Parsing with CCG

- Lecture 6-

Syntactic formalisms for natural language parsing

FI MU autumn 2011

Categorial Grammar is

: a lexicalized theory of grammar along with other theories of grammar such as HPSG, TAG, LFG, . . .

: linguistically and computationally attractive

→ language invariant combination rules, high efficient parsing

Outline

1. **A-B categorial system**
2. **Lambek calculus**
3. **Extended Categorial Grammar**
   - Variation based on Lambek calculus
     - Abstract Categorial Grammar, Categorial Type Logic
   - Variation based on Combinatory Logic
     - Combinatory Categorial Grammar (CCG)
     - Multi-modal Combinatory Categorial Grammar

Main idea in CG and *application* operation

- All natural language consists of operators and of operands.
  - Operator (functor) and operand (argument)
  - Application: (operator(operand))
  - Categorial type: typed operator and operand

1. A-B categorial system

The product of the directional adaptation by Bar-Hillel (1953) of Ajdukiewicz's calculus of syntactic connection (Ajdukiewicz, 1935)

Definition 1 (AB categories).

Given A, a finite set of atomic categories, the set of categories C is the smallest set such that:

- $A \subseteq C$
- $(X\backslash Y), (X/Y) \in C$ *if* $X, Y \in C$

- **Categories** (type): primitive categories and derivative categories
    - Primitive: S for sentence, N for nominal phrase
    - Derivative: S/N, N/N, (S\N)/N, NN/N, S/S...

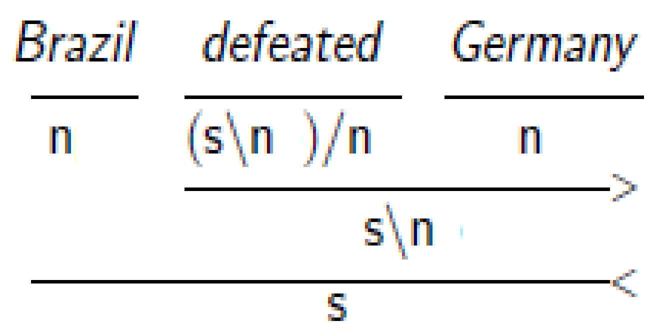
- Forward(>) and backward (<) **functional application**

    a. $X/Y \; Y \Rightarrow X$ (>)

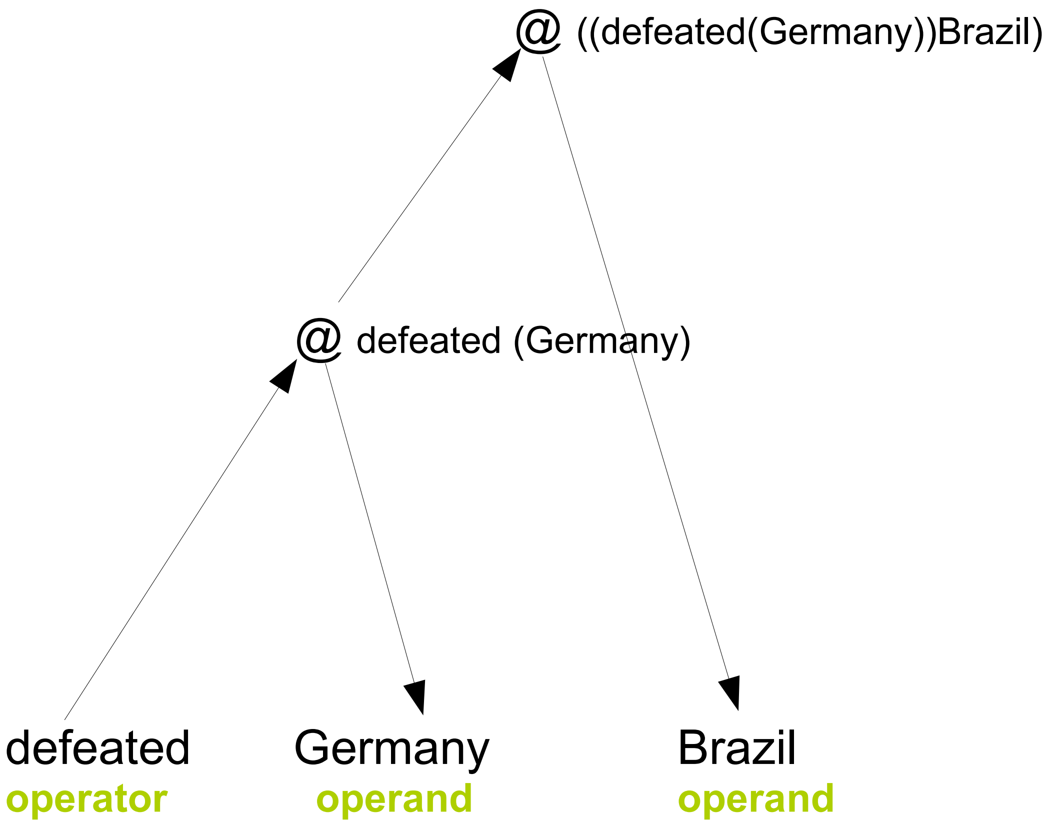
    b. $Y \; X\backslash Y \Rightarrow X$ (<)

