# Discrete maths

Completely identical to a past paper

# Logic

## ai)

Show the following using truth tables:

A close-up of a symbol

Description automatically generated

## aii)

## A black text on a white background Description automatically generated aiii)

A math equations on a white background

Description automatically generated

## bi)A) (skip all the explanation in the exam because you will not have time)

1. The sentence means “Every object seen by x sees something that sees itself”. 1 is the only object that sees itself.

* 1 works because 1 only sees 1, 1 sees something that sees itself, namely, 1.
* 2 works because 2 only sees 1, 1 sees something that sees itself, namely, 1.
* 3 doesn’t work because 3 sees 4 which doesn’t see anything that sees itself.
* 4 doesn’t work because 4 sees 6 which doesn’t see anything that sees itself.
* 5 works because 5 only sees 2, 2 sees something that sees itself, namely, 1.
* 6 works because 6 doesn’t see anything.

So x = 1, 2, 5, or 6.

1. The sentence means “x is seen by at most one circle”.

* 1 works because 1 is only seen by one circle (1).
* 2 doesn’t work because 2 is seen by two circles (1 and 5).
* 3 works because no circle sees 3.
* 4 works because no circle sees 4.
* 5 works because no circles sees 5.
* 6 works because no circle sees 6.

So x = 1, 3, 4, 5, or 6.

1. The sentence means “x sees exactly two objects”.

* 1 works because 1 sees 1 and 2.
* 2 doesn’t work because 2 only sees 1.
* 3 works because 3 sees 2 and 4.
* 4 works because 4 sees 5 and 6.
* 5 doesn’t work because 5 only sees 2.
* 6 doesn’t work because 6 sees nothing.

So x = 1, 3, or 4.

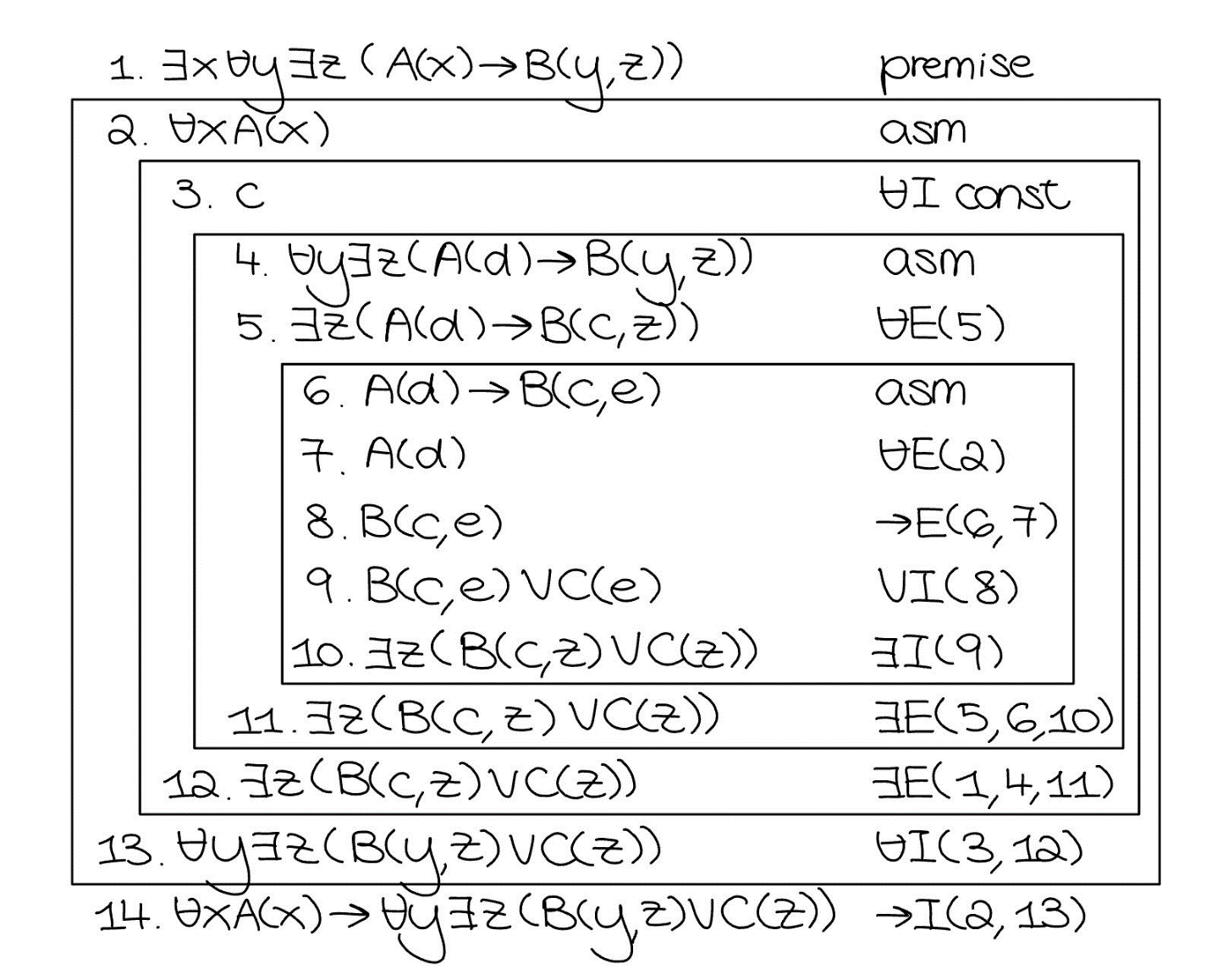
## bi)B)

Q(x) iff x doesn’t see anything that is in Q.

