Workshop Testing and Formal Methods, Week 3: Answers

This workshop is about predicate logic. Predicate logic is explained in chapter 2 of "The Haskell Road" and in Chapter 4 of the "Logic in Action" ebook, available from `www.logicinaction.org`. If the following exercises are difficult for you, you should study the relevant material (again).

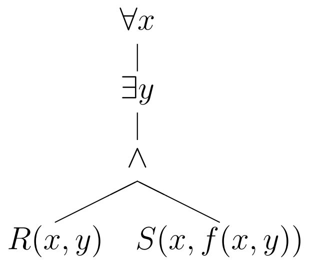
Question 1 The language of predicate logic is given by the following grammar (see the references above):

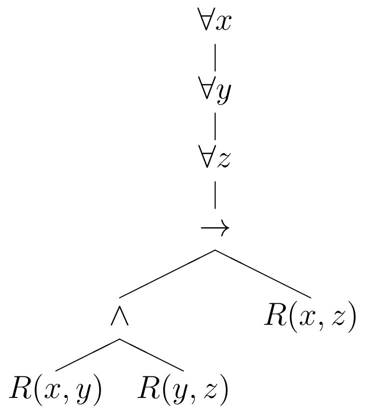
$$\begin{array}{lcl} t & ::= & v \mid f(t,\dots,t) \\ \varphi & ::= & P(t_1,\dots,t_n) \mid t_1 = t_2 \mid (\neg\varphi) \mid (\varphi\wedge\varphi) \mid (\varphi\vee\varphi) \mid (\varphi\rightarrow\varphi) \mid (\varphi\leftrightarrow\varphi) \mid (\forall x\varphi) \mid (\exists x\varphi) \end{array}$$

Draw parse trees for the following formulas:

1. $(\forall x(\exists y(R(x,y)\wedge S(x,f(x,y)))))$.
2. $(\forall x(\forall y(\forall z((R(x,y)\wedge R(y,z))\rightarrow R(x,z)))))$.

Answer:





Question 2 Analogous to the case in propositional logic, a literal is a basic predication or its negation, where the basic predications are

$$P(t_1,\dots,t_n) \text{ and } t_1 = t_2.$$

Give a definition of the grammar of predicate logical formulas in negation normal form (no arrows or double arrows, negations can only occur in literals).

Answer:

$$\begin{array}{lcl} L & ::= & P(t_1,\dots,t_n) \mid t_1 = t_2 \mid \neg P(t_1,\dots,t_n) \mid \neg t_1 = t_2 \\ \psi & ::= & L \mid (\psi\wedge\psi) \mid (\psi\vee\psi) \mid (\forall x\psi) \mid (\exists x\psi) \end{array}$$

Question 3 Is it possible to find for every predicate logical formula an equivalent formula in negation normal form? If your answer is 'no', explain. If your answer is 'yes', give an algorithm for transforming into negation normal form.

Answer: Yes, this is possible. Here is the definition (assume the formulas are $\rightarrow$- and $\leftrightarrow$-free):

$$\begin{array}{rcl}
\text{NNF}(P(t_1,\ldots,t_n)) & := & P(t_1,\ldots,t_n) \\
\text{NNF}(t_1 = t_2) & := & t_1 = t_2 \\
\text{NNF}(\neg\neg\varphi) & := & \text{NNF}(\varphi) \\
\text{NNF}(\neg P(t_1,\ldots,t_n)) & := & \neg P(t_1,\ldots,t_n) \\
\text{NNF}(\neg t_1 = t_2) & := & \neg t_1 = t_2 \\
\text{NNF}(\varphi_1 \wedge \varphi_2) & := & \text{NNF}(\varphi_1) \wedge \text{NNF}(\varphi_2) \\
\text{NNF}(\varphi_1 \vee \varphi_2) & := & \text{NNF}(\varphi_1) \vee \text{NNF}(\varphi_2) \\
\text{NNF}(\forall x\varphi) & := & \forall x\text{NNF}(\varphi) \\
\text{NNF}(\exists x\varphi) & := & \exists x\text{NNF}(\varphi) \\
\text{NNF}(\neg(\varphi_1 \wedge \varphi_2)) & := & \text{NNF}(\neg\varphi_1) \vee \text{NNF}(\neg\varphi_2) \\
\text{NNF}(\neg(\varphi_1 \vee \varphi_2)) & := & \text{NNF}(\neg\varphi_1) \wedge \text{NNF}(\neg\varphi_2) \\
\text{NNF}(\neg\forall x\varphi) & := & \exists x\text{NNF}(\neg\varphi) \\
\text{NNF}(\neg\exists x\varphi) & := & \forall x\text{NNF}(\neg\varphi)
\end{array}$$

It is left to you to turn this into a Haskell implementation.

