Lexical-Functional Grammar

Ash Asudeh
Carleton University

University of Iceland
July 3, 2009
Architecture and Structures
Basic Syntactic Architecture of LFG

• Two basic, simultaneous representations of syntax:
  
  • **C**(onstituent)-**s**tructure: constituency, dominance, word order, phrase structure
  Annotated trees
  
  • **F**(unctional)-**s**tructure: abstract grammatical relations/functions (subject, object, etc.), tense, case, agreement, predication, local and non-local dependencies
  Feature structures/attribute-value matrices

• Kaplan & Bresnan (1982):

  \[
  \phi
  \]

  constituent structure \(\rightarrow\) functional structure
LFG’s Parallel Projection Architecture


\[ \text{Form} \xrightarrow{\pi} \text{c-structure} \xrightarrow{\phi} \text{f-structure} \xrightarrow{\alpha} \text{anaphoric structure} \]

\[ \xrightarrow{\sigma} \text{semantic structure} \xrightarrow{\delta} \text{discourse structure} \]

\[ \text{Meaning} \]
LFG’s Parallel Projection Architecture

• Asudeh (2006):
Design Principles

• **Principle I: Variability**
  External structures (modelled by LFG c-structures) vary across languages.

• **Principle II: Universality**
  Internal structures (modelled by LFG f-structures) are largely invariant across languages.

• **Principle III: Monotonicity**
  The mapping from c-structure to f-structure is not one-to-one, but it is monotonic (information-preserving).
Nonconfigurationality

- Two fundamental ways for language to realize underlying concepts:
  - Phrase structure (groups)
  - Morphology (shapes)
  - English: phrase structure strategy (configurational)
  - Warlpiri: morphological strategy (nonconfigurational)
English

• Underlying meaning:
  That of ‘the two small children are chasing that dog’
English and Warlpiri

• English:

(1)  
  a. The two small children are chasing that dog.
  b. * The two small are chasing that children dog.
  c. * The two small are dog chasing children that.
  d. * Chasing are the two small that dog children.
  e. * That are children chasing the two small dog.

• Warlpiri:

  • All of the permutations in (1) are grammatical ways to express the same underlying concept of ‘the two small children are chasing that dog’
  • Even more permutations than this are possible
  • Only restriction: Aux must be in second position
    (Note: this is a slight simplification)
Warlpiri

- Underlying meaning:
  That of ‘the two small children are chasing that dog’
Abstract Syntax

• Despite the striking structural differences between English and Warlpiri, there are nevertheless common syntactic constraints on the two languages.

• Example: a subject can bind an object reflexive, but not vice versa

(1)  
  a. Lucy is hitting herself.
  b. * Herself is hitting Lucy.

(2)  
  a. Napaljarri-rl - ka-nyanu - paka-rni
     Napaljarri-ERG   PRES-REFL  hit-NONPAST
     ‘Napaljarri is hitting herself.’

  b. * Napaljarri - ka-nyanu paka-rni
     Napaljarri.ABS   PRES-REFL  hit-NONPAST
     ‘Herself is hitting Napaljarri.’

➡ How should abstract grammatical relations be captured?

**Transformational Grammar**: configurationally, using a uniform syntactic representation

**LFG**: non-configurationally, using a separate syntactic representation
C-structure

• Language variation in phrasal expression:
  • Basic word order:
    • SVO (English), SOV (Japanese), VSO (Irish), VOS (Malagasy)
  • Constituency:
    • Grouping of verb and complements,
    • Grouping of noun and modifiers
  • Strict vs. free word order:
    • configurational languages vs. case-marking languages
Constraints on C-structures: Phrase Structure Rules

- LFG distinguishes between the objects in the model and descriptions of those objects (i.e. *constraints* on the objects).

- C-structure trees are *constrained* by phrase structure rules.

\[
\begin{align*}
\text{IP} & \rightarrow \text{NP} \, \text{I}' \\
\text{IP} & \rightarrow \text{NP} \, \text{I}' \\
\end{align*}
\]

- Right-hand side of LFG phrase structure rules are *regular expressions*:

  - disjunction, optionality, arbitrary repetition (Kleene plus [+] and star [*])

\[
V' \rightarrow (V) \, (NP) \, \text{PP}^* 
\]
F-structures

- F-structures represent abstract **grammatical functions** (subject, object, etc.), **grammatical features** (tense, case, person, number, etc.), and **grammatical dependencies** (raising, control, unbounded dependencies)

(1) David devoured a sandwich.

```
[ PRED ‘DEVOUR<SUBJ, OBJ>’ ]
[ SUBJ [ PRED ‘DAVID’ ] ]
[ OBJ [ SPEC A ]
  [ PRED ‘SANDWICH’ ] ]
```
Anatomy of an F-structure
General Constraints on F-structures: Completeness, Coherence, Uniqueness

• **Completeness:**
  All the grammatical functions subcategorized by a predicate must be present in the f-structure.

  (1)* David devoured.

  **Devour <SUBJ, OBJ>**

• **Coherence:**
  Only the grammatical functions subcategorized by a predicate may be present in the f-structure.

  (2)* David devoured a sandwich that it was raining.

• **Uniqueness:**
  No attribute may have more than one value.
Uniqueness and Semantic Forms

- Semantic forms (values of PRED features) are unique.

\[
\begin{align*}
\text{PRED} & \quad \text{‘DEVOUR}_{37}(\text{SUBJ}, \text{OBJ}) \\
\text{SUBJ} & \quad \begin{bmatrix}
\text{PRED} & \quad \text{‘DAVID}_{42}' \\
\end{bmatrix} \\
\text{OBJ} & \quad \begin{bmatrix}
\text{SPEC} & \quad \text{A} \\
\text{PRED} & \quad \text{‘SANDWICH}_{14}' \\
\end{bmatrix}
\end{align*}
\]

- Multiple instances of semantic forms cannot unify, even if the semantic forms are otherwise compatible.

