Combinatory Categorial Grammar (CCG)

11-711 Algorithms for NLP

8 November 2018

(With thanks to Alan W Black)

Goals of CCG

- Simplify the (*combinatory*) rules
- Move complexity from rules to (*categorial*) lexical entries
- More tightly couple with semantics, particularly lambda calculus
- One-to-one relationship between syntactic and semantic constituents

Five (5) rules!

- Application:
  - Forward: A/B + B = A
  - Backward: B + A\B = A
- Composition:
  - A/B + B/C = A/C
- Coordination:
  - X CONJ X’ = X”
- Type raising:
  - A = X/(X\A)

John = np *(an argument category)*

Mary = np

likes = (s\np)/np *(a functor category)*

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							

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
```

```
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
```

```
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
```

Handling 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 sports.

    ???

Coordination

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**

Coordination

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

Type Raising

TR: X => Y/(Y\X)

COMP: A/B + B/C => A/C

John	likes	and	**Mary**	dislikes	Bob
np	(s\np)/np	conj	**np**	(s\np)/np	np
------TR			**-------TR**		
s/(s\np)			**s/(s\np)**		

Type Raising

TR: X => Y/(Y\X)

COMP: A/B + B/C => A/C

```
John likes        and      Mary dislikes      Bob
np     (s\np)/np conj      np       (s\np)/np   np
------TR                  -------TR
s/(s\np)                   s/(s\np)
----------------COMP ----------------COMP
        s/np                        s/np
```

Type Raising

TR: X => Y/(Y\X)

COMP: A/B + B/C => A/C

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

Type Raising

TR: X => Y/(Y\X)

COMP: A/B + B/C => A/C

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

Type Raising

- Computationally unbounded:
  - could happen for any category
  - makes parsing intractable
- Controlled type raising
  - needs to be *guarded*
  - only allowed for some lexical items

CCG Semantics

- Remember goals:
  - More tightly couple with semantics, particularly lambda calculus
  - One-to-one relationship between syntactic and semantic constituents
- Add semantics to CCG rules
  - Lambda calculus (again)
- “Montague semantics”

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

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

John np:j

walks s\np: λX.walks(X)

John walks

np:j s\np: λX.walks(X)

----

s : walks(j)

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

np:j + s\np: λX.walks(X)

s : λX.walks(X) . j

s : walks(j)

$A/B:S + B:T = A:S.T$

$B:T + A\backslash B:S = A:S.T$

John np:j

walks s\np: λX.walks(X)

John walks

np:j s\np: $\lambda X.walks(X)$

----

s : walks(j)

$B:T + A\backslash B:S = A:S\ .\ T$

$np:j + s\backslash np: \lambda X.walks(X)$

$s : \lambda X.walks(X)\ .\ j$

$s : walks(j)$

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

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

John np:j

walks s\np: λX.walks(X)

John **walks**

np:j **s\np: λX.walks(X)**

----

s : walks(j)

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

np:j + s\np: λX.walks(X)

s : λX.walks(X) . j

s : walks(j)

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

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

John   np:j

walks   s\np: λX.walks(X)

John   walks

np:j   s\np: λX.walks(X)

----

s : walks(j)

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

np:j + s\np: λX.walks(X)

s : λX.walks(X) . j

s : walks(j)

John   np:j

Mary   np:m

likes (s\np)/np: λY.λX.likes(X,Y)

John   likes   Mary

np:j   (s\np)/np:λY.λX.likes(X,Y)   m

----

s\np: λX.likes(X,m)

----

s likes(j,m)

λY.λX.likes(X,Y) . m

λX.likes(X,m)

λX.likes(X,m) . j

likes(j,m)

John np:j

Mary np:m

likes (s\np)/np: λY.λX.likes(X,Y)

John	**likes**	Mary
np:j	**(s\np)/np:λY.λX.likes(X,Y)**	m
	--	
	s\np: λX.likes(X,m)	
--		
	s likes(j,m)	

λY.λX.likes(X,Y) . m

λX.likes(X,m)

λX.likes(X,m) . j

likes(j,m)

John   np:j

Mary   np:m

likes (s\np)/np: λY.λX.likes(X,Y)

John   **likes**   Mary

np:j   (s\np)/np:λY.λX.likes(X,Y) m

----

s\np: λX.likes(X,m)

---

s likes(j,m)

λY.λX.likes(X,Y) . m

λX.likes(X,m)

λX.likes(X,m) . j

likes(j,m)

Coordination:

X:A **CONJ** X':A' = X'': λ**S.(**A . **S &** A'. **S)**

Composition:

X/Y:A Y/Z:B => X/Z: λQ.(A . (B . Q))

Type raising:

NP:a -> T/(T\NP): λR.(R . a)

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

Coordination:

X:A CONJ X’:A’ = X’’: λS.(A . S & A’. S)

Composition:

X/Y:A Y/Z:B => X/Z: **λQ.**(A . (B . **Q**))

Type raising:

NP:a -> T/(T\NP): λR.(R . a)

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

Coordination:

X:A CONJ X’:A’ = X’’: λS.(A . S & A’. S)

Composition:

X/Y:A Y/Z:B => X/Z: λQ.(A . (B . Q))

Type raising:

NP:a -> T/(T\NP): **λR.(R** . a)

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

Coordination:

X:A CONJ X':A' = X'': λS.(A . S & A'. S)

Composition:

X/Y:A Y/Z:B => X/Z: **λQ.(A . (B . Q))**

Type raising:

NP:a -> T/(T\NP): **λR.(R . a)**

John likes ...

np:**j** (s\np)/np: **λY.λX.likes(X,Y)**

---**TR**

s/(s\np): **λR.(R . j)**

---------------------------**COMP**

s/np:

λQ.(A . (B . Q))

λQ.((λR.(R . j)) . (**λY.λX.likes(X,Y)** . Q))

λQ.((**λR.(R . j)**) . (λX.likes(X,Q)))

λQ.(λX.likes(X,Q) . j)

λQ.(likes(j,Q))

Coordination:

X:A CONJ X':A' = X'': λS.(A . S & A'. S)

Composition:

X/Y:A Y/Z:B => X/Z: λQ.(A . (B . Q))

Type raising:

NP:a -> T/(T\NP): λR.(R . a)

... Mary dislikes ...

np:m (s\np)/np:λY.λX.dislikes(X,Y)

---TR

s/(s\np): λR.(R . m)

-------------------------COMP

s/np:

λQ.(A   . (B   . Q))

λQ.((λR.(R . m)) . (**λY.λX.dislikes(X,Y)** . Q))

λQ.((**λR.(R . m)**) . (λX.dislikes(X,Q)))

λQ.(λX.dislikes(X,Q) . m)

λQ.(dislikes(m,Q))

Coordination:

X:A CONJ X':A' = X'': **<u>λS.(A . S & A'. S)</u>**

Composition:

X/Y:A Y/Z:B => X/Z: λQ.(A . (B . Q))

Type raising:

NP:a -> T/(T\NP): λR.(R . a)

John likes and Mary dislikes Bob

.... CONJ .... np:b

--------COMP --------COMP

s/np: **λQ.(likes(j,Q))**

s/np: **λQ.(dislikes(m,Q))**

--------------------------- CONJ

s/np: λS.(**λQ.(likes(j,Q))** . S &

λQ.(dislikes(m,Q)) . S)

s/np: λS.(likes(j,S) &

dislikes(m,S))

----------------------COMP

s: **<u>λS.(likes(j,S) & dislikes(m,S)) </u>** . 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**]

