Categorial Grammar

- Basic Categorial Grammar:
  – core "5" rules
- Semantics in Categorical Grammar
- Categorial Unification Grammar
  – and how it can be used in other formalisms

Categorial Grammar

- Simplfy the rules
- Move complexity from rules to lexical entries
- More tightly coupled with semantics:
  – particularly lambda calculus
- One to one relationship from
  – syntactic and semantic constituents

5 rules

- application:
  - Forward: A/B + B = A
  - Backward: B + A\B = A
- composition:
  - A/B + B/C = A/C
- coordination:
  - A CONJ A' = A"
- type raising:
  - A = X/(X\A)

```
John = np
Mary = np
likes = (s\np)/np

Forward application
 X/Y Y => X
Backward application
 Y X\Y => X

Thus
    John    likes    Mary
     np   (s\np)/np)  np
             ---------- Forward
                s\np
      ----------- Backward
            s
```

```
a,the np/n
old n/n
in  (np\np)/np
man,ball,park n
kicked (s\np)/np

the  old man kicked    a    ball    in     the   park
np/n n/n n  (s\np)/np np/n   n  (np\np)/np np/n  n
     -----             ------                -----
       n                 np                    np
  ------                            -----------
    np                                np\np
                        -----------------
                                np
               ----------------
                     s\np
    -------------------
             s
```

Conjunction

```
    X CONJ X' => X''

John likes     Mary and   dislikes   Bob
np   (s\np)/np np   conj (s\np)/np   np
       ---------FA          ---------FA
           s\np                 s\np
            -----------------------CONJ
                     s\np
-----------------------BA
          s
```

Coordination

- Constituent Coordination
  - John and Mary like books
  - (NP and NP) VP
  - John likes fishing and dislikes baseball.
  - NP (VP and VP)
- Non-constituent coordination
  - John likes and Mary dislikes sport.
  - ???

Type raising

```
  X => Y/(Y\X)

John     likes       and   Mary dislikes   Bob
np      (s\np)/np   conj   np  (s\np)/np  np
---T                       ---T
s/(s\np)                  s/(s\np)
  --------FC                --------FC
   s/np                      s/np
    ----------------------------  CONJ
               s/np
                --------------------------
                                s
```

Type Raising

- Computationally unbounded:
  - could happen for any category
  - Makes parsing intractable

- Controlled type rasing
  - needs to be guarded
  - only (some) lexical items

Categorial Grammar Semantics

- Each consituent mapped to lambda expression:
  - *Mary* $\rightarrow$ m
  - *walks* $\rightarrow$ lambda X walks(X)

