Comp24412 Symbolic AI

Lecture 13: Natural Language Semantics I

Ian Pratt-Hartmann

Room KB2.38
e-mail: ipratt@cs.man.ac.uk

2016–17
Outline

A notation for functions

Natural language semantics

Prolog implementation
Some functions defined on the integers:

\[ \lambda x [2x] = \{ \langle 0, 0 \rangle, \langle 1, 2 \rangle, \langle 2, 4 \rangle, \ldots \} \]

\[ \lambda x [x^2] = \{ \langle 0, 0 \rangle, \langle 1, 1 \rangle, \langle 2, 4 \rangle, \langle 3, 9 \rangle, \ldots \} \]

\[ \lambda x [(x + 4)^3] = \{ \langle 0, 64 \rangle, \langle 1, 125 \rangle, \langle 2, 216 \rangle, \ldots \} \]
• Some more functions

\[ \lambda x[x] = \lambda x[x + 0] = \{\langle 0, 0\rangle, \langle 1, 1\rangle, \langle 2, 2\rangle, \ldots\} \]
\[ \lambda x[x + 1] = \{\langle 0, 1\rangle, \langle 1, 2\rangle, \langle 2, 3\rangle, \ldots\} \]
\[ \lambda x[x + 2] = \{\langle 0, 2\rangle, \langle 1, 3\rangle, \langle 2, 4\rangle, \ldots\} \]

• Functions with functions as values

\[ \lambda y[\lambda x[x + y]] = \{\langle 0, \{\langle 0, 0\rangle, \langle 1, 1\rangle, \ldots\}\rangle,\]
\[ \langle 1, \{\langle 0, 1\rangle, \langle 1, 2\rangle, \ldots\}\rangle,\]
\[ \langle 2, \{\langle 0, 2\rangle, \langle 1, 3\rangle, \ldots\}\rangle, \ldots\} \]
• Applying functions:

\[(\lambda x[2x] \ 3) = 2.3 = 6\]
\[(\lambda x[x^2] \ 5) = 5^2 = 25\]
\[(\lambda x[(x + 4)^3] \ 2) = (2 + 4)^3 = 216\]
\[(\lambda y[\lambda x[x + y]] \ 2) = \lambda x[x + 2]\]
\[((\lambda y[\lambda x[x + y]] \ 2) \ 3) = (\lambda x[x + 2] \ 3)\]
\[= 2 + 3\]
\[= 5\]
• Note: some authors (actually, just Blackburn and Bos) use the @ notation, thus:

\[(\lambda x[2x])@3 = 2.3 = 6\]
\[(\lambda x[x^2])@5 = 5^2 = 25\]
\[(\lambda y[\lambda x[x + y]])@2)@3 = \lambda x[x + 2]@3\]
\[= 2 + 3\]
\[= 5\]
• Higher-order functions take functions as arguments.
• Here is a function which takes function on numbers, and applies it to 5:
  \[ \lambda f[(f \ 5)] \]
• Then we can compute
  \[
  (\lambda f[(f \ 5)] \ \lambda x[x + 1]) = (\lambda x[x + 1] \ 5) \\
  = 5 + 1 = 6.
  \]
• Notice that this procedure is completely mechanical.
• Functions with other domains and ranges

\[ \lambda x[(\text{prime } x)] = \{ \langle 0, F \rangle, \langle 1, F \rangle, \langle 2, T \rangle, \langle 3, T \rangle, \langle 4, F \rangle, \ldots \} \]

• Let \{john, mary, \ldots, jane\} be a set of individuals

\[ \lambda x[(\text{boy } x)] = \{ \langle john, T \rangle, \langle mary, F \rangle, \ldots, \langle jane, F \rangle \} \]

• and of course

\[ \lambda x[\lambda y[(\text{kissed } x) y]] = \{ \langle john, \{ \langle john, F \rangle, \langle mary, T \rangle, \ldots, \langle jane, F \rangle \} \rangle, \langle mary, \{ \langle john, T \rangle, \langle mary, F \rangle, \ldots, \langle jane, F \rangle \} \rangle, \ldots \langle jane, \{ \langle john, F \rangle, \langle mary, F \rangle, \ldots, \langle jane, F \rangle \} \rangle \} \]
• Again, we can have higher-order functions over such domains.
• the property of being a property possessed by Mary:
\[ \lambda p[(p \text{ mary})] \]
• the property of being a property possessed by every girl:
\[ \lambda p[\forall x((\text{girl } x) \to (p \ x))] \]
• the property of being a property possessed by some boy:
\[ \lambda p[\exists x((\text{boy } x) \land (p \ x))] \]
• The lambda calculus features two operations on lambda-expressions:

**α-conversion:** renaming of bound variables:

\[ \lambda x[\varphi(x)] \Rightarrow \lambda y[\varphi(y)] \]

where \( y \) is a variable not occurring free in \( \varphi \)

**β-reduction:** applying lambda functions to their arguments:

\[ (\lambda x[\varphi(x)] \psi) \Rightarrow \varphi(\psi) \]

where \( \psi \) is an expression of the appropriate type. Here, the expression on the left-hand-side is called the **redex** and the expression on the right, the **reduct**.
Outline

