Lab & More on Quantification

Ling 233B 1/29/02

Getting Started

Copy the following files from the class directory to your own glue directory

`cp /afs/ir/class/linguist233b/prolog/semlex.pl ~/glue`
A preliminary semantic lexicon

`cp /afs/ir/class/linguist233b/prolog/class_doc.txt ~/glue`
Some documentation

The lexicon in semlex.pl contains some illustrative entries for proper names and intransitive verbs

Running Glue & XLE

```
1. In your ~/glue directory, invoke emacs
2. ESC-x shell
    % to start up a shell
3. ./glue_image
    % to start the prolog / glue process
4. | ?- go.
    % type "go." at the prolog prompt to start the glue loop
5. Split the emacs window
6. ESC-x run-new-xle
    % start the XLE in the other window
7. parse "Chris saw David"
    % input to the XLE window
    % will open up four new XLE windows displaying results.
8. On the lower left hand XLE window opened, select
        Prove semantics
```


Proper Name Entries

In semlex.pl you have the following entries

```
sem_template(name, _Name, LogicalConstant, true,
             sigma(^) -~> LogicalConstant
    ).

lex_sem('Chris', chris, 'N', name).
```

The format of these entries is:

```
sem_template(<template name>,          lex_sem(<word form>,
             <word form>,                      <semantic predicate>,
             <semantic predicate>,             <basic category>,
             <fstructure description>,         <template name>).
             <glue formula>).
```

Note Prolog Syntax

Every entry must be terminated with a period.

Use of underscore for anonymous variables

Atoms beginning with capital letters must be quoted (otherwise they are interpreted as prolog variables

Intransitive Verb Entries

```
sem_template(v_intrans, _Verb, Pred, true,
        &(X, sigma(p:[^,'SUBJ']) -~> X  -*   sigma(^) +~> [Pred, X])
    ).
lex_sem(yawns,    yawn,    'V',   v_intrans).
```

- `&(X, ...)`
  to represent universal quantification
- `sigma(^)`
  to indicate the semantic projection of ↑
- `sigma(p:[^,'SUBJ'])`
  to indicate semantic projection of ↑'s SUBJ
- `p:[^,'SUBJ']` is a *path* expression
- Two typed meaning assignments
  `r -~> M`: resource r has a type e(ntity) meaning M
  `r +~> M`: resource r has a type t(ruth) meaning M

Modifying and Loading Your Lexicon

Add new entries to your semlex.pl file

To load these new entries, first get to a prolog prompt

- Either hit Ctl-C in the prolog window
- Or select "Break semantics" from the fstructure window commands

At the prolog prompt, enter

```
| ?-  load_lex.
```

The lexicon compiler tries to produce some helpful warnings if the file contains errors (see class_doc.txt)

Hint: VAR and RESTR

To represent VARs and RESTRs in semantic projections, try the following kind of glue formula

```
&(X, 'VAR'(sigma(^)) -~> X
    -*
     'RESTR'(sigma(^)) +~> [man,X]
 )
```

Quantification

Today

- Quantified NPs v. Proper Names
- Quantifier scope ambiguity

Next week

- View quantified NPs as modifiers

The Problem with Quantifiers

Meanings for proper names are straight forward
Entity denoting arguments to which predicates apply

$$\frac{yawn : g \multimap f \qquad chris : g}{yawn(chris) : f}$$

But quantified NPs don't act as simple arguments
Someone yawned $\Rightarrow$ $some(X, person(X), yawn(X))$

Treat quantified NPs as functions looking for predicates as argument

$$\frac{yawn : g \multimap f \qquad \lambda P.some(X, person(X), P(X)) : (g \multimap f) \multimap f}{some(X, person(X), yawn(X)) : f}$$

Digression: Type Raising

Can also treat proper names as functions looking for predicates, via type raising

$$\frac{\dfrac{\dfrac{[P : g \multimap f]^1 \qquad chris : g}{P(chris) : f}}{\lambda P.P(chris) : (g \multimap f) \multimap f}\,{\multimap_{\mathcal{I},1}} \qquad yawn : g \multimap f}{yawn(chris) : f}$$

Type raising is often regarded as deeply mysterious.

In glue, it is just a consequence of the standard inference rules for implication

Scope Ambiguity: Everyone saw someone

Suppose we have the instantiated premises

- $\lambda x, y.see(x, y) : g \multimap (h \multimap f)$
- $\lambda P.every(X, person(X), P(X)) : (g \multimap f) \multimap f$
- $\lambda Q.some(Y, person(Y), Q(Y)) : (h \multimap f) \multimap f$

There are two distinct derivations of f

$$\frac{\dfrac{\dfrac{\dfrac{\dfrac{g \multimap (h \multimap f) \quad [g]^1}{h \multimap f} \quad [h]^2}{f}}{g \multimap f}\,{\multimap_{\mathcal{I},1}} \quad (g \multimap f) \multimap f}{f}}{h \multimap f}\,{\multimap_{\mathcal{I},2}} \quad (h \multimap f) \multimap f}{f}$$

$$\frac{\dfrac{\dfrac{\dfrac{\dfrac{g \multimap (h \multimap f) \quad [g]^1}{h \multimap f} \quad [h]^2}{f}}{h \multimap f}\,{\multimap_{\mathcal{I},2}} \quad (h \multimap f) \multimap f}{f}}{g \multimap f}\,{\multimap_{\mathcal{I},1}} \quad (g \multimap f) \multimap f}{f}$$

