PHIL3110 - Assignment 2

Maxwell Bo

May 20, 2018

Problem 1

Definition 1 *$f : \mathcal{X} \rightarrowtail \mathcal{Y}$ is is injective $\Leftrightarrow \forall x_1, x_2 \in \mathcal{X}$ if $F(x_1) = F(x_2)$ then $x_1 = x_2$.*

Theorem 1 *If $f : \mathcal{X} \rightarrowtail \mathcal{Y}$ is injective and $g : \mathcal{Y} \rightarrowtail \mathcal{Z}$ is injective, then $g \circ f$ is injective.*

Proof 1 *Suppose $f : \mathcal{X} \rightarrowtail \mathcal{Y}$ is injective and $g : \mathcal{Y} \rightarrowtail \mathcal{Z}$ is injective. We must show that $g \circ f$ is injective. Suppose x_1 and x_2 are elements of $\mathcal{X}$ such that*

$$(g \circ f)(x_1) = (g \circ f)(x_2)$$

By definition of composition of functions,

$$g(f(x_1)) = g(f(x_2))$$

Since g is injective

$$f(x_1) = f(x_2)$$

And since f is injective

$$x_1 = x_2$$

Theorem 2 *If there is some injection $f : A \rightarrowtail B$, then $A \precsim B$, and vice-versa.*

If A, B and C are sets such that $A \precsim B$ and $B \precsim C$, there exists an injective $f : A \rightarrowtail B$ and injective $g : B \rightarrowtail C$. Per theorem 1, $g \circ f : A \rightarrowtail C$ is injective. Per thereom 2, $g \circ f$ implies $A \precsim C$.

Problem 2

1. $$\varphi_2 := (\exists x)(\exists y)(x \neq y)$$

   says that there at least 2 objects.

   Thus

   $$\varphi_n := (\exists x_1)(\exists n_2) \dots (\exists x_n)(x_1 \neq \; x_2 \neq \dots \neq x_n)$$

   says that for any natural number n, there are at least n objects.

2.

Problem 3

```
; This program accepts a block of n-many 1s and outputs a block of 2n-many 1s,
; after the original block with a single blank space separating them;

; Henceforth:
; - the n-many 1's will be referred to as the "parameter array"
; - the 2n-many 1's will be referred to as the "accumulator array"
; - the single blank space seperating them will be referred to as "the divider"

; ALGORITHM SUMMARY: Move a loop pointer through the parameter array,
; terminating the loop when the pointer reaches the end of the parameter array.
; On each loop, append two 1s to the end of the accumulator array.

; ### State 0: as per the assignment sheet, the head should start under the 1th cell
0 -  -R 1

; ### State 1 deals with placing our loop pointer, and halting the loop
; Our loop pointer is a blank, that shifts through our parameter array.
; where [_ 1 1 1] starts the loop, and [1 1 1 1] halts the loop

; This instruction puts down the new loop pointer.
; This either initializes it (if we came from State 0),
; or increments it (if we came from State 6)
1 1 -R 2

; ### State 2 deals with getting to the start of the accumulator array
; Glide over the parameter array
2 1 1 R 2

; Jump over the divider
2 --R 3

; ### State 4 and 5 deal with getting to the end of the accumulator,
; and appending two 1s.
; Glide over the accumulator...
3 1 1 R 3

; ... until we pop out the end of the accumulator. Put down a 1...
3 -1 R 4

; ... and another one. Now we've gotta turn back around and increment the loop variable.
4 -1 L 5

; ### State 5 deals with trying to get back to the end of the parameter array
; Glide back over the accumulator array
5 1 1 L 5
; Jump over the divider
5 --L 6

; ### State 6 deals with incrementing our loop variable
; Glide over our parameter array...
6 1 1 L 6

; ...until we hit our loop pointer. We clear the current loop pointer, shift the
; head right, and loop back to state 0, so that it may deal with the next loop iteration.
```

```
6 -1 R 1

; This is our loop halting condition. State 6 has cleared the loop pointer,
; and pushed the head onto the divider. We still have to move the head to the 1th cell.
1 --L 7

; #### State 7 deals with getting to the 1th cell, and halting
; Glide back over our parameter array...
7 1  1 L 7

; ... and when we pop out past the head of the parameter array, shift right to
; the 1th cell, and halt
7 --R halt
```