\ b)] \\ \wedge \forall b1, b2 : BExp.[P(b1) \wedge P(b2) \rightarrow P(BAnd\ b1\ b2)] \longrightarrow \forall b : BExp.P(b)$$

- data T = C1 [Int] | C2 Int T

$$\forall is : [Int].P(C1\ is) \wedge \forall i : Int.\forall t : T.[P(t) \rightarrow P(C2\ i\ t)] \longrightarrow \forall t : T.P(t)$$

- data Reds = BaseR | Red Greens
  data Greens = BaseG | Green Reds

$$P(BaseR) \wedge \forall g : Greens.[Q(g) \rightarrow P(Red\ g)] \\ \wedge Q(BaseG) \wedge \forall r : Reds.[P(r) \rightarrow Q(Green\ r)] \\ \longrightarrow \forall r : Reds.P(r) \wedge \forall g : Greens.Q(g)$$

- data Cactus = Root Tree
  data Tree = Leaf | Node Trees
  data Trees = Empty | Cons Tree Trees

$$P(Leaf) \wedge \forall ts : Trees.[Q(ts) \rightarrow P(Node\ ts)] \wedge \\ Q(Empty) \wedge \forall t : Tree.\forall ts : Trees.[P(t) \wedge Q(ts) \rightarrow Q(Cons\ t\ ts)] \\ \longrightarrow \forall t : Tree.P(t) \wedge \forall ts : Trees.Q(ts)$$

1.2.3 Two Approaches

1. Invent an Auxiliary Lemma
2. Strengthen the original property

1.3 General Induction

1.3.1 Inductively Defined Sets

- $S_{\mathbb{N}}$ defined over Zero and Succ through
  $Zero \in S_{\mathbb{N}}$
  $\forall n.[n \in S_{\mathbb{N}} \rightarrow Succ\ \ n \in S_{\mathbb{N}}]$

$$Q(Zero) \wedge \forall m \in S_{\mathbb{N}}.[Q(m) \rightarrow Q(Succ\ m)] \longrightarrow \forall n \in S_{\mathbb{N}}.Q(n)$$

- Tree
  $i \in \mathbb{N} \rightarrow Leaf\ i \in Tree$
  $\forall t1, t2 \in Tree.\forall c \in Char.Node\ c\ t1\ t2 \in Tree$

$$\forall i \in \mathbb{N}.Q(Leaf\ i) \\ \wedge \forall t1, t2 \in Tree.\forall c \in Char.[Q(t1) \wedge Q(t2) \rightarrow Q(Node\ c\ t1\ t2)] \\ \longrightarrow \forall t \in Tree.Q(t)$$

- $OL \subseteq \mathbb{N}^*$
  $[] \in OL$
  $\forall i \in \mathbb{N}.i : [] \in OL$
  $\forall i, j \in \mathbb{N}, js \in \mathbb{N}^*.[i \leq j \wedge j : js \in OL \rightarrow i : j : js \in OL]$

$$Q([]) \wedge \forall i \in \mathbb{N}.Q(i : []) \\ \wedge \forall i, j \in \mathbb{N}, js \in \mathbb{N}^*.[i \leq j \wedge j : js \in OL \wedge Q(j : js) \rightarrow Q(i : j : js)] \\ \longrightarrow \forall ns \in OL.Q(ns)$$

1.3.2 Inductively Defined Relations

- $SL \subseteq \mathbb{N} \times \mathbb{N}$
  $\forall k \in \mathbb{N}.SL(0, k+1)$
  $\forall m, n \in \mathbb{N}.[SL(m, n) \rightarrow SL(m+1, n+1)]$

$$\forall k \in \mathbb{N}.Q(0, k+1) \\ \wedge \forall m, n \in \mathbb{N}.[SL(m.n) \wedge Q(m, n) \rightarrow Q(m+1, n+1)] \\ \longrightarrow \forall m, n \in \mathbb{N}.[SL(m, n) \rightarrow Q(m, n)]$$

