Computational Formal Semantics Notes: Part 4

Adrian Brasoveanu*

November 14, 2013

Contents

1 The syntax of our English fragment

The syntax of our Eng. fragment is the same as before. We just had one determiner in there – *Most* – for which we didn't have a semantics before:

```
ghci 1> :l EF1syn
```

```
ghci 2> :i DET
data DET = The | Every | Some | No | Most |
A -- Defined at EF1syn.hs:7:6 instance Show DET – Defined at EF1syn.hs:7:57
```

2 The (direct) semantics of our English fragment

We define a direct, compositional interpretation for our Eng. fragment.

```
ghci 3> :l EF2sem
```

Eng. expressions are interpreted as higher-order functions of various types. The two basic types are *Entity* (*e* in Montague semantics) and *Bool* (*t* in Montague semantics).

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

```
ghci 4> :i Entity
data Entity = Alice | Bob | Cyrus | Dorothy | Ellie | Fred | Goldilocks |
Hillary | Irene | Jim | Kim | Linda | LittleMook | Noah | Ollie | Penny |
Quine | Remmy | SnowWhite | Tom | Uli | Victor | Willie | Xena | Atreyu |
Zorba -- Defined at Model.hs:5:6 instance Bounded Entity – Defined at Model.hs:6:24 instance Enum Entity –
```

```
ghci 5> :i Bool
data Bool = False | True -- Defined in 'GHC.Types' instance Bounded Bool – Defined in 'GHC.Enum' instance
```

This is the interpretation of CNs:

```
ghci 6> :t intCN Boy
intCN Boy :: Entity → Bool
```

```
ghci 7> :t intCN
intCN :: CN → Entity → Bool
```

```
ghci 8> :i intCN Boy
intCN :: CN → Entity → Bool -- Defined at EF2sem.hs:38:1 data CN = ... Boy ... – Defined at EF1syn.hs:8:22
```

This is the interpretation of proper names:

```
ghci 9> :t intNP ALICE
intNP ALICE :: (Entity → Bool) → Bool
```

Determiners and NP headed by determiners have translations of the expected Montague-style types:

```
ghci 10> :t intDET Every
intDET Every :: (Entity → Bool) → (Entity → Bool) → Bool
```

```
ghci 11> :t intDET Most
intDET Most :: (Entity → Bool) → (Entity → Bool) → Bool
```

```
ghci 12> :t intNP $ NP1 Every Boy
intNP $ NP1 Every Boy :: (Entity → Bool) → Bool
```

```
ghci 13> :t intNP $ NP1 Most Sword
intNP $ NP1 Most Sword :: (Entity → Bool) → Bool
```

The translations for VPs containing intrasitive, transitive and ditransive verbs have the expected Montagovian form:

```
ghci 14> :t intVP Laughed
intVP Laughed :: Entity → Bool
```

```
ghci 15> :t intVP $ VP1 Helped (NP1 Every Boy)
intVP $ VP1 Helped (NP1 Every Boy) :: Entity → Bool
```

```
ghci 16> :t intVP $ VP2 Gave (NP1 Every Boy) (NP1 A Sword)
intVP $ VP2 Gave (NP1 Every Boy) (NP1 A Sword) :: Entity → Bool
```

We can now translate full sentences:

```
ghci 17> :t intSent $ Sent (NP1 No Girl) Laughed
intSent $ Sent (NP1 No Girl) Laughed :: Bool
```

```
ghci 18> :t intSent $ Sent (NP1 No Girl) (VP1 Helped (NP1 Every Boy))
intSent $ Sent (NP1 No Girl) (VP1 Helped (NP1 Every Boy)) :: Bool
```

```
ghci 19> :t intSent $ Sent (NP1 No Girl) (VP2 Gave (NP1 Every Boy) (NP1 A Sword))
intSent $ Sent (NP1 No Girl) (VP2 Gave (NP1 Every Boy) (NP1 A Sword)) :: Bool
```

Finally, restrictive relative clauses with a subject or object gap are translated in the expected Montagovian way:

```
ghci 20> :t intRCN $ RCN1 Boy That Laughed
intRCN $ RCN1 Boy That Laughed :: Entity → Bool
```

```
ghci 21> :t intNP $ NP2 Every (RCN1 Boy That Laughed)
intNP $ NP2 Every (RCN1 Boy That Laughed) :: (Entity → Bool) → Bool
```

```
ghci 22> :t intSent $ Sent (NP2 Every (RCN1 Boy That Laughed)) Smiled
intSent $ Sent (NP2 Every (RCN1 Boy That Laughed)) Smiled :: Bool
```

```
ghci 23> :t intRCN $ RCN2 Boy That (NP1 A Girl) Loved
intRCN $ RCN2 Boy That (NP1 A Girl) Loved :: Entity → Bool
```

```
ghci 24> :t intNP $ NP2 Every (RCN2 Boy That (NP1 A Girl) Loved)
intNP $ NP2 Every (RCN2 Boy That (NP1 A Girl) Loved) :: (Entity → Bool) → Bool
```

```
ghci 25> :t intSent $ Sent (NP2 Every (RCN2 Boy That (NP1 A Girl) Loved)) Smiled
intSent $ Sent (NP2 Every (RCN2 Boy That (NP1 A Girl) Loved)) Smiled :: Bool
```

The interpretation of full sentences yields truth values, for example, since:

- the set of boys in the model is {*LittleMook*, *Atreyu*}
- the set of girls in the model is {*SnowWhite*, *Alice*, *Dorothy*, *Goldilocks*}
- the set of love-pairs in the model is {(*Atreyu*, *Ellie*), (*Bob*, *SnowWhite*), (*Remmy*, *SnowWhite*), (*SnowWhite*, *LittleMook*)}
- the set of smilers in the model is {*Alice*, *Bob*, *Cyrus*, *Dorothy*, *Ellie*, *Fred*, *Goldilocks*, *LittleMook*}

Therefore, *Every boy that a girl loved smiled* is true b/c *LittleMook* is the only boy loved by a girl and *LittleMook* is in the set of smilers:

```
ghci 26> intSent $ Sent (NP2 Every (RCN2 Boy That (NP1 A Girl) Loved)) Smiled
True
```

And *No boy that a girl loved smiled* is false:

```
ghci 27> intSent $ Sent (NP2 No (RCN2 Boy That (NP1 A Girl) Loved)) Smiled
False
```

An example using name constants:

```
ghci 28> intSent $ Sent SNOWWHITE (VP1 Loved LITTLEMOOK)
True
```

More examples (from the *Comp. Sem.* textbook):

```
ghci 29> intSent $ Sent (NP1 The Princess) Laughed
True
```

```
ghci 30> intSent $ Sent (NP1 The Giant) Shuddered
False
```

```
ghci 31> intSent $ Sent (NP1 A Dwarf) Cheered
False
```

```
ghci 32> intSent $ Sent (NP1 No Wizard) Laughed
True
```

```
ghci 33> intSent $ Sent (NP1 A Dwarf) (VP1 Defeated (NP1 A Giant))
    True
```