(1) * David devoured a sandwich a sandwich.
Features and the Lexicon in LFG
Lexical Entries in LFG

**yawns** V

(!↑ PRED)=‘yawn{SUBJ}’
(!↑ VFORM)=FINITE
(!↑ TENSE)=PRES
(!↑ SUBJ PERS)=3
(!↑ SUBJ NUM)=SG

F(unctional)-description, made up of functional schemata
Two Main Kinds of F-structure Constraints: Defining Equations and Constraining Equations

• Functional schemata and functional descriptions are often referred to as equations. This is a little inaccurate, because equality is not always the relevant relation, but it is certainly the most common way of specifying constraints on f-structures in LFG. So the term has stuck.

• There are two main classes of f-structure constraints in LFG:

  1. Defining Equations

     These equations define the f-structure by specifying which features have which values. They ‘make it so’. Defining equations are stated with a simple equality (or other relation symbol).

     \[(f \text{ SUBJ NUM}) = \text{SG}\]
Two Main Kinds of F-structure Constraints: Defining Equations and Constraining Equations

- There are two main classes of f-structure constraints in LFG:

  2. Constraining Equations
  These equations further constrain the f-structure once it has been constructed.

  In other words:
  1. Satisfy defining equations, setting aside constraining equations, to get minimal model.
  2. Satisfy constraining equations.

  There are a number of different kinds of constraining equations, but the ones that check feature-value pairs are written with a subscript $c$ on the equality like this:

  $$(f \text{ SUBJ NUM}) = _c \text{ SG}$$
Other Kinds of Constraining Equations

Negative equation: \((f \text{TENSE}) \neq \text{PRESENT}\)

Existential constraint: \((f \text{TENSE})\)

Negative existential constraint: \(\neg(f \text{TENSE})\)
Optionality, Disjunction, Conjunction, Negation

sneeze  \((f \ PRED) = \text{‘SNEEZE<SUBJ>’}\)  Conjunction (implicit)
\{(f \ VFORM) = \text{BASE} |\)
\((f \ \text{TENSE}) = \text{PRES} \)
\(\neg \{(f \ \text{SUBJ PERS}) = 3\} \)
\((f \ \text{SUBJ NUM}) = \text{SG}\}\}

Negation \(\neg A \) or \(\neg \{ \ldots \} \)

Disjunction \(\{ A | B \} \)

- The lexical entry for ‘sneeze’ (from Dalrymple 2001:87) says the following:
  The PRED of ‘sneeze’ is ‘SNEEZE<SUBJ>’. Also (conjunction): Either (disjunction) the
  VFORM is BASE (i.e. it’s a non-finite form) or it has present tense and it is not the case
  that (negation) its subject has third person singular agreement features (cf. She sneeze.)

\(( (f \ \text{SUBJ PRED}) = \text{‘PRO’} ) \)

Optionality (A)

Hint: ‘pro-drop’ in LFG!
Outside-In and Inside-Out equations

- Outside-in equations with respect to an f-structure \( f \) make specifications about paths leading \textit{in} from \( f \):

\[
(\uparrow \text{COMP TENSE}) = \text{PRESENT}
\]

- Inside-out equations with respect to an f-structure \( f \) make specifications about paths leading \textit{out} from \( f \):

\[
(\text{COMP} \uparrow)
\]

- The two kinds of equation can be combined:

\[
((\text{COMP} \uparrow) \text{ TENSE}) = \text{PRESENT}
\]
Outside-In and Inside-Out equations

• Outside-in equations with respect to an f-structure $f$ make specifications about paths leading in from $f$:

$$ (f \ \text{COMP TENSE}) = \text{PRESENT} $$

• Inside-out equations with respect to an f-structure $f$ make specifications about paths leading out from $f$:

$$ (\text{COMP} \ f) $$

• The two kinds of equation can be combined:

$$ ((\text{COMP} \ f) \ \text{TENSE}) = \text{PRESENT} $$
Functional Uncertainty

- Simple or limited functional uncertainty can be expressed by defining abbreviatory symbols disjunctively:

  \[ GF = \{ \text{SUBJ} | \text{OBJ} | \text{OBJ}_\theta | \text{OBL} | \text{COMP} | \text{XCOMP} | \text{ADJ} | \text{XADJ} \} \]

- Unlimited functional uncertainty can be expressed with Kleene star (*) or Kleene plus (+), where \( X^* \) means ‘0 or more \( X \)’ and \( X^+ \) means ‘1 or more \( X \)’:

  \[ (\uparrow \text{FOCUS}) = (\uparrow \{ \text{XCOMP} | \text{COMP} \}^* \text{GF}) \]

  \[ (\uparrow \text{INDEX}) = (\text{GF}^+ \uparrow) \text{SUBJ INDEX} \]

- Note that f-descriptions are therefore written in a regular language, as is also the case for the right-hand side of c-structure rules.
Functional Descriptions and Subsumption

- F-descriptions are true of not just the smallest, ‘intuitively intended’ f-structure, but also any larger f-structure that contains the same information.*

* This relationship is called **subsumption**: In general, a structure A subsumes a structure B if and only if A and B are identical or B contains A and additional information not included in A.

$$ f \left[ \begin{array}{c} \text{PRED} \ '\text{GO(SUBJ)}' \\ \text{SUBJ} \ [\text{NUM} \ \text{SG}] \end{array} \right] \quad g \left[ \begin{array}{c} \text{PRED} \ '\text{GO(SUBJ)}' \\ \text{TENSE} \ \text{FUTURE} \\ \text{SUBJ} \ [\text{CASE} \ \text{NOM} \ [\text{NUM} \ \text{SG}]] \end{array} \right] $$

- An f-description is therefore true of not just the **minimal** f-structure that satisfies the description: the f-description is also true of the infinitely many other f-structures that the intended, minimal f-structure subsumes.
Minimization

- There is a general requirement on LFG’s solution algorithm that it yield the **minimal** solution: no features that are not mentioned in the f-description may be included.

