Zum Beispiel in der Sprache Curry:

```
coin :: Int
coin = 0
coin = 1
```

```
double :: Int → Int
double x = x + x
```

```
> coin
0
More?
1
More?
No more Solutions
```

```
> double  coin
0
More?
2
More?
No more Solutions
```

```
coin :: Int
coin = 0 ? 1
```

Zum Beispiel in der Sprache Curry:

```
f :: a → [a] → [a]
f x ys       =  x : ys
f x (y : ys) =  y : f x ys
```

```
g :: [a] → [a]
g [ ]       =  [ ]
g (x : xs)  =  f x (g xs)
```

```
> f 3 [1,2]
[1,2,3]
More?
[1,3,2]
More?
[3,1,2]
More?
No more Solutions
```

```
> g [1,2,3]
[3,2,1]
More?
[3,1,2]
More?
[2,3,1]
More?
…
```

Zum Beispiel in der Sprache Curry:

```
list :: [Int]
list = ys ++ [1]
    where  ys free
```

```
f :: [a] → a
f xs | ys ++ [y] == xs  =  y
    where  ys, y free
```

```
> list
[1]
More?
[_a,1]
More?
[_a,_b,1]
More?
…
```

```
> f [1..4]
4
More?
No more Solutions
```

```
f :: [a] → a
f (_ ++ [y]) =  y
```

Zebra-Puzzle funktional-logisch (1)

```
data Color       =  Red | Yellow | Blue | Green | Ivory
data Nationality =  Norwegian | Englishman | Spaniard | Ukrainian | Japanese
data Drink       =  Coffee | Tea | Milk | Juice | Water
data Pet         =  Dog | Horse | Snails | Fox | Zebra
data Smoke       =  Winston | Kools | Chesterfield | Lucky | Parliaments

right_of :: a → a → [a] → Success
right_of r l (h1 : h2 : hs)  = (l =:= h1 & r =:= h2)  ?  right_of r l (h2 : hs)

next_to :: a → a → [a] → Success
next_to x y = right_of  x y
next_to x y = right_of  y x

member :: a → [a] → Success
member x (y : ys)  = x =:= y  ?  member x ys
```

Zebra-Puzzle funktional-logisch (2)

```
zebra :: ([(Color,Nationality,Drink,Pet,Smoke)], Nationality)
zebra  |   member (Red, Englishman, _, _, _) houses
           & member (_, Spaniard, _, Dog, _) houses
           & member (Green, _, Coffee, _, _) houses
           & member (_, Ukrainian, Tea, _, _) houses
           & right_of (Green, _, _, _, _) (Ivory, _, _, _, _) houses
           & member (_, _, _, Snails, Winston) houses
           & member (Yellow, _, _, _, Kools) houses
           & next_to (_, _, _, _, Chesterfield) (_, _, _, Fox, _) houses
           & next_to (_, _, _, _, Kools) (_, _, _, Horse, _) houses
           & member (_, _, Juice, _, Lucky) houses
           & member (_, Japanese, _, _, Parliaments) houses
           & next_to (_, Norwegian, _, _, _) (Blue, _, _, _, _) houses
           & member (_, zebraOwner, _, Zebra, _) houses
           & member (_, _, Water, _, _) houses
       = (houses, zebraOwner)
       where
           houses = [(_, Norwegian, _, _, _), _, (_, _, Milk, _, _), _, _]
           zebraOwner = _
```