- **Calculus on types** in CG are analogue to **arithmetic subtraction**

x/y x → y ≈ 2/4 * 2 = 4



Applicative tree of *Brazil defeated Germany*



Limitation of AB system

1. Relative construction

a. team$_i$ that t_i defeated Germany

b. team$_i$ that Brazil defeated t_i

a′. that (n\n)/(s\n)

b′. that (n\n)/(s/n)

team [that]$_{(n\backslash n)/(s\backslash n)}$ [defeated Germany]$_{s\backslash n}$

team [that]$_{(n\backslash n)/(s/n)}$ [Brazil defeated]$_{s/n}$

team	that	Brazil	defeated
	(n\n)/(s/n)	n	(s\n)/n
		←	**(?)**

2. Agrammatical sentence considered as well-formed structure

**a man good*

n/n n n\n

←

n : ((good)man)

→

n : (a((good)man))

a good man

n/n n\n n

→

n : ((good)man)

→

n : (a((good)man))

3. Many others complex phenomena

- Coordination
- Object extraction, unbounded dependencies,...

4. AB's generative power is too weak.

2. Lambek calculus (Lambek, 1958, 1961)

- on the calculus of syntactic types

The axioms of Lambek calculus are the following:

1 . $x \to x$

2 . $(xy)z \to x(yz) \to (xy)z$ (the axioms 1, 2 with inference rules, 3, 4, 5)

3 . If $xy \to z$ then $x \to z/y$, if $xy \to z$ then $y \to x\backslash z$;

4 . If $x \to z/y$ then $xy \to z$, if $y \to x\backslash z$ then $xy \to z$;

5 . If $x \to y$ and $y \to z$ then $x \to z$.

The rules obtained from the previous axioms are the following:

1 . Hypothesis: if x and y are types, then x/y and y\x are types.

2 . Application rules : (x/y)y → x, y (y\x) → x

ex: *Poor John works.*

3 . Associativity rule : (x\y)/z ↔ x\(y/z)

ex: *John likes Jane.*

4. Composition rules : (x/y)(y/z) → x/z, (x\y)(y\z) → x\z

ex: *He likes him.*

s/(n\s) n\s/n

5. Type-raising rules : x → y/(x/y), x → (y/x)\y

3. Combinatory Categorial Grammar

- Developed originally by M. Steedman (1988, 1990, 2000, ...)
- Combinatory Categorial Grammar (CCG) is a grammar formalism equivalent to Tree Adjoining Grammar, i.e.
  - it is lexicalized
  - it is parsable in polynomial time (See Vijay-Shanker and Weir, 1990)
  - it can capture cross-serial dependencies
- Just like TAG, CCG is used for grammar writing
- CCG is especially suitable for statistical parsing

- several of the **combinators which Curry and Feys** (1958) use to define **the λ-calculus** and applicative systems in general are of considerable syntactic interest (Steedman, 1988)
- The relationships of these combinators to terms of the λ-calculus are defined by the following equivalences (Steedman, 2000b):

a. $\mathbf{B}fg \equiv \lambda x.f(g\ x)$

b. $\mathbf{T}x \equiv \lambda f.f\ x$

c. $\mathbf{S}fg \equiv \lambda x.fx(g\ x)$

CCG categories

- Atomic categories: S, N, NP, PP, TV. . .
- Complex categories are built recursively from atomic categories and slashes

