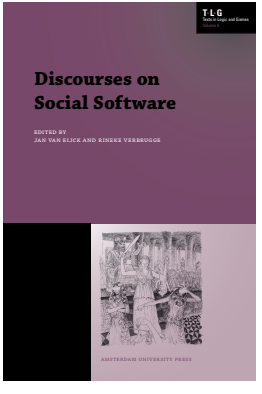
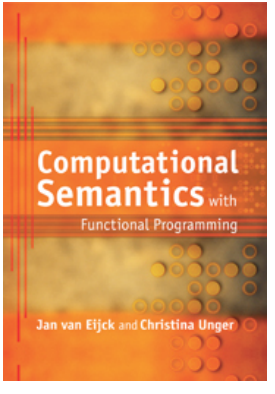
A Program for Computational Semantics

Jan van Eijck

CLCG, 5 November 2010





Abstract

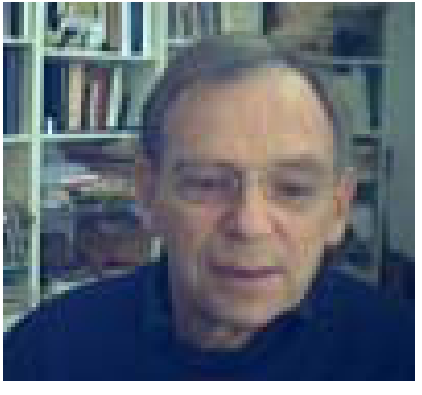
Just as war can be viewed as continuation of diplomacy using other means, computational semantics is continuation of logical analysis of natural language by other means. For a long time, the tool of choice for this used to be Prolog. In our recent textbook we argue (and try to demonstrate by example) that lazy functional programming is a more appropriate tool. In the talk we will lay out a program for computational semantics, by linking computational semantics to the general analysis of procedures for social interaction. The talk will give examples of how Haskell can be used to begin carrying out this program.

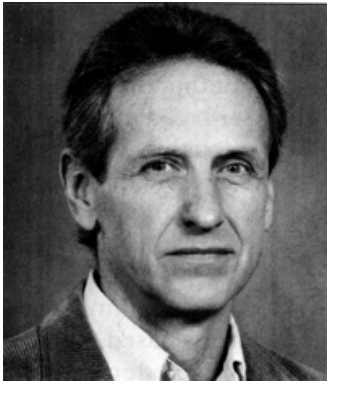
http://www.cambridge.org/vaneijck-unger

http://www.computational-semantics.eu/

http://www.cwi.nl/~jve/nias/discourses/discourses.pdf

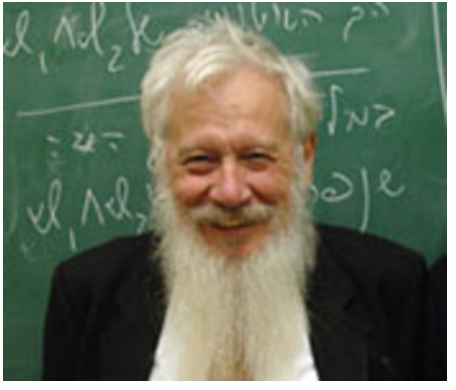
Old Heroes

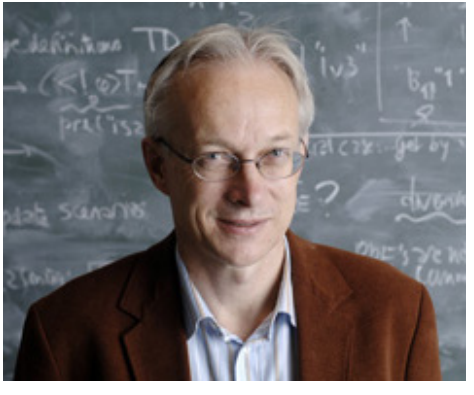




New Heroes

And Some More





What is Computational Semantics? (Wikipedia)

Computational semantics is the study of how to automate the process of constructing and reasoning with meaning representations of natural language expressions. It consequently plays an important role in natural language processing and computational linguistics.

Some traditional topics of interest are: construction of meaning representations, semantic underspecification, anaphora resolution, presupposition projection, and quantifier scope resolution. Methods employed usually draw from formal semantics or statistical semantics. Computational semantics has points of contact with the areas of lexical semantics (word sense disambiguation and semantic role labeling), discourse semantics, knowledge representation and automated reasoning [. . .].

SIGSEM

http://www.sigsem.org/wiki/

If you click on ‘books’ you get the same list as in the wikipedia article.

I did not succeed in editing the page, to add a recent book ☺

What is computational semantics?
The art and science of computing or processing meanings

- ‘meaning’: informational content, as opposed to syntactic ‘form’
- ‘computing’: what computers do
- ‘processing’: what happens in the human brain

A Brief History of Formal Linguistics

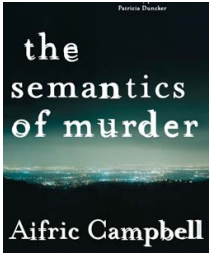


1916 Ferdinand de Saussure, Cours de linguistique générale published posthumously. Natural language may be analyzed as a formal system.

1957 Noam Chomsky, Syntactic Structures, proposes to define natural languages as sets of grammatical sentences, and to study their structure with formal (mathematical) means. Presents a formal grammar for a fragment of English.

1970 Richard Montague, English as a Formal Language, proposes to extend the Chomskyan program to semantics and pragmatics. Presents a formal grammar for a fragment of English, including semantics (rules for computing meanings). Links the study of natural language to the study of formal languages (languages from logic and computer science).

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 [10], EFL [9], PTQ [8]
- 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

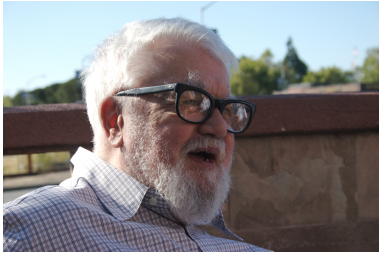
where **every** is a name for the constant $\lambda P \lambda Q . \forall x (Px \to Qx)$.