- Let’s look at an example from Dalrymple (2001).

(1) David sneezed.

- F-description:

  \[
  \begin{align*}
  (f \text{ PRED}) &= \text{‘SNEEZE(SUBJ)’} \\
  (f \text{ TENSE}) &= \text{PAST} \\
  (f \text{ SUBJ}) &= g \\
  (g \text{ PRED}) &= \text{‘DAVID’}
  \end{align*}
  \]

(2) **Minimal consistent f-structure**

(3) **Consistent but non-minimal f-structure**

\[
\begin{align*}
(f \text{ PRED}) &= \text{‘SNEEZE(SUBJ)’} \\
(f \text{ TENSE}) &= \text{PAST} \\
(f \text{ SUBJ}) &= g \\
(g \text{ PRED}) &= \text{‘DAVID’}
\end{align*}
\]
Lexical Generalizations in LFG

<table>
<thead>
<tr>
<th>yawns</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>(↑ PRED)=‘yawn{SUBJ}’</td>
<td></td>
</tr>
<tr>
<td>(↑ VFORM)=FINITE</td>
<td></td>
</tr>
<tr>
<td>(↑ TENSE)=PRES</td>
<td></td>
</tr>
<tr>
<td>(↑ SUBJ PERS)=3</td>
<td></td>
</tr>
<tr>
<td>(↑ SUBJ NUM)=SG</td>
<td></td>
</tr>
</tbody>
</table>

A lot of this f-description is shared by other verbs.
LFG Templates: Relations between Descriptions

\[ \text{yawns} \quad (\uparrow \text{PRED}) = \text{`yawn<SUBJ>'} \]
\[ (\uparrow \text{VFORM}) = \text{FINITE} \]
\[ (\uparrow \text{TENSE}) = \text{PRES} \]
\[ (\uparrow \text{SUBJ PERS}) = 3 \]
\[ (\uparrow \text{SUBJ NUM}) = \text{SG} \]

\[ \downarrow \]

\[ \text{yawns} \quad (\uparrow \text{PRED}) = \text{`yawn<SUBJ>'} \]
\[ \text{PRESENT} = (\uparrow \text{VFORM}) = \text{FINITE} \]
\[ (\uparrow \text{TENSE}) = \text{PRES} \]
\[ 3\text{SG} = (\uparrow \text{SUBJ PERS}) = 3 \]
\[ (\uparrow \text{SUBJ NUM}) = \text{SG} \]

\[ \text{@PRESENT} \]
\[ \text{@3SG} \]
Templates: Factorization and Hierarchies

\[
\begin{align*}
\text{FINITE} &= (\uparrow \text{vform})=\text{FINITE} \\
\text{PRES-TENSE} &= (\uparrow \text{tense})=\text{PRES} \\
\text{PRESENT} &= @\text{FINITE} \\
&\quad @\text{PRES-TENSE} \\
\downarrow \\
\text{PRES-TENSE} &\quad \text{FINITE} \\
&\quad \text{PRESENT} \\
\downarrow \\
\text{3PERSONSUBJ} &\quad \text{FINITE} \\
&\quad \text{PRESENT} \\
\text{SINGSUBJ} &= (\uparrow \text{subj pers})=3 \\
\text{3SG} &= @3\text{PERSONSUBJ} \\
&\quad @\text{SINGSUBJ} \\
\downarrow \\
\text{3PERSONSUBJ} &\quad \text{SINGSUBJ} \\
&\quad 3\text{SG}
\end{align*}
\]
Templates: Factorization and Hierarchies

\[
yawns \quad (\uparrow \text{PRED})=\text{‘yawn\{SUBJ\}’} \quad \text{Pres3Sg} = \text{@present} @3\text{Sg}
\]

\[
\downarrow
\]

\[
yawns \quad (\uparrow \text{PRED})=\text{‘yawn\{SUBJ\}’} \quad @\text{Pres3Sg}
\]

\[
\begin{array}{c}
\text{Pres-Tense} \\
\text{Finite} \\
3\text{PersonSubj} \\
\text{SingSubj}
\end{array}
\]

\[
\begin{array}{c}
\text{Present} \\
3\text{Sg} \\
\text{Pres3Sg}
\end{array}
\]
Templates: Boolean Operators

\[
\text{PresNot3SG} = \begin{array}{c}
\text{@PRESENT} \\
\neg@3SG
\end{array}
\Rightarrow
\begin{align*}
(\uparrow \text{VFORM}) &= \text{FINITE} \\
(\uparrow \text{TENSE}) &= \text{PRES} \\
\neg\{(\uparrow \text{SUBJ PERS}) &= 3 \\
(\uparrow \text{SUBJ NUM}) &= \text{SG}\}
\end{align*}
\]

\[
\text{PRES-TENSE} \quad \text{FINITE} \quad 3\text{PersonSubj} \quad \text{SingSubj}
\]

\[
\text{PRESENT} \quad 3\text{SG}
\]

\[
\text{PresNot3SG} \quad \text{Pres3SG}
\]
Hierarchies: Templates vs. Types

• Type hierarchies are *and/or* lattices:
  • Motherhood: *or*
  • Multiple Dominance: *and*

• Type hierarchies encode inclusion/inheritance and place constraints on how the inheritance is interpreted.

• LFG template hierarchies encode only inclusion: multiple dominance not interpreted as conjunction, no real status for motherhood.

• LFG hierarchies relate descriptions only: mode of combination (logical operators) is determined contextually at invocation or is built into the template.

• HPSG hierarchies relate first-class ontological objects of the theory.