- Example complex categories for verbs:
  - intransitive verb: S\NP *walked*
  - transitive verb: (S\NP)/NP *respected*
  - ditransitive verb: ((S\NP)/NP)/NP *gave*

Lexical categories in CCG

- An elementary syntactic structure – a lexical category – is assigned to each word in a sentence, eg:

  walked: S\NP ‘give me an NP to my left and I return a sentence’

- Think of the lexical category for a verb as a function: NP is the argument, S the result, and the slash indicates the direction of the argument

The typed lexicon item

- The CCG lexicon assigns categories to words, i.e. it specifies which categories a word can have.
- Furthermore, the lexicon specifies the semantic counterpart of the syntactic rules, e.g.:

  love (S\NP)/NP *λxλy.loves'xy*

- Combinatory rules determine what happens with the category and the semantics on combination

- **Attribution of types to lexical items: examples**

Predicate

ex: *is* as an identificator of nominal

as an **operator of predication** from a nominal → (S\NP)/NP

from an adjective → (S\NP)/(N/N)

from an adverb → (S\NP)/(S\NP)\(S\NP)

from a preposition → (S\NP)/((S\NP)\(S\NP)/NP)

ex: verbs → unary (S\NP)

→ binary (S\NP)/NP

→ ternary (S\NP)/NP/NP

Adverbs

Adverb of verb

(S\NP)/(S\NP)

(S\NP)/NP/(S\NP)/NP

Adverb of adverb

(S\NP)/(S\NP)/(S\NP)/(S\NP)

(S\NP)/NP/(S\NP)/NP/(S\NP)/NP/(S\NP)/NP

Adverb of adjective

(N/N)/(N/N)

(N\N)/(N\N)

Adverb of proposition

S/S

Adverb: operator of determination of type (X/X)

Preposition

Prep. 1:

constructor of adverbial phrase

(S\NP)\(S\NP)/NP

(S/S)/NP

(S/S)/N

Prep. 2:

constructor of adjectival phrase

(N\N)/NP

(N\N)/N

Preposition: constructor of determination of type (X/X)

Dictionary of typed words

Syntactic categories	Syntactic types	Lexical entries
Nom.	N	*Olivia, apple...*
Completed nom.	NP	*an apple, the school*
Pron.	NP	*She, he...*
Adj.	(N/N), (N\N)	***pretty** woman,...*
Adv.	(N/N)/(N/N), (S\NP)\(S\NP)...	***very** delicious,...*
Vb	(S\NP), (S\NP)/NP...	*run, give...*
Prep.	(S\NP)\(S\NP)/NP (NP\NP)/NP...	*run **in** the park, book **of** John,...*
Relative	(S\NP)/S...	*I believe that...*

Combinatorial categorial rules

- Functional application (>,<)
- Functional composition (>**B**, <**B**)
- Type-raising (<**T**, >**T**)
- Distribution (<**S**, >**S**)
- Coordination (<**Φ**, >**Φ**)

Functional application (FA)

X/Y:*f* Y:*a* ⇒ X:*fa* (forward functional application, **>**)

Y:*a* X\Y:*f* ⇒ X:*fa* (backward functional application, **<**)

- Combine a function with its argument:

NP S\NP
S
Mary sleeps → (sleeps (Mary))

NP (S\NP)/NP NP
S\NP → (likes (Mary))
S
John likes Mary → ((likes (Mary))John)

- Direction of the slash indicates position of the argument with respect to the function

Derivation in CCG

- The combinatorial rule used in each derivation step is usually indicated on the right of the derivation line
- Note especially what happens with the semantic information

$$
\dfrac{\mathit{John}}{NP : John'} \quad \dfrac{\mathit{loves}}{(S\backslash NP)/NP : \lambda x \lambda y.loves'xy} \quad \dfrac{\mathit{Mary}}{NP : Mary'}
$$

$$
\dfrac{(S\backslash NP)/NP : \lambda x \lambda y.loves'xy \quad NP : Mary'}{S\backslash NP : \lambda y.loves'\, Mary'\, y} >
$$

$$
\dfrac{NP : John' \quad S\backslash NP : \lambda y.loves'\, Mary'\, y}{S : loves'\, Mary'\, John'} <
$$

