Artificial Intelligence Tutorial 3 - Answers

1a) Translate the following sentences into first order logic.

(i) All dogs are mammals
(ii) Fido is a dog
(iii) Fido is a mammal
(iv) All mammals produce milk

1b) Use the Modus Ponens deduction rule to deduce sentence (iii) from (i) and (ii). What did you unify in order to use Modus Ponens, and what substitution made the unification possible?

1c) Write the answers from part (a) in conjunctive normal form.

1d) Translate the following sentence into CNF: “There exists a dog which doesn’t produce milk.” [You will have to use *skolemisation* for this – see the lecture notes. Also Russell & Norvig give a good explanation.]

1e) Explain, in English, why this new sentence contradict sentences (i) to (iv) from part (a).

1a) The translation into first order logic is fairly straightforward:

(i) $\forall X\ (dog(X) \rightarrow mammal(X))$.
(ii) dog(fido).
(iii) mammal(fido).
(iv) $\forall X\ (mammal(X) \rightarrow milk(X))$.

1b) To use Modus Ponens, we need a sentence that A implies B, and a sentence stating that A is true. We will then deduce B from it. Our “A implies B” sentence will be (i) above and our “A is true sentence” will be (ii) above. However, the left hand side of (i) is: dog(X), whereas (ii) is: dog(fido), so, as it stands, we cannot use Modus Ponens. So, we must unify dog(X) and dog(fido). This is done by substituting the constant fido for X. We can now apply this substitution as part of the deduction step. To show that the deduction has occurred, we write it as follows:

$$\frac{\forall X\ (dog(X) \rightarrow mammal(X))\ ,\quad dog(fido).}{mammal(fido)}$$

Because mammal(fido) appears below the line, we have deduced this sentence using our substitution and the Modus Ponens rule.

1c) Remember that CNF sentences are disjunctions of literals, where a literal can be a predicate or negated predicate. We also need to remember the rewrite rules the lecture notes prescribed for turning a normal first order sentence into a CNF sentence. For these particular sentences, we need to remember the rewrite rule

R: $X \rightarrow Y$ can be re-written as $\neg X \vee Y$ and vice versa

Also remember that we assume universal quantification for CNF statements. The four sentences are therefore expressed in CNF as:

(i) $\neg\ dog(X) \vee mammal(X)$
(ii) dog(fido)
(iii) mammal(fido)
(iv) $\neg\ mammal(X) \vee milk(X)$

1d) This new sentence is a little trickier to translate into CNF. Firstly, we need to write it in standard first order logic. We see that it is an existence statement, therefore requires an existential quantifier. Also, there is a conjunction in there (an object which is a dog *and* doesn't produce milk) and the *doesn't* will be modelled with a negate sign. Thus, we get:

∃X (dog(X) ∧ ¬milk(X))

Now, our first problem is that we cannot have existentially quantified variables in CNF, because variables are assumed to be universally quantified, and we can't have any exceptions. Remember the existential elimination rule? We can do something similar here: if we know that some animal exists which is a non-milk-producing dog, then why don't we just give it a name? We have to be careful not to use a constant name which has been used already (so, lets avoid fido), but that is the only restriction. Lets use the name: some_animal. This means we can remove the existential quantifier and simply write:

dog(some_animal) ∧ ¬ milk(some_animal)

Note that the process of introducing a new constant or function in order to remove the existential quantification is known as Skolemisation, although the full Skolemisation process is a little more complicated. Don't think we've finished, however: we have a conjunction, so our sentence is not in conjunctive normal form. In fact, we need to think of this as two clauses, both of which will be in CNF:

dog(some_animal)
¬ milk(some_animal)

As CNF is taken to be a big conjunction of all the disjunctions, we haven't lost anything by thinking of our single sentence as two others. In English, we're saying that there is some animal which is a dog and that the same some animal does not drink milk. It should be clear that this is equivalent to the original sentence.

1e) Dogs are mammals and mammals produce milk, so dogs produce milk. That means there can't be a dog which doesn't produce milk, but our answer from part (d) says there is. So (d) must be contradicting (i) to (iv).

2a) Write down a substitution which unifies the following sentences:

(i) likes(homer, dinner(X)) ∧ (today(A) → dinner(X)).
(ii) likes(homer, dinner(cooked_by(Y))) ∧ (today(Z) → dinner(cooked_by(marge))).

[Universal quantification of variables is assumed.]

2b) What are the compound operators in sentence (i) from part (a)? For each compound expression X, write down the results of the functions args(X) and op(X). [Refer to the unification algorithm given in lecture 8 for the explanation of these functions].

2c) Explain why these two sentences cannot be unified:

(i) p(X) ∨ (q(f(X)) ∧ r(Y)).
(ii) p(X) ∨ (q(X) ∧ r(s)).

