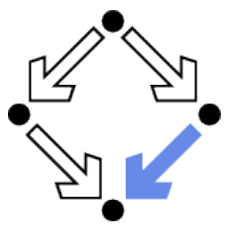
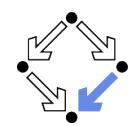
First Order Predicate Logic

Formal Reasoning in Special Domains

Wolfgang Schreiner and Wolfgang Windsteiger
Wolfgang.(Schreiner|Windsteiger)@risc.jku.at

Research Institute for Symbolic Computation (RISC)
Johannes Kepler University (JKU), Linz, Austria
http://www.risc.jku.at



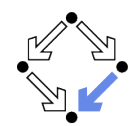


Formal Reasoning in Special Domains

We will consider methods for reasoning about

- natural numbers,
- algebraic data types,
- computer programs

all of which are based on the principle of induction.



Mathematical Induction

A method to prove statements over the natural numbers.

- Goal: prove

$$\forall x \in \mathbb{N} : F$$

  i.e., formula F holds for all natural numbers.

- Rule:

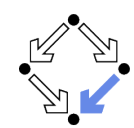
$$\frac{K \vdash F[0/x] \quad K \vdash (\forall y \in \mathbb{N} : F[y/x] \rightarrow F[y+1/x])}{K \vdash \forall x \in \mathbb{N} : F}$$

  $F[t/x]$: *F where every free occurrence of x is replaced by t.*

- Proof Steps:
  - *Induction base:* prove that F holds for 0.
  - *Induction hypothesis:* assume that F holds for new constant $\overline{x}$.
  - *Induction step:* prove that then F also holds for $\overline{x}+1$.

    Often the constant symbol x itself is chosen rather than $\overline{x}$.

Works because every natural number is reachable by a finite number of increments starting from 0.



Example

We prove Gauss's sum formula

$$\forall n \in \mathbb{N} : \sum_{i=1}^{n} i = \frac{n \cdot (n+1)}{2}$$

by induction on n:

- Induction Base:

$$\sum_{i=1}^{0} i = 0 = \frac{0 \cdot (0+1)}{2}$$

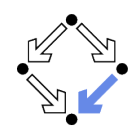
- Induction Hypothesis:

$$\sum_{i=1}^{\overline{n}} i = \frac{\overline{n} \cdot (\overline{n}+1)}{2} \qquad (*)$$

- Induction Step:

$$\sum_{i=1}^{\overline{n}+1} i = (\overline{n}+1) + \sum_{i=1}^{\overline{n}} i \overset{(*)}{=} (\overline{n}+1) + \frac{\overline{n} \cdot (\overline{n}+1)}{2}$$

$$= \frac{2 \cdot (\overline{n}+1) + \overline{n} \cdot (\overline{n}+1)}{2} = \frac{(\overline{n}+2) \cdot (\overline{n}+1)}{2} \qquad \square$$



Choice of Induction Variable

We define addition on $\mathbb{N}$ by primitive recursion:

$$x + 0 := x \tag{1}$$
$$x + (y + 1) := (x + y) + 1 \tag{2}$$

Our goal is to prove the associativity law

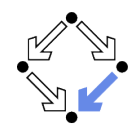
$$\forall x \in \mathbb{N}, y \in \mathbb{N}, z \in \mathbb{N} : x + (y + z) = (x + y) + z$$

For this purpose, we prove

$$\forall z \in \mathbb{N} : \underbrace{\forall x \in \mathbb{N}, y \in \mathbb{N} : x + (y + z) = (x + y) + z}_{F}$$

by induction on z.

Sometimes the appropriate choice of the induction variable is critical.