Function composition (FC)

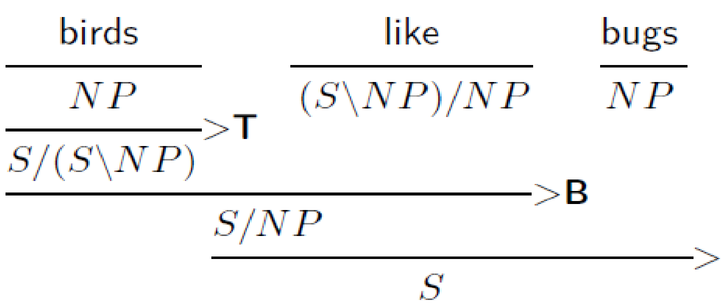
Generalized forward composition (>Bn)

$X/Y{:}f \quad Y/Z{:}g \Rightarrow_{\mathbf{B}} X/Z{:}\lambda x.f(gx) \quad (>\mathbf{B})$

- Functional composition composes two complex categories (two functions):

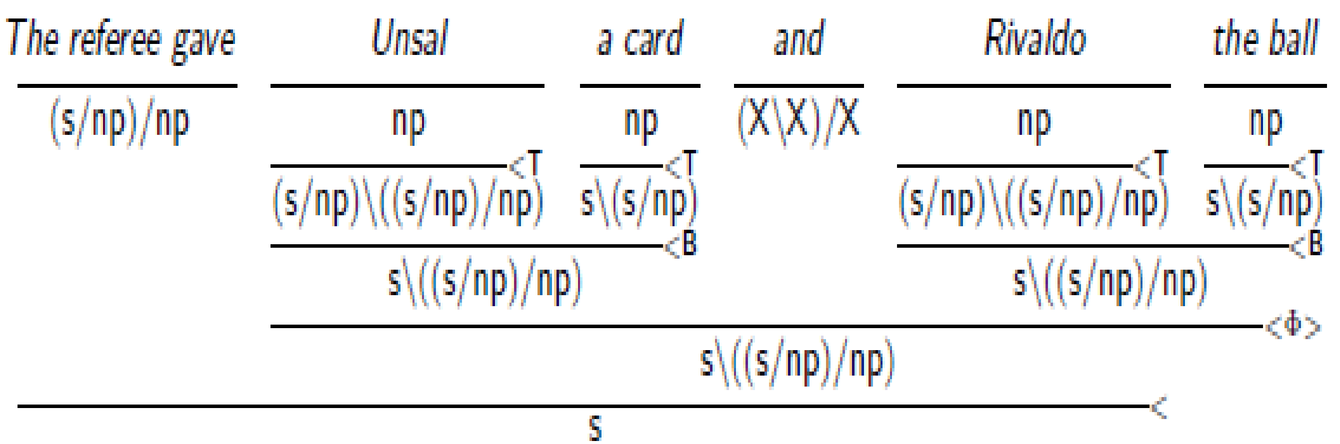
$(S\backslash NP)/PP \quad (PP/NP) \Rightarrow_{\mathbf{B}} (S\backslash NP)/NP$

$S/(S\backslash NP) \quad (S\backslash NP)/NP \Rightarrow_{\mathbf{B}} S/NP$



Generalized backward composition (<Bn)

$$Y\backslash Z{:}f \quad X\backslash Y{:}g \;\Rightarrow_{\mathbf{B}}\; X\backslash Z{:}\lambda x.f(gx) \qquad (<\mathbf{B})$$



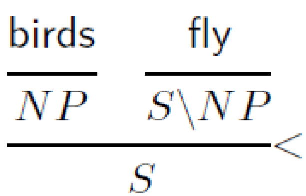
Type-raising (T)