• LFG hierarchies are abbreviatory only and have no real ontological status.
Hierarchies: Templates vs. Types

HPSG

HEAD

NOUN

RELATIONAL

C-NOUN

GERUND

VERB

LFG

PRES-TENSE

FINITE

3PERSON

SUBJ

SING

SUBJ

PRESENT

3SG

PRES

NOT 3SG

PRES

3SG
Parameterized Templates

\[ \text{yawns} \quad (\uparrow \text{PRED})=\text{‘yawn} \langle \text{SUBJ} \rangle \text{’} \quad \text{INTRANSITIVE}(P) = (\uparrow \text{PRED})=\text{‘P} \langle \text{SUBJ} \rangle \text{’} \]
\@PRES3SG

\[ \downarrow \]

\[ \text{yawns} \quad \@\text{INTRANSITIVE}(\text{yawn}) \]
\@PRES3SG
Parameterized Templates

\[
\text{TRANSITIVE}(p) = (\uparrow \text{PRED})=\text{P}(\text{SUBJ}, \text{OBJ})
\]

\[
\text{TRANS-OR-INTRANS}(p) = \text{@TRANSITIVE}(p) \lor \text{@INTRANSITIVE}(p)
\]

\[
(\uparrow \text{PRED})='eat(\text{SUBJ}, \text{OBJ})' \lor (\uparrow \text{PRED})='eat(\text{SUBJ})'
\]
Temple Hierarchy with Lexical Leaves

```
                3PersonSubj
                  /
                 PRESENT
                   |
                  3sg
                   |
               Pres3sg
                      |
                     yawns
```

```
                SingSubj
                  /
                 INTRANSITIVE
                   |
                  TRANSITIVE
                   |
               TRANS-OR-INTRANS
                      |
                     eats
                      |
                     cooked
```
Defaults in LFG

\[ (↑\text{CASE}) ∨ (↑\text{CASE}) = \text{NOM} \]

The f-structure must have case and if nothing else provides its case, then its case is nominative.

\[ \text{DEFAULT}(D \hspace{0.5em} V) = D ∨ D = V \]

Parameterized template for defaults.

Also illustrates that parameterized templates can have multiple arguments.

\[ @\text{DEFAULT}((↑\text{CASE}) \text{ NOM}) \]
C-structure Annotation of Templates

\[
\begin{align*}
\text{VP} & \longrightarrow V \quad \text{ADVP}^* \\
& \uparrow=\downarrow \quad \downarrow \in (\uparrow \text{ADJUNCT}) \\
& (\downarrow \text{ADJUNCT-TYPE})=\text{VP-ADJ} \\
\text{ADJUNCT}(P) &= \downarrow \in (\uparrow \text{ADJUNCT}) \quad @\text{ADJUNCT-TYPE}(P) \\
\text{ADJUNCT-TYPE}(P) &= (\downarrow \text{ADJUNCT-TYPE})=P
\end{align*}
\]
Features in the Minimalist Program
Features and Explanation

• The sorts of features that are associated with functional heads in the Minimalist Program are well-motivated morphosyntactically, although other theories may not draw the conclusion that this merits phrase structural representation (cf. Blevins 2008).

• Care must be taken to avoid circular reasoning in feature theory:
  • The ‘strong’ meta-feature: “This thing has whatever property makes things displace, as evidenced by its displacement.”
  • The ‘weak’ meta-feature: “This thing lacks whatever property makes things displace, as evidenced by its lack of displacement.”
  • The EPP feature: “This thing has whatever property makes things move to subject position, as evidenced by its occupying subject position.”
Features and Simplicity

- Adger (2003, 2008) considers three kinds of basic features:
  - Privative, e.g. [singular]
  - Binary, e.g. [singular +]
  - Valued, e.g. [number singular]
- Adger considers the privative kind the simplest in its own right.
- This may be true, but only if it does not introduce complexity elsewhere in the system (Culicover & Jackendoff 2005: ‘honest accounting’).
- Notice that only the final type of feature treats number features as any kind of natural class within the theory (as opposed to meta-theoretically).
Kinds of Feature-Value Combinations

- Adger (2003):
  - Privative
    - [singular], [V], ...
  - Binary
    - [singular: +] (?)
  - Attribute-value
    - [Tense: past]
Interpreted vs. Uninterpreted Features

- Interpreted features:
  - [F]

- Uninterpreted features:
  - [uF]

- All uninterpreted features must be eliminated (‘checked’).

- Interpreted features are interpreted by the semantics.
  - Presupposes an interpretive (non-combinatorial) semantics.

[Notation from Adger 2003]
Feature Strength

• Strong features must be checked locally: Trigger Move/Internal Merge/Remerge
  • \([F^*]\)

• Weak features do not have to be checked locally: Do not trigger Move
  • \([F]\)

[Notation from Adger 2003]
An Example: Auxiliaries

- Adger (2003:181)

“When [uInfl:] on Aux is valued by T, the value is strong; when [uInfl:] on v is valued by T, the value is weak.”

```
TP
  /   \                  
 T[subj] T[aux]           
  /   \                  
 NegP NegP               
  /   \                  
 ⟨ subj ⟩ vP             
  /   \                  
 Verb + v[uInfl]...     
```
Locality of Feature Matching

• Adger (2003:218)

**Locality of Matching**
A*gree holds between a feature $F$ on $X$ and a matching feature $F$ on $Y$ if and only if there is no intervening $Z[F]$.

**Intervention**
In a structure $[X \ldots Z \ldots Y]$, $Z$ intervenes between $X$ and $Y$ iff $X$ c-commands $Z$ and $Z$ c-commands $Y$. 
Feature-Value Unrestrictiveness & Free Valuation

• Asudeh & Toivonen (2006) argue that the Minimalist feature system of Adger (2003) has two undesirable properties.

**Feature-value unrestrictiveness**
Feature valuation is unrestricted with respect to what values a valued feature may receive.

**Free valuation**
Feature valuation appears freely, subject to locality conditions.

• This results in a very unconstrained theory of features.