One Derivation, with Meaning Terms

$$\dfrac{\dfrac{\dfrac{\dfrac{\lambda x, y.see(x,y) : g \multimap (h \multimap f) \quad [X' : g]^1}{\lambda y.see(X',y) : h \multimap f} \quad [Y' : h]^2}{see(X',Y') : f}}{\lambda X'.see(X',Y') : g \multimap f}\multimap_{\mathcal{I},1}}{\dfrac{\dfrac{\lambda X'.see(X',Y') : g \multimap f \quad \lambda P.every(X, person(X), P(X)) : (g \multimap f) \multimap f}{every(X, person(X), see(X,Y') : f}}{\dfrac{\lambda Y'.every(X, person(X), see(X,Y')) : h \multimap f \quad \lambda Q.some(Y, person(Y), Q(Y)) : (h \multimap f) \multimap f}{some(Y, person(Y), every(X, person(X), see(X,Y))) : f}}}$$

Scope Across Clauses
Every man believed a woman yawned

Quantified NP "a woman" can scope over main or subordinate clause.

```
every(x, man(x),
         believe(x, some(y, woman(y), yawn(y))))

every(x, man(x),
         some(y, woman(y), believe(x, yawn(y))))

some(y, woman(y),
        every(x, man(x), believe(x, yawn(y))))
```

To allow quantifiers to 'float up', glue formula universally quantifies over scope resource, e.g.

$$\lambda P.every(X, man(X), P(X)) : \forall H.(g \multimap H) \multimap H$$

where H can be any (well, almost any!) semantic resource serving as the nuclear scope of the quantifier

F-structure & Premises

$$f : \begin{bmatrix} \text{PRED} & \text{believe} \\ \text{SUBJ} & g : \begin{bmatrix} \text{PRED} & \text{everyone} \end{bmatrix} \\ \text{COMP} & h : \begin{bmatrix} \text{PRED} & \text{yawn} \\ \text{OBJ} & k : \begin{bmatrix} \text{PRED} & \text{someone} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

- $believe : g \multimap (h \multimap f)$
- $yawn : k \multimap h$
- $everyone : \forall H_1.\ (g \multimap H_1) \multimap H_1$
- $someone : \forall H_2.\ (k \multimap H_2) \multimap H_2$

Derivations

every(x, man(x), believe(x, some(y, woman(y), yawn(y))))

$$\dfrac{\dfrac{\dfrac{k \multimap h \quad \dfrac{\forall H_2.\ (k \multimap H_2) \multimap H_2}{(k \multimap h) \multimap h}}{h} \quad \dfrac{[g] \quad g \multimap (h \multimap f)}{h \multimap f}}{f}}{\dfrac{g \multimap f \quad \dfrac{\forall H_1.\ (g \multimap H_1) \multimap H_1}{(g \multimap f) \multimap f}}{f}}$$

some(y, woman(y), every(x, man(x), believe(x, yawn(y))))

$$\dfrac{\dfrac{\dfrac{\dfrac{k \multimap h \quad [k]}{h} \quad \dfrac{[g] \quad g \multimap (h \multimap f)}{h \multimap f}}{f}}{g \multimap f} \quad \dfrac{\forall H_1.\ (g \multimap H_1) \multimap H_1}{(g \multimap f) \multimap f}}{\dfrac{f}{k \multimap f} \quad \dfrac{\forall H_2.\ (k \multimap H_2) \multimap H_2}{(k \multimap f) \multimap f}}{f}$$

A Non-Derivation

If the "someone" quantifier can float up, why can't the "everyone" quantifier float down?

Why prevents us from deriving the ill-formed meaning

```
believe(x, some(y, woman(y),
                   every(x, man(x), yawn(y)))
```

The quantifier meaning constructor

$$everyone : \forall H.\ (g \multimap H) \multimap H$$

says: freely choose a scope H, **provided that H depends on g.**

The subordinate clause cannot be a scope for the main clause subject, since its meaning does **not** depend on the main clause subject.

e and t Resources

To get scoping right, we need to restrict possible scopes to semantic resources that correspond to propositional meanings.

We thus sort all semantic resources into two types:

- t-resources (truth value denoting)
- e-resources (entity denoting)

The quantified variable, H in $everyone : \forall H.\ (g \multimap H) \multimap H$ needs to be restricted to range over only t-resources.

Note that g is an e-resource.

From the glue formula $(g_e \multimap H_t) \multimap H_t$
we can read of the type of the meaning expression: $((e,t),t)$.

Experiment: see what happens of you allow the variable H to range over type e resources.

To float the main clause subject down, we would have to be able to take a derivation step

$$\dfrac{\dfrac{[k] \quad k \multimap h}{h} \qquad \dfrac{\forall H.\ (g \multimap H) \multimap H}{(g \multimap h) \multimap h}}{h}$$

There is no way to do this