IPython Lambda Notebook: a system for digital fragments

Kyle Rawlins
rawlins@cogsci.jhu.edu
Johns Hopkins University

LSA Annual Meeting
Jan 3, 2014

https://github.com/rawlins/lambda-notebook

Overview

A key part of doing linguistic semantics:
Developing and conveying a formal analysis of some natural language data.

❖❖❖

Classical approach: the 'method of fragments' (Montague 1970, Partee 1979, Partee & Hendriks 1997)

- Precise and complete statement of a grammar for some sublanguage of a natural language.
- Fragments have fallen by the wayside. (Not necessarily for bad reasons...)

❖❖❖

Proposal: the method of fragments can and should be resurrected, in digital form.

- Digital fragments preserve the positives of the traditional approach.
- Digital fragments mitigate the negatives. Key points: modularity and interactivity.

The method of fragments

The positives.

What does 'formal' mean? In principle:

1. Mathematically precise (lambda calculus, type theory, logic, model theory(?), ...).
2. Complete (covers 'all' the data).
3. Predictive (like any scientific theory).
4. Consistent, or at least compatible (with itself, analyses of other phenomena, some unifying conception of the grammar).

The method of fragments provided a structure for meeting these criteria.

- Paper with a fragment is guaranteed (or at least likely) to provide a working system.
- Explicit outer bound for empirical coverage.
- Typically, explicit integration with a particular theory of the grammar.
- Explicit answer to relevant questions not necessarily dealt with in text.

Summary: fragments are a method for replicability, similar to a computional researcher providing an explicit statement of their model.

Useful internal check for researcher.

> "...But I feel strongly that it is important to try to [work with fully explicit fragments] periodically, because otherwise it is extremely easy to think that you have a solution to a problem when in fact you don't."
> (Partee 1979, p. 41)

❖❖❖

The downsides.

Part 1 of the above quote:

> "It can be very frustrating to try to specify frameworks and fragments explicitly; this project has not been entirely rewarding. I would not recommend that one always work with the constraint of full explicitness."
> (*Ibid.*)

- Fragments can be tedious and time-consuming to write (not to mention hard).
- Fragments are in practice not easy for a reader to actually use.
  - Dense/unapproachable: With exactness comes a huge chunk of hard-to-digest formalism. (E.g. in Partee (1979), about 10% of the paper.)
  - Monolithic/non-modular: everything relevant to phenomena under investigation is (or should be) specified, rather than left to our work.
  - Exact opposite of modern method – researchers typically hold most aspects of the grammar constant (implicitly) while varying a few things.

Digital fragments

Summary: the scientific motivations for fragments remain strong, but they have fallen out of use because in practice the payoff is not [perceived to be] large enough for the writer or reader of the fragment.

Proposal: shift the problem from one of giving a logical fragment, to specifying a computational model.

- Digital fragments: both of the key problems in left column (fragments are dense, monolothic) can be solved by interactivity.
- Interface along the lines of Mathematica notebooks – mix documentation and code.
- Payoff for reader/consumer greatly increased!
- Payoff for writer? Hard to tell yet – currently programming experience is required. Hope to improve as time goes on.

Existing/prior infrastructure for linguistic semantics:

- van Eijck & Unger (2010): treatment of many core topics in compositional semantics in Haskell. Thorough and rich, but not easy to use.
- `nltk.sem` module: implementation of typed lambda calculus as part of NLTK. (Bird et al. 2009)
- Champollion et al. (2007): teaching oriented implementation of lambda calculus-style composition in Java.

Infrastructure for this style of model development: IPython Notebook (Pérez & Granger 2007).

- Interactive Mathematica-style notebooks for scientific computing.
- A notebook mixes code fragments (Python), markdown formatting, graphical content.
- Key feature: MathJax support for rendering mathematical expressions.

Architecture of the IPython Lambda Notebook:

- Layer 1: Typed logical metalanguage, with limited type inference. Comparable to `nltk.sem` with a richer type system. Less sophisticated than van Eijck & Unger (2010).
- Layer 2: Framework for writing composition systems on trees (or tree-like) objects in natural language. Integrates with `nltk.tree`.
- Layer 3: Frontend in IPython Notebook. Uses cell magics (see demo) for a lambda calculus mini-language based on python syntax. A fragment is a notebook.

Layer 1 largely replicates existing systems. Layers 2 and 3 (and especially the interfaces between layers 1,2 and 3) are new.

❖❖❖

Writing a digital fragment (not necessarily in this order):

