Computational Formal Semantics Notes: Part 2

Adrian Brasoveanu*

November 14, 2013

Contents

1 First-order predicate logic: Syntax

We load the *PredLSyn* module, i.e., the module that contains our implementation of the syntax of First Order Logic (FOL):

```
ghci 1> :l PredLsyn
```

Note that this module imports *Data.List*.

1.1 The definition of terms

(1) **type** *Name* = *String*
type *Index* = [*Int*]
data *Term* = *Var Name Index* | *Struct Name* [*Term*] **deriving** (*Eq*, *Ord*)

1.2 Atomic formulas

Here are a couple of examples of atomic formulas:

```
ghci 2> let form1 = Atom "left" [x]
```

*Code based on *Computational Semantics with Functional Programming* by Jan van Eijck & Christina Unger, http://www.computational-semantics.eu.

```
ghci 3> form1
    left [x]
```

```
ghci 4> :t form1
    form1 :: Formula
```

```
ghci 5> let form2 = Atom "hugged" [y,z]
```

```
ghci 6> form2
    hugged [y,z]
```

```
ghci 7> :t form2
    form2 :: Formula
```

```
ghci 8> let form3 = Atom "it's raining" []
```

```
ghci 9> form3
    it's raining
```

```
ghci 10> :t form3
    form3 :: Formula
```

1.3 More on variables

The variables x, y, z (a subtype of terms) are predefined:

```
ghci 11> x
    x
```

```
ghci 12> :t x
    x :: Term
```

We can define additional variables. For example, we can build variables out of a variable name and an index:

```
ghci 13> :t Var
Var :: Name → Index → Term
```

```
ghci 14> let u = Var "u" []
```

```
ghci 15> u
u
```

```
ghci 16> :t u
u :: Term
```

```
ghci 17> let u1 = Var "u" [1]
```

```
ghci 18> u1
u1
```

```
ghci 19> :t u1
u1 :: Term
```

```
ghci 20> let u2 = Var "u" [1,2,3,4]
```

```
ghci 21> u2
u1_2_3_4
```

```
ghci 22> :t u2
u2 :: Term
```

And we can define new formulas containing these new variables:

```
ghci 23> let form4 = Atom "hugged" [u1,u2]
```

ghci 24> *form4*
hugged [*u1*, *u1_2_3_4*]

ghci 25> :*t form4*
form4 :: *Formula*

1.4 Identity

And this is the identity relation (we're working with FOL w/ identity):

ghci 26> :*t Eq*
Eq :: *Term* → *Term* → *Formula*

ghci 27> **let** *form5* = *Eq u1 u2*

ghci 28> *form5*
u1 = *u1_2_3_4*

ghci 29> :*t form5*
form5 :: *Formula*

1.5 Propositional operators

Let's turn now to propositional operators.

ghci 30> :*t Neg*
Neg :: *Formula* → *Formula*

ghci 31> **let** *form6* = *Neg form5*

ghci 32> *form6*
∼*u1* = *u1_2_3_4*

ghci 33> :*t form6*
form6 :: *Formula*

ghci 34> $:t\ Impl$
$Impl :: Formula \to Formula \to Formula$

ghci 35> **let** $form7 = Impl\ form6\ form4$

ghci 36> $form7$
$(\sim u1 = u1\_2\_3\_4 \Rightarrow hugged\ [u1, u1\_2\_3\_4])$

ghci 37> $:t\ form7$
$form7 :: Formula$

ghci 38> $:t\ Equi$
$Equi :: Formula \to Formula \to Formula$

ghci 39> **let** $form8 = Equi\ form6\ form4$

ghci 40> $form8$
$(\sim u1 = u1\_2\_3\_4 <==> hugged\ [u1, u1\_2\_3\_4])$

ghci 41> $:t\ form8$
$form8 :: Formula$

ghci 42> $:t\ Conj$
$Conj :: [Formula] \to Formula$

ghci 43> **let** $form9 = Conj\ [form6, form4]$

ghci 44> $form9$
$\&[\sim u1 = u1\_2\_3\_4, hugged\ [u1, u1\_2\_3\_4]]$

```
ghci 45> :t form9
form9 :: Formula
```

```
ghci 46> :t Disj
Disj :: [Formula] → Formula
```

```
ghci 47> let form10 = Disj [form6,form4]
```

```
ghci 48> form10
V [~u1 = u1_2_3_4,hugged [u1,u1_2_3_4]]
```

```
ghci 49> :t form10
form10 :: Formula
```

1.6 Complex terms

We can build complex / structured terms out of a function name and a list of terms

```
ghci 50> :t Struct
Struct :: Name → [Term] → Term
```

```
ghci 51> let t1 = Struct "mother_of" [u1]
```

```
ghci 52> t1
mother_of [u1]
```

```
ghci 53> :t t1
t1 :: Term
```

```
ghci 54> let form13 = Atom "hugged" [u1,t1]
```

```
ghci 55> form13
    hugged [u1, mother_of [u1]]
```

```
ghci 56> :t form13
    form13 :: Formula
```

1.7 Quantifiers

Finally, we turn to quantifiers:

```
ghci 57> :t Forall
    Forall :: Term → Formula → Formula
```

```
ghci 58> :t Exists
    Exists :: Term → Formula → Formula
```