- $Even \subseteq S_{\mathbb{N}}$
  $Even(Zero)$
  $\forall n \in S_{\mathbb{N}}.[Even(n) \rightarrow Even(Succ\ (Succ\ n))]$

$$Q(Zero) \\ \wedge \forall n \in S_{\mathbb{N}}.[Even(n) \wedge Q(n) \rightarrow Q(Succ\ (Succ\ n))] \\ \longrightarrow \forall n \in S_{\mathbb{N}}.[Even(n) \rightarrow Q(n)]$$

- $Odd \subseteq S_{\mathbb{N}}$
  $Odd(Succ\ Zero)$
  $\forall n \in S_{\mathbb{N}}.[Odd(n) \rightarrow Odd(Succ\ (Succ\ n))]$

$$Q(Succ\ Zero) \\ \wedge \forall n \in S_{\mathbb{N}}.[Odd(n) \wedge Q(n) \rightarrow Q(Succ\ (Succ\ n))] \\ \longrightarrow \forall n \in S_{\mathbb{N}}.[Odd(n) \rightarrow Q(n)]$$

1.3.3 Inductively Defined Functions

- F 0 = 0
  F i = 1 + F(i - 3)

$$Q(0, 0) \\ \wedge \forall j, k : \mathbb{Z}.[j \neq 0 \wedge F(j-3) = k \wedge Q(j-3, k) \rightarrow Q(j, k+1) \\ \longrightarrow \forall j, k : \mathbb{Z}.[F\ j = k \rightarrow Q(j,k)]$$

- G'(i,j,cnt,acc)
  | i==cnt = acc
  | otherwise = G'(i,j,cnt+1,acc+j)

$$\forall i, j, acc : \mathbb{N}.Q(i, j, i, acc, acc) \\ \wedge \forall i, j, acc, cnt, r : \mathbb{N}.[i \neq cnt \wedge G'(i, j, cnt+1, acc+j) = r \\ \wedge Q(i, j, cnt+1, acc+j, r) \rightarrow Q(i, j, cnt, acc, r)] \\ \longrightarrow \forall i, j, acc, cnt, r : \mathbb{N}.[G'(i, j, cnt, acc) = r \rightarrow Q(i, j, cnt, acc, r)$$

- DM'(i,j,cnt,acc)
  | acc+j > i = (cnt,i-acc)
  | otherwise = DM'(i,j,cnt+1,acc+j)

  $\forall i, j, cnt, acc : \mathbb{N}.[acc + j > i \rightarrow Q(i, j, cnt, acc, cnt, i - acc)]$
  $\wedge \forall i, j, acc, cnt, k1, k2 : \mathbb{N}.[acc + j \leq i \wedge DM'(i, j, cnt + 1, acc + j) = (k1, k2)$
  $\wedge Q(i, j, cnt + 1, acc + j, k1, k2) \rightarrow Q(i, j, cnt, acc, k1, k2)]$
  $\longrightarrow \forall i, j, acc, cnt, k1, k2 : \mathbb{N}.[DM'(i, j, cnt, acc) = (k1, k2) \rightarrow Q(i, j, cnt, acc, k1, k2)]$

- M'(i,cnt,acc)
  | i==cnt = acc
  | otherwise = M'(i,cnt+1,2*acc)

  $\forall i, acc : \mathbb{N}.Q(i, i, acc, acc)$
  $\forall i, cnt, acc, r : \mathbb{N}.[i \neq cnt \wedge M'(i, cnt + 1, 2 * acc) = r \wedge Q(i, cnt + 1, 2 * acc, r)$
  $\rightarrow Q(i, cnt, acc, r)]$
  $\longrightarrow \forall i, cnt, acc, r : \mathbb{N}.[M'(i, cnt, acc) = r \rightarrow Q(i, cnt, acc, r)]$

2 Imperative Programs

2.1 Program Specifications

2.1.1 Hoare Logic

$$\frac{P[x \mapsto x_{old}] \wedge x = E[x \mapsto x_{old}] \longrightarrow Q}{\{P\} \quad x = E; \quad \{Q\}}$$