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 [2]
- All this in [4]

A Brief History of Functional Programming






1932 Alonzo Church presents the lambda calculus

1937 Alan Turing proves that lambda calculus and Turing machines have the same computational power.

1958 John McCarthy starts to implement LISP.

1978-9 Robin Milner cs develop ML.

1987 Agreement on a common standard for lazy purely functional programming: Haskell.

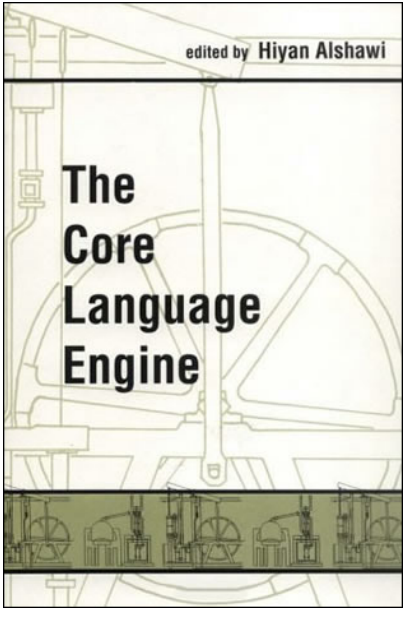
http://www.haskell.org

Natural language analysis and functional programming

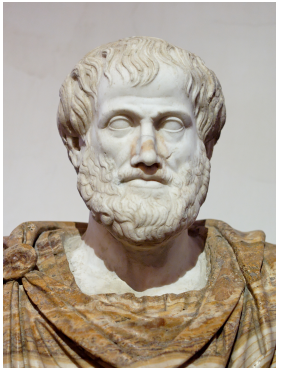
- Usefulness of typed lambda calculus for NL analysis.
- Linguist Barbara Partee: “Lambda’s have changed my life.”
- Computational linguistics: From Prolog to Haskell?
- Appeal of Prolog: Prolog-style unification [12], ‘Parsing as Deduction’ [11], useful didactic tool [1].
- But a new trend is emerging [5, 6]
- NLP Resources in Haskell: see

  `http://www.haskell.org/haskellwiki/Applications_and_libraries/Linguistics`

Using Prolog for NL Semantics: An Ancient Art (1992)



The Simplest Natural Language Engine You Can Get



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.

What can we ask or state with the Aristetelian quantifiers?

Questions and Statements (PN for plural nouns):

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

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

Example Interaction

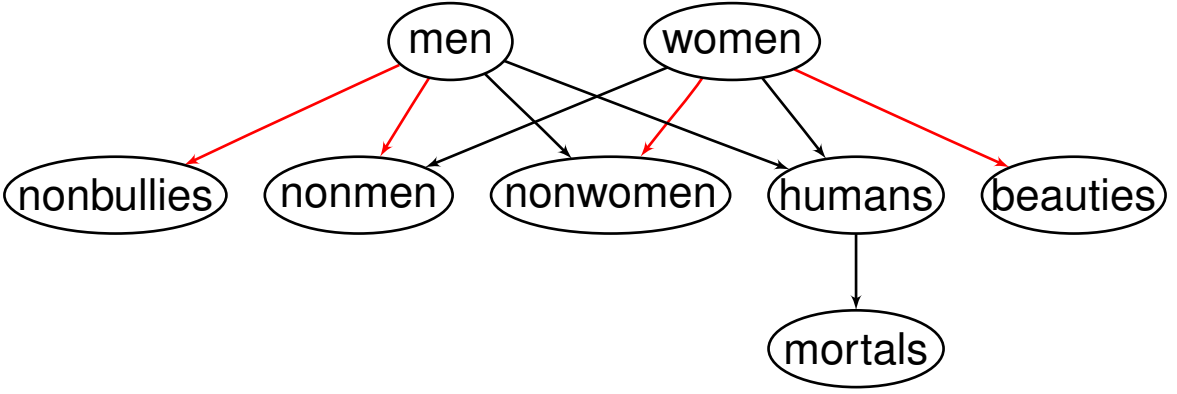
```
jve@vuur:~/courses/lot2009$ ./Main
Welcome to the Knowledge Base.
Update or query the KB:
How about women?
All women are humans.
No women are men.

Update or query the KB:
All mammals are animals.
I knew that already.

Update or query the KB:
No mammals are birds.
```

```
OK.

Update or query the KB:
How about women?
All women are humans.
No women are men.

Update or query the KB:
All humans are mammals.
OK.

Update or query the KB:
How about women?
All women are animals.
All women are humans.
All women are mammals.
```

No women are birds.
No women are men.
No women are owls.

Update or query the KB:

Example Knowledge Base



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 $\widetilde{A} = \overline{A}$.

Computing the subset relation from the knowledge base:

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

Computing the non-subset relation from the knowledge base:

$$\frac{A \not\subseteq B}{\widetilde{B} \not\subseteq \widetilde{A}} \qquad \frac{A \supseteq B \quad B \not\subseteq C}{A \not\subseteq C} \qquad \frac{B \not\subseteq C \quad C \supseteq D}{B \not\subseteq D}$$

Reflexivity and existential import:

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

Consistency of a Knowledge Base

A Knowledge Base **K** is inconsistent if for some $A \subseteq B$:

$$\frac{\mathbf{K}}{A \subseteq B} \qquad \frac{\mathbf{K}}{A \nsubseteq B}$$

Otherwise **K** is consistent.

Soundness and Completeness of Inference System

Exercise 1 *An inference system is called sound if all conclusions that can be derived are valid, i.e. if all axioms are true and all inference rules preserve truth. Show that the inference system for Aristotelian syllogistics is sound.*

Exercise 2 *An inference system is called complete if it can derive all valid conclusions from a set of premisses. In other words: if* $A \subseteq B$ *does not follow from a knowledge base, then there is a class model for the knowledge base where* $A \not\subseteq B$*, and if* $A \not\subseteq B$ *does not follow from a knowledge base, then there is a class model for the knowledge base where* $A \subseteq B$*. Show that the inference system for Aristotelian syllogistics is complete.*

Implementation (in Haskell)

We will need list and character processing, and we want to read natural language sentences from a file, so we import the I/O-module `System.IO`.

```
import List
import Char
import System.IO
```

In our Haskell implementation we can use `[(a,a)]` for relations.

```
type Rel a = [(a,a)]
```

If $R \subseteq A^2$ and $x \in A$, then $xR := \{y \mid (x, y) \in R\}$.

```
rSection :: Eq a => a -> Rel a -> [a]
rSection x r = [ y | (z,y) <- r, x == z ]
```

`Eq a` indicates that a is in the equality class.