Choice of Induction Variable

We prove by induction on z

$$\forall z \in \mathbb{N} : \forall x \in \mathbb{N}, y \in \mathbb{N} : x + (y + z) = (x + y) + z$$

- Induction base: we prove

$$\forall x \in \mathbb{N}, y \in \mathbb{N} : x + (y + 0) = (x + y) + 0$$

We prove for arbitrary $x_0, y_0 \in \mathbb{N}$

$$x_0 + (y_0 + 0) \overset{(1)}{=} x_0 + y_0 \overset{(1)}{=} (x_0 + y_0) + 0$$

- Induction hypothesis (*): we assume

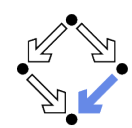
$$\forall x \in \mathbb{N}, y \in \mathbb{N} : x + (y + z) = (x + y) + z$$

- Induction step: we prove

$$\forall x \in \mathbb{N}, y \in \mathbb{N} : x + (y + (z + 1)) = (x + y) + (z + 1)$$

We prove for arbitrary $x_0, y_0 \in \mathbb{N}$

$$x_0 + (y_0 + (z + 1)) \overset{(2)}{=} x_0 + ((y_0 + z) + 1) \overset{(2)}{=} (x_0 + (y_0 + z)) + 1$$
$$\overset{(*)}{=} ((x_0 + y_0) + z) + 1 \overset{(2)}{=} (x_0 + y_0) + (z + 1) \quad \square$$



Induction with a Different Starting Value

- Goal: prove

$$\forall x \in \mathbb{N} : x \geq b \rightarrow F$$

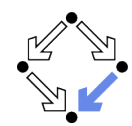
  i.e., formula F holds for all natural numbers greater than or equal to some natural number b.

- Rule:

$$\frac{K \vdash F[b/x] \quad K \vdash (\forall y \in \mathbb{N} : y \geq b \wedge F[y/x] \rightarrow F[y+1/x])}{K \vdash (\forall x \in \mathbb{N} : x \geq b \rightarrow F)}$$

- Proof Steps:
  - *Induction base:* prove that F holds for b.
  - *Induction hypothesis:* assume that F holds for $\overline{x} \geq b$.
  - *Induction step:* prove that then F also holds for $\overline{x}+1$.

Induction works with arbitrary starting values.



Example

We prove

$$\forall n \in \mathbb{N} : n \geq 4 \rightarrow n^2 \leq 2^n$$

- Induction base: we show

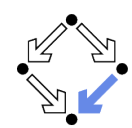
$$4^2 = 16 = 2^4$$

- Induction hypothesis: we assume for $n \geq 4$

$$n^2 \leq 2^n \qquad (*)$$

- Induction step: we show

$$(n+1)^2 = n^2 + 2n + 1 \overset{1 \leq n}{\leq} n^2 + 2n + n = n^2 + 3n \overset{0 \leq n}{\leq} n^2 + 4n$$
$$\overset{4 \leq n}{\leq} n^2 + n \cdot n = n^2 + n^2 = 2n^2 \overset{(*)}{\leq} 2 \cdot 2^n = 2^{n+1} \qquad \square$$



Complete Induction

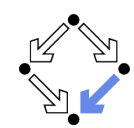
A generalized form of the induction method.

- Rule:

$$\frac{K \vdash (\forall x \in \mathbb{N} : (\forall y \in \mathbb{N} : y < x \to F[y/x]) \to F)}{K \vdash \forall x \in \mathbb{N} : F}$$

- Proof steps:
  - *Induction hypothesis:* assume that F holds for all y less than $\overline{x}$.
  - *Induction step:* prove that F then also holds for $\overline{x}$.

The induction assumption is applied not only to the direct predecessor.