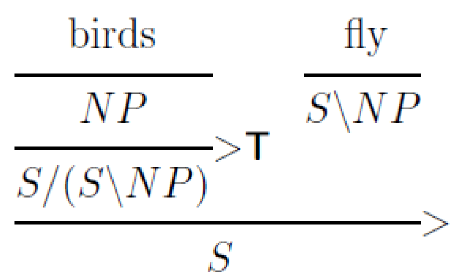
Forward type-raising (>T)

X:a $\Rightarrow$ T/(T\X):*λf.fa* (>**T**)

- Type-raising turns an <u>argument</u> into a <u>function</u> (e.g. for case assignment)

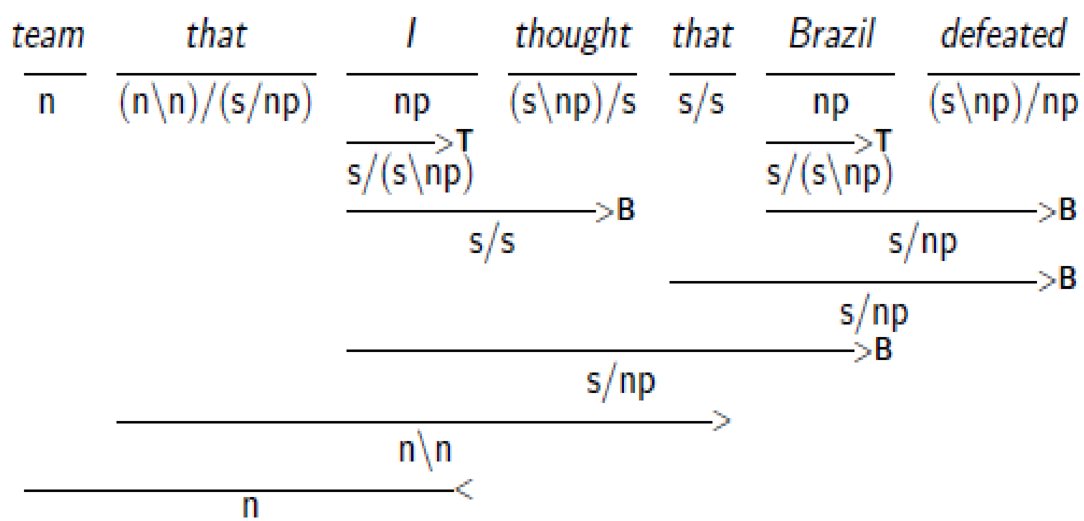
NP $\Rightarrow$ S/(S\NP) (nominative)





- This must be used *e.g.* in the case of WH-movement

Example of <u>functional composition (>B)</u> and <u>type-raising (T)</u>

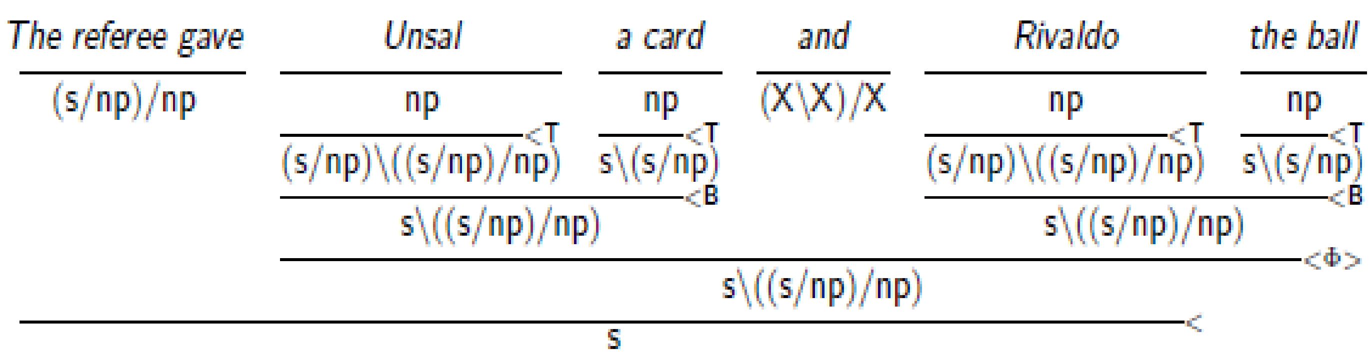


Backward type-raising (<T)

$$X{:}a \quad \Rightarrow \quad T\backslash(T/X){:}\lambda f.fa \qquad (<\mathbf{T})$$