• This may sound good, because it’s less stipulative and hence more Minimal, but from a theory perspective it is bad: unconstrained theories are less predictive.
Example: English Subject Agreement

(1) Gilgamesh missed Enkidu

(2) Gilgamesh misses Enkidu

- Contrast with HPSG: MP has no typing of values (feature value unrestrictiveness)
- Contrast with LFG: MP has valuation without specification (free valuation)
Two Contrasting Feature Theories

• HPSG (Pollard & Sag 1994): features are not just valued, the values are also *typed*

  • If two values can unify, they must be in a typing relation (one must be a subtype of the other).

  • Feature values in HPSG are thus tightly restricted by types.

• LFG (Kaplan & Bresnan 1982, Bresnan 2001): features are not restricted, but there is no free valuation

  • A feature cannot end up with a given value unless there is an explicit equation in the system.
Feature Simplicity and Constraint Types

• LFG offers the opportunity to consider Adger’s three feature types in light of a single feature type, with varying constraint types.

• LFG features are valued ($f$ is an LFG f(unctional)-structure):

$$f\left[\text{NUMBER singular}\right]$$

• Types of LFG feature constraints.
  • Defining equation: $(f\ \text{NUMBER}) = \text{singular}$
  • Existential constraint: $(f\ \text{NUMBER})$
  • Negative existential constraint: $\neg(f\ \text{NUMBER})$
  • Constraining equation: $(f\ \text{NUMBER}) =_c \text{singular}$
  • Negative constraining equation: $(f\ \text{NUMBER}) \neq \text{singular}$
Feature Simplicity and Constraint Types

- All features treated as valued features: no restriction on constraint types
- All features treated as binary features: only positive and negative constraining equations allowed
- All features treated as privative: only negative and existential constraints allowed

  - This understanding of privative features actually does treat number as a natural class.

  - This treats the notion of feature simplicity as a kind of meta-theoretical statement in an explicit, non-ad-hoc feature theory.
Control and Raising
Lexical Entries

**tried**  V  \((\uparrow \text{PRED}) = \text{‘try}\langle \text{SUBJ}, \text{XCOMP}\rangle\text{’}\)  
\((\uparrow \text{SUBJ}) = (\uparrow \text{XCOMP SUBJ})\)

**seemed**  V  \((\uparrow \text{PRED}) = \text{‘seem}\langle \text{CF}\rangle\text{SUBJ}\text{’}\)  
\{  
\((\uparrow \text{SUBJ}) = (\uparrow \text{XCOMP SUBJ})\)  
|  
\((\uparrow \text{SUBJ PRONTYPE}) = \text{EXPLETIVE}\)  
\((\uparrow \text{SUBJ FORM}) = \text{IT}\)  
\((\uparrow \text{COMP}) \}\)
Raising to Subject/Subject Control C-structure
F-structures

\[
\begin{array}{c}
\text{PRED} & \text{‘seem} (XCOMP)\text{‘(SUBJ)}' \\
\text{SUBJ} & \text{PRED} \quad \text{‘Gonzo'} \\
\text{XCOMP} & \text{PRED} \quad \text{‘leave} (XCOMP)\text{‘(SUBJ)}' \\
\text{SUBJ} & \text{PRED} \quad \text{‘Gonzo'} \\
\text{XCOMP} & \text{PRED} \quad \text{‘leave}(XCOMP)\text{‘(SUBJ)}'
\end{array}
\]
Copy Raising
Data

(1) Thora seems like she enjoys hot chocolate.
(2) Thora seems like Isak pinched her again.
(3) Thora seems like Isak ruined her book.
(4)* Thora seems like Isak enjoys hot chocolate.
(5)* Thora seems like Isak pinched Justin again.
(6)* Thora seems like Isak ruined Justin’s book.
Data

(7) It seems like there is a problem here.

(8) It seems like Thora is upset.

(9) It seems like it rained last night.

(10) There seems like there’s a problem here.

(11) * There seems like it rained last night.
Lexical Entries

$like_1 \quad P^0 \quad (\uparrow \text{PRED}) = \text{‘like}$\langle\text{SUBJ,COMP}\rangle$

$like_2 \quad P^0 \quad (\uparrow \text{PRED}) = \text{‘like}$\langle\text{CF}\rangle\text{SUBJ}$

\{ (\uparrow \text{SUBJ}) = (\uparrow \text{XCOMP SUBJ}) \mid \\
(\uparrow \text{SUBJ PRONTYPE}) = \text{EXPLETIVE} \\
(\uparrow \text{SUBJ FORM}) = \text{IT} \\
(\uparrow \text{COMP}) \}\
C-structure

IP
  (↑ SUBJ) = ↓  ↑ = ↓
  DP
     ↓
     Richard
  ↑ = ↓
  VP
       ↑ = ↓
       V^0
           ↓
           seems
               ↑ = ↓
               P'
                   ↑ = ↓
                   P^0
                       ↓
                       like
                           ↑ = ↓
                           (↑ SUBJ) = ↓
                           DP
                               ↓
                               he
                                   ↑ = ↓
                                   VP
                                       ↓
                                       smokes
F-structure

[SUBJ]

[PRED] ‘seem’

[XCOMP]

[SUBJ]

[PRED] ‘like’

[SUBJ]

[PRED] ‘Richard’

[COMP]

[PRED] ‘smoke’

[SUBJ]

[PRED] ‘pro’

[PERS] 3

[NUM] sg

[GEND] masc
C-structure
F-structure

```
[ PRED
  \[ SUBJ
    \[ PRED
      \[ SUBJ
        \[ PRED
          \[ SUBJ
            \[ PRED
              \[ SUBJ
                \[ PRED
                  \[ EXPL there
                    \[ PRED
                      \[ SPEC
                        \[ PRED a
                          \[ XCOMP
                            \[ PRED 'be'
                              \[ XCOMP
                                \[ PRED 'like'
                                  \[ XCOMP
                                    \[ PRED 'seem'
                                      \[ XCOMP
                                        \[ SUBJ
                                          \[ OBJ
                                            \[ SPEC
                                              \[ PRED 'problem'
                                                \[ SPEC
                                                  \[ PRED 'a'
                                                    \[ SPEC
                                                      \[ PRED 'the'
                                                        \[ SPEC
                                                          \[ PRED 'there'
                                                            \[ SPEC
                                                              \[ PRED 'a'
        \[ SUBJ
      \[ PRED
  \[ SUBJ
```