Example

We take function $T : \mathbb{N} \to \mathbb{N}$ where

$$T(n) = \begin{cases} 0 & \text{if } n = 0 \\ 2 \cdot T(n/2) & \text{if } n > 0 \wedge 2|n \\ 1 + 2 \cdot T((n-1)/2) & \text{else} \end{cases}$$

and prove by complete induction on n

$$\forall n \in \mathbb{N} : T(n) = n$$

- Induction hypothesis:

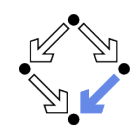
$$\forall m \in \mathbb{N} : m < n \to T(m) = m \qquad (*)$$

- Induction step:
  - Case $n = 0$: we know $T(n) = T(0) = 0 = n$
  - Case $n > 0 \wedge 2|n$: we know

$$T(n) = 2 \cdot T(n/2) \overset{(*)}{=} 2 \cdot (n/2) = n$$

  - Case $n > 0 \wedge \neg(2|n)$: we know

$$T(n) = 1 + 2 \cdot T((n-1)/2) \overset{(*)}{=} 1 + 2 \cdot ((n-1)/2) = 1 + (n-1) = n \quad \square$$



Induction on Sequence of Subdomains

We would like to prove

$$\forall x \in S : F$$

where S can be decomposed into an infinite union $S = S_0 \cup S_1 \cup S_2 \ldots$:

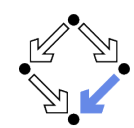
$$S = \bigcup_{n \in \mathbb{N}} S_i = \{x \mid \exists n \in \mathbb{N} : x \in S_n\}$$

Since thus

$$x \in S \Leftrightarrow \exists n \in \mathbb{N} : x \in S_n$$

it suffices to prove

$$\begin{aligned}
& \forall x : (\exists n \in \mathbb{N} : x \in S_n) \rightarrow F \\
\Leftrightarrow\ & \forall x : (\exists n \in \mathbb{N} : x \in S_n) \rightarrow F \\
\Leftrightarrow\ & \forall x : \neg(\exists n \in \mathbb{N} : x \in S_n) \vee F \\
\Leftrightarrow\ & \forall x : (\forall n \in \mathbb{N} : x \notin S_n) \vee F \\
\Leftrightarrow\ & \forall x : (\forall n \in \mathbb{N} : x \notin S_n \vee F) \\
\Leftrightarrow\ & \forall x : (\forall n \in \mathbb{N} : x \in S_n \rightarrow F) \\
\Leftrightarrow\ & \forall n \in \mathbb{N} : \forall x \in S_n : F
\end{aligned}$$



We can prove this by induction on n.

Example

We want to prove

$$\forall s \in \mathit{FiniteSet} : |\mathcal{P}(s)| = 2^{|s|}$$

where

$$\mathit{FiniteSet} = \bigcup_{n \in \mathbb{N}} \{s \mid |s| = n\}$$

We prove

$$\forall n \in \mathbb{N} : \forall s : |s| = n \rightarrow |\mathcal{P}(s)| = 2^{|s|}$$

by induction on n.

- Induction base: take arbitrary s with $|s| = 0$, i.e., $s = \emptyset$.

$$|\mathcal{P}(\emptyset)| = |\{\emptyset\}| = 1 = 2^0 = 2^{|\emptyset|}$$

- Induction hypothesis: we assume

$$\forall s : |s| = n \rightarrow |\mathcal{P}(s)| = 2^{|s|}$$

- Induction step: we show

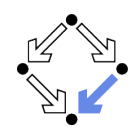
$$\forall s : |s| = n+1 \rightarrow |\mathcal{P}(s)| = 2^{|s|}$$

Take arbitrary s with $|s| = n+1$. Since thus $|s| > 0$, we know $s \neq \emptyset$ and thus have some $x \in s$. Define $s' := s \backslash \{x\}$, i.e., $|s'| = |s| - 1 = n$. We then know

$$\mathcal{P}(s) = \mathcal{P}(s') \cup \{\{x\} \cup p \mid p \in \mathcal{P}(s')\}$$

From $|s'| = n$ and the induction hypothesis, we know $|\mathcal{P}(s')| = 2^{|s'|} = 2^n$. Since $x \notin s'$, we have $\forall p \in \mathcal{P}(s') : x \notin p$ and thus

$$|\mathcal{P}(s)| = |\mathcal{P}(s')| + |\{\{x\} \cup p \mid p \in \mathcal{P}(s')\}| = 2 \cdot |\mathcal{P}(s')| = 2 \cdot 2^n = 2^{n+1} = 2^{|s|} \quad \square$$