We can define new formulas like this:

```
ghci 59> let form11 = Forall u1 $ Exists u2 $ Conj [form6, form4]
```

```
ghci 60> form11
    A u1 E u1_2_3_4 & [~u1 = u1_2_3_4, hugged [u1, u1_2_3_4]]
```

```
ghci 61> :t form11
    form11 :: Formula
```

```
ghci 62> let form12 = Forall u1 $ Exists u2 $ Impl form6 form4
```

```
ghci 63> form12
    A u1 E u1_2_3_4 (~u1 = u1_2_3_4 ⇒ hugged [u1, u1_2_3_4])
```

```
ghci 64> :t form12
    form12 :: Formula
```

```
ghci 65> let form14 = Forall u1 $ Atom "hugged" [u1, t1]
```

```
ghci 66> form14
    A u1 hugged [u1, mother_of [u1]]
```

```
ghci 67> :t form14
    form14 :: Formula
```

We also have two convenience functions, one determining whether a term is a variable or not, another determining the set of variables in a term:

```
ghci 68> isVar u1
    True
```

```
ghci 69> isVar u2
    True
```

```
ghci 70> isVar t1
    False
```

```
ghci 71> varsInTerm u1
    [u1]
```

```
ghci 72> t1
    mother_of [u1]
```

```
ghci 73> varsInTerm t1
    [u1]
```

```
ghci 74> let t2 = Struct "gift_from_to" [u2, t1]
```

```
ghci 75> t2
    gift_from_to [u1_2_3_4, mother_of [u1]]
```

```
ghci 76> varsInTerm t2
    [u1_2_3_4, u1]
```

2 First-order predicate logic: Semantics

```
ghci 77> :l PredLsem
```

Let's take an atomic formula saying that x laughed:

```
ghci 78> let form1 = Atom "laugh" [x]
```

```
ghci 79> form1
    laugh [x]
```

```
ghci 80> :t form1
    form1 :: Formula
```

We will interpret this formula relative to a basic interpretation function *int0* (predefined in the *PredLsem* module) and a variable assignment *ass0* (also predefined in *PredLsem*).

Both *int0* and *ass0* are defined relative to the model in the *Model* module, in particular relative to the domain of entities defined there. The set of laughing entities is $laugh = \{Alice, Goldilocks, Ellie\}$ and the denotation of the intransitive verb `"laugh"` is this set, coded as follows:

$$int0\ \texttt{"laugh"} = \lambda[x] \rightarrow laugh\ x$$

The variable assignment *ass0* maps all the variables to entity *Alice*:

$$ass0 = \lambda v \rightarrow Alice$$

The evaluation function takes a domain of entities (this is needed for quantified sentences), a basic interpretation function and an assignment and returns a truth value for the formula *form1*:

```
ghci 81> eval entities int0 ass0 form1
    True
```

We can evaluate the negation of *form1* etc.

```
ghci 82> Neg form1
    ~laugh [x]
```

```
ghci 83> eval entities int0 ass0 (Neg form1)
    False
```

```
ghci 84> Conj [form1, Neg form1]
    &[laugh [x], ~laugh [x]]
```

```
ghci 85> eval entities int0 ass0 (Conj [form1, Neg form1])
    False
```

```
ghci 86> Disj [form1, Neg form1]
    V [laugh [x], ~laugh [x]]
```

```
ghci 87> eval entities int0 ass0 (Disj [form1, Neg form1])
    True
```

We can evaluate the universal and existential generalizations of *form1* too:

```
ghci 88> Forall x form1
    A x laugh [x]
```

```
ghci 89> eval entities int0 ass0 (Forall x form1)
    False
```

```
ghci 90> Exists x form1
    E x laugh [x]
```

```
ghci 91> eval entities int0 ass0 (Exists x form1)
    True
```

And here are a couple of examples of more complicated formulas:

```
ghci 92> let form2 = Forall x (Atom "love" [x, x])   -- we test whether "love" is reflexive (it's not)
```

```
ghci 93> form2
    A x love [x, x]
```

```
ghci 94> :t form2
    form2 :: Formula
```

```
ghci 95> eval entities int0 ass0 form2
    False
```

```
ghci 96> let form3 = Forall x ((Atom "boy" [x]) `Impl` (Exists y (Conj [Atom "girl" [y], Atom "love" [y, x]])))
```

```
ghci 97> form3
    A x (boy [x] ⇒ E y & [girl [y], love [y, x]])
```

```
ghci 98> :t form3
    form3 :: Formula
```

```
ghci 99> eval entities int0 ass0 form3
    False
```

We can also add 0-arity (name) constants:

```
ghci 100> let form4 = Atom "laugh" [Struct "Alice" []]
```

```
ghci 101> form4
    laugh [Alice]
```

```
ghci 102> :t form4
    form4 :: Formula
```

```
ghci 103> evl entities int0 fint0 ass0 form4
    True
```

```
ghci 104> let form5 = Atom "laugh" [Struct "Dorothy" []]
```

```
ghci 105> form5
    laugh [Dorothy]
```

ghci 106> *:t form5*
form5 :: *Formula*

ghci 107> *evl entities int0 fint0 ass0 form5*
False

ghci 108> *Neg form5*
$\sim$*laugh* [*Dorothy*]

ghci 109> *evl entities int0 fint0 ass0* (*Neg form5*)
True