Unbounded Dependencies
Filler-Gap Dependencies
Functional Uncertainty

- The syntactic relationship between the top and bottom of an unbounded dependency is represented with a functional uncertainty:

- Top = MiddlePath-Func-Uncertainty  Bottom-Func-Uncertainty

(1) [What] [did Kim claim that Sandy suspected that Robin knew]   

(↑ FOCUS) = (↑ COMP* {OBJ | OBJ_θ})

(2) [What] [did Kim claim that Sandy suspected that Robin gave Bo]   

(↑ FOCUS) = (↑ COMP* {OBJ | OBJ_θ})
**Wh-Questions: Example**

Who does David like?
Wh-Questions: Annotated PS Rule

$$CP \quad \rightarrow \quad \left( \begin{array}{l}
\text{QuesP} \\
(\uparrow \text{FOCUS}) = \downarrow \\
(\uparrow \text{FOCUS}) = (\uparrow \text{QFOCUSPATH}) \\
(\uparrow Q) = (\uparrow \text{FOCUS WHPATH}) \\
(\uparrow Q \text{ PRONTYPE}) =_c WH
\end{array} \right) \quad \left( \begin{array}{c}
C' \\
\uparrow = \downarrow
\end{array} \right)$$
Wh-Questions: QuesP Metacategory

\[ \text{QuesP} \equiv \{ \text{NP} \mid \text{PP} \mid \text{AdvP} \mid \text{AP} \} \]

1. NP: Who do you like?
2. PP: To whom did you give a book?
3. AdvP: When did you yawn?
4. AP: How tall is Chris?
Wh-Questions: Unbounded Dependency Equation

English QFOCUSPATH:

\[
\{ \text{XCOMP} | \text{COMP} \rightarrow \text{LDD} \neq \text{OBJ} \rightarrow \text{TENSE} \}^* \{ (\text{ADJ} \in \text{TENSE} \rightarrow \text{TENSE}) (\text{GF}) \mid \text{GF} \} \]
Wh-Questions: Pied Piping

English WhPATH:
\{ \text{SPEC}^* \mid \text{OBJ} \} \\

(1) [Whose book] did you read?  
(2) [Whose brother’s book] did you read?  
(3) [In which room] do you teach?
Relative Clauses: Example

*a man who Chris saw*

(a man who Chris saw)
Relative Clauses: Annotated PS Rule

\[
\text{CP} \rightarrow \begin{pmatrix}
\text{RelP} \\
(\uparrow \text{TOPIC}) = \downarrow \\
(\uparrow \text{TOPIC}) = (\uparrow \text{RTOPICPath}) \\
(\uparrow \text{RELPRO}) = (\uparrow \text{TOPIC RELPath}) \\
(\uparrow \text{RELPRO PRONTYPE}) \equiv_c \text{REL}
\end{pmatrix} \quad \begin{pmatrix}
\text{C'} \\
\uparrow = \downarrow
\end{pmatrix}
\]
Relative Clauses: RelP Metacategory

\[ \text{RelP} \equiv \{ \text{NP} \mid \text{PP} \mid \text{AP} \mid \text{AdvP} \} \]

(1) NP: a man who I selected
(2) PP: a man to whom I gave a book
(3) AP: the kind of person proud of whom I could never be
(4) AdvP: the city where I live
Relative Clauses: Unbounded Dependency Equation

English RTOPICPATH:

\[
\{ \text{XCOMP} \mid \text{COMP} \ \mid \text{OBJ} \}^* \ {\text{(ADJ}} \in \text{(GF)} \mid \text{GF})
\]

\[
- (\rightarrow \text{TENSE})
\]
Relative Clauses: Pied Piping

**English RELPATH:**

\[ \{ \text{SPEC}^* \mid [\text{(OBL}_θ\text{OBJ})^*] \} \]

1. the man [who] I met
2. the man [whose book] I read
3. the man [whose brother’s book] I read
4. the report [the cover of which] I designed
5. the man [faster than whom] I can run
6. the kind of person [proud of whom] I could never be
7. the report [the height of the lettering on the cover of which] the government prescribes
Relative Clauses: Pied Piping Example

a man whose book Chris read

[Diagram of syntactic structure showing the pied piping example]
Constraints on Extraction
Empty Category Principle/\textit{That}-Trace

(1) Who do you think [\_ left]?

(2) * Who do you think [that \_ left]?

(3) * What do you wonder [if \_ smells bad]?

(4) Who do you think [\_ should be trusted]?

(5) * Who do you think [that \_ should be trusted]?

(6) Who do you think [that, under no circumstances, \_ should be trusted]?

(7) Who do you wonder [if, under certain circumstances, \_ could be trusted]?
That-Trace in LFG

- LFG has a relation called f-precedence that uses the native precedence of c-structure to talk about precedence between bits of f-structure.
- F-precedence relies on LFG’s projection architecture and the inverse of the c-structure–f-structure mapping function $\phi$.
- The inverse is written $\phi^{-1}$ and returns the set of c-structure nodes that map to its argument f-structure node.

**F-precedence**
An f-structure $f$ f-precedes an f-structure $g$ ($f <_f g$) if and only if for all $n_1 \in \phi^{-1}(f)$ and for all $n_2 \in \phi^{-1}(g)$, $n_1$ c-precedes $n_2$. 
That-Trace in LFG

- We can leverage LFG’s projection architecture to capture the fact that That-Trace is a ‘surfacy’ phenomenon (cf. ECP as a PF constraint in recent Minimalism).

Form

string \quad c\text{-}structure \quad f\text{-}structure
**That-Trace in LFG**

- Assume a native precedence relation on strings, yielding a notion of element that is string-adjacent to the right (‘next string element’), which we define as $\text{Right}_{\text{string}}(\pi^{-1}(\ast))$, where $\ast$ designates the current c-structure node in a phrase structure rule element or lexical entry.