- Type-raising turns an argument into a function (e.g. for case assignment)

$$NP \Rightarrow (S\backslash NP)\backslash((S\backslash NP)/NP) \quad \text{(accusative)}$$



Coordination (&)

$$X \text{ CONJ } X \Rightarrow_{\Phi} X \qquad \text{(Coordination } (\Phi))$$

```
   give          a dog              a bone     and     a policeman        a flower
---------- ------------------<T --------<T ---- ------------------<T --------<T
(VP/NP)/NP (VP/NP)\((VP/NP)/NP) VP\(VP/NP) conj (VP/NP)\((VP/NP)/NP) VP\(VP/NP)
           -------------------------------<B    -------------------------------<B
                    VP\((VP/NP)/NP)                       VP\((VP/NP)/NP)
                    -------------------------------------------------------<&>
                                 VP\((VP/NP)/NP)
-------------------------------------------------<
                      VP
```

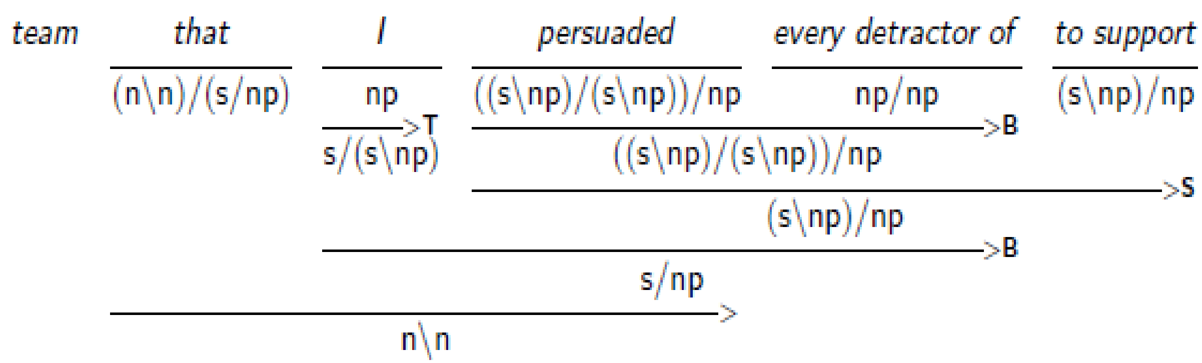
Substitution (S)

Forward substitution (>S)

$$(X/Y)/Z \;\; Y/Z \Rightarrow_S X/Z$$

- Application to parasitic gap such as the following:

  a. team that I persuaded every detractor of to support



Substitution (S)

Backward crossed substitution (<S×)

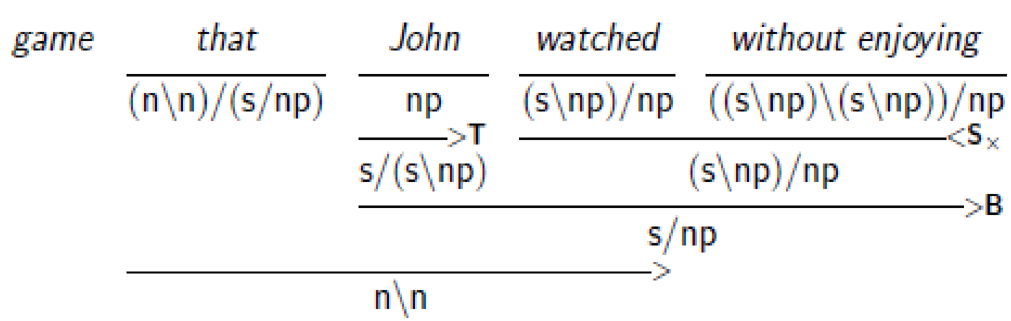
$$Y/Z\ (X\backslash Y)/Z \Rightarrow_S X/Z$$

- Application to parasitic gap such as the following:

  a. John watched without enjoying the game between Germany and Paraguay.

  b. game that John watched without enjoying

game that John [watched]$_{(s\backslash np)/np}$ [without enjoying]$_{((s\backslash np)\backslash(s\backslash np))/np}$



Limit on possible rules

- **The Principle of Adjacency:**

Combinatory rules may only apply to entities which are linguistically realised and adjacent.

