Overall goal

Montagovian semantics for Computer Scientists, or Derivation calculators for Semanticists

Derivations and normalizations are boring, let the computer do it

Gains

- for NL researchers: a helpful tool
  not just for counting words, but
  complement to pen-and-paper theory building
- for PL researchers: an interesting application to build tools for

Beginning of a beautiful friendship
(or, collaboration, or at least mutual comprehension)

`http://okmij.org/ftp/gengo/NASSLLI10/`

Grand goal

NL researchers will

- gain rational reconstruction of Montagovian tricks
- import developed CS ideas:
  side effects, continuations, regions, staging, dependent types

PL researchers will

- export developed CS ideas:
  side effects, continuations, regions, staging, dependent types
- build theories of programming language competence

All would benefit from connections with logic and probability theory

Plan

- Making (intuitive) sense of our metalanguage (Haskell)
- CFG: writing and (*re-*)interpreting derivations
  overall: how to embed (object) languages and represent (grammar/type) derivations
- Propositional and predicate logic as an object language
- Language transformations and simplifications: teaching the computer equational reasoning
- Data types and capturing the structure of a domain
- Approaches to quantification

- Expressives
- Theories of intensionality

- Embedding Combinatorial Categorial Grammars
- Dynamic logic and donkey anaphora
- Scope and inverse linking in continuation semantics

Main ideas

- *Calculemus:* yield*s*, denotation*s*
- *Many* fragments, languages, interpretations
- Growing fragments and languages
- *Interactivity*
- Montagovian tradition
- Representing published analyses and theories (de Groote, Potts, Pollard's APWS, Zimmermann, boot camp, ...)

`http://homepages.cwi.nl/~jve/HR/`
`http://lambda.jimpryor.net/`

The look of Haskell

- GHCi prompt
- Arithmetic, Logic, Strings
- Abstractions and applications
- Types, type annotations, type errors
- Definitions, parametrized definitions

`http://tryhaskell.org`

http://www.haskell.org/platform/

Exercises 0

Fill in the blanks

Prelude> True && False

$\boxed{}$

Prelude> :t (||)

$\boxed{}$

Prelude> :t $\boxed{}$

$\boxed{}$ :: [Char] → [Char]

Exercises 1

```
twice = \f -> \x -> f (f x)
```

- How else we can write this definition?
- Does this term reminds us something from lambda-calculus?
- How to quickly verify that?

Exercises 2

1. Write Church numeral for 0
2. Write increment incr. How to test it?
3. Write addition, multiplication, exponentiation, decrement

Further look at Haskell

Pairs (products)

introduction, elimination, pattern-matching in definitions

Sums (co-products)

introduction, elimination, defining by clauses

Why pairs are called products and why Either is called a sum or a co-product?

Polymorphic types

Exercises 3

Write functions of these types:

$$((),\ a) \to a$$
$$a \to ((),\ a)$$
$$\text{Either}\ a\ b \to (a \to c) \to (b \to c) \to c$$
$$((a,b) \to c) \to (a \to b \to c)$$
$$(a \to b \to c) \to ((a,b) \to c)$$
$$a \to ((a \to f) \to f)$$
$$(((a \to f) \to f) \to f) \to (a \to f)$$
$$(\text{Either}\ a\ b \to f) \to (a \to f,\ b \to f)$$
$$((a,b) \to f) \to (((\text{Either}\ (a \to f)\ (b \to f)) \to f) \to f)$$

- what do these functions do?
- What do these types remind you of?
- What do the terms your wrote signify?

Exercises 4

1. How polymorphic types relate to universals?
2. Why existentials in Haskell look the way they do?

Exercises 5

1. Add ditransitive verbs
2. Add some sort of agreement
   The yield of a derivation (the phonetics generated from the derivation) should show the agreement between a verb and its arguments (in number, case, gender, etc.) Extend the fragment appropriately. You can use any language for phonetics.