Lambda Calculus

- Syntax (of LC):
  - First Order Predicate logic, plus
  - lambda VAR TERM
  - where VAR scopes over all occurrences of VAR in TERM
  - lambda X walk(X)
- Lambda application:
  - lambda X walks(X) composed with m =
  - lambda X walks(X) . m
  - walks(m) by beta-reduction

```
A/B:S + B:T = A:S.T
B:T + A\B:S = A:S.T

John np:j
walks (s\np):lambda X walks(X)

John            walks
np:j  s\np:lambda X walks(X)
  -----------------
     s : walks(j)

B:T + A\B:S = A:S . T
np:j + s\np:lambda X walks(X)
s : lambda X walks(X) . j
s : walks(j)
```

```
John np:j
Mary np:m
likes (s\np)/np: lambda Y lambda X likes(X,Y)

John                    likes                       Mary
np:j  (s\np)/np: lambda Y lambda X likes(X,Y)   m
                         --------------------------
                        s\np: lambda X likes(X,m)
  -------------------------------
         s likes(j,m)

lambda Y lambda X likes(X,Y) . m
lambda X likes(X,m)

lambda X likes(X,m) . j
likes(j,m)
```

```
Conj
   X:A CONJ X':A' = X'':lambda S (A . S & A'. S)
Composition
   X/Y:A Y/Z:B =>  X/Z: lambda Q ( A . (B . Q))
Type raising
   NP:a -> T/(T\NP): lambda R (R . a)

John     likes       and   Mary dislikes   Bob
np      (s\np)/np   conj   np  (s\np)/np   np
---T                      ---T
s/(s\np)                 s/(s\np)
  --------FC                --------FC
   s/np                      s/np
    ----------------------------  CONJ
               s/np
                ----------------------------
                             s
```

```
Conj
   X:A CONJ X':A' = X'':lambda S (A . S & A'. S)
Composition
   X/Y:A Y/Z:B =>  X/Z: lambda Q ( A . (B . Q))
Type raising
   NP:a -> T/(T\NP): lambda R (R . a)

John                    likes                           ...
np:j    (s\np)/np:lambda Y lambda X likes(X,Y)
---T
s/(s\np): lambda R (R . j)
  --------------------------FC
s/np:
 lambda Q (        A           . (         B                    . Q))
 lambda Q ((lambda R (R . j)) . (lambda Y lambda X likes(X,Y) . Q))
 lambda Q ((lambda R (R . j)) . (      lambda X likes(X,Q)       ))
 lambda Q ( lambda X likes(X,Q) . j )
 lambda Q ( likes(j,Q) )
```

```
Conj
   X:A CONJ X':A' = X'':lambda S (A . S & A'. S)
Composition
   X/Y:A Y/Z:B =>  X/Z: lambda Q ( A . (B . Q))
Type raising
   NP:a -> T/(T\NP): lambda R (R . a)

 ...   Mary     dislikes ...
      np:m  (s\np)/np:lambda Y lambda X dislikes(X,Y)
      ---T
     s/(s\np): lambda R (R . m)
       --------------------------FC
s/np:
lambda X (        A           . (             B                    . Q))
lambda Q ((lambda R (R . m)) . (lambda Y lambda X dislikes(X,Y) . Q))
lambda Q ((lambda R (R . m)) . (      lambda X dislikes(X,Q)       ))
lambda Q ( lambda X dislikes(X,Q) . m )
lambda Q ( dislikes(m,Q) )
```

```
Conj
   X:A CONJ X':A' = X'':lambda S (A . S & A'. S)
Composition
   X/Y:A Y/Z:B =>  X/Z: lambda Q ( A . (B . Q))
Type raising
   NP:a -> T/(T\NP): lambda R (R . a)

John     likes       and   Mary dislikes   Bob
      ....           CONJ      ....        np:b
   --------FC                 --------FC
s/np: lambda Q (likes(j,Q))
                    s/np: lambda Q ( dislikes(m,Q) )
        ----------------------------  CONJ
      s/np: lambda S ( lambda Q (likes(j,Q)) . S &
                       lambda Q (dislikes(m,Q)) . S )
      s/np: lambda S ( likes(j,S) &
                       dislikes(m,S) )
                      ------------------------
          s : lambda S ( likes(j,S) & dislikes(m,S) ) . b
          s : likes(j,b) & dislikes(m,b)
```

Compositionality and Incrementality

- Compositionality:
  - all constituents have a denotation
- Incrementality:
  - all initial substrings have a denotation
  - all substrings have a denotation (stronger)

Categorical Unification Grammar

- Extending the formalism to allow features:
  – agreement, grammatical relations
- Embedding CCG techniques in other formalisms
  – SUBCAT, predicate/arguments

```
the np/n
boy n
boys n
walk s\np
walks s\np

Forward application
 X/Y Y => X
Backward application
 Y X\Y => X

Thus
    the        boy    walks
    np/n        n      s\np
     ----------- FA
          np
           ------------- BA
                 s
```

```
the   [cat: np]/[cat: n]
boy   [cat: n]
boys  [cat: n]
walk  [cat: s]\[cat: np]
walks [cat: s]\[cat: np]

Forward application
 X/Y Y => X
Backward application
 Y X\Y => X

Thus
    the              boy         walks
[cat: np]/[cat: n]  [cat: n]   [cat: s]\[cat: np]
     ----------------- FA
          [cat: np]
               ----------------- BA
                    [cat: s]
```

```
the   [cat: np num: !X]/
       [cat: n num: !X]
boy   [cat: n num: sg]
boys  [cat: n num: pl]
walk  [cat: s]\
       [cat: np num: pl]
walks [cat: s]\
       [cat: np num: sg]
```