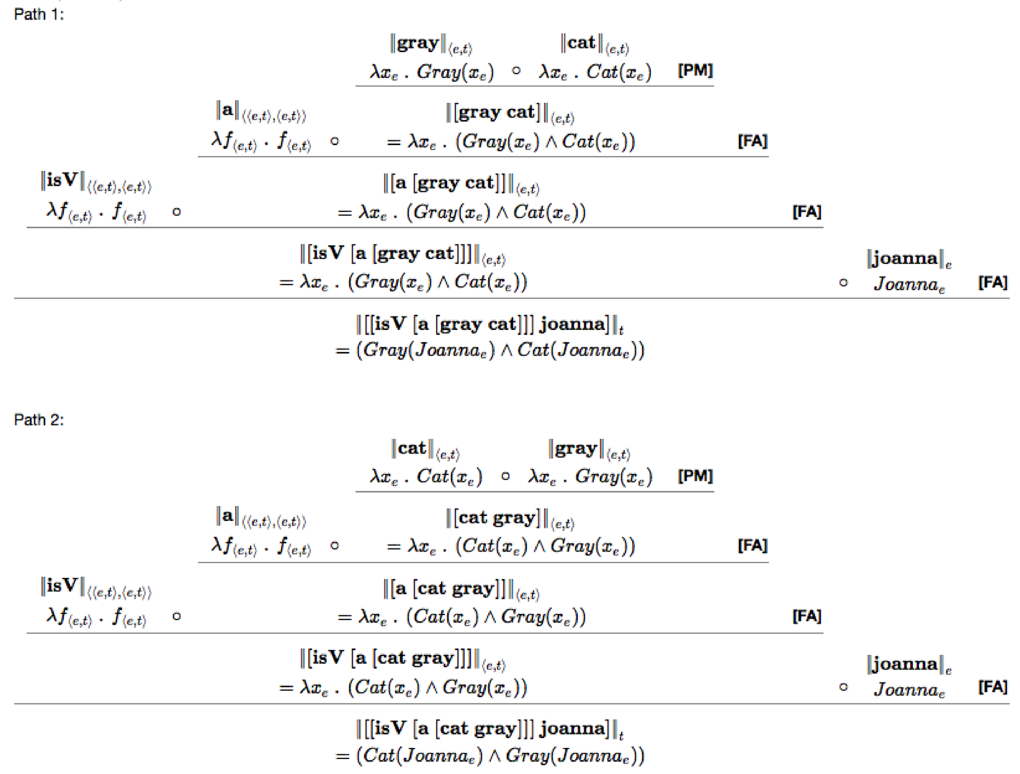
1. State background assumptions about syntax, composition. In practice, the aim is for core assumptions to be provided in a modular fashion, not rewritten each time. E.g. most of the core of Heim & Kratzer (1998) is currently in place.
2. Add components unique to particular analysis. Composition rules (python code), lexical entries.
3. State key examples in notebook form. (Lexical entries, trees.)

A tiny example

What does this look like in action? An example after Heim & Kratzer:

In [9]:
```
%%lamb reset
||gray|| = lambda x_e : Gray(x)
||cat|| = lambda x_e : Cat(x)
||joanna|| = Joanna_e
||isV|| = lambda f_<e,t> : f # 'is' is a python reserved word
||a|| = lambda f_<e,t> : f
```
```
warning: coerced guessed type t for 'Gray' into <e,t>, to match argument 'x'
warning: coerced guessed type t for 'Cat' into <e,t>, to match argument 'x'
```
Out[9]:
$\|\mathbf{gray}\|_{\langle e,t\rangle} = \lambda x_e\,.\,Gray(x_e)$
$\|\mathbf{cat}\|_{\langle e,t\rangle} = \lambda x_e\,.\,Cat(x_e)$
$\|\mathbf{joanna}\|_e = Joanna_e$
$\|\mathbf{isV}\|_{\langle\langle e,t\rangle,\langle e,t\rangle\rangle} = \lambda f_{\langle e,t\rangle}\,.\,f_{\langle e,t\rangle}$
$\|\mathbf{a}\|_{\langle\langle e,t\rangle,\langle e,t\rangle\rangle} = \lambda f_{\langle e,t\rangle}\,.\,f_{\langle e,t\rangle}$

In [10]:
```
r = joanna * (isV * (a * (gray * cat)))
r.latex_step_tree()
```
Out[10]: 2 composition paths:
Path 1:



Summary

Digital fragments mitigate challenges of the 'method of fragments'.

- Range of technology largely exists for writing digital fragments (Bird et al. 2009, van Eijck & Unger 2010).
- IPython Lambda Notebook brings together many of these ideas into a package aimed at linguistic semanticists.

Current state of the project:

- Code is alpha quality, much development and testing still needed!
- Programming experience currently necessary for use.
- Short term goal: develop more fragments, reduce required programming knowledge at least for readers of fragments.
- Long term goal: put the field in a place where it is possible for anyone to provide an interactive digital fragment along with a research paper!

❖❖❖

Selected references

Bird, Steven, Edward Loper & Ewan Klein. 2009. *Natural Language Processing with Python*. O'Reilly Media Inc.
Champollion, L., J. Tauberer & M. Romero. 2007. The Penn Lambda Calculator: Pedagogical software for natural language semantics. In T. Holloway King & E. M. Bender (eds.), *Proceedings of the Grammar Engineering across Frameworks (GEAF) 2007 Workshop*, .
van Eijck, Jan & Christina Unger. 2010. *Computational semantics with functional programming*. Cambridge University Press.
Heim, Irene & Angelika Kratzer. 1998. *Semantics in Generative Grammar*. Malden: Blackwell.
Montague, Richard. 1970. English as a formal language. In *Linguaggi nella società e nella tecnica*, 189–224. Edizioni di Comunità.
Partee, Barbara H. 1979. Constraining Montague grammar: a framework and a fragment. In S. Davis & M. Mithun (eds.), *Linguistics, Philosophy, and Montague Grammar*, 51–101. University of Texas Press.
Partee, Barbara H. & Herman L. W. Hendriks. 1997. Montague grammar. In Johan van Benthem & Alice G. B. ter Meulen (eds.), *Handbook of Logic and Language*, 5–91. Elsevier and MIT Press.
Pérez, Fernando & Brian E. Granger. 2007. IPython: a System for Interactive Scientific Computing. *Comput. Sci. Eng.* 9(3). 21–29. `http://ipython.org`.

❖❖❖

Acknowledgements: For discussion of this work and relatied issues, I'm grateful to Emily Bender, Adrian Brasoveanu, Barbara Partee, Tim Vieira, members of the JHU Semantics lab, and audiences at JHU.