

## Léibniz Axiom

(defunc f (x)  
 $\vdots$  ic (natp x)  
 $\vdots$  oc (natp (fx))  
~~(+1 (fx)))~~

A1 ( $\frac{IC}{\text{natp } x} \Rightarrow \text{natp } (fx)$ )

A2 ( $\frac{IC}{\text{natp } x} \Rightarrow \text{function } = \frac{\text{body}}{+B}$ )

T1 ( $\text{natp } x \Rightarrow (x \neq (x+1))$ )  
 $\equiv \{ T1 |_{(x (fx))} \}$

( $\text{natp } (fx) \Rightarrow ((fx) \neq ((fx)+1))$ )

$A \triangleright C$

$A \Rightarrow (B \Rightarrow C) \quad (\text{if } A) \quad (\text{if } B)$

$(A \Rightarrow B) \Rightarrow C \quad (\text{if } (\text{if } A \ C))$

$(\text{in a } (\text{rest } x)) \Rightarrow \underline{C}$

$\Rightarrow \{ A \}$

$(\text{natp } x) \Rightarrow ((fx) \neq ((fx)+1))$

$\equiv \{ \text{Combine with A2} \}$

$(\text{natp } x) \Rightarrow B \wedge \neg B$

$\equiv \{ P \wedge \neg P \equiv \text{nil} \}$

$\text{natp } x \Rightarrow \text{nil}$

$\emptyset$   
 $B((x 4))$   
 $\equiv \text{nil}$   
 $t \Rightarrow \text{nil}$   
 $\equiv \text{nil}$

$Q_{BS}$

(defunc f (x)  
:ic (natp x)  
:oc (nulp (f x))  
(+ x y))

### Definitional Principle

The function f defined as

(defunc f ( $x_1 \dots x_n$ )

: input-contrad ic  
: output-contrad oc

<body>

is admissible provided

1) f is a new function symbol

2) The variable  $x_i$  are distinct

~~(defunc f (x . x))~~ ~~call(f 23)~~

3) A body is a term possibly using f recursively as a function symbol mentioning no variables freely.  
Term = an expression that is well formed given the context of a history.

(defunc f (x)  
: ic (natp x)  
: oc (notp y)  
(super<sup>t</sup> (x) y)))  
Possibly bad  
BAD BAD BAD

(defunc g (x)  
: ic (and (natp x)  
          (notp x))  
: oc (integerp (g x))  
(+ 1 x))

4) The function must terminate. This is the "non-trivial" obligation

5)  $IC \Rightarrow OC$  (<sup>(output)</sup>  
Contract violation)

6) The body contracts hold under the assumption that ic  
holds. (Body Contract Violation)

gives us:  $IC \Rightarrow (f x) \equiv \langle \text{body} \rangle$

---

```
(defunc f (x)
  :ic (natp x)
  :oc (integerp (f x))
  (if (eql x 0)
      1
      (+ 1 (f (- x 1)))))
```

(defunc f (x y)
 :ic (and (notp x) (notp y))
 :oc (~~notp~~ rationalp (fx y)))

---

```
(defunc f (x y)
  :ic (and (notp x) (notp y))
  :oc (integerp (f x y))
  (cond ((eql x 0) 1)
        ((< x 0) (f -1 -1))
        (t (+ 1 (f (- x 1) y))))))
```

A function  $m$  is a measure of  $f$  if:

- 1)  $m$  has the same IC as  $f$
- 2) the OC for  $m$  is  $\text{natp}$
- 3)  $m$  has the same parameters as  $f$  and is admissible
- 4) For each recursive call in  $f$ , given the conditions that lead to recursion  $m$  must decrease with the change in parameters.

$$\begin{array}{c} (\text{natp } x) \wedge (\text{listp } y) \wedge (\text{endp } y) \\ \wedge (x \neq \text{nil}) \Rightarrow (m \text{ } x \text{ } y) > (m \text{ } (-x \text{ } 1) \text{ } (\text{cons } x \text{ } y)) \end{array}$$