- **The Principle of Directional Consistency:**

All syntactic combinatory rules must be consistent with the directionality of the principal function. ex: X\Y Y ≠> X

- **The Principle of Directional Inheritance:**

If the category that results from the application of a combinatory rule is a function category, then the slash defining directionality for a given argument in that category will be the same as the one defining directionality for the corresponding arguments in the input functions. ex: X/Y Y/Z ≠> X\Z.

Semantic in CCG

- CCG offers a syntax-semantics interface.
- The lexical categories are augmented with an explicit identification of their semantic interpretation and the rules of functional application are accordingly expanded with an explicit semantics.
- Every syntactic category and rule has a semantic counterpart.
- The lexicon is used to pair words with syntactic categories and semantic interpretations:

love (S\NP)/NP $\Rightarrow \lambda x \lambda y.loves'xy$

- The semantic interpretation of all combinatory rules is fully determined by the **Principle of Type Transparency**:
  - **Categories**: All syntactic categories reflect the semantic type of the associated logical form.
  - **Rules**: All syntactic combinatory rules are type-transparent versions of one of a small number of semantic operations over functions including application, composition, and type-raising.

proved := (S\NP$_{3s}$)/NP : *λxλy.prove'xy*

- *the semantic type of the reduction is the same as its syntactic type, here functional application.*

$$\dfrac{\dfrac{\text{Marcel}}{NP_{3sm} : marcel'} \quad \dfrac{\dfrac{\text{proved}}{(S\backslash NP_{3s})/NP : \lambda x \lambda y.prove'xy} \quad \dfrac{\text{completeness}}{NP : completeness'}}{S\backslash NP_{3s} : \lambda y.prove'completeness'y}>}{S : prove'completeness'marcel'}<$$

CCG with semantics : *Mary will copy and file without reading these articles*

```
Mary  will     copy       and          file       without                  reading        these  articles.
------------- --------  ----------   -------   -----------             -----------   ---------------------
 S/VP          VP/NP    CONJ         VP/NP    (VP\VP)/VPing       VPing/NP          NP
:p.Mary'   λp.will'    : copy'       : and'    : file'     : λp. λq.without'pq    : read'        : articles'
                                              ------------------------------------> B
                                                        (VP\VP) /NP
                                                  : λx. λq.without'(read' x)q
                                      ----------------------------------------------------< S
                                                     VP/NP
                                               : λx. without'(read' x)(file' x)
                        -----------------------------------------------------------------------------<Φ>
                                              VP/NP
                           : λx. and'(without'(read' x)(file' x))(copy' x)
                        ------------------------------------------------------------------------------------------<
                                           VP
                         : and'(without')(read'articles')(file'articles'))(copy'articles')
                ----------------------------------------------------------------------------------------------->
                                                       S
                        : will'(and'(without')(read'articles')(file'articles'))(copy'articles'))mary'
```

Parsing a sentence in CCG

Step 1: tokenization

Step 2: tagging the concatenated lexicon

Step 3: calculate on types attributed to the concatenated lexicons by applying the adequate combinatorial rules

Step 4: eliminate the applied combinators (we will see how to do on next week)

Step 5: finding the parsing results presented in the form of an operator/operand structure (predicate -argument structure)

Example: ***I requested and would prefer musicals***

STEP 1 : tokenization/lemmatization → ex) POS Tagger, tokenizer, lemmatizer

a. I-requested-and-would-prefer-musicals

b. I-request-ed-and-would-prefer-musical-s

STEP 2 : tagging the concatenated expressions → ex) Supertagger, Inventory of typed words

I	NP
Requested	(S\NP)/NP
And	CONJ
Would	(S\NP)/VP
Prefer	VP/NP
musicals	NP

STEP 3 : categorial calculus

a. apply the type-raising rules → *Subject Type-raising:* ($>\mathbf{T}$)
$$NP : a \Rightarrow T/(T\backslash NP) : \mathbf{T}\,a$$

b. apply the functional composition rules → *Forward Composition:* ($>\mathbf{B}$)
$$X/Y : f \quad Y/Z : g \Rightarrow X/Z : \mathbf{B}\,fg$$

c. apply the coordination rules → *Coordination:* ($< \& >$)
$$X \quad conj \quad X \Rightarrow X$$

