Andreas van Cranenburgh 0440949
Core Logic
Sat Oct 17 12:40:30 CEST 2009

Exercise 9

a) Doubt is cast on the claim that "among the Peripatetics only Theophrastus and Eudemus made even the barest beginnigs" of a theory of hypothetical syllogisms by the work of Avicenna. It seems Boethius and Avicenna did not know of or use each other's work, although their work is similar in some respects.

b) Abelaerd's principle

c)

1. The Stoics
2. Abaelard
3. Logicians of the twelfth century
4. Frege

Exercise 10

1) Exhibit A: The theory of "obligationes."
Exhibit B: The theory of "exposition" and exponible" propositions.
Exhibit C the theory of "proofs of propositions."
Exhibit D, the theory of "supposition."

2) Immediate terms are things that need to be verified empirically by direct perception or intuited instead of by recourse to proofs in terms of other propositions.

3) The "proof" doesn't reduce the statement that a man runs to immediate terms like it should according to the theory, and it appears this is generally not possible in this theory. The "proof" appears to be circular.

4)

a) I think Spade would not subscribe to this statement, because the mysteriousness of medieval logic appears to be a specific case, whereas in other areas such problems do not arise with "even the most basic starting points".

b) I think also this claim should be rejected, because it rather appears that the theories suddenly appear "full-grown from the head of Zeus". It seems implausible that not having access to the fundamental texts would happen repeatedly for various theories of medieval logic. Instead it might be the case that the theories developed as an oral tradition.

Exercise 11

For the case n = 2 there is exactly one operator which defines medieval disjunction, exclusive or:

A	B	A xor B
0	0	0
0	1	1
1	0	1
1	1	0

It is easy to see that any other binary truth function would fail.
Problems immediately arise when exclusive or is applied to a third argument:

A	B	xor	C	xor
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	0	0	0
0	0	0	1	1
0	1	1	1	0
1	0	1	1	0
1	1	0	1	1

We see that in the first seven cases all is well, but in the last case where all operands are true the outcome should be false but isn't. Since we already exhausted all 16 possible binary truth functions in the case n = 2, it has to be concluded that there is no binary thruth function which induces medieval disjunction.

Exercise 12

1)
The rules of the (strictly constructive) game:
 - In each move, the action and the announcment have to fit together
 - In round n + 1, the Opponent has to either attack or defend against round n
 - The Proponent may only attack or defend against the previous round.
 - The Opponent may assert any atomic formulas
 - The Proponent may assert only atomic formulas that have been asserted by the
   Opponent before.

2)

Opponent	Proponent
0	assert(p -> p)
1 attack(0) what if? assert(p)	
2	defend(1) assert(p)
3 ---	

Opponent	Proponent
0	assert(~p -> ~p)
1 attack(0) what if? assert(~p)	
2	defend(1) assert(~p)
3 ---	

3)
in |= dialog:

Opponent	Proponent
0	assert(p^q -> p)
1 attack(0) what if? assert(p^q)	
2	attack(1) left?
3 defend(2) assert(p)	
4	defend(0) assert(p)
5 ---	

in |= sc

Opponent	Proponent
0	assert(p^q -> p)
1 attack(0) what if? assert(p^q)	
2	attack(1) left?
3 defend(2) assert(p)	
4	---

Exercise 13

```
In |= class
 Opponent                      Proponent
0                             assert(~~~p -> ~p)
1 attack(0) what if? assert(~~~p)
2                             attack(1) what if? assert(~~p)
3 attack(2) what if? assert(~p)
4                             defend(1) assert(~p)
5 ---
```

In |= dialog the same proof can't be made, but the formula is provable in intuitionistic logic. Here we make use of the fact that an attack on ~p can't be defended against:

```
In |= dialog
 Opponent                      Proponent
0                             assert(~~~p -> ~p)
1 attack(0) what if? assert(~~~p)
2                             attack(1) what if? assert(~~p)
3 attack(2) what if? assert(~p)
4                             attack(3) assert(p)
5 ---
```

```
In |= class
 Opponent                      Proponent
0                             assert(((p->q) ^ ~q) -> ~p)
1 attack(0) what if? assert((p->q) ^ ~q)
2                             attack(1) left?
3 defend(2) assert(p->q)
4                             attack(1) right?
5 defend(4) assert(~q)
6                             attack(3) what if? assert(p)
7 ---
```

In |= dialog the same proof can be made.