(defunc  $f$  ( $x$   $y$ )  
; ic (and (natp  $x$ ) (listp  $y$ ))  
; oc (listp ( $f$   $x$   $y$ ))  
(cond ((endp  $y$ )  
((equal  $x$  0)  
((equal  $x$  1)  
((equal  $x$  2)  
((equal  $x$  3)  
((equal  $x$  4)  
((equal  $x$  5)  
((equal  $x$  6)  
((equal  $x$  7)  
((equal  $x$  8)  
((equal  $x$  9)  
((equal  $x$  10)  
((equal  $x$  11)  
((equal  $x$  12)  
((equal  $x$  13)  
((equal  $x$  14)  
((equal  $x$  15)  
((equal  $x$  16)  
((equal  $x$  17)  
((equal  $x$  18)  
((equal  $x$  19)  
((equal  $x$  20)  
((equal  $x$  21)  
((equal  $x$  22)  
((equal  $x$  23)  
((equal  $x$  24)  
((equal  $x$  25)  
((equal  $x$  26)  
((equal  $x$  27)  
((equal  $x$  28)  
((equal  $x$  29)  
((equal  $x$  30)  
((equal  $x$  31)  
((equal  $x$  32)  
((equal  $x$  33)  
((equal  $x$  34)  
((equal  $x$  35)  
((equal  $x$  36)  
((equal  $x$  37)  
((equal  $x$  38)  
((equal  $x$  39)  
((equal  $x$  40)  
((equal  $x$  41)  
((equal  $x$  42)  
((equal  $x$  43)  
((equal  $x$  44)  
((equal  $x$  45)  
((equal  $x$  46)  
((equal  $x$  47)  
((equal  $x$  48)  
((equal  $x$  49)  
((equal  $x$  50)  
((equal  $x$  51)  
((equal  $x$  52)  
((equal  $x$  53)  
((equal  $x$  54)  
((equal  $x$  55)  
((equal  $x$  56)  
((equal  $x$  57)  
((equal  $x$  58)  
((equal  $x$  59)  
((equal  $x$  60)  
((equal  $x$  61)  
((equal  $x$  62)  
((equal  $x$  63)  
((equal  $x$  64)  
((equal  $x$  65)  
((equal  $x$  66)  
((equal  $x$  67)  
((equal  $x$  68)  
((equal  $x$  69)  
((equal  $x$  70)  
((equal  $x$  71)  
((equal  $x$  72)  
((equal  $x$  73)  
((equal  $x$  74)  
((equal  $x$  75)  
((equal  $x$  76)  
((equal  $x$  77)  
((equal  $x$  78)  
((equal  $x$  79)  
((equal  $x$  80)  
((equal  $x$  81)  
((equal  $x$  82)  
((equal  $x$  83)  
((equal  $x$  84)  
((equal  $x$  85)  
((equal  $x$  86)  
((equal  $x$  87)  
((equal  $x$  88)  
((equal  $x$  89)  
((equal  $x$  90)  
((equal  $x$  91)  
((equal  $x$  92)  
((equal  $x$  93)  
((equal  $x$  94)  
((equal  $x$  95)  
((equal  $x$  96)  
((equal  $x$  97)  
((equal  $x$  98)  
((equal  $x$  99)  
((equal  $x$  100)))  
; or (natp ( $m$   $x$   $y$ ))  
(+  $x$  100)))

(defunc  $f(x,y)$   
 :ic (and (natp  $x$ ) (natp  $y$ ))  
 :oc (integerp ( $f x y$ ))  
 (cond ((equal ( $x 0$ ) 1)  
 (( $< x 0$ ) ( $f x y$ ))  
 (t ( $\exists_1 (f (-x 1) y)$ )))  
  
 C.I. (natp  $x$ )  
 (2 (natp  $y$ ))  
 C. ( $x \neq 0$ )  
 $(m x y)$   
 $= \{ \text{Def } m \}$   
 $x$

$\frac{\text{Obl.}}{\text{C}}$

$\Rightarrow \{ \text{Arithmet.} \}$   
 $\equiv \{ \text{Def } m \}$   
 $(m(-x 1) y)$

$$\frac{\text{Obligation 1}}{\text{nat } x \cdot (\text{nat } y) \wedge (x \neq 0) \wedge x < 0} \quad \text{false}$$
$$\Rightarrow (m \times y) > (m \times y)$$

Obligation

$$\cancel{(\text{nat}_p x) \wedge (\text{nat}_p y) \wedge (x \neq 0)} \quad (\cancel{x \neq 0})$$

$$\Rightarrow ((m x y) \downarrow (m (\neg x 1) y))$$

$$\begin{aligned}
 & \text{C1. } (\text{natp } x) \\
 & \text{C2. } (\text{notp } y) \\
 & \text{C3. } (x \neq 0) \\
 & (m \times y) \\
 & = \left\{ \text{Def } m \right\} \\
 & \quad \Rightarrow \left\{ \begin{array}{l} \sum_{i=1}^n i \\ \text{Arithmetical} \end{array} \right\} \\
 & = \left\{ \begin{array}{l} (-x)^1 \\ \text{Def } m \end{array} \right\}^{a_1, c_3} \dots \Rightarrow (m \times y) = x > (-x) = (m(-x))y \\
 & \quad \Rightarrow (m \times y) = x > (-x) = (m(-x))y
 \end{aligned}$$

```
(defunc f (x1 ... xi ... xn)
  :ic (and (...)(listp xi)...)
  :oc (...(f x1 ... xi ... xn))
```

```
(cond ((endp xi) <stop>
        ((...) (f ... (rest x) ...))
        ((...) (f ... (rest x) ...)))
```

```
(defunc m (x1 ... xi ... xn)
  :ic (and (...)(listp xi)...)
  :oc (natp (f x1 ... xi ... xn))
  (len xi))
```

```
(defunc f (x1 ... xi ... xn)
  :ic (and (...)(natp xi)...)
  :oc (... (f x1 ... xi ... xn))
  (cond ((equal xi 0) <stop>
         ((...) (f ... (- xi 1) ...))
         ((...) (f ... (- xi 1) ...))))
```

```
(defunc m (x1 ... xi ... xn)
  :ic (and (...)(natp xi)...)
  :oc (natp (f x1 ... xi ... xn))
  x)
```

```
(defunc f (r)
  :ic (rationalp r)
  :oc (rationalp (f r))
  (if ( $\leq$  r 0)
      0
      (f (/ r 2))))
```

```
(defunc f (i)
  :ic (integerp i)
  :oc (integerp (f i))
  (if ( $\leq$  i -5)
      i
      (f (- i 1))))
```

```
(defunc m (r)
  :ic (rationalp r)
  :oc (natp (m r))
  r)
}
(defunc m (i)
  :ic (integerp i)
  :oc (integerp (m i))
  (* (+ i 5) (+ i 5)))
```

