Encoding glue premises in the f-structure
Glue meeting, November 2019

1 Representing glue premises

Proper name:

(1) a. *David*: d_e

b. $$d : \left[\begin{array}{ll} \textsc{pred} & \textsc{David} \\ \textsc{glue} & \left\{\left[\begin{array}{ll} \textsc{rel} & \textsc{David} \\ \textsc{semstr} & d \\ \textsc{type} & e \end{array}\right]\right\} \end{array}\right]$$

Intransitive verb:

(2) a. *yawn*: $d_e \multimap y_t$

b. $$y : \left[\begin{array}{ll} \textsc{pred} & \text{‘}\textsc{yawn}\text{’}\!<\!\textsc{subj}\!>\text{’} \\ \textsc{subj} & d : [\ ] \\ \textsc{glue} & \left\{\left[\begin{array}{ll} \textsc{rel} & \textsc{yawn} \\ \textsc{arg1} & \left[\begin{array}{ll} \textsc{semstr} & d \\ \textsc{type} & e \end{array}\right] \\ \textsc{semstr} & y \\ \textsc{type} & t \end{array}\right]\right\} \end{array}\right]$$

Transitive verb:

(3) a. *see*: $d_e \multimap (c_e \multimap s_t)$

b. $$s : \left[\begin{array}{ll} \textsc{pred} & \text{‘}\textsc{see}\!<\!\textsc{subj,obj}\!>\text{’} \\ \textsc{subj} & d : [\ ] \\ \textsc{obj} & c : [\ ] \\ \textsc{glue} & \left\{\left[\begin{array}{ll} \textsc{rel} & \textsc{see} \\ \textsc{arg1} & \left[\begin{array}{ll} \textsc{semstr} & d \\ \textsc{type} & e \end{array}\right] \\ \textsc{arg2} & \left[\begin{array}{ll} \textsc{semstr} & c \\ \textsc{type} & e \end{array}\right] \\ \textsc{semstr} & s \\ \textsc{type} & t \end{array}\right]\right\} \end{array}\right]$$

Common noun:

(4) a. *man*: $p_e \multimap p_t$

b. $$m : \left[\begin{array}{ll} \text{PRED} & p : \text{‘MAN’} \\ \text{GLUE} & \left\{ \left[\begin{array}{ll} \text{REL} & \text{MAN} \\ \text{ARG1} & \left[\begin{array}{ll} \text{SEMSTR} & p \\ \text{TYPE} & e \end{array}\right] \\ \text{SEMSTR} & p \\ \text{TYPE} & t \end{array}\right] \right\} \end{array}\right]$$

Determiner:

(5) a. *a*: $\forall F.(p_e \multimap p_t) \multimap (m_e \multimap F_t) \multimap F_t$

b. $$m : \left[\begin{array}{ll} \text{PRED} & p \\ \text{GLUE} & \left\{ \left[\begin{array}{ll} \text{REL} & a \\ \text{ARG1} & \left[\begin{array}{ll} \text{ARG1} & \left[\begin{array}{ll} \text{SEMSTR} & p \\ \text{TYPE} & e \end{array}\right] \\ \text{SEMSTR} & p \\ \text{TYPE} & t \end{array}\right] \\ \text{ARG2} & \left[\begin{array}{ll} \text{ARG1} & \left[\begin{array}{ll} \text{SEMSTR} & m \\ \text{TYPE} & e \end{array}\right] \\ \text{SEMSTR} & F \\ \text{TYPE} & t \end{array}\right] \\ \text{FORALL} & F \\ \text{SEMSTR} & F \\ \text{TYPE} & t \end{array}\right] \right\} \end{array}\right]$$

NB: For some reason the “FORALL” attribute does not show up in the f-structure display, but it is there in the Prolog file.