* 1 must not be in Q, since 1 sees itself.
* 2 must be in Q, since 2 only sees 1 which cannot be in Q.
* 3 and 5 must not be in Q, since 3 sees 2 which must be in Q.
* 6 must be in Q, since 6 sees nothing.
* 4 must not be in Q, since 4 sees 6 which must be in Q.

So 2, 6 must be in Q.

## c



# Reasoning

## 1ai)

p1 weighs 1 + (1 + 2 + 4) = 7

p2 weighs 1 + 9 = 10

p3 weighs 1 + (2 + 5) = 8

so 7 + 10 + 8 = 25

## aii)

[Weight 1, Weight 2, Weight 4, Weight 9, Weight 2, Weight 5]

## bi)

A black text on a white background

Description automatically generated

## ii)

Base case 1

To show: weight Empty >= leight (repack Empty)

weight Empty

= 0 by def of weight

= leight [] by def of leight

= leight (repack Empty) by def of repack

Base case 2

Take arbitrary bs: [Box].

To show: weight (Pallet bs) >= leight (repack (Pallet bs))

weight (Pallet bs)

= 1 + leight bs by def of weight

> leight bs by arith

= leight (repack (Pallet bs)) by def of repack

Inductive step

Take arbitrary o1, o2: Order.

(IH1) weight o1 >= leight (repack o1)

(IH2) weight o2 >= leight (repack o2)

To show: weight (Truckload o1 o2) >= leight (repack (Truckload o1 o2))

weight (Truckload o1 o2)

= weight o1 + o2 by def of weight

>= leight (repack o1) + leight (repack o2) by (IH1) and (IH2)

= leight (repack o1 ++ repack o2) by (leight++)

= leight (repack (Truckload o1 o2)) by def of repack

## ci)

w? is the number of pallets in o

## cii)

numPallets :: Order -> Int

numPallets Empty = 0

numPallets (Pallet \_) = 1

numPallets (Truckload o1 o2) = numPallets o1 + numPallets o2

## di)

A close-up of a number

Description automatically generated

## dii)

Base case

To show: Even(double(Succ(Zero)))

1. Even(Succ(Succ(Zero))) by (R5) and (R6)
2. Even(Succ(Succ(double(Zero))) by (1), def of double
3. Even(double(Succ(Zero))) by (2), def of double

Inductive step

Take arbitrary n in S\_N.

(Ass) Odd(n)

(IH) Even(double(n))

To show: Even(double(Succ(Succ(n)))

1. Even(Succ(Succ(double(n))) by def of Even
2. Even(double(Succ(Succ(n))) by (1) def of double

Notice we didn’t actually need to do induction here, so the proof can simply be (not sure how much Mark cares about the word “hence”):

Take arbitrary n in S\_N.

(Ass) Odd(n)

To show: Even(double(Succ(Succ(n)))

1. Even(Succ(Succ(double(n))) by def of Even
2. Even(double(Succ(Succ(n))) by (1) def of double

Done.

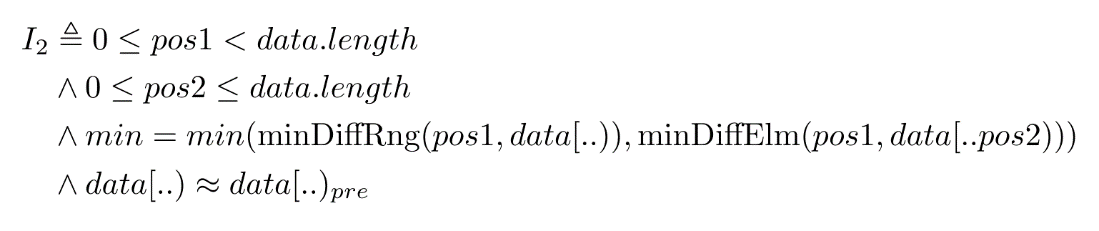
## 2a

|  |  |  |  |
| --- | --- | --- | --- |
| Pos1 | Pos2 | Diff | Min |
| 0 | 1 | - | infinity |
| 0 | 2 | 5 | 5 |
| 0 | 3 | 3 | 3 |
| 1 | 1 | 5 | 3 |
| 1 | 2 | - | 3 |
| 1 | 3 | 2 | 2 |
| 2 | 1 | 3 | 2 |
| 2 | 2 | 2 | 2 |
| 2 | 3 | - | 2 |

## bi)



## bii)



Notice pos1 is strictly less than data.length since we are in the body of the outer loop.

## c

We don’t need to calculate the difference between the element at pos1 and elements at any pos < pos1, since we would have already calculated the difference on the inner loop iteration where pos1 = pos and pos2 = pos1. This relies on the mathematical property that |x – y| = |y – x|.

The case where pos2 = pos1 is redundant as it will be skipped over by the if condition.

## d

A white background with black dots

Description automatically generated

A white background with black and white clouds

Description automatically generated

A white background with black text

Description automatically generated