2.1.2 Straight Line Code

$$\frac{\{P\} \quad code1 \quad \{R\} \quad \{R\} \quad code2 \quad \{Q\}}{\{P\} \quad code1; code2 \quad \{Q\}}$$

2.2 Conditional Branches

$$\frac{\{P \wedge cond\} \quad code1 \quad \{Q\} \quad \{P \wedge \neg cond\} \quad code2 \quad \{Q\}}{\{P\} \quad if(cond)\{code1\}else\{code2\} \quad \{Q\}}$$

$$M_1 \triangleq \mathtt{y} = \mathtt{x} \ \wedge\ \mathtt{x} = \mathtt{x}_{pre}$$
$$M_2 \triangleq \mathtt{y} = \mathtt{x}_{pre} \ \wedge\ \mathtt{x} = \mathtt{x}_{pre} + 1$$
$$M_3 \triangleq \mathtt{y} = 2 * \mathtt{x}_{pre}$$

2.3 Method Calls

line 7: $M_1[\mathtt{y} \mapsto \mathtt{y}_{old}] \ \wedge\ \mathtt{y} = \mathtt{y}_{old} + \mathtt{x} \ \longrightarrow\ M_2$

$$\mathtt{y}_{old} = \mathtt{x} \ \wedge\ \mathtt{y} = \mathtt{y}_{old} + \mathtt{x}$$
$$\longrightarrow$$
$$\mathtt{y} = 2 * \mathtt{x}$$

void someMethod(type x_1, ..., type x_n)
//Pre: R
//Post: S

$$\frac{\begin{array}{c} P \longrightarrow R[\overline{x} \mapsto \overline{v}] \\ P[\overline{v}[..) \mapsto \overline{v}[..)_{old}] \wedge S[\overline{x} \mapsto \overline{v}][\overline{v}[..)_{pre} \mapsto \overline{v}[..)_{old}] \longrightarrow Q \end{array}}{\{P\} \quad someMethod(v_1, ..., v_n) \quad \{Q\}}$$

$$\frac{\begin{array}{c} P \longrightarrow R[\overline{x} \mapsto \overline{v}] \\ P[\overline{v}[..) \mapsto \overline{v}[..)_{old}][res_{old} \mapsto res] \wedge res = r \\ \wedge S[\overline{x} \mapsto \overline{v}][\overline{v}[..)_{pre} \mapsto \overline{v}[..)_{old}][res \mapsto res_{old}] \longrightarrow Q \end{array}}{\{P\} \quad res = someMethod(v_1, ..., v_n) \quad \{Q\}}$$

2.4 Iteration

1. I holds before the loop is entered
2. Given condition, the loop re-establishes I
3. Termination of loop and I establishes Q

$$\frac{P \longrightarrow I \quad \{I \wedge cond\} \ body \ \{I\} \quad I \wedge \neg cond \longrightarrow Q}{\{P\} \quad while(cond)\{body\} \quad \{Q\}}$$

$$\frac{I[\overline{mod} \mapsto \overline{mod}_{old}] \wedge cond[\overline{mod} \mapsto \overline{mod}_{old}] \wedge body\text{-}effect \longrightarrow I}{\{I \wedge cond\} \quad body \quad \{I\}}$$

4. V is bounded
5. V decreases with each iteration

$$I[\overline{mod} \mapsto \overline{mod}_{old}] \wedge cond[\overline{mod} \mapsto \overline{mod}_{old}] \wedge body\text{-}effect$$
$$\longrightarrow V \geq n \wedge V[\overline{mod} \mapsto \overline{mod}_{old}] > V$$

6. Array access are legal

$$I[\overline{mod} \mapsto \overline{mod}_{old}] \wedge cond[\overline{mod} \mapsto \overline{mod}_{old}] \longrightarrow 0 \leq x \leq a.length$$

   for any array a and access x (i_{old} or i)
7. No integer overflows — Assume perfect machine