Trees for Predicate Logic

It is a simple matter to combine the rules you learned for the quantifiers with the (single sided) tree rules to create a method for testing for validity in predicate logic. The tree rules for the quantifiers are exactly the same as those you learned for proofs, namely the following:

∀Out	∃Out	QE	
<u>∀xAx</u>	<u>∃xAx</u>	<u>-∀xAx</u>	<u>-∃xAx</u>
An	An	∃x-Ax	∀x-Ax
	n is new		

Here is a tree that shows that the translation of the first argument presented in this class is valid, namely the UH students are Nazis Argument: ∀x(Ux→Nx), ∀x(Nx→Bx) ⊢ ∀x(Ux→Bx). We start by negating the conclusion.

∀x(Ux→Nx)
∀x(Nx→Bx)
-∀x(Ux→Bx)

Then we apply the predicate logic rules in the familiar order, first we do QE.

∀x(Ux→Nx)
∀x(Nx→Bx)
(1) -∀x(Ux→Bx)
∃x-(Ux→Bx)

Then ∃Out:

∀x(Ux→Nx)
∀x(Nx→Bx)
(1) -∀x(Ux→Bx)
(2) ∃x-(Ux→Bx)
-(Ua→Ba) a is new

Note that we do ∃Out first so that we can avoid violating the restriction that the name must be new. Now we do ∀Out on the top two steps.

```
(3)   ∀x(Ux→Nx)
(4)   ∀x(Nx→Bx)
(1)   -∀x(Ux→Bx)
(2)   ∃x-(Ux→Bx)
      -(Ua→Ba)
      Ua→Na
      Na→Ba
```

All the quantifier steps are now done, so all that remains is to apply the propositional logic tree rules to the last three lines.

```
(3)   ∀x(Ux→Nx)
(4)   ∀x(Nx→Bx)
(1)   -∀x(Ux→Bx)
(2)   ∃x-(Ux→Bx)
(5)   -(Ua→Ba)
(6)   Ua→Na
(7)   Na→Ba
      Ua
      -Ba
      /  \
  -Ua     Na
    *     /  \
       -Na   Ba
         *    *
```

Since all branches are closed, the argument is valid, which is what we claimed in the first week of class.

We also made the point then that the Canaries are Pets Argument was invalid. Here is a tree that verifies that. The argument is ∀x(Cx→Fx), ∃x(Px&Fx) ⊢ ∃x(Px&Cx). We negate the conclusion and apply QE to the last step:

```
      ∀x(Cx→Fx)
      ∃x(Px&Fx)
(1)   -∃x(Px&Cx)
      ∀x-(Px&Cx)
```

Now we take off the quantifiers making sure to do ∃Out first so as not to violate the restriction on that rule:

```
(3)    ∀x(Cx→Fx)
(2)    ∃x(Px&Fx)
(1)    -∃x(Px&Cx)
(4)    ∀x-(Px&Cx)
       Pa&Fa
       Ca→Fa
       -(Pa&Ca)
```

Now we apply the propositional rules to the remaining three steps:

```
(3)    ∀x(Cx→Fx)
(2)    ∃x(Px&Fx)
(1)    -∃x(Px&Cx)
(4)    ∀x-(Px&Cx)
(5)    Pa&Fa
(7)    Ca→Fa
(6)    -(Pa&Ca)
       Pa
       Fa
       /  \
   -Pa     -Ca
    *       /  \
         -Ca   Fa
          o     o
```

This time we got open branches on the right, so the argument is invalid as we claimed. (One open branch would be enough to show invalidity.)