Computational Semantics with Functional Programming

Jan van Eijck, CWI

Software Techology Colloquium, Utrecht, 12 March 2009

Abstract

Almost forty years ago Richard Montague proposed to analyse natural language with the same tools as formal languages. In particular, he gave formal semantic analyses of several interesting fragments of English in terms of typed logic. This led to the development of Montague grammar as a particular style of formal analysis of natural language.

Pure functional programming languages are in fact implementations of the typed lambda calculus, and implementing a Montague style fragment of English in Haskell is a breeze. In the talk we will first explain the program of Montague style natural language analysis, and next show how this can be carried out with functional programming. Examples will be taken from a textbook on computational semantics, Computational Semantics with Functional Programming (CUP, to appear). Draft version of the book: `http://www.cwi.nl/~jve/cs/`

Overview

- A natural language engine for talking about classes.
- Demo
- Natural language analysis and functional programming
- The program of Montague Grammar
- From semantics to pragmatics
- Challenge: linking up with cognitive realities.

Aristotelian Quantifiers: The Square of Opposition

All A are B	No A are B
Some A are B	Not all A are B

Aristotle interprets his quantifiers with existential import: All A are B and No A are B are taken to imply that there are A.

The Simplest Natural Language Engine You Can Get

Questions and Statements (PN for plural nouns):

$$\begin{array}{rcl} Q & ::= & \text{Are all PN PN?} \\ & | & \text{Are no PN PN?} \\ & | & \text{Are any PN PN?} \\ & | & \text{Are any PN not PN?} \\ & | & \text{What about PN?} \end{array}$$

$$\begin{array}{rcl} S & ::= & \text{All PN are PN.} \\ & | & \text{No PN are PN.} \\ & | & \text{Some PN are PN.} \\ & | & \text{Some PN are not PN.} \end{array}$$

The Simplest Knowledge Base You Can Get

The two relations we are going to model in the knowledge base are that of inclusion $\subseteq$ and that of non-inclusion $\not\subseteq$.

'all A are B' $\rightsquigarrow A \subseteq B$

'no A are B' $\rightsquigarrow A \subseteq \overline{B}$

'some A are not B' $\rightsquigarrow A \not\subseteq B$

'some A are B' $\rightsquigarrow A \not\subseteq \overline{B}$ (equivalently: $A \cap B \neq \emptyset$).

A **knowledge base** is a list of triples

$$(\text{Class}_1, \text{Class}_2, \text{Boolean})$$

where $(A, B, \top)$ expresses that $A \subseteq B$,

and $(A, B, \bot)$ expresses that $A \not\subseteq B$.

Rules of the Inference Engine

Let $\widetilde{A}$ be given by: if A is of the form $\overline{C}$ then $\widetilde{A} = C$, otherwise $\widetilde{A} = \overline{A}$. Let $A \Longrightarrow B$ express $A \subseteq B$. Let $A \not\Longrightarrow B$ express $A \not\subseteq B$.

Computing the subset relation from the knowledge base:

$$\frac{A \Longrightarrow B}{\widetilde{B} \Longrightarrow \widetilde{A}} \qquad \frac{A \Longrightarrow B \qquad B \Longrightarrow C}{A \Longrightarrow C}$$

Computing the non-subset relation from the knowledge base:

$$\frac{A \not\Longrightarrow B}{\widetilde{B} \not\Longrightarrow \widetilde{A}} \qquad \frac{A \Longleftarrow B \quad B \not\Longrightarrow C \qquad C \Longleftarrow D}{A \not\Longrightarrow D}$$

Reflexivity and existential import:

$$\frac{}{A \Longrightarrow A} \qquad \frac{}{A \not\Longrightarrow \widetilde{A}}\;\text{A not of the form $\overline{C}$}$$

Soundness and Completeness

Aristotelian class model: A universe U with a list of non-empty subsets.

The calculus is **sound** for Aristetelian class models. Why?

The calculus is **complete** for Aristotelian class models. Why?

Because a consistent knowledge base can be turned into an Aristotelian class model.