Structural Induction

Let T be an algebraic data type

$$\textbf{type } T := c_1(T,\ldots) + \ldots + c_n(T,\ldots)$$

with n constructors $c_1,\ldots,c_n$. In order to prove

$$\forall x \in T : F$$

it suffices to prove n closed formulas ("cases")

$$\forall x_1 \in T,\ldots : F[x_1/x] \wedge \ldots \rightarrow F[c_1(x_1,\ldots)/x] \;//\; \text{case } x = c_1(x_1,\ldots)$$

$$\ldots$$

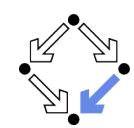
$$\forall x_1 \in T,\ldots : F[x_1/x] \wedge \ldots \rightarrow F[c_n(x_1,\ldots)/x] \;//\; \text{case } x = c_n(x_1,\ldots)$$

where all arguments $x_1,\ldots$ in constructor terms $c_i(x_1,\ldots)$ are new quantified variables. The proof of each "case" $x = c_i(x_1,\ldots)$ consists of:

- Induction hypothesis: assume that F holds for $x_1,\ldots$.

  For all variables $x_1,\ldots$ of type T that appear in $c_i(x_1,\ldots)$.

- Induction step: prove that then F also holds for $x = c_i(x_1,\ldots)$.



Induction can be also applied to algebraic data types.

Example

Given the type and function definitions

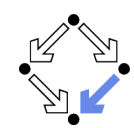
$$\textbf{type } List(E) := empty + cons(E, List(E))$$

$$length : List(E) \to \mathbb{N}$$
$$length(empty) := 0 \tag{1}$$
$$length(cons(e, l)) := 1 + length(l) \tag{2}$$

$$append : List(E) \times List(E) \to List(E)$$
$$append(empty, r) := r \tag{3}$$
$$append(cons(e, l), r) := cons(e, append(l, r)) \tag{4}$$

we want to prove

$$\forall l \in List(T), r \in List(T) : length(append(l, r)) = length(l) + length(r)$$



Example (Contd)

Given the type

$$\textbf{type } List(E) := empty + cons(E, List(E))$$

in order to prove

$$\forall l \in List(E) : F$$

it suffices to prove

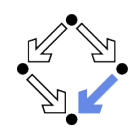
$$F[empty/l]$$
$$\forall e \in E, l \in List(E) : F \rightarrow F[cons(e, l)/l]$$

Therefore, for

$$F \equiv \forall r \in List(E) : length(append(l, r)) = length(l) + length(r)$$

it suffices to prove

$$\forall r \in List(E) : length(append(empty, r)) = length(empty) + length(r)$$
$$\forall e \in T, l \in List(E) :$$
$$\forall r \in List(E) : length(append(l, r)) = length(l) + length(r) \rightarrow$$
$$\forall r \in List(E) : length(append(cons(e, l), r)) =$$
$$length(cons(e, l)) + length(r)$$



Example (Contd)

We prove

$$\forall l \in List(E) : \forall r \in List(E) : length(append(l, r)) = length(l) + length(r)$$

by structural induction on l.

- Case $l = empty$: we know for arbitrary r

$$\begin{aligned} length(append(empty, r)) &\overset{(3)}{=} length(r) = 0 + length(r) \\ &\overset{(1)}{=} length(empty) + length(r) \end{aligned}$$

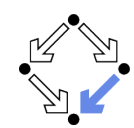
- Case $l = cons(e_0, l_0)$: take arbitrary $e_0 \in E, l_0 \in List(E)$ and assume

$$\forall r \in List(T) : length(append(l_0, r)) = length(l_0) + length(r) \qquad (*)$$