A notation for functions

Natural language semantics

Prolog implementation
Consider the (old-style) grammar rules:

\[
\begin{align*}
S & \rightarrow \text{NP, VP} \\
\text{NP} & \rightarrow \text{PropN} \\
\text{VP} & \rightarrow \text{Vi}
\end{align*}
\]

and lexicon

\[
\begin{align*}
\text{PropN} & \rightarrow \text{Mary} \\
\text{Vi} & \rightarrow \text{coughed}
\end{align*}
\]
• This grammar assigns the sentence *Mary coughed* the structure:
Let us now assign meanings as follows:

\[
[[\text{NP Mary}]] = \lambda p[(p \text{ mary})]
\]
\[
[[\text{Vi coughed}]] = \lambda x[(\text{coughed } x)]
\]

And let us augment the grammar rules with semantic information:

\[
S/(\varphi \psi) \rightarrow \text{NP}/\varphi, \text{VP}/\psi
\]
\[
\text{VP}/\varphi \rightarrow \text{Vi}/\varphi
\]
These meaning-assignments and grammar rule tell us that

$$[[S \text{ Mary coughed}]] = ([[\text{NP} \text{Mary}]] \ [ [\text{VP} \text{coughed}]]))$$
$$= (\lambda p[p(\text{mary})] \ \lambda x[(\text{coughed} \ x)])$$
$$= (\text{coughed} \ \text{mary})$$
• We can illustrate this process diagrammatically:

```
S
  coughed(mary)
    NP
      \lambda p[p(mary)]
    VP
      \lambda x[coughed(x)]

  NP
    Mary
  VP
    V
      \lambda x[coughed(x)]
```

\textit{coughed}
• Similarly, consider the sentence *Mary kissed John*, with structure
Let us assign meanings as follows:

\[[\text{NP John}] = \lambda p[(p \ john)]\]
\[[\text{NP Mary}] = \lambda p[(p \ mary)]\]
\[[\text{V kissed}] = \lambda s[\lambda x[\lambda y[((\text{kissed} x) y) y] y]]\]

And let us augment the usual VP rule with semantic information:

\[\text{VP}/(\varphi \ \psi) \rightarrow \text{V}/\varphi, \text{NP}/\psi\]
• This leads to the following meaning computation:

$$\text{kissed}(\text{mary}, \text{john}) = \lambda p[p(\text{mary})](\lambda x[\text{kissed}(x, \text{john})])$$
Outline

A notation for functions

Natural language semantics

Prolog implementation
• Lambda-expressions can easily be encoded in Prolog

Lambda-calculus: $\lambda x[(\text{boy } x)]$
Prolog: lbd(x, boy(x))

Lambda-calculus: $\lambda p[\lambda q[\forall x((p \ x) \rightarrow (q \ x))]]$
Prolog: lbd(p,lbd(q,forall(x,px -> qx)))
I have written a predicate `var_replace/2` that performs α-conversion:

\[ \lambda x[(f (g ((h x) x)))] \rightarrow \lambda x'[f (g ((h x') x'))] \]

?- var_replace(lbd(x,f(g(h(x,x)))),L).

L = lbd(x2, f(g(h(x2, x2)))) ;
No

\[ \lambda p[\lambda q[\exists x((p x) \land (q x))]] \rightarrow \lambda p'[\lambda q'[\exists x'((p' x') \land (q' x'))]] \]

var_replace(lbd(p,
    lbd(q,
    exists(x, (p@x & q@x)))),L).

L = lbd(p1, lbd(q1, exists(x1, p1@x1&q1@x1))) ;
No
var_replace(X,X):- atomic(X),!.

var_replace(X,Result):-
    X =.. [Head,Var,Expr], variable_binder(Head), !,
    gensym(Var,Result_var),
    substitute(Var,Result_var,Expr,Result_expr1),
    var_replace(Result_expr1,Result_expr),
    Result =.. [Head,Result_var,Result_expr].
var_replace(X,Result):-
    X =.. XList, maplist(var_replace,XList,ResultList),
    Result =.. ResultList.

variable_binder(lbd).
variable_binder(forall).
variable_binder(exists).
We define the substitution predicate as:

% Expression B is the result of substituting
% an occurrence of expression Y for each
% occurrence of expression X in expression A.

substitute(X,Y,X,Y):- !.
substitute(_X,_Y,A,A):- atomic(A), !.
substitute(X,Y,A,B):-
  A =.. A_list,
  maplist(substitute(X,Y),A_list,B_list),
  B =.. B_list.
Similarly, I have written a predicate `beta/2` that implements \( \beta \)-reduction assuming that there are no variable clashes:

\[
\begin{align*}
\lambda q[ \lambda p[\forall x(q(x) \to p(x))] & \quad \lambda x_1[\text{boy}(x_1)] = \lambda p[\forall x(\text{boy}(x) \to p(x))]
\end{align*}
\]

```prolog
beta(lbd(q, lbd(p, forall(x, (q@x -> p@x))))); 
L = lbd(p, forall(x, boy@x->p@x));
```

No
• Now `beta` is implemented as follows:

```prolog
beta(Exp, Exp):-
    atomic(Exp), !.
beta(lbd(V, F_body)@Exp, Result):-
    !,
    substitute(V, Exp, F_body, Result1),
    beta(Result1, Result).
beta(Exp, Result):-
    Exp=..ExpList,
    maplist(beta, ExpList, ResultList),
    Result=..ResultList.
```

• To think about: does the order in which the redexes are reduced matter?