Thus

```
    the              boy         walks
   [cat: np         [cat: n     [cat: s]\
    num: !X]/        num: sg]    [cat: np
    [cat: n                       num: sg]
     num: !X]
     ----------------- FA
          [cat: np
           num: sg]
              ----------------- BA
                   [cat: s]
```

```
the   [cat: np num: !X]/
       [cat: n num: !X]
boy   [cat: n num: sg]
boys  [cat: n num: pl]
walk  [cat: s]\
       [cat: np num: pl]
walks [cat: s]\
       [cat: np num: sg]
```

Thus

```
    the              boys       walk
   [cat: np         [cat: n     [cat: s]\
    num: !X]/        num: pl]    [cat: np
    [cat: n                       num: pl]
     num: !X]
     ----------------- FA
          [cat: np
           num: pl]
              ----------------- BA
                   [cat: s]
```

```
the   [cat: np num: !X]/
       [cat: n num: !X]
boy   [cat: n num: sg]
boys  [cat: n num: pl]
walk  [cat: s]\
       [cat: np num: pl]
walks [cat: s]\
       [cat: np num: sg]
```

Thus

```
    the              boys       walks
   [cat: np         [cat: n     [cat: s]\
    num: !X]/        num: pl]    [cat: np
    [cat: n                       num: sg]
     num: !X]
     ----------------- FA
          [cat: np
           num: pl]
                -------------------- ****
```

```
the   [becomes [cat: np num: !X]
       direction: forward
       wants: [cat: n num: !X]]
boy   [cat: n num: sg]
walks [becomes: [cat: s]
       direction: backward
       wants: [cat: np num: sg]]

Forward Apllication
   [becomes: !X
    direction: forward   +  !Y   =  !X
    wants: !Y]
Backward Apllication
           [becomes: !X
    !Y  +   direction: backward  =  !X
            wants: !Y]
```

```
the   [becomes [cat: np num: !X]
       direction: forward
       wants: [cat: n num: !X]]
boy   [cat: n num: sg]
walks [becomes: [cat: s]
       direction: backward
       wants: [cat: np num: sg]]
```

Thus

```
    the                        boy        walks
 [becomes [cat: np num: !X]                 ...
  direction: forward       [cat: n num: sg]
  wants: [cat: n num: !X]]
     ------------------------ FA
          [cat: np
           num: sg]
              ---------------------------- BA
                   [cat: s]
```

SUBCAT feature

- In GPSG and HPSG:
  - SUBCAT feature identifies features of argument(s)
  - [SUBCAT [NP]]
- This is actually CG-like:
  - S\NP is a verb looking for one argument
  - (S\NP)/NP is a verb looking for two arguments
- Typing the arguments:
  - number/case etc

CG Parsing

- CG:

forward application	A/B + B = A
backward application	B + A\B = A
composition	A/B + B/C = A/C
conjunctions	A CONJ A' = A"

- Simple bottom-up algorithm:
  - basic chart technique
  - Insert lexical entries in chart
  - **do**
  - **for each** v vertice
  - **for each** e1 edge from v
  - **for each** e2 edge from e1's end
  - **if** e1 + e2 match rule
  - apply and insert new edge
  - **until** new new edges inserted

CCG Parsing

forward application	A/B + B = A
backward application	B + A\B = A
composition	A/B + B/C = A/C
conjunctions	A CONJ A' = A"
type raising	A = X/(X\A)

- Simple bottom-up algorithm:
  - basic chart technique
  - Insert lexical entries in chart
  - apply restricted type raising
  - **do**
  - **for each** v vertice
  - **for each** e1 edge from v
  - **for each** e2 edge from e1's end
  - **if** e1 + e2 match rule
  - apply and insert new edge
  - **until** new new edges inserted

Some other examples

- gapping:
  - Harry will buy bread and Barry, potatoes

Some Properties

- mildly context sensitive
- weakly equvalent to LTAG:
  - Lexicalized Tree Adjoining Grammars
- Relationship to intonation:
  - Steedman 1991
- Lambek calculus
  - change argument order
- Complexity:
  - unrestricted type raising: unbounded
  - restricted type raising: $O(n^3)$
  - on CCG minus CONJ which can be (S\S)/S