We then know for arbitrary $r_0 \in List(T)$

$$\begin{aligned} length(append(cons(e_0, l_0), r_0)) &\overset{(4)}{=} length(cons(e_0, append(l_0, r_0))) \\ &\overset{(2)}{=} 1 + length(append(l_0, r_0)) \\ &\overset{(*)}{=} 1 + (length(l_0) + length(r_0)) \\ &= (1 + length(l_0)) + length(r_0) \\ &\overset{(2)}{=} length(cons(e_0, l_0)) + length(r_0) \end{aligned}$$

□



Example

Given the type and function definitions

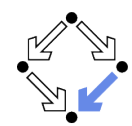
$$\textbf{type } \mathit{Tree}(E) := \mathit{empty} + \mathit{node}(E, \mathit{Tree}(E), \mathit{Tree}(E))$$

$$\mathit{weight} : \mathit{Tree}(\mathbb{N}) \to \mathbb{N}$$
$$\mathit{weight}(\mathit{empty}) := 0 \tag{1}$$
$$\mathit{weight}(\mathit{node}(n, t_1, t_2)) := n + \mathit{weight}(t_1) + \mathit{weight}(t_2) \tag{2}$$

$$\mathit{double} : \mathit{Tree}(\mathbb{N}) \to \mathit{Tree}(\mathbb{N})$$
$$\mathit{double}(\mathit{empty}) := \mathit{empty} \tag{3}$$
$$\mathit{double}(\mathit{node}(n, t_1, t_2)) := \mathit{node}(2n, \mathit{double}(t_1), \mathit{double}(t_2)) \tag{4}$$

we want to prove

$$\forall t \in \mathit{Tree}(\mathbb{N}) : \mathit{weight}(\mathit{double}(t)) = 2 \cdot \mathit{weight}(t)$$



Example (Contd)

Given the type

$$\textbf{type}\ \mathit{Tree}(E) := \mathit{empty} + \mathit{node}(E, \mathit{Tree}(E), \mathit{Tree}(E))$$

in order to prove

$$\forall t \in \mathit{Tree}(E) : F$$

it suffices to prove

$F[\mathit{empty}/t]$

$\forall e \in E, t_1 \in \mathit{Tree}(E), t_2 \in \mathit{Tree}(E) : F[t_1/t] \wedge F[t_2/t] \rightarrow F[\mathit{node}(e, t_1, t_2)/t]$

Therefore, for

$$F \equiv \mathit{weight}(\mathit{double}(t)) = 2 \cdot \mathit{weight}(t)$$

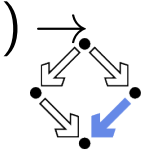
it suffices to prove

$\mathit{weight}(\mathit{double}(\mathit{empty})) = 2 \cdot \mathit{weight}(\mathit{empty})$

$\forall e \in T, t_1 \in \mathit{Tree}(E), t_2 \in \mathit{Tree}(E) :$

$\quad \mathit{weight}(\mathit{double}(t_1)) = 2 \cdot \mathit{weight}(t_1) \wedge \mathit{weight}(\mathit{double}(t_2)) = 2 \cdot \mathit{weight}(t_2) \rightarrow$

$\quad\quad \mathit{weight}(\mathit{double}(\mathit{node}(e, t_1, t_2))) = 2 \cdot \mathit{weight}(\mathit{node}(e, t_1, t_2))$



Example (Contd)

We prove

$$\forall t \in \mathit{Tree}(\mathbb{N}) : \mathit{weight}(\mathit{double}(t)) = 2 \cdot \mathit{weight}(t)$$

by structural induction on t.