- Let’s abbreviate the right string-adjacent element to $\ast$ as $\succ$.

- The semantics of $\succ$ is ‘the string element that is right string-adjacent to me’.

- Note that $\pi^{-1}$ returns string elements, not sets of string elements, because $\pi$ is bijective, since c-structures are trees.
**That-Trace in LFG**

- We can use f-precedence and $>$ to capture the surfacy nature of That-Trace.

- Basically, English has a (somewhat arbitrary) constraint that the right-adjacent string element to the complementizer must be locally realized.

- This can be stated by requiring that any unbounded dependency function in the f-structure corresponding to the element that occurs in the string immediately after the complementizer should not f-precede the complementizer’s f-structure.
Left Branch Constraint

(1) Whose car did you drive __?

(2)* Whose did you drive [__ car]?
Left Branch Constraint in LFG

- Do not include SPEC/POSS in GFs of possible extraction sites.
- Note that the equation we looked at previously already disallows the extraction from passing through a SPEC in the first part.
- We modify the equation as follows

\[
\{XCOMP \mid \text{COMP} \not\in (\rightarrow \text{TENSE}) \}^* \{\text{ADJ} \in \rightarrow \text{TENSE} \} \text{ (GF) | GF - SPEC}\]
**Wh-Islands in LFG: Off-Path Constraints**

- The off-path metavariable $\leftarrow$ refers to the f-structure that contains the attribute that the constraint is attached to.

- The off-path metavariable $\rightarrow$ refers to the f-structure that is the value of the attribute that the constraint is attached to.

\[
\begin{align*}
\{ \text{XCOMP} & | \text{COMP} \rightarrow \text{LDD} \neq - \rightarrow \text{TENSE} \}^* \{ (\text{ADJ} \notin \rightarrow \text{TENSE}) \text{ (GF) } | \text{GF - SPEC} \} \\
\{ \text{XCOMP} & | \text{COMP} \rightarrow \text{LDD} \neq - \rightarrow \text{TENSE} \}^* \{ (\text{ADJ} \notin \rightarrow \text{TENSE}) \text{ (GF) } | \text{GF - SPEC} \} \end{align*}
\]

- Use $\leftarrow$ to state the bottom cannot be in an f-structure that has an unbounded dependency function UDF, where UDF = \{TOPIC | FOCUS\}.

\[
\{ \text{XCOMP} | \text{COMP} \rightarrow \text{LDD} \neq - \rightarrow \text{TENSE} \}^* \{ (\text{ADJ} \notin \rightarrow \text{TENSE}) \text{ (GF) } | \text{GF - SPEC} \} \not\leftarrow (\leftarrow \text{UDF})
\]
Successive Cyclic Effects
Successive Cyclicity

- Data from languages such as Irish and Chamorro, which show successive marking along the extraction path, have motivated the claim that extraction/movement is ‘cyclic’ (not all at once). Cf. Phases in Minimalism.

- Of course, this data does not argue for movement per se, as some have wrongly assumed, but rather that unbounded dependencies should

  1. Be made up of a series of local relations; or

  2. Have a way to refer to their environments as the dependency is constructed.

- HPSG has adopted the first approach, LFG the second.
Data: Irish

- Note: Date from McCloskey via Bouma et al. (2001).

a. Shíl mé goN mbeadh sé ann
   thought I PRT would-be he there
   I thought that he would be there.

b. Dúirt mé gurL shíl mé goN mbeadh sé ann
   said I goN+PAST thought I PRT would-be he there
   I said that I thought that he would be there.

c. an fear aL shíl mé aL bheadh __ ann
   [the man] j PRT thought I PRT would-be __j there
   the man that I thought would be there

d. an fear aL dúirt mé aL shíl mé aL bheadh __ ann
   [the man] j PRT said I PRT thought I PRT would-be __j there
   The man that I said I thought would be there

e. an fear aL shíl __ goN mbeadh sé ann
   [the man] j PRT thought __j PRT would-be he there
   the man that thought he would be there
Irish Successive Cyclicity in LFG

\[ aL \hat{C} (\uparrow \text{UDF}) = (\uparrow \text{CF}* \text{GF}) (\rightarrow \text{UDF}) = (\uparrow \text{UDF}) \]

Note: UDF = \{TOPIC | FOCUS\}, CF = \{XCOMP | COMP\}

\[ goN \hat{C} (\uparrow \text{TENSE}) \neg(\uparrow \text{UDF}) \]
Glue Semantics
Glue Semantics

• Glue Semantics is a type-logical semantics that can be tied to any syntactic formalism that supports a notion of headedness.
• Glue Semantics can be thought of as *categorial semantics without categorial syntax*.
• The independent syntax assumed in Glue Semantics means that the logic of composition is *commutative*, unlike in Categorial Grammar.
• Selected works:
Glue Semantics

- Lexically-contributed meaning constructors :=

  Meaning language term \( \mathcal{M} : G \)  Composition language term

  Meaning language := some lambda calculus
  - Model-theoretic
  - Composition language := linear logic
  - Proof-theoretic
  - Curry Howard Isomorphism between formulas (meanings) and types (proof terms)
  - Successful Glue Semantics proof:

    \[ \Gamma \vdash \mathcal{M} : G_t \]
Key Glue Proof Rules with Curry-Howard Terms

Application : Implication Elimination

\[
\begin{array}{c}
\text{Application : Implication Elimination} \\
\hline
\vdots \quad \vdots \\
a : A \\ f : A \rightarrow \sigma B \\
\hline
f(a) : B \\
\end{array}
\]

Abstraction : Implication Introduction

\[
\begin{array}{c}
\text{Abstraction : Implication Introduction} \\
\hline
[x : A]^1 \\
\vdots \\
f : B \\
\hline
\lambda x.f : A \rightarrow \sigma B \\
\end{array}
\]