• To think about: does the process even terminate?
• Now for the clever bit
• We can transcribe the augmented grammar rules
  \[ S/\varphi(\psi) \rightarrow NP/\varphi, VP/\psi \]
  \[ VP/\varphi(\psi) \rightarrow V/\varphi, NP/\psi \]
  as the Prolog DCG rules
  \[
  s(SSem) \rightarrow \\
  np(NPSem), vp(VPSem), \\
  \{var\_replace(NPSem,NPSem1), \ 
  beta(NPSem1@VPSem,SSem)\}.
  \]

  \[
  vp(VPSem) \rightarrow \\
  v(VSem), np(NPSem), \\
  \{var\_replace(VSem,VSem1), \ 
  beta(VSem1@NPSem,VPSem)\}.
  \]
And we can encode the meaning-assignments

\[
[[\text{NP John}]] = \lambda p[(p \ john)]
\]
\[
[[\text{NP Mary}]] = \lambda p[(p \ mary)]
\]
\[
[[\text{V kissed}]] = \lambda s[\lambda x[s(\lambda y[((\text{kissed} \ x) \ y))]]]
\]
as the the Prolog DCG rules

\[
\text{np(lbd(p,p@john)) --> [john].}
\]
\[
\text{np(lbd(p,p@mary)) --> [mary].}
\]
\[
\text{v(lbd(s,lbd(x,s@lbd(y,kissed(x,y)))))) --> [kissed].}
\]
• The result is:

?- s(SSem,[john,kissed,mary],[]).

SSem = kissed(john, mary);

• Prolog has performed the calculation carried out above
• Consider the sentence *Every boy loves some girl*.
• We assign meanings as follows:

[[Det every]] = \( \lambda q \lambda p[\forall x((q \ x) \rightarrow (p \ x))] \)

[[Det some]] = \( \lambda q \lambda p[\exists x((q \ x) \land (p \ x))] \)

[[V loves]] = \( \lambda s[\lambda x[(s \ \lambda y[((\text{loves} \ x) \ y))])] \)

[[N boy]] = \( \lambda x[(\text{boy} \ x)] \)

[[N girl]] = \( \lambda x[(\text{girl} \ x)] \)

• And we augment the usual NP rule with semantic information:

\[ \text{NP} / (\varphi \ \psi) \rightarrow \text{Det} / \varphi, \text{N} / \psi \]
• We encode all of this as Prolog DCG rules in the usual way:

\[
np(NPSem) \rightarrow \\
\quad \text{det(DetSem), n(NSem),} \\
\quad \{\text{var_replace(DetSem,DetSem1),} \\
\quad \text{beta(DetSem1@NSem,NPSem)}\}.
\]

\[
det(lbd(q,lbd(p,\exists(x,(q@x \& p@x)))))) \rightarrow [\text{some}].
\]

\[
det(lbd(q,lbd(p,\forall(x,(q@x -> p@x)))))) \rightarrow [\text{every}].
\]

\[
v(lbd(s,lbd(x,s@lbd(y, (\text{loves(x,y)})))))) \rightarrow [\text{loves}].
\]

\[
n(lbd(x,\text{boy}(x)))\rightarrow [\text{boy}].
\]

\[
n(lbd(x,\text{girl}(x)))\rightarrow [\text{girl}].
\]
• Lo and behold:

\[
\text{?- } s(SSem, [\text{every, boy, loves, some, girl}], []), \text{ pp(SSem)}.
\]

\[
(\text{x11}) (\text{boy(x11)} \Rightarrow (\exists \text{x2})(\text{girl(x2) \& loves(x11, x2)}))
\]

• which, when we change the fonts a bit, becomes

\[
\forall \text{x11}(\text{boy(x11)} \rightarrow \exists \text{x2}(\text{girl(x2) \& loves(x11, x2)}))
\]
Similarly:

?- s(SSem, [some, girl, loves, some, girl], []), pp(SSem).

(Ex41)(girl(x41) &
  (Ex5)(girl(x5) &
    loves(x41, x5)))

which, when we change the fonts a bit, becomes

\[ \exists x_{41} (girl(x_{41}) \land \exists x_{5} (girl(x_{5}) \land loves(x_{41}, x_{5}))) \]

Wow!
Summary:

- The Lambda calculus:
  - $\alpha$-conversion
  - $\beta$-reduction

- Semantics:
  - Meaning assignment to terminals
  - Semantically augmented grammar
  - Computation of sentence meaning

- Transcription into Prolog
- Illustrations