The composition of two relations R and S on A.

```
(@@) :: Eq a => Rel a -> Rel a -> Rel a
r @@ s = nub [ (x,z) | (x,y) <- r, (w,z) <- s, y == w ]
```

Computation of transitive closure using a function for least fixpoint.
TC(R) = lfp $(\lambda S.S \cup S \cdot S)\, R$.

```
tc :: Ord a => Rel a -> Rel a
tc = lfp (\ s -> (sort.nub) (s ++ (s@@s)))

lfp :: Eq a => (a -> a) -> a -> a
lfp f x | x == f x  = x
        | otherwise = lfp f (f x)
```

Assume that each class has an opposite class. The opposite of an opposite class is the class itself.

```
data Class = Class String | OppClass String
             deriving (Eq,Ord)

instance Show Class where
  show (Class xs)    = xs
  show (OppClass xs) = "non-" ++ xs

opp :: Class -> Class
opp (Class name)    = OppClass name
opp (OppClass name) = Class name
```

Declaration of the knowledge base:

```
type KB = [(Class, Class, Bool)]
```

A data types for statements and queries:

```
data Statement =
     All  Class   Class | No      Class Class
   | Some Class   Class | SomeNot Class Class
   | AreAll Class Class | AreNo   Class Class
   | AreAny Class Class | AnyNot  Class Class
   | What   Class
  deriving Eq
```

Show function for statements:

```
instance Show Statement where
  show (All as bs) =
    "All " ++ show as ++ " are " ++ show bs ++ "."
  show (No as bs)  =
    "No " ++ show as ++ " are " ++ show bs ++ "."
  show (Some as bs)   =
    "Some " ++ show as ++ " are " ++ show bs ++ "."
  show (SomeNot as bs) =
    "Some " ++ show as ++ " are not " ++ show bs ++ "."
```

and for queries:

```
show (AreAll as bs) =
  "Are all " ++ show as ++ show bs ++ "?"
show (AreNo as bs) =
  "Are no " ++ show as ++ show bs ++ "?"
show (AreAny as bs) =
  "Are any " ++ show as ++ show bs ++ "?"
show (AnyNot as bs) =
  "Are any " ++ show as ++ " not " ++ show bs ++ "?"
show (What as) =
  "What about " ++ show as ++ "?"
```

Classification of statements:

```
isQuery :: Statement -> Bool
isQuery (AreAll _ _)  = True
isQuery (AreNo _ _)   = True
isQuery (AreAny _ _)  = True
isQuery (AnyNot _ _)  = True
isQuery (What _)      = True
isQuery  _            = False
```

Negations of queries:

```
neg :: Statement -> Statement
neg (AreAll as bs) = AnyNot as bs
neg (AreNo as bs)  = AreAny as bs
neg (AreAny as bs) = AreNo as bs
neg (AnyNot as bs) = AreAll as bs
```

Use the transitive closure operation to compute the subset relation from the knowledge base.

```
subsetRel :: KB -> [(Class,Class)]
subsetRel kb =
  tc ([(x,y)         | (x,y,True) <- kb ]
  ++  [(opp y,opp x) | (x,y,True) <- kb ]
  ++  [(x,x)         | (x,_,_)    <- kb ]
  ++  [(opp x,opp x) | (x,_,_)    <- kb ]
  ++  [(y,y)         | (_,y,_)    <- kb ]
  ++  [(opp y,opp y) | (_,y,_)    <- kb ])
```

The supersets of a particular class are given by a right section of the subset relation. I.e. the supersets of a class are all classes of which it is a subset.

```
supersets :: Class -> KB -> [Class]
supersets cl kb = rSection cl (subsetRel kb)
```

Computing the non-subset relation from the knowledge base:

```
nsubsetRel :: KB -> [(Class,Class)]
nsubsetRel kb =
  let
   r = nub ([(x,y) | (x,y,False) <- kb ]
         ++ [(opp y,opp x) | (x,y,False) <- kb ]
         ++ [(Class xs,OppClass xs) |
                    (Class xs,_,_)      <- kb ]
         ++ [(Class ys,OppClass ys) |
                    (_,Class ys,_)      <- kb ]
         ++ [(Class ys,OppClass ys) |
                    (_,OppClass ys,_)   <- kb ])
   s = [(y,x) | (x,y) <- subsetRel kb ]
  in s @@ r @@ s
```

The non-supersets of a class:

```
nsupersets :: Class -> KB -> [Class]
nsupersets cl kb = rSection cl (nsubsetRel kb)
```

Query of a knowledge base by means of yes/no questions is simple:

```
deriv :: KB -> Statement -> Bool
deriv kb (AreAll as bs) = elem bs (supersets as kb)
deriv kb (AreNo  as bs) = elem (opp bs) (supersets as kb)
deriv kb (AreAny as bs) = elem (opp bs) (nsupersets as kb)
deriv kb (AnyNot as bs) = elem bs (nsupersets as kb)
```

Caution: there are three possibilities:

- `deriv kb stmt` holds. So the statement is derivable, hence true.
- `deriv kb (neg stmt)` holds. So the negation of `stmt` is derivable, hence true. So `stmt` is false.
- neither `deriv kb stmt` nor `deriv kb (neg stmt)` holds. So the knowledge base has no information about `stmt`.

Open queries ("How about A?") are slightly more complicated.