- Case $t = \mathit{empty}$: we know

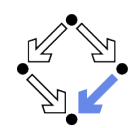
$$\mathit{weight}(\mathit{double}(\mathit{empty})) \overset{(3)}{=} \mathit{weight}(\mathit{empty}) \overset{(1)}{=} 0 \overset{(1)}{=} 2 \cdot \mathit{weight}(\mathit{empty})$$

- Case $t = \mathit{node}(n, t_1, t_2)$: take $n \in \mathbb{N}, t_1, t_2 \in \mathit{Tree}(\mathbb{N})$ and assume

$$\mathit{weight}(\mathit{double}(t_1)) = 2 \cdot \mathit{weight}(t_1) \qquad (*_1)$$
$$\mathit{weight}(\mathit{double}(t_2)) = 2 \cdot \mathit{weight}(t_2) \qquad (*_2)$$

We then know

$$\begin{aligned}
\mathit{weight}(\mathit{double}(\mathit{node}(n, t_1, t_2))) &\overset{(4)}{=} \mathit{weight}(\mathit{node}(2n, \mathit{double}(t_1), \mathit{double}(t_2))) \\
&\overset{(2)}{=} 2n + \mathit{weight}(\mathit{double}(t_1)) + \mathit{weight}(\mathit{double}(t_2)) \\
&\overset{(*)}{=} 2n + 2 \cdot \mathit{weight}(t_1) + 2 \cdot \mathit{weight}(t_2) \\
&= 2 \cdot (n + \mathit{weight}(t_1) + \mathit{weight}(t_2)) \\
&\overset{(2)}{=} 2 \cdot \mathit{weight}(\mathit{node}(n, t_1, t_2)) \quad \square
\end{aligned}$$



Automated Reasoning

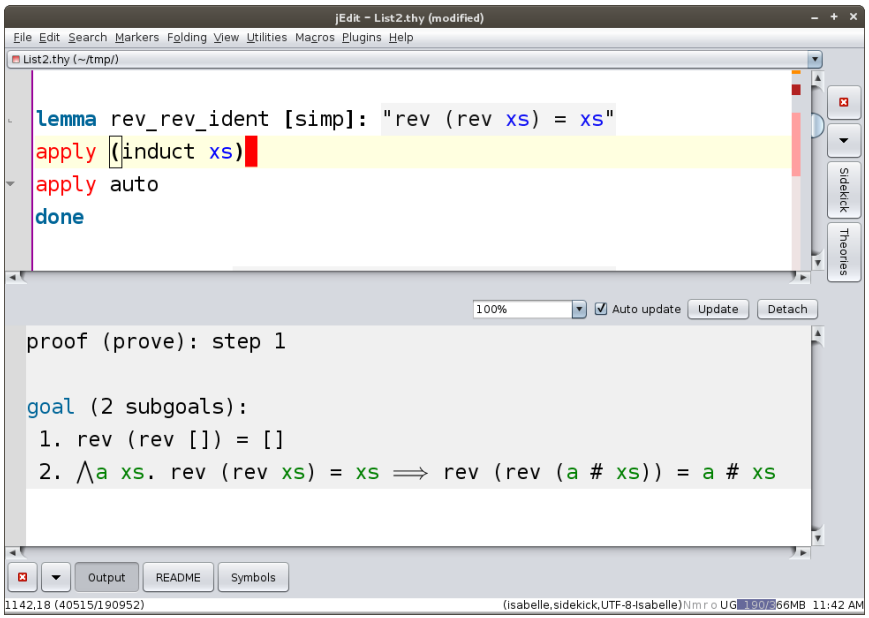
A structural induction proof in the theorem prover Isabelle.

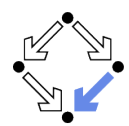
```
datatype 'a list =
    Nil    ("[]")
  | Cons 'a  "'a list"

...

primrec rev ::
  "'a list => 'a list" where
"rev [] = []" |
"rev (x # xs) = rev xs @ [x]"

lemma rev_rev_ident: "rev (rev xs) = xs"
apply (induct xs)
apply auto
done
```





Computer Programs

Also the correctness of loop-based programs can be proved by induction.