Completeness

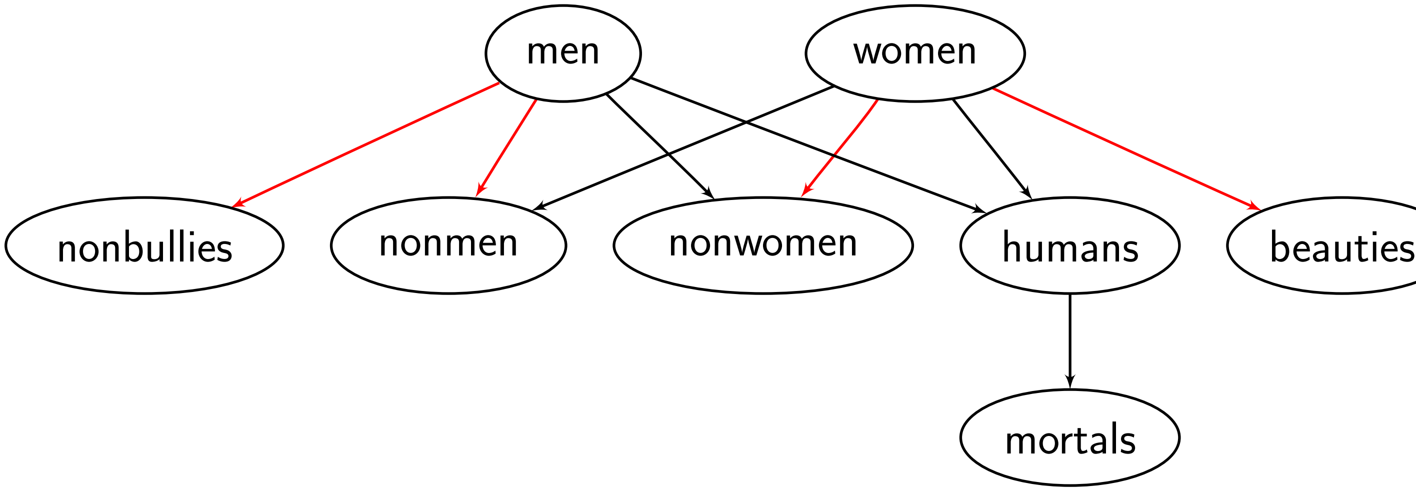
Short digest of the reasoning. Let ϕ be one of $A \Longrightarrow B$, $A \not\Longrightarrow B$.

- Suppose KB $\not\vdash \phi$ (ϕ is not derivable from the KB).
- Then KB' $=$ KB $+\neg\phi$ is consistent.
- Construct a complete graph from KB'.
- Initially put $A = \{a\}$ for every non-complement class A and $\overline{B} = \emptyset$ for each complement class $\overline{B}$.
- Carry out the $\Longrightarrow$ inclusion instructions until the process stabilizes (which will always be the case, as the graph is finite).
- The result will be an Aristotelian class model for KB that falsifies ϕ. Hence KB $\not\models \phi$.

Implementation: see Section 5.7 of [2].

Idea: `http://rubyquiz.com/` (Ruby Quiz # 37).

Example:



Demo

. . .

Natural language analysis and functional programming

- Usefulness of typed lambda calculus for NL analysis.
- Linguist Barbara Partee: “Lambda’s have changed my life.”
- Computational linguistics: From Prolog to Haskell?
- Appeal of Prolog: Prolog-style unification [10], ‘Parsing as Deduction’ [8]
- But a new trend is emerging [3, 4]
- NLP Resources in Haskell: see `http://www.haskell.org/haskellwiki/Applications_and_libraries/Linguistics`

Richard Montague (1930-1971)

Developed higher-order typed intensional logic with a possible-worlds semantics and a formal pragmatics incorporating indexical pronouns and tenses.

Program in semantics (around 1970): universal grammar.

Towards a philosophically satisfactory and logically precise account of syntax, semantics, and pragmatics, covering both formal and natural languages.

"The Proper Treatment of Quantification was as profound for semantics as Chomsky's Syntactic Structures was for syntax." (Barbara Partee on Montague, in the Encyclopedia of Language and Linguistics.)

- Chomsky: English can be described as a formal system.
- Montague: English can be described as a formal system with a formal semantics, and with a formal pragmatics.

Montague's program can be viewed as an extension of Chomsky's program.

The Program of Montague Grammar

- Montague's thesis: there is no essential difference between the semantics of natural languages and that of formal languages (such as that of predicate logic, or programming languages).
- The method of fragments: UG [7], EFL [6], PTQ [5]
- The misleading form thesis (Russell, Quine)
- Proposed solution to the misleading form thesis
- Key challenges: quantification, anaphoric linking, tense, intensionality.

Misleading Form

Aristotle's theory of quantification has two logical defects:

1. Quantifier combinations are not treated; only one quantifier per sentence is allowed.
2. 'Non-standard quantifiers' such as most, half of, at least five, . . . are not covered.

Frege's theory of quantification removed the first defect.

The Fregean view of quantifiers in natural language: quantified Noun Phrases are systematically misleading expressions.

Their natural language syntax does not correspond to their logic:

"Nobody is on the road" $\rightsquigarrow$ $\neg\exists x(\text{Person}(x) \wedge \text{OnTheRoad}(x))$

Solution to the Misleading Form Thesis

expression	translation	type
every	**every**	$(e \to t) \to ((e \to t) \to t)$
princess	P	$(e \to t)$
every princess	**every** P	$(e \to t) \to t$
laughed	S	$(e \to t)$
every princess laughed	$($**every** $P)$ S	t

where **every** is a name for the constant $\lambda P \lambda Q.\forall x(Px \to Qx)$.

Generalized Quantifiers and Continuations

Continuations: invented by Adriaan van Wijngaarden (1964), reinvented many times after [9].