We should take care to select the most natural statements to report on a class:

$A \subseteq B$ is expressed with 'all',

$A \subseteq \overline{B}$ is expressed with 'no',

$A \not\subseteq B$ is expressed with 'some not',

$A \not\subseteq \overline{B}$ is expressed with 'some'.

```
f2s :: (Class, Class, Bool) -> Statement
f2s (as, Class bs, True)     = All as (Class bs)
f2s (as, OppClass bs, True)  = No as (Class bs)
f2s (as, OppClass bs, False) = Some as (Class bs)
f2s (as, Class bs, False)    = SomeNot as (Class bs)
```

Giving an explicit account of a class:

```
tellAbout :: KB -> Class -> [Statement]
tellAbout kb as =
  [All as (Class bs) |
      (Class bs) <- supersets as kb,
      as /= (Class bs) ]
  ++
  [No as (Class bs) |
      (OppClass bs) <- supersets as kb,
      as /= (OppClass bs) ]
```

A bit of pragmatics: do not tell ‘Some A are B’ if ‘All A are B’ also holds.

```
++
[Some as (Class bs) |
    (OppClass bs) <- nsupersets as kb,
    as /= (OppClass bs),
    notElem (as,Class bs) (subsetRel kb) ]
```

Do not tell ‘Some A are not B’ if ‘No A are B’ also holds.

```
++
[SomeNot as (Class bs) |
    (Class bs) <- nsupersets as kb,
    as /= (Class bs),
    notElem (as,OppClass bs) (subsetRel kb) ]
```

To build a knowledge base we need a function for updating an existing knowledge base with a statement.

If the update is successful, we want an updated knowledge base. If it is not, we want to get an indication of failure. The Haskell `Maybe` data type gives us just this.

```
data Maybe a = Nothing | Just a
```

The update function checks for possible inconsistencies. E.g., a request to add an $A \subseteq B$ fact to the knowledge base leads to an inconsistency if $A \not\subseteq B$ is already derivable.

```
update  :: Statement -> KB -> Maybe (KB,Bool)

update (All as bs) kb
  | elem bs (nsupersets as kb)  = Nothing
  | elem bs (supersets as kb)   = Just (kb,False)
  | otherwise                   =
                  Just (((as,bs,True): kb),True)
```

A request to add $A \subseteq \overline{B}$ leads to an inconsistency if $A \not\subseteq \overline{B}$ is already derivable.

```
update (No as bs) kb
  | elem bs' (nsupersets as kb) = Nothing
  | elem bs' (supersets as kb)  = Just (kb,False)
  | otherwise                   =
                  Just (((as,bs',True):kb),True)
 where bs' = opp bs
```

Similarly for the requests to update with $A \not\subseteq \overline{B}$ and with $A \not\subseteq B$:

```
update (Some as bs) kb
  | elem bs' (supersets as kb)  = Nothing
  | elem bs' (nsupersets as kb) = Just (kb,False)
  | otherwise                   =
                  Just (((as,bs',False):kb),True)
 where bs' = opp bs

update (SomeNot as bs) kb
  | elem bs (supersets as kb)   = Nothing
  | elem bs (nsupersets as kb)  = Just (kb,False)
  | otherwise                   =
                   Just (((as,bs,False):kb),True)
```

Use this to build a knowledge base from a list of statements. Again, this process can fail, so we use the `Maybe` datatype.

```
makeKB :: [Statement] -> Maybe KB
makeKB = makeKB' []
     where
         makeKB' kb [] = Just kb
         makeKB' kb (s:ss) =
             case update s kb of
               Just (kb',_) -> makeKB' kb' ss
               Nothing      -> Nothing
```

Preprocessing of strings, to prepare them for parsing:

```
preprocess :: String -> [String]
preprocess = words . (map toLower) .
      (takeWhile (\ x -> isAlpha x || isSpace x))
```

This will map a string to a list of words:

```
Main> preprocess "Are any women sailors?"
["are","any","women","sailors"]
```

A simple parser for statements:

```
parse :: String -> Maybe Statement
parse = parse' . preprocess
  where
    parse' ["all",as,"are",bs] =
      Just (All (Class as) (Class bs))
    parse' ["no",as,"are",bs] =
      Just (No (Class as) (Class bs))
    parse' ["some",as,"are",bs] =
      Just (Some (Class as) (Class bs))
    parse' ["some",as,"are","not",bs] =
      Just (SomeNot (Class as) (Class bs))
```

and for queries:

```
parse' ["are","all",as,bs] =
  Just (AreAll (Class as) (Class bs))
parse' ["are","no",as,bs] =
  Just (AreNo (Class as) (Class bs))
parse' ["are","any",as,bs] =
  Just (AreAny (Class as) (Class bs))
parse' ["are","any",as,"not",bs] =
  Just (AnyNot (Class as) (Class bs))
parse' ["what", "about", as] = Just (What (Class as))
parse' ["how", "about", as]  = Just (What (Class as))
parse' _ = Nothing
```

Parsing a text to construct a knowledge base:

```
process :: String -> KB
process txt = maybe [] id
              (mapM parse (lines txt) >>= makeKB)
```

This uses the `maybe` function, for getting out of the `Maybe` type. Instead of returning `Nothing`, this returns an empty knowledge base.

```
maybe :: b -> (a -> b) -> Maybe a -> b
maybe _ f (Just x) = f x
maybe z _ Nothing  = z
```

```
mytxt = "all bears are mammals\n"
     ++ "no owls are mammals\n"
     ++ "some bears are stupids\n"
     ++ "all men are humans\n"
     ++ "no men are women\n"
     ++ "all women are humans\n"
     ++ "all humans are mammals\n"
     ++ "some men are stupids\n"
     ++ "some men are not stupids"
```

```
Main> process mytxt
[(men,stupids,False),(men,non-stupids,False),
 (humans,mammals,True),(women,humans,True),
 (men,non-women,True),(men,humans,True),
 (bears,non-stupids,False),(owls,non-mammals,True),
 (bears,mammals,True)]
```

Now suppose we have a text file of declarative natural language sentences about classes. Here is how to turn that into a knowledge base.

```
getKB :: FilePath -> IO KB
getKB p = do
    txt <- readFile p
    return (process txt)
```

And here is how to write a knowledge base to file:

```
writeKB :: FilePath -> KB -> IO ()
writeKB p kb = writeFile p
                (unlines (map (show.f2s) kb))
```