- We consider loops of form

  for(i=0; i<n; i++) x = t(x,i);

- We want to prove that
  - if a precondition $P(x)$ holds before the execution of the loop,
  - then a postcondition $Q(x)$ holds afterwards.
- First we prove by induction that, for all $i \leq n$, some suitable loop invariant $I(x,i)$ holds after i iterations of the loop:
  - I holds initially, i.e., after 0 iterations:

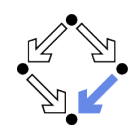
    $$P(x) \to I(x,0)$$

  - If I holds after $i < n$ iterations, then it also holds after $i+1$ iterations:

    $$I(x,i) \wedge i < n \to I(t(x,i), i+1)$$

- It then suffices to prove that at the termination of the loop ($i = n$) the invariant implies the postcondition:

  $$I(x,n) \to Q(x)$$



Example

- Program

  for(i=0; i<n; i++) x = x+2·i+1;

- Precondition $P(x) :\Leftrightarrow x = 0$

x	0	1	4	9	16
i	0	1	2	3	$4{=}n$

- Postcondition $Q(x) :\Leftrightarrow x = n^2$
- Loop invariant $I(x, i) :\Leftrightarrow x = i^2$
  - $P(x) \to I(x, 0)$

    $$x = 0 \to x = 0^2$$

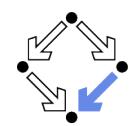
  - $I(x, i) \wedge i < n \to I(x + 2 \cdot i + 1, i + 1)$

    $$x = i^2 \wedge i < n \to x + 2 \cdot i + 1 = (i + 1)^2$$

  - $I(x, n) \to Q(x)$

    $$x = n^2 \to x = n^2$$

The computation of a square as a sum of odd numbers.



Example

- Program

  for(i=0; i<n; i++) x = x+$\frac{1}{2^i}$;

- Precondition $P(x) :\Leftrightarrow x = 0$

x	0	1	$\frac{3}{2}$	$\frac{7}{4}$	$\frac{15}{8}$
i	0	1	2	3	$4{=}n$

- Postcondition $Q(x) :\Leftrightarrow x + \frac{1}{2^{n-1}} = 2$
- Loop invariant $I(x,i) :\Leftrightarrow x + \frac{1}{2^{i-1}} = 2$
  - $P(x) \to I(x,0)$

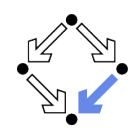
    $$x = 0 \to x + \frac{1}{2^{0-1}} = 2$$

  - $I(x,i) \wedge i < n \to I(x + \frac{1}{2^i}, i+1)$

    $$x + \frac{1}{2^{i-1}} = 2 \wedge i < n \to x + \frac{1}{2^i} + \frac{1}{2^i} = 2$$

  - $I(x,n) \to Q(x)$

    $$x + \frac{1}{2^{n-1}} = 2 \to x + \frac{1}{2^{n-1}} = 2$$



The approximation of a value by a convergent series.

Example

- Program

  for(i=0; i<n; i++) x = x+$a(i)$;

- Precondition $P(x) :\Leftrightarrow x = 0$

x	0	2	5	10	17
i	0	1	2	3	4=n

$a = [2, 3, 5, 7]$

- Postcondition $Q(x) :\Leftrightarrow x = \sum_{j=0}^{n-1} a(j)$
- Loop invariant $I(x, i) :\Leftrightarrow x = \sum_{j=0}^{i-1} a(j)$
  - $P(x) \rightarrow I(x, 0)$

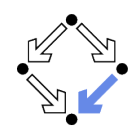
    $$x = 0 \rightarrow x = \sum_{j=0}^{-1} a(j)$$

  - $I(x, i) \wedge i < n \rightarrow I(x + a(j), i + 1)$

    $$x = \sum_{j=0}^{i-1} a(j) \wedge i < n \rightarrow x + a(i) = \sum_{j=0}^{i} a(j)$$

  - $I(x, n) \rightarrow Q(x)$

    $$x = \sum_{j=0}^{n-1} a(j) \rightarrow x = \sum_{j=0}^{n-1} a(j)$$



The summation of an array of values.