Montague's idea of lifting a proper name to the type of a quantifier is a form of continuation. Programs in Continuation Passing Style (CPS) have an extra argument for 'what to do next'.

What is the 'what to do next' of the interpretation of a proper name? Use it as argument of a predicate, of course.

Jan translates to $\lambda Q.Qj$

Every princess translates to $\lambda Q.\forall x(\text{Princess } x \rightarrow Qx)$.

```
intNP :: NP -> (Entity -> Bool) -> Bool
intNP Ann = \ p -> p ann
intNP (NP det cn)  = (intDET det) (intCN cn)
```

Generalized Quantifier Theory: Numerical Trees

Suppose a quantifier Q has A as a first and B as a second argument. Q can then be characterized as a subset of a tree of numbers.

The first number in each number pair is $|A - B|$, the second one $|A \cap B|$.

$$
\begin{array}{ll|ccccccccccc}
|A| & = 0 & & & & & & 0,0 & & & & & \\
|A| & = 1 & & & & & 1,0 & & 0,1 & & & & \\
|A| & = 2 & & & & 2,0 & & 1,1 & & 0,2 & & & \\
|A| & = 3 & & & 3,0 & & 2,1 & & 1,2 & & 0,3 & & \\
|A| & = 4 & & 4,0 & & 3,1 & & 2,2 & & 1,3 & & 0,4 & \\
|A| & = 5 & 5,0 & & 4,1 & & 3,2 & & 2,3 & & 1,4 & & 0,5 \\
 & \vdots & & & & & & \vdots & & & & &
\end{array}
$$

all A are B

$$\begin{array}{ccccccccccc}
 & & & & & + & & & & & \\
 & & & & - & & + & & & & \\
 & & & - & & - & & + & & & \\
 & & - & & - & & - & & + & & \\
 & - & & - & & - & & - & & + & \\
- & & - & & - & & - & & - & & + \\
 & & & & & \vdots & & & & &
\end{array}$$

```haskell
tree :: Integer -> [(Integer,Integer)]
tree n = [(n-x,x) | x <- [0..n] ]

treeOfNumbers :: [(Integer,Integer)]
treeOfNumbers = concat [ tree n | n <- [0..] ]

type Quant = (Integer -> Bool) -> [Integer] -> Bool

check :: Quant -> (Integer,Integer) -> Bool
check q (n,m) = q (\ x -> 0 < x && x <= m) [1..n+m]

genTree :: Quant -> [(Integer,Integer)]
genTree q = filter (check q) treeOfNumbers
```

From semantics to pragmatics

- Analysing communication as flow of knowledge.
- Logical tool: Dynamic Epistemic Logic
- Computational tool: Dynamic Epistemic Model Checking
- DEMO: Epistemic Model Checker in Haskell [1]
- Book code contains a mini version of this in the final chapter.

Conclusion

- Computational linguistics and computational semantics provides lots of challenges for grammar engineers, and functional programmers.
- Main challenge: Linking up with cognitive realities.
- Want to learn more? Read our book!
- `www.cwi.nl/~jve/cs/`

References

[1] Jan van Eijck. DEMO — a demo of epistemic modelling. In Johan van Benthem, Dov Gabbay, and Benedikt Löwe, editors, Interactive Logic — Proceedings of the 7th Augustus de Morgan Workshop, number 1 in Texts in Logic and Games, pages 305–363. Amsterdam University Press, 2007.

[2] Jan van Eijck and Christina Unger. Computational Semantics with Functional Programming. To appear with Cambridge University Press, 2009.

[3] R. Frost and J. Launchbury. Constructing natural language interpreters in a lazy functional language. The Computer Journal, 32(2):108–121, 1989.

[4] Richard A. Frost. Realization of natural language interfaces using lazy functional programming. ACM Comput. Surv., 38(4), 2006.

[5] R. Montague. The proper treatment of quantification in ordinary English. In J. Hintikka, editor, Approaches to Natural Language, pages 221–242. Reidel, 1973.

[6] R. Montague. English as a formal language. In R.H. Thomason, editor, Formal Philosophy; Selected Papers of Richard Montague, pages 188–221. Yale University Press, New Haven and London, 1974.

[7] R. Montague. Universal grammar. In R.H. Thomason, editor, Formal Philosophy; Selected Papers of Richard Montague, pages 222–246. Yale University Press, New Haven and London, 1974.

[8] F.C.N. Pereira and H.D. Warren. Parsing as deduction. In Proceedings of the 21st Annual Meeting of the ACL, pages 137–111. MIT, Cambridge, Mass., 1983.

[9] J.C. Reynolds. The discoveries of continuations. Lisp and Symbolic Computation, 6(3–4):233–247, 1993.

[10] S.M. Shieber. An Introduction to Unification Based Approaches to Grammar, volume 4 of CSLI Lecture Notes. CSLI, Stanford, 1986. Distributed by University of Chicago Press.