defunc f (i)

```
:ic (integerp i)
:oc (integerp (f i))
(if (> i 1000)
    0
    (f (+ i 1))))
}
(defunc m (i)
  :ic (integerp i)
  :oc (natp (m i))
  (if (> i 1000)
      -1000
      i))
```

```

(defun f (i l)
  :ic (and (integerp i) (listp l))
  :oc (natp (f i l))
  (cond ((and (equal i 0) (endp l)) 0)
        → ((equal i 0) (f i (rest l)))
        → ((< i 0) (f (+ i 1) l))
        → (t (f (- i 1) l))))

```

```

(defun m (i l)
  :ic (and (integerp i) (listp l))
  :oc (natp (m i l))
  (+ (len l) (if (< i 0) (- i) i))
  or
  (+ (* i i) (len l))
  or
  (+ i (len l))

```

Ob 1  
 $\frac{(\text{integerp } i) \wedge (\text{listp } l) \wedge ((i=0) \wedge (\text{endp } l))}{(\text{integerp } i) \wedge (\text{listp } l) \wedge ((i=0) \Rightarrow (m i l) > (m i (\text{rest } l)))}$

Ob 3  
 $\frac{\overline{(\text{integerp } i)} \wedge (\text{listp } l) \wedge ((i \neq 0) \wedge (\text{endp } l))}{\overline{(\text{integerp } i)} \wedge (\text{listp } l) \wedge ((i \neq 0) \Rightarrow (m i l) > (m (- i 1) l))}$

- ① Turing's Indecidability Problem.
- ② It justifies recursion.
- ③ It justifies the design recipe
- ④ It justifies generative recursion
- ⑤ It justifies induction

Assume

$$\phi_{n-1}$$

$\Rightarrow$

$$\phi_n$$

$$\phi_0 \Rightarrow \phi_1 \Rightarrow \phi_2 \dots \phi_{n-1} \Rightarrow \phi_n$$

..

```
(defunc nind (n)
  :sic (natp n)
  :oc (natp (nind n))
  (if (equal n 0)
      0
      (nind (- n 1))))
```

```
(defunc sum (n)
  :rc (notp n)
  :oc (notp (sum n))
  (if (equal n 0)
      0
      (+ n (sum (- n 1)))))
```

Induction Scheme for nind  
(Is that nind gives rises to)

- ①  $\neg(\text{natp } n) \Rightarrow \phi$
- ②  $(\text{natp } n) \wedge (n=0) \Rightarrow \phi$
- ③  $(\text{natp } n) \wedge (n \neq 0) \wedge \phi_{((n-1))} \Rightarrow \phi$

$$\varphi_{\text{app.assoc}} \quad (\text{list}x), (\text{list}y), (\text{list}z) \Rightarrow ((\text{app}(\text{app}xy)z) = (\text{app}x(\text{app}yz)))$$

$$\varphi_{\text{sumn}} \quad (\text{natp}n) \Rightarrow ((\text{sumn } n) = (n * (n+1)) / 2)$$

$$\frac{\text{Obligation 1}}{(\text{natp}n) \Rightarrow ((\text{natp}n) \Rightarrow ((\text{sumn } n) = (n * (n+1)) / 2))}$$

$$\frac{\text{Obl.gation 2}}{(\text{natp}n), (n=0) \Rightarrow ((\text{natp}n) \Rightarrow ((\text{sumn } n) = (n * (n+1)) / 2))}$$

$$\frac{\text{Obligation 3}}{(\text{natp}n), (n \neq 0) \cdot \left[ (\text{natp}(-n)) \Rightarrow ((\text{sumn } (-n)) = ((-n) * ((-n)+1)) / 2) \right] \\ \Rightarrow ((\text{natp}n) \Rightarrow ((\text{sumn } n) = (n * (n+1)) / 2))}$$

C1.  $\neg(\text{nat}^n)$   
C2.  $\underline{\neg \text{nat}^n}$   
-  
C3.  $\text{nil} \in \{C1, C2\}$   
 $n \neq 1 \Rightarrow \varphi \equiv \text{true}$

If  $\varphi_{\text{app-assoc}}$   
C1.  $\neg(\text{nat}^n)$   
C2.  $(\text{list}^n x)$   
C3.  $(\text{list}^n y)$   
C4.  $(\text{list}^n z)$