Pairwise Conjunction

Substitution : Elimination

\[
\begin{array}{c}
\text{Pairwise Conjunction} \\
\hline
[x : A]^1 \\ [y : B]^2 \\
\vdots \\
a : A \otimes B \\
\hline
\text{Substitution : Elimination} \\
f : C \\
\otimes \varepsilon, 1, 2 \\
\end{array}
\]

let \(a \times b\) be \(x \times y\) in \(f\)  \(\Rightarrow_\beta f[a/x, b/y]\)

Beta reduction for let:

\[
\begin{array}{c}
\text{let } a \times b \text{ be } x \times y \text{ in } f \Rightarrow_\beta f[a/x, b/y] \\
\end{array}
\]
Example: Mary laughed

1. \( \text{mary} : \uparrow_{\sigma_e} \)  
2. \( \text{laugh} : (\uparrow \text{SUBJ})_{\sigma_e} \xrightarrow{\sigma} \uparrow_{\sigma_t} \)

1'. \( \text{mary} : g_{\sigma_e} \)

2'. \( \text{laugh} : g_{\sigma_e} \xrightarrow{\sigma} f_{\sigma_t} \)

1''. \( \text{mary} : m \)

2''. \( \text{laugh} : m \xrightarrow{\sigma} l \)

\[
\begin{align*}
\text{Proof} \\
1. \text{mary} : m & \quad \text{Lex. Mary} \\
2. \text{laugh} : m \xrightarrow{\sigma} l & \quad \text{Lex. laughed} \\
3. \text{laugh(mary)} : l & \quad E \xrightarrow{\sigma}, 1, 2
\end{align*}
\]

\[
\begin{align*}
\text{Proof} \\
\text{mary} : m \quad \text{laugh} : m \xrightarrow{\sigma} l \quad \equiv \quad \\
\text{laugh(mary)} : l
\end{align*}
\]

\[
\begin{array}{c}
PRED \ \text{‘laugh(SUBJ)’} \\
\text{SUBJ} \quad g[PRED \ \text{‘Mary’}] \\
\end{array}
\]

\[
\begin{array}{c}
\text{mary} : m \\
\text{laugh} : m \xrightarrow{\sigma} l \\
\end{array}
\]

\[
\begin{array}{c}
\text{laugh(mary)} : l
\end{array}
\]
Example: *Most presidents speak*

1. $\lambda R \lambda S.\text{most}(R, S) : (v \rightarrow r) \rightarrow \forall X.[(p \rightarrow X) \rightarrow X]$  
   Lex. *most*

2. $\text{president}^* : v \rightarrow r$  
   Lex. *presidents*

3. $\text{speak} : p \rightarrow s$  
   Lex. *speak*

\[
\begin{align*}
\lambda R \lambda S.\text{most}(R, S) : & \quad \text{president}^* : \\
(v \rightarrow r) \rightarrow \forall X. [(p \rightarrow X) \rightarrow X] & \quad v \rightarrow r
\end{align*}
\]

\[
\begin{align*}
\lambda S.\text{most}(\text{president}^*, S) : & \quad \text{speak} : \\
\forall X. [(p \rightarrow X) \rightarrow X] & \quad p \rightarrow s
\end{align*}
\]

\[
\begin{align*}
\text{most}(\text{president}^*, \text{speak}) : s & \quad \neg \varepsilon, [s/X]
\end{align*}
\]
Example:

**Most presidents speak at least one language**

```
[ PRED 'Speak⟨SUBJ, OBJ⟩' ]
[ SUBJ ]
[ PRED 'president' ]
[ SPEC [ PRED 'most' ] ]
[ OBJ ]
[ PRED 'language' ]
[ SPEC [ PRED 'at-least-one' ] ]
```

1. $\lambda R \lambda S. most(R, S) :$
   
   $\lambda v1 \rightarrow r1 \rightarrow \forall X . [(p \rightarrow X) \rightarrow X]$
   
2. `president* : v1 \rightarrow r1`
3. `speak : p \rightarrow l \rightarrow s`
4. $\lambda P \lambda Q. at-least-one(P, Q) :$
   
   $\lambda v2 \rightarrow r2 \rightarrow \forall Y . [(l \rightarrow Y) \rightarrow Y]$
5. `language : v2 \rightarrow r2`

**Lex. most**

**Lex. presidents**

**Lex. speak**

**Lex. at least one**

**Lex. language**

Single parse

⇒

Multiple scope possibilities
(Underspecification through quantification)
Most presidents speak at least one language

Subject wide scope
Most presidents speak at least one language
Object wide scope
Anaphora in Glue Semantics

• Variable-free: pronouns are functions on their antecedents (Jacobson 1999, among others)

• Commutative logic of composition allows pronouns to compose directly with their antecedents.

• No need for otherwise unmotivated additional type shifting (e.g. Jacobson’s z-shift)
Anaphora in Glue Semantics

1. Joe said he bowls.

- Pronominal meaning constructor:

\[ \lambda z. z \times z : A \rightarrow (A \otimes P) \]

\begin{align*}
\text{joe} & : \lambda z. z \times z : \\
\text{j} & : j \rightarrow (j \otimes p) \\
\text{joe \times joe} & : j \otimes p \\
\text{let joe \times joe be} & \text{x \times y in } say(x, bowl(y)) : s \\
\Rightarrow & \beta \\
\text{say}(joe, bowl(joe)) & : s
\end{align*}
Further Points of Interest

• Glue Semantics can be understood as a *representationalist* theory, picking up on a theme from Wednesday’s semantics workshop.

• Proofs can be reasoned about as representations (Asudeh & Crouch 2002a,b).

• Proofs have strong identity criteria: normalization, comparison.

• Glue Semantics allows recovery of a non-representationalist notion of *direct compositionality* (Asudeh 2005, 2006).

⇒ Flexible framework with lots of scope for exploration of questions of compositionality and semantic representation.