The inference engine in action:

```
chat :: IO ()
chat = do
 kb <- getKB "kb.txt"
 putStrLn "Update or query the KB:"
 str <- getLine
 if str == "" then return ()
  else do
   case parse str of
     Just (What as) -> let info = tellAbout kb as in
       if info == [] then putStrLn "No info.\n"
       else putStrLn (unlines (map show info))
     Just stmt      ->
      if isQuery stmt then
        if deriv kb stmt then putStrLn "Yes.\n"
```

```
        else if deriv kb (neg stmt)
                then putStrLn "No.\n"
                else putStrLn "I don't know.\n"
      else case update stmt kb of
        Just (kb',True) -> do
                    writeKB "kb.txt" kb'
                    putStrLn "OK.\n"
        Just (_,False)  -> putStrLn
                    "I knew that already.\n"
        Nothing         -> putStrLn
                       "Inconsistent with my info.\n"
  Nothing        -> putStrLn "Wrong input.\n"
chat
```

```
main = do
        putStrLn "Welcome to the Knowledge Base."
        chat
```

Summary

- Cognitive research focusses on this kind of quantifier reasoning. Links with cognition by refinement of this calculus . . . The "natural logic for natural language" enterprise: special workshop during Amsterdam Colloquium 2009 (see http://www.illc.uva.nl/AC2009/)
- "Our ultimate goal is to form an adequate model of parts of our language competence. Adequate means that the model has to be realistic in terms of complexity and learnability. We will not be so ambitious as to claim that our account mirrors real cognitive processes, but what we do claim is that our account imposes constraints on what the real cognitive processes can look like." [4]

The Muddy Children Puzzle

a (1) clean, b (2), c (3) and d (4) muddy.

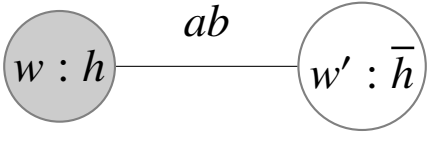
a	b	c	d
○	●	●	●
?	?	?	?
?	?	?	?
?	!	!	!
!	!	!	!

Individual Ignorance

You have to finish a paper, and you are faced with a choice: do it today, or put it off until tomorrow. You decide to play a little game with yourself. You will flip a coin, and you promise yourself: "If it lands heads I will have to do it now, if it lands tails I can postpone it until tomorrow. But wait, I don't have to know right away, do I? I will toss the coin in a cup." And this is what you do. Now the cup is upside down on the table, covering the coin. The coin is showing heads up, but you cannot see that.

Multi Agent Ignorance

Suppose Alice and Bob are present, and Alice tosses a coin under a cup. We will use a for Alice and b for Bob. The result of a hidden coin toss with the coin heads up:

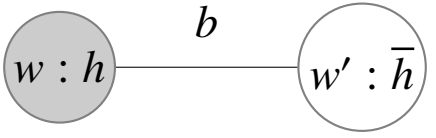


After Alice has taken a look

Assume that Alice is taking a look under the cup, while Bob is watching.

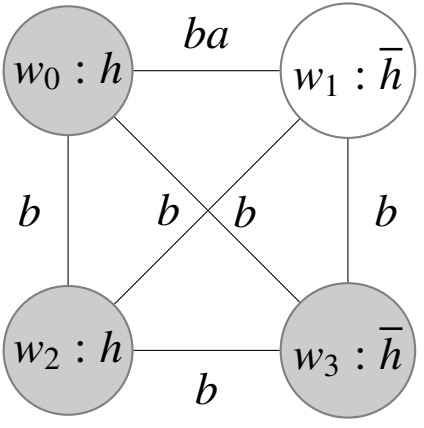
Now Alice knows whether the coin shows heads or tails.

Bob knows that Alice knows the outcome of the toss, but he does not know the outcome himself.



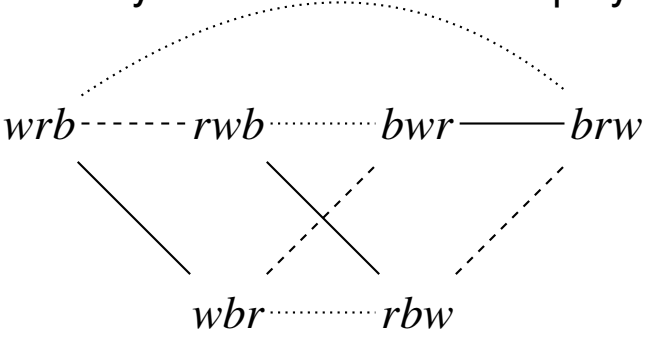
Bob leaves the room and returns

The coin gets tossed under the cup and lands heads up. Alice and Bob are present. Now Bob leaves the room for an instant. After he comes back, the cup is still over the coin. Bob realizes that Alice might have taken a look. She might even have reversed the coin! Alice also realizes that Bob considers this possible.

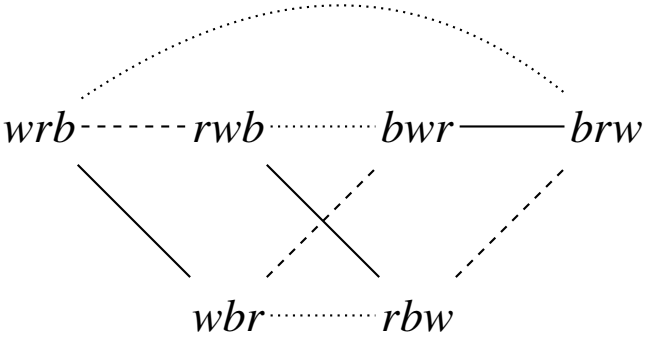


Card Deals

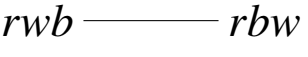
Three players, Alice, Bob and Carol, each draw a card from a stack of three cards. The cards are red, white and blue and their backsides are indistinguishable. Let rwb stand for the deal of cards where Alice holds the red card, Bob the white card, and Carol the blue card. There are six different card deals. Players can only see their own card, but not the cards of other players. They do see that other players also only hold a single card, and that this cannot be their own card, and they do know that all the players know that.