Exercises 6

1. Define the data type of Pizzas
   The datatype describes which baked thing can be considered a pizza and which cannot.
2. Define a data type for burrito

Exercises 7

Think about representing the derivation of, and computing yield and truth values of two sample sentences from the Semantics boot camp:

- Düsseldorf is hot
- Düsseldorf is in Germany

Elizabeth Coppock. Semantics bootcamp handouts (Part III, §1.1 and §1.2) NASSLLI 2012, June 16, 2012
`http://nasslli2012.com/bootcamp`

Grammatical Framework

"GF, Grammatical Framework, is a programming language for multilingual grammar applications."
`http://www.grammaticalframework.org/`

- EDSL vs. stand-alone language
  - implementation effort
  - flexibility
  - polish and convenience
  - error messages
  - parsers, syntactic sugar
- *Implementing* an ACG/CCG interpreter in Haskell vs. using Haskell as a *metalanguage* to express ACG/CCG: understanding a foreign language by translation vs. thinking in a foreign language

Lecture 3

1. Truth values vs. truth conditions,
   why we want to see logical formulas
2. Logic as a language
   - 2.1 Embedding the propositional logic:
     syntax, semantics (models), simplification (consequence)
     - Data types in Haskell: a more general view
   - 2.2 Embedding higher-order languages
   - 2.3 Predicate logic and logical quantification
3. Putting it all together: seeing logical formulas for a
   sentence and its constituents

Natural and Formal languages

"I reject the contention that an important theoretical difference exists between formal and natural languages. ... In the present paper I shall accordingly present a precise treatment, culminating in a theory of truth, of a formal language that I believe may reasonably be regarded as a fragment of ordinary English. ... The treatment given here will be found to resemble the usual syntax and model theory (or semantics) [due to Tarski] of the predicate calculus, but leans rather heavily on the intuitive aspects of certain recent developments in intensional logic [due to Montague himself]. (Montague 1970b, p.188 in Montague 1974)"

[Quoted from Semantics bootcamp handouts (Part I) by Elizabeth Coppock. NASSLLI 2012, June 16, 2012]

Understanding type classes

	Logic	Language	Information
`class`	Σ	(embedded) Language	Interface
`instance`	Model $< D, I >$	Interpretation	Implementation

Compositionality

Compositionality Principle

The meaning of an expression is uniquely determined by the meaning of its parts and the manner in which they are combined.

$\Rightarrow$

The substitution principle

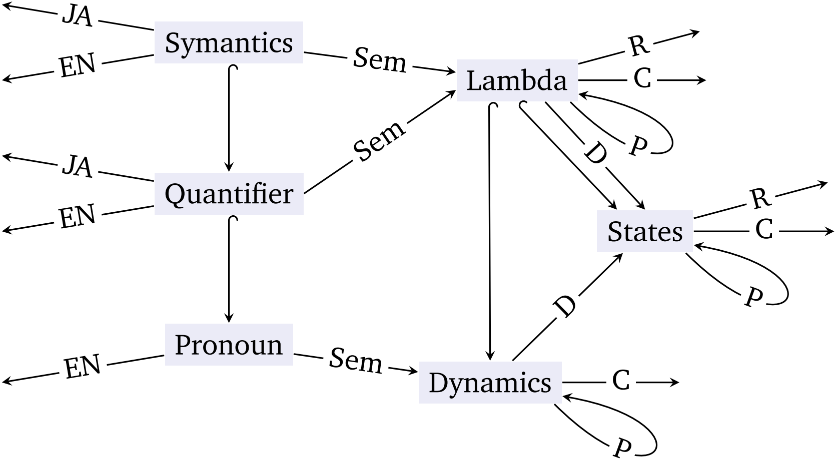
If two expressions have the same meanings, they may replace each other in all contexts (in *all* positions within any bigger expression) without affecting the truth conditions.

Intensionality

Hesperus is Venus.
Venus is a planet. $\Rightarrow$
Hesperus is a planet.

Hesperus is Venus.
John wants to find Venus. $\not\Rightarrow$
John wants to find Hesperus.

Map



JA
EN
Symantics
Sem
Lambda
R
C
P
D
JA
EN
Quantifier
Sem
States
R
C
P
D
EN
Pronoun
Sem
Dynamics
C
P