Question 4 Consider the following formulas (from now on we adopt some obvious conventions for leaving out parentheses, and we also leave out commas between terms):

1. $\forall x\forall yRxy$
2. $\forall x\exists yRxy$
3. $\exists x\forall yRxy$
4. $\exists x\forall y(x = y \vee Rxy)$
5. $\forall x\exists y(Rxy \wedge \neg\exists z(Rxz \wedge Rzy))$.
6. $\forall x\neg Rxx$.

For each of these formulas, find a model where the formula is true and a model where the formula is false.

Answer:

1. $\forall x\forall yRxy$ is true in a model where R is interpreted as the universal relation, and false in any other model.

2. $\forall x \exists y Rxy$ is true in $\mathbb{N}$, with R interpreted as $<$, and false in $\mathbb{N}$ with R interpreted as $>$.

3. $\exists x \forall y Rxy$ is true in $\mathbb{N}$, with R interpreted as $\leq$, and false in $\mathbb{N}$ with R interpreted as $\geq$.

4. $\exists x \forall y(x = y \vee Rxy)$ is true in $\mathbb{N}$, with R interpreted as $<$, and false in $\mathbb{N}$ with R interpreted as $>$.

5. $\forall x \exists y(Rxy \wedge \neg \exists z(Rxz \wedge Rzy))$ is true in $\mathbb{N}$ with R interpreted as $<$, and false in $\mathbb{Q}$ with R interpreted as $<$.

6. $\forall x \neg Rxx$ is true in $\mathbb{N}$ with R interpreted as $<$, and false in $\mathbb{N}$ with R interpreted as $\leq$.

Question 5 Consider the following three predicate logical sentences:

$$\begin{aligned}
\varphi_1 &\overset{\text{def}}{=} \forall x Pxx. \\
\varphi_2 &\overset{\text{def}}{=} \forall x \forall y(Pxy \to Pyx). \\
\varphi_3 &\overset{\text{def}}{=} \forall x \forall y \forall z((Pxy \wedge Pyz) \to Pxz).
\end{aligned}$$

These sentences together express that a certain binary relation P is an equivalence relation (reflexive, symmetric and transitive). Show that none of these sentences is semantically entailed by the other ones by choosing for each pair of sentences a model that makes these two sentences true but makes the third sentence false. In other words: find three examples of binary relations, each satisfying just two of the properties. This shows, essentially, that the definition of being an equivalence cannot be simplified (why?).

Note: later, once you get more Haskell experience, you will be able to program the example in Haskell, and let Haskell do the work for you.

Answer: Example of a relation that is reflexive and symmetric but not transitive: take $\{1, 2, 3\}$ with $R = \{(1, 1), (2, 2), (3, 3), (1, 2), (2, 1), (2, 3), (3, 2)\}$. Draw a picture. This is not transitive, for $(1, 3)$ and $(3, 1)$ are missing.

Example of a relation that is reflexive and transitive but not symmetric: take $\{1, 2\}$ with $R = \{(1, 1), (2, 2), (1, 2)\}$. Draw a picture. This is not symmetric, for $(2, 1)$ is missing.

Example of a relation that is symmetric and transitive but not reflexive: $\{1\}$ with $R = \emptyset$. This is not reflexive for $(1, 1)$ is missing. But it satisfies the requirements for symmetry and transitivity.

Question 6 Consider again the definition of terms in predicate logic:

$$t \ ::= \ v \mid f(t, \ldots, t)$$

$t^v_{t'}$ is the new term that is the result of replacing v everywhere in t by t'.

Can you give a recursive definition of this substitution operation?

Answer: Distinguish the two cases where t equals a variable w and a complex term $f(t_1 \cdots t_n)$.

$$w^v_{t'} = \begin{cases} t' & \text{if } w = v \\ w & \text{otherwise} \end{cases}$$

$$f(t_1 \cdots t_n)^v_{t'} = f({t_1}^v_{t'} \cdots {t_n}^v_{t'}).$$

Question 7 Here is a Haskell version of the definition of terms:

```haskell
data Term = V String | F String [Term] deriving (Eq,Ord,Show)
```

Implement the substitution operation for this datatype. The type of this operation is:

```haskell
subst :: String -> Term -> Term -> Term
...
```

`subst v t' t` is the Haskell version of $t^v_{t'}$. Replace the `...` by your definition.

Answer:

```haskell
subst :: String -> Term -> Term -> Term
subst name t (V name') = if name == name' then t else (V name')
subst name t (F name' ts) = F name' (map (subst name t) ts)
```

Question 8 Suppose you want to extend this notion to substitution in formulas. You have to be careful now, for φ^v_t is the result of replacing all *free occurrences* of v in φ by t.

Can you give a recursive definition of this substitution operation? Hint: you should use the operation $t^v_{t'}$.

Answer: can be found in Chapter 4 of "Logic in Action".