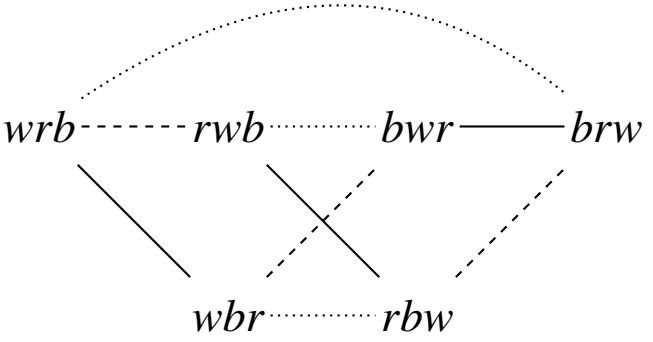
Communication



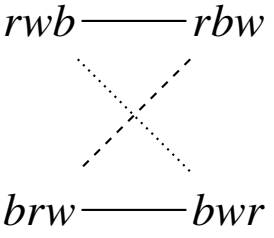
Alice says: 'I hold the red card'. What is the effect of this?



Communication 2

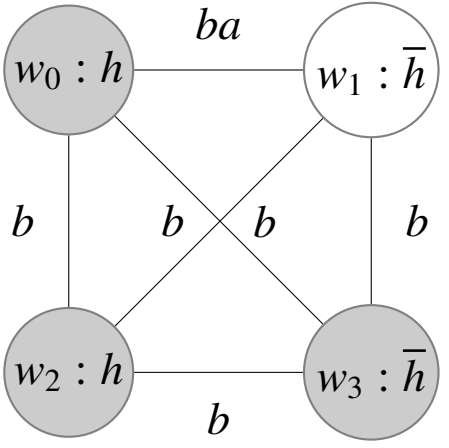


Alice says: 'I do not hold the white card'. What is the effect of this?



Epistemic formulas and their interpretation

$K_a\phi$: agent a knows that ϕ

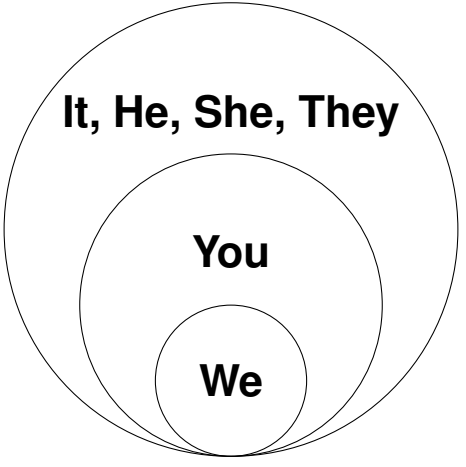


Now evaluate some formulas: $K_a h$, $K_b K_a h$, $K_a h \vee K_a \neg h$, $\neg K_b(K_a h \vee K_a \neg h)$, $K_a \neg K_b(K_a h \vee K_a \neg h)$.

Epistemic Model Checking: Muddy Children

- initMuddy: model where children cannot see their own state
- m1: model after the public announcement that at least one child is muddy.
- m2: model after public announcement that none of them knows their state.
- m3: model after public announcement that none of them knows their state.
- m4: model after public announcement that b, c, d know their state.

Aim of Communicative Discourse



Aim of a communicative discourse is to create common knowledge between us and you.

Common Knowledge

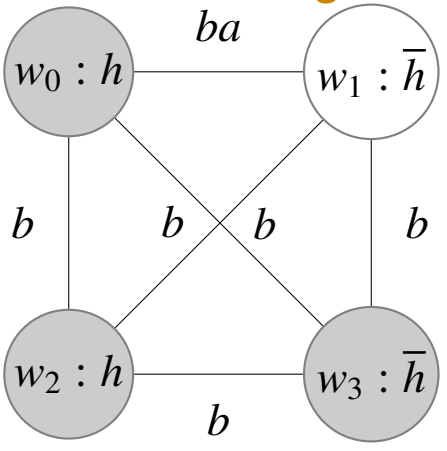
What does it mean to create common knowledge? If we inform you that something is the case, say that there is a party on tonight, then not only do you know that there is a party on, but also we know that you know, and you know that we know that you know. And so on, ad infinitum. This is common knowledge.

If a denotes Alice's accessibility relation and b Bob's, and we use $\cup$ for union of relations, and * for reflexive transitive closure, then $(a \cup b)^*$ expresses the common knowledge of Alice and Bob.

Formula for this: $C_{a,b}\phi$.

Exercise 3 *When money is paid out to you by an ATM, does this create common knowledge between you and the machine? Why (not)?*

Exercises about Common Knowledge

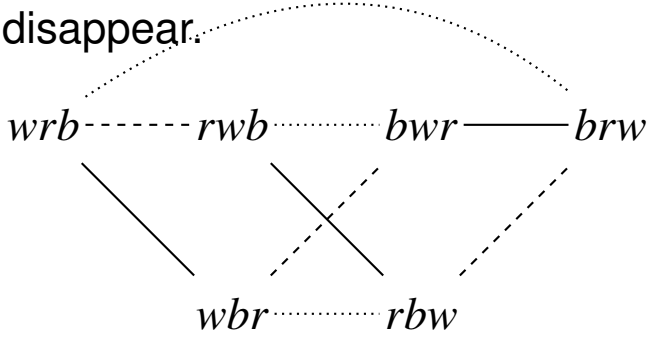


Exercise 4 *Is it common knowledge between* a *and* b *in* w_1 *that Bob does not know that Alice does not know the outcome of the toss? Formula for this:* $C_{a,b}\neg K_b(K_a h \vee K_a \neg h)$.

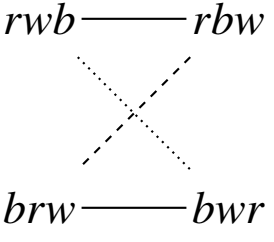
Exercise 5 *Is it common knowledge between* a *and* b *in* w_0 *that Alice knows that Bob does not know the outcome of the toss? Formula for this:* $C_{a,b}K_a\neg(K_b h \vee K_b \neg h)$.