```
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]

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]
```

```
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]
           ---------------- *****
```

SUBCAT Feature

- In GPSG and HPSG
  - SUBCAT feature identifies features of arguments:

    [SUBCAT [NP]]   *like +np feature in previous lecture*
- This is actually CCG-like
  - S\NP is verb looking for one argument
  - (S\NP)/NP is verb looking for two arguments
- Can be extended to full SUBCAT feature
  - required PPs, VCOMP, etc.

Some properties

- Mildly context sensitive
- Weakly equivalent to LTAG (Lexicalized TAG)
- Derived/gapped categories prevent non-projective parses
- Complexity:
  - unrestricted type raising: unbounded
  - restricted type raising: $O(n^3)$
    - *(also without general Coordination)*

Some other points of interest

- Free word order languages: “|”
- Relationship to intonation (Steedman 1991)
- Extension to Lambek calculus to allow changes in argument order and real incr. processing
- CCGBank: Julia Hockenmaier and Steedman
  - CCG version of Penn Treebank
- PCCG: Luke Zettlemoyer and Collins
  - Learn to produce logical form statistically

Sentence 2

```
{S[dcl] {S[dcl] {NP {N {N/N  Mr.}
                  {N  Vinken}}}
         {S[dcl]\NP {(S[dcl]\NP)/NP  is}
                    {NP {NP {N  chairman}}
                        {NP\NP {(NP\NP)/NP  of}
                               {NP {NP {N {N/N  Elsevier}
                                          {N  N.V.}}}
                                   {NP[conj] {,  ,}
                                             {NP {NP[nb]/N  the}
                                                 {N {N/N  Dutch}
                                                    {N {N/N  publishing}
                                                       {N  group}}}}}}}}}}}
  {.  .}}
```

Mr.	**(N/N)**	Vinken	
is	**((S[dcl]\NP)/NP)**	Vinken	chairman
of	**((NP\NP)/NP)**	chairman	N.V., group
Elsevier	**(N/N)**	N.V.	
the	**(NP[nb]/N)**	group	
Dutch	**(N/N)**	group	
publishing	**(N/N)**	group	

Learning a PCCG

- See slides and videos courtesy of Yoav Artzi, Nicholas FitzGerald and Luke Zettlemoyer
- http://yoavartzi.com/tutorial/