Home > &c > FOL Evaluator

FOL Evaluator

The FOL Evaluator is a semantic calculator which will evaluate a well-formed formula of first-order logic on a user-specified model. In its output, the program provides a description of the entire evaluation process used to determine the formula's truth value. For a list of the symbols the program recognizes and some examples of well-formed formulas involving those symbols, see below. Click the "Sample Model" button for an example of the syntax to use when you specify your own model.

Enter a Model:

Domain	1,2,3
P	1,2
Q	(1,1),(3,1),(3,2),(2,3)

+

Load Model Clear Model Sample Model

```
Loaded Model

Domain: [1,2,3]
Q: [(1,1),(3,1),(3,2),(2,3)]
P: [1,2]
```

Enter a Formula:

```
AzPzvAyEx(Qxy&Py)
```

Evaluate!

```
(AzPzvAyEx(Qxy&Py)) is false on this model


Evaluation History:
-------------------

(AzPzvAyEx(Qxy&Py)) is false on {}
      AzPz is false on {}
            Pz is true on {"z":1}
            Pz is true on {"z":2}
            Pz is false on {"z":3}
      AyEx(Qxy&Py) is false on {}
            Ex(Qxy&Py) is true on {"y":1}
                  (Qxy&Py) is true on {"y":1,"x":1}
                        Qxy is true on {"y":1,"x":1}
                        Py is true on {"y":1,"x":1}
                  (Qxy&Py) is false on {"y":1,"x":2}
                        Qxy is false on {"y":1,"x":2}
                        Py is true on {"y":1,"x":2}
                  (Qxy&Py) is true on {"y":1,"x":3}
                        Qxy is true on {"y":1,"x":3}
                        Py is true on {"y":1,"x":3}
            Ex(Qxy&Py) is true on {"y":2}
                  (Qxy&Py) is false on {"y":2,"x":1}
                        Qxy is false on {"y":2,"x":1}
                        Py is true on {"y":2,"x":1}
                  (Qxy&Py) is false on {"y":2,"x":2}
                        Qxy is false on {"y":2,"x":2}
                        Py is true on {"y":2,"x":2}
                  (Qxy&Py) is true on {"y":2,"x":3}
                        Qxy is true on {"y":2,"x":3}
                        Py is true on {"y":2,"x":3}
            Ex(Qxy&Py) is false on {"y":3}
                  (Qxy&Py) is false on {"y":3,"x":1}
                        Qxy is false on {"y":3,"x":1}
                        Py is false on {"y":3,"x":1}
```

```
                (Qxy&Py) is false on {"y":3,"x":2}
                      Qxy is true on {"y":3,"x":2}
                      Py is false on {"y":3,"x":2}
                (Qxy&Py) is false on {"y":3,"x":3}
                      Qxy is false on {"y":3,"x":3}
                      Py is false on {"y":3,"x":3}
```

Here is a list of the symbols the program recognizes (note that since the letter 'v' is used for disjunction, it cannot be used as a variable or individual constant):

Logical Symbols

Negation	~
Conjunction	&
Disjunction	v
Conditional	>
Biconditional	<>
Existential Quantifier	E
Universal Quantifier	A
Identity Relation	=
Absurdity/Falsum	#

Non-Logical Symbols

Individual Constants	a, b, c ... z (except v)
Variables	a, b, c ... z (except v)
Propositional Constants	A, B, C, ... Z
1- and 2-Place Predicates	A, B, C, ... Z

Here are some examples of well-formed formulas the program will accept:

Fa > ~Raa
ExRxa
ExAy(Rxy <> Ryx)
Ax(Fx > EyRxy) & EzRaz
AxEyx=y
AxAy(x=y > (Fx > Fy))
Ex(P > Fx) > (P > ExFx)

If you load the "sample model" above, these formulas will all successfully evaluate in that model. In general, in order for a formula to be evaluable in a model, the model needs to assign an extension to every non-logical constant the formula contains.

Notice that only binary connectives introduce parentheses, whereas quantifiers don't, so e.g. 'ExRxa' and 'Ex(Rxa & Fx)' are well-formed but 'Ex(Rxa)' is not. In the above examples, I've left off the outermost parentheses on formulas that have a binary connective as their main connective (which the program allows). In general, the formal grammar that the program implements for complex wffs is:

φ := ~φ | (φ & φ) | (φ v φ) | (φ > φ) | (φ <> φ) | Evφ | Avφ |

One final point: if you load a model that assigns an empty extension to a predicate, the program has no way of anticipating whether you intend to use that predicate as a 1-place predicate or a 2-place predicate. Internally it therefore adds two versions of the predicate to the model, a 1-place version and a 2-place version, each with an empty extension. (Extensions for sentences and individual constants can't be empty, and neither can domains.)