Public Announcement

The effect of a public announcement ‘I have the red card’ or ‘I do not have the white card’ is that the worlds where these announcements are false disappear.



‘I do not hold the white card’

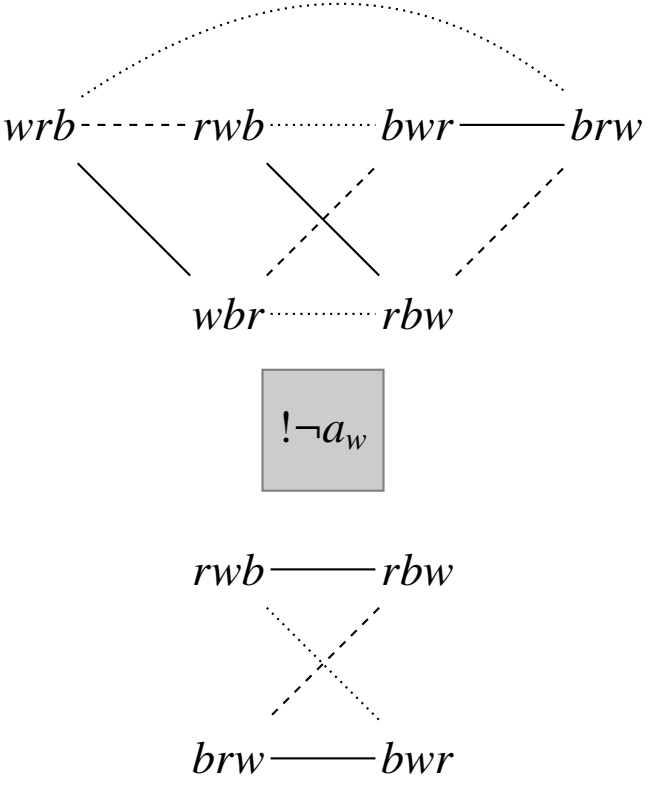


Effect of Public Announcement, Formally

Use $[!\phi]\psi$ for: 'after public announcement of ϕ, ψ holds'.

Formally: $M \models_w [!\phi]\psi$ iff ($M \models_w \phi$ implies $M \mid \phi \models_w \psi$).

$M \mid \phi$ is the result of removing all non-ϕ worlds from M.


wrb
rwb
bwr
brw
wbr
rbw
!¬a_w
rwb
rbw
brw
bwr

Presupposition

A presupposition of an utterance is an implicit assumption about the world or a background belief shared by speaker and hearer in a discourse.

“Shall we do it again?”

Presupposition: we have done it before.

“Jan is a bachelor.”

Presupposition: ‘Jan’ refers to a male person. (True in the Netherlands and Poland, false in the United Kingdom.)

Second presupposition: ‘Jan’ refers to an adult.

So: ‘bachelor’ presupposes ‘male’ and ‘adult’, and conveys ‘unmarried’.

Presupposition and Common Knowledge [3]

Extend the language with public announcements.

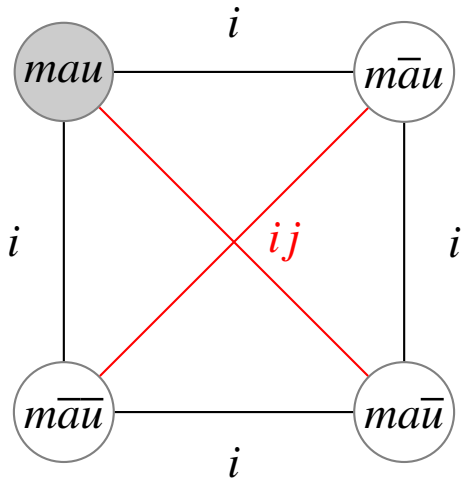
$[!\phi]\psi$ expresses that after public announcement of ϕ, ψ holds.

Now consider the special case of an update of the form "it is common knowledge between i and j that ϕ".

Formally: $!C_{i,j}\phi$.

- In case ϕ is already common knowledge, this update does not change the model.
- In case ϕ is not yet common knowledge, the update leads to a model without actual worlds.

Example



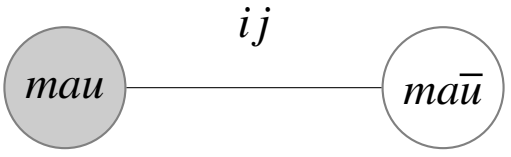
m for ‘male’, a for ‘adult’, u for ‘unmarried’.

j does not know about u

i does not know about a, u.

$C_{ij}m$ holds, $C_{ij}a$ and $C_{ij}u$ do not hold.

Analysis of Presupposition in terms of Common Knowledge



A presupposition is a piece of common knowledge between speaker and hearer in a discourse.

'bachelor' has presupposition 'male' and 'adult', and conveys information 'unmarried'.

$$!(C_{ij}(m \wedge a) \wedge u)$$

Update result:

Facts About Public Announcement of Common Knowledge

$$M \models_w [!C_{ij}\phi]\psi \text{ iff } M \models_w C_{ij}\phi \rightarrow \psi.$$

Public announcement of common knowledge has the force of an implication.

$$M \models_w [!(C_{ij}\phi \wedge \phi')]\psi \text{ iff } M \models_w [!C_{ij}\phi][!\phi']\psi.$$

Putting a presupposition before an assertion has the same effect as lumping them together.

Presupposition Projection

Example: update without presupposition $!m$ (the statement **male**) followed by the update for **bachelor**).

$$
\begin{aligned}
[!m][!(C_{ij}(m \wedge a) \wedge u)]\chi &\leftrightarrow [!(m \wedge [!m](C_{ij}(m \wedge a) \wedge u))]\chi \\
&\leftrightarrow [!(m \wedge [!m]C_{ij}m \wedge [!m]C_{ij}a \wedge [!m]u)]\chi \\
&\leftrightarrow [!(m \wedge [!m]C_{ij}a \wedge [!m]u)]\chi \\
&\leftrightarrow [!(m \wedge C_{ij}(m, a) \wedge m \rightarrow u)]\chi \\
&\leftrightarrow [!(C_{ij}(m, a) \wedge m \wedge u)]\chi
\end{aligned}
$$

$C_{ij}(\phi, \psi)$ for $[!\phi]C_{ij}\psi$.