Modifier, 2 premises:

(6) a. *interesting*: $(ip_e \multimap i_t)$,
$\lambda P.\lambda Q.\lambda x.and(P(x), Q(x)) : (ip_e \multimap i_t) \multimap (p_e \multimap p_t) \multimap (p_e \multimap p_t)$

b.
$$m : \begin{bmatrix} \text{PRED} & p \\ \text{ADJ} & \left\{ i : \begin{bmatrix} \text{PRED} & ip\text{: `INTERESTING'} \end{bmatrix} \right\} \\ \text{GLUE} & \left\{ \begin{matrix} \begin{bmatrix} \text{ARG1} & \begin{bmatrix} \text{SEMSTR} & ip \\ \text{TYPE} & e \end{bmatrix} \\ \text{REL} & \text{INTERESTING} \\ \text{SEMSTR} & i \\ \text{TYPE} & t \end{bmatrix} \\ \\ \begin{bmatrix} \text{ARG1} & \begin{bmatrix} \text{ARG1} & \begin{bmatrix} \text{SEMSTR} & ip \\ \text{TYPE} & e \end{bmatrix} \\ \text{SEMSTR} & i \\ \text{TYPE} & t \end{bmatrix} \\ \text{ARG2} & \begin{bmatrix} \text{ARG1} & \begin{bmatrix} \text{SEMSTR} & p \\ \text{TYPE} & e \end{bmatrix} \\ \text{SEMSTR} & p \\ \text{TYPE} & t \end{bmatrix} \\ \text{ARG3} & \begin{bmatrix} \text{SEMSTR} & p \\ \text{TYPE} & e \end{bmatrix} \\ \text{REL} & \text{/P./Q./x.AND(P(x),Q(x))} \\ \text{SEMSTR} & p \\ \text{TYPE} & t \end{bmatrix} \end{matrix} \right\} \end{bmatrix}$$

2 The premise rewriting component

The component that transfers f-structures to premises operates as follows:

- Read in the Prolog representation of the parsed sentence. Throw away the c-structure, and unpack the f-structure.
- For each f-structure, gather up the values of the GLUE attributes – these are the f-structure encodings of the glue premises. The value of the GLUE attribute can be a set of premises (this has been tested) or a single premise (this has not been tested). Throw the rest of the f-structure away.
- Transfer each f-structure premise to the prover format.
- Collect up the output premise sets for each f-structure, and get rid of duplicate premise sets.
- Pass all of the premise sets to the prover.

3 F-structure attributes and values in premises

Format for f-structure premises:

- Each premise is expected to have three attributes:
  - REL, whose value is the meaning side of the premise.
  - SEMSTR, whose value is an f-structure (standing in for the semantic structure)
  - TYPE, the type of SEMSTR (generally e or t, though this is not checked by the transfer component)
- Premises can also have arguments. Each argument should contain a SEMSTR and a TYPE. It can also contain argument attributes and values, for embedded implications.
- Attribute names for arguments: I have used ARG1-ARGn in the sample grammar. Actually, the transfer component does not care what the names are. It assumes that there are 4 special attributes (REL, SEMSTR, TYPE, and FORALL), and any other attributes are expected to encode arguments, with alphabetical order determining the order of arguments. So ARG1-ARGn should work, or A1-An, or A,B,C,... (but this has not been tested).
- FORALL is a special attribute with an f-structure value. Assume that its value is the f-structure that is labeled 18 in the Prolog representation. Then the resulting glue formula will have a universal quantifier binding f-structure 18 added at the beginning (`AF18`) and all occurrences of 18 in the rest of the formula will be prefixed by F, indicating that it is a variable.

4 Input format to prover

The prover accepts premises that look like this, with meanings like `/u.dog(u)`:

```
/u.dog(u) : (g_e -o g_t)
/v.bone(v) : (h_e -o h_t)
/P./Q./z.every(z,P(z),Q(z)) : ((g_e -o g_t) -o AX_t.(i_e -o X_t) -o X_t)
/R./S./z.a(z,P(z),Q(z)) : ((h_e -o h_t) -o AY_t.(j_e -o Y_t) -o Y_t)
/x./y.eat(x,y) : (i_e -o (j_e -o f_t))
```

Characters like the forward slash/lambda, comma, etc. in values of REL in the grammar must be escaped with backquote:

```
(^ REL) = `/P`.`/Q`.`/x`.and`(P`(x`)`,Q`(x`)`)
```