2a) We are looking to replace variables with terms which make the two sentences look the same. For these sentences, if we replace X by cooked_by(Y), then the sentences become:

likes(homer, dinner(cooked_by(Y))) ∧ (today(A) → dinner(cooked_by(Y))).
likes(homer, dinner(cooked_by(Y))) ∧ (today(Z) → dinner(cooked_by(marge))).

These sentences look more similar, but, looking at the end of sentence 2, we see that Y will need to be replaced by marge. This gives us:

likes(homer, dinner(cooked_by(marge))) ∧ (today(A) → dinner(cooked_by(marge))).
likes(homer, dinner(cooked_by(marge))) ∧ (today(Z) → dinner(cooked_by(marge))).

Now we simply have to substitute A for Z (or vice versa). This gives us:

likes(homer, dinner(cooked_by(marge))) ∧ (today(A) → dinner(cooked_by(marge))).
likes(homer, dinner(cooked_by(marge))) ∧ (today(A) → dinner(cooked_by(marge))).

Which are exactly the same. Hence the substitution is {X/cooked_by(Y), Y/marge, Z/A}. As we are substituting Y by marge, we should do this in the cooked_by replacement for X. Hence, we should write the substitution as: {X/cooked_by(marge), Y/marge, Z/A}.

2b) Firstly, the compound operators (as described in the unification algorithm) are either predicates, functions or connectives. Hence in the sentence (i) from part (a), the following are compound operators: likes, dinner, today, ∧, →. These symbols would also be returned by the op function, by definition (see the lecture notes).

The args function returns a list of arguments for a predicate, function or connective. Hence, for the predicates and functions, the args function would calculate:

args(likes(homer, dinner(X))) = [homer, dinner(X)]
args(dinner(X)) = [X]
args(today(A)) = [A]
args(dinner(Y)) = [Y]

and for the connectives, the args function would calculate:

args(today(A) → dinner(Y)) = [today(A), dinner(Y)]
args(likes(homer, dinner(X)) ∧ (today(A) → dinner(Y))) =
[likes(homer, dinner(X)), (today(A) → dinner(Y))]

2c) Here, we see that substituting ground symbol s for variable Y doesn't cause a problem. It appears, however, that we also need to substitute f(X) for X. If we do this to both sentences, we get:

p(X) ∨ (q(f(f(X))) ∧ r(s))
p(X) ∨ (q(f(X)) ∧ r(s))

So, the substitution has not unified the two sentences. If we continued trying to substitute X by f(X), we would just get longer sentences, not a unification. This is called an *occurs-check error*. The unification algorithm is designed to check that something which is being substituted for something else does not contain the thing it is being substituted for (as above), and when this is the case, it fails, as the occurs-check error would happen.

3) Rewrite the following propositional sentence to be in conjunctive normal form:

$(P \rightarrow Q) \vee \neg(Q \vee \neg R)$

We are going to use the propositional rewrite rules to turn the sentence into conjunctive normal form, i.e., a conjunction of bracketed disjunctiones, e.g., $(X \vee Y \vee Z) \wedge (A \vee B)$, etc. There is some trial and error involved here – don't be surprised if some re-writing choices you make turn out to be bad choices. There is a skill to using these rewrite rules, which can be learned over time.

We start with:

$(P \rightarrow Q) \vee \neg(Q \vee \neg R)$

Whenever you see an implication, it's a good idea to get rid of it straight away. So, using the equivalence that $(P \rightarrow Q)$ can be rewritten as $\neg P \vee Q$ we get:

$(\neg P \vee Q) \vee \neg(Q \vee \neg R)$

Similarly, whenever you see a negation around a bracket (which isn't allowed in CNF), it's a good idea to get rid of this as soon as possible. Hence, using one of de Morgan's laws, we see that $\neg(Q \vee \neg R)$ can be written as $(\neg Q \wedge \neg\neg R)$. We will probably want to remove the double negation in the same step, giving us:

$(\neg P \vee Q) \vee (\neg Q \wedge R)$

Introducing the conjunction sign seems to be a step backwards, but it's not a problem if we move it outside of the bracket. We can use the distributivity of or over and to do this: $X \vee (Y \wedge Z)$ can be written as $(X \vee Y) \wedge (X \vee Z)$. If we make X be $(\neg P \vee Q)$, Y be $\neg Q$ and Z be R, then the rewritten sentence is:

$((\neg P \vee Q) \vee \neg Q) \wedge ((\neg P \vee Q) \vee R)$

If we now flatten the disjunctions, we get:

$(\neg P \vee Q \vee \neg Q) \wedge (\neg P \vee Q \vee R)$

but, we can replace $Q \vee \neg Q$ in the first clause to give us:

$(\neg P \vee \text{True}) \wedge (\neg P \vee Q \vee R)$

and $(\neg P \vee \text{True})$ is always true. Hence the left hand clause is always true, so the truth of the sentence only depends on the right hand clause. Hence, we can rewrite the sentence, finally, as:

$(\neg P \vee Q \vee R)$