So the presuppositional part of the combined statement is $C_{ij}(m, a)$.

The assertional part is $m \wedge u$.

Presupposition Accommodation

Suppose p is common knowledge.

Then updating with statement $!(Cp \wedge q)$ has the same effect as updating with $!q$.

Suppose p is true in the actual world but not yet common knowledge.

Updating with $!(Cp \wedge q)$ wil lead to an inconsistent state

Updating with $!p$ followed by an update with $!(Cp \wedge q)$ will not.

Accommodation of the presupposition would consist of replacement of $!(Cp \wedge q)$ by $[!p][!(Cp \wedge q)]$.

By invoking the Gricean maxim 'be informative' one can explain why $[!p][!(Cp \wedge q)]$ is not appropriate in contexts where p is common knowledge.

Public Change

Extend the language with public change.

$[p := \phi]\psi$.

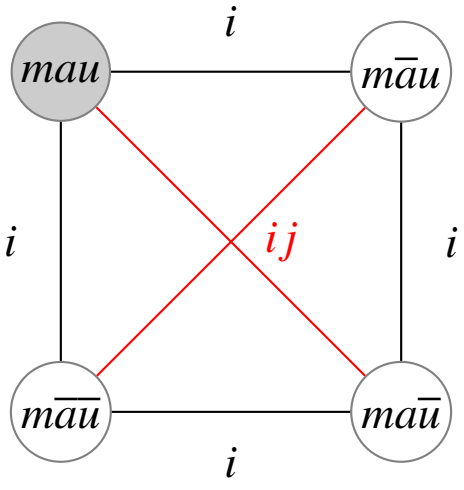
True in world w of M if ψ is true in world $w^{p:=[\![\phi]\!]_w}$ of $M^{p:=[\![\phi]\!]}$.

$p := \phi$ changes the model M to $M^{p:=[\![\phi]\!]}$.

Performative speech acts are examples:

- 'I call you Adam'
- 'Call me Ishmael'
- 'I declare you man and wife'

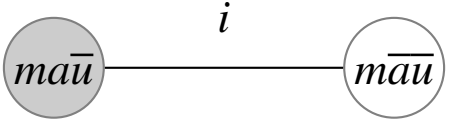
Marriage



Public change:

$$u := \bot$$

Result:



Epistemic Analysis of Yes/No Questions [7]

Let f be a propositional variable for question focus.

Analyse a Yes/No Question $\phi?$ as:

$$f := \phi$$

Analyze the answer ‘yes’ as:

$$!f$$

Analyze the answer ‘no’ as:

$$!\neg f$$

Questions and Appropriate Answers

Question: ‘Is Johnny married?’ Answer: ‘Johnny is not an adult.’

This answer is appropriate: updating with this answer makes ‘John is not married’ common knowledge. The update entails the answer ‘no’.

Question:

$$f := \phi$$

Answer:

$$!\psi$$

This is appropriate if either updating with ψ has the effect that f becomes common knowledge, or updating with ψ has the effect that $\neg f$ becomes common knowledge.

Dynamic Semantics

"Dynamic semantics is a perspective on natural language semantics that emphasises the growth of information in time. It is an approach to meaning representation where pieces of text or discourse are viewed as instructions to update an existing context with new information, with an updated context as result. In a slogan: meaning is context change potential. Prime source of inspiration for this dynamic turn is the way in which the semantics of imperative programming languages like C is defined."

Entry 'Dynamic Semantics' in the Stanford Encyclopedia of Philosophy http://plato.stanford.edu/

Connection with Dynamic Epistemic Logic

The SEP entry goes on:

> "Dynamic semantics comes with a set of flexible tools, and with a collection of 'killer applications', such as the compositional treatment of Donkey sentences, the account of anaphoric linking, the account of presupposition projection, and the account of epistemic updating. It is to be expected that advances in dynamic epistemic logic will lead to further integration. Taking a broader perspective, Some would even suggest that dynamic semantics is (nothing but) the application of dynamic epistemic logic in natural language semantics. But this view is certainly too narrow, although it is true that dynamic epistemic logic offers a promising general perspective on communication."

Conclusions: Program for Computational Semantics

- Common Knowledge and Common Belief Central Notions in Discourse Analysis, in Social Software, in Interaction Protocols
- Program: Analyzing Discourse as sequences of public announcements
- Program: Analyze Presupposition Projection and Accommodation in terms of common knowledge
- Analyze yes/no questions as public change of focus, Analyze appropriate answers in terms of 'same update effect'
  Program: extend this to a full semantic/pragmatic theory of questions and answers.
- Program: Analyze (language use in) social software protocols, and develop model checking tools for this.

References

[1] P. Blackburn and J. Bos. Representation and Inference for Natural Language; A First Course in Computational Semantics. CSLI Lecture Notes, 2005.

[2] 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 305ŋ–363. Amsterdam University Press, 2007.

[3] Jan van Eijck and Christina Unger. The epistemics of presupposition projection. In Maria Aloni, Paul Dekker, and Floris Roelofsen, editors, Proceedings of the Sixteenth Amsterdam Colloquium, December 17–19, 2007, pages 235–240, Amsterdam, December 2007. ILLC.

[4] Jan van Eijck and Christina Unger. Computational Semantics with Functional Programming. Cambridge University Press, 2010.

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

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

[7] J. Groenendijk and M. Stokhof. Studies on the Semantics of Questions and the Pragmatics of Answers. PhD thesis, University of Amsterdam, 1984.

[8] R. Montague. The proper treatment of quantification in or-

dinary English. In J. Hintikka, editor, Approaches to Natural Language, pages 221–242. Reidel, 1973.

[9] 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.

[10] 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.

[11] 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.

[12] S.M. Shieber. An Introduction to Unification Based Ap-

proaches to Grammar, volume 4 of CSLI Lecture Notes. CSLI, Stanford, 1986. Distributed by University of Chicago Press.