1 WFI or not?

```
take 0 xs     = []
take n []     = []
take n (x:xs) = [x]++(take (n-1) xs)

drop 0 xs     = xs
drop n []     = []
drop n (x:xs) = drop (n-1) xs
```

Claim 1.1. The following equation holds.

```
(take n xs)++(drop n xs) = xs
```

Proof. by induction over n.

BC: n=0:

```
  (take n xs)++(drop n xs)
= (take 0 xs)++(drop 0 xs)
= []++xs
= xs
```

IH: `(take n' xs)++(drop n' xs) = xs`

holds for an arbitrary but fixed n' and all xs.

IS: n=n'+1

Case 1: xs=[]

```
  (take n xs)++(drop n xs)
= (take (n'+1) [])++(drop (n'+1) [])
= []++[]
= []
= xs
```

Case 2: xs=(x:xs')

```
  (take n xs)++(drop n xs)
= (take (n'+1) (x:xs'))++(drop (n'+1) (x:xs'))
= [x]++(take n' xs')++(drop n' xs')
= [x]++xs'
= xs
```

□

```
zip [] ys         = []
zip xs []         = []
zip (x:xs) (y:ys) = [(x,y)]++(zip xs ys)

length []     = 0
length (x:xs) = 1+(length xs)

min x y
   |x>y   = y
   |true  = x
```

Claim 1.2. The following equation holds.

```
length (zip xs ys) = min (length xs) (length ys)
```

Proof. by structural induction over xs.

BC: xs=[]

```
  length (zip xs ys)
= length (zip [] ys)
= length []
= 0
= min 0 (length ys) --since (length ys) >= 0
= min (length xs) (length ys)
```

IH: `length (zip xs' ys) = min (length xs') (length ys)`

holds for an arbitrary but fixed xs' and all ys.

IS: xs=(x:xs')

Case 1: ys=[]

```
  length (zip xs ys)
= length (zip (x:xs') [])
= length []
= min (length (x:xs')) 0 --since (length (x:xs')) > 0
= min (length xs) (length ys)
```

Case 2: ys=(y:ys')

```
  length (zip xs ys)
= length (zip (x:xs') (y:ys'))
= length ([(x,y)]++(zip xs' ys'))
= 1+length (zip xs' ys')
= 1+(min (length xs') (length ys'))
= min (1+(length xs')) (1+(length ys'))
= min (length (x:xs')) (length (y:ys'))
= min (length xs) (length ys)
```

□

But now just for the fun of it let us proof Claim 1.2 by well founded induction.

$$(xs, ys) \prec ((x : xs), (y : ys))$$

We denote the transitive closure of $\prec$ by $<$. Obviously $<$ is a well-founded order.

Proof. by well-founded induction with the predicate

```
P((xs,ys))=(length (zip xs ys) = min (length xs) (length ys))
```

xs=[]: (Note that ([],ys) has no successors w.r.t. $<$.)

```
  length (zip xs ys)
= length (zip [] ys)
= length []
= 0
= min 0 (length ys) --since (length ys) >= 0
= min (length xs) (length ys)
```

ys=[]: This case can be treated analogously to xs=[].

xs=(x:xs'),ys=(y:ys)

```
  length (zip xs ys)
= length (zip (x:xs') (y:ys'))
= length ([(x,y)]++(zip xs' ys'))
= 1+(length (zip xs' ys'))
= 1+(min (length xs') (length ys')) {-since (xs',ys')<(xs,ys)
                                        P((xs',ys')) holds-}
= min (1+(length xs')) (1+(length ys'))
= min (length (x:xs')) (length (y:ys'))
= min (length xs) (length ys)
```

□

```
a 0 m = m+1
a n 0 = a (n-1) 1
a n m = a (n-1) (a n (m-1))

alist []     ys     = [1]++ys
alist (x:xs) []     = alist xs [x]
alist (x:xs) (y:ys) = alist xs (alist (x:xs) ys)
```

Claim 1.3. The following equation holds.

```
length (alist xs ys) = a (length xs) (length ys)
```

$$(xs, ys) \prec (\quad xs\ , (y : ys))$$
$$(xs, zs) \prec ((x : xs),\quad ys\ )$$

```
r []     = []
r (x:xs) = r (xs++[(x+1)]++[(x+1)])
```

Claim 1.4. The following equation holds.

```
r xs = []
```