	I-	requested-	and-	would-	prefer-	musicals	
1/	NP	(S\NP)/NP	CONJ	(S\NP)/VP	VP/NP	NP	
2/	S/(S\NP)	(S\NP)/NP	CONJ	(S\NP)/VP	VP/NP	NP	(>**T**)
3/	S/(S\NP)	(S\NP)/NP	CONJ	(S\NP)/NP		NP	(>**B**)
4/	S/(S\NP)		(S\NP)/NP			NP	(>**Φ**)
5/	S/(S\NP)		(S\NP)/NP			NP	(>**B**)
6/		S/NP				NP	(>)
7/			**S**				

STEP 4 : semantic representation (predicate-argument structure)

I requested and would prefer musicals

1/ :i' :request' :and' : will' :prefer' : musicals'

2/ :λf.f I'

3/ : λx.λy.will'(prefer'x)y

4/ : λtvλxλy.and'(will'(prefer'x)y))(tv xy)

5/ : λxλy.and'(will'(prefer'x)y)(request'xy)

6/ :λy.and'(would'(prefer' musicals')y)(request' musicals' y)

7/S: and'(will'(prefer' musicals') i')(request' musicals' i')

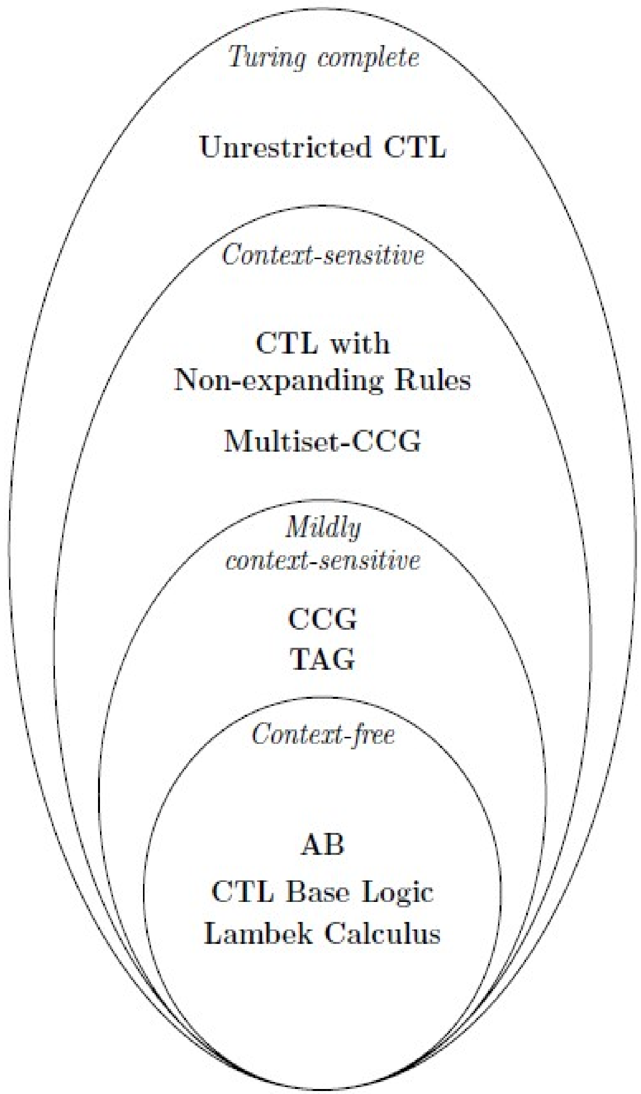
Variation of CCG : Multi-modal CCG (Baldridge, 2002)

- Modalized CCG system
- Combination of Categorial Type Logic (CTL, Morrill, 1994; Moortgat, 1997) into the CCG (Steedman, 2000)
- Rules restrictions by introducing the modalities: *, x, •, ◊
- Modalized functional composition rules

$$(>\mathbf{B}) \quad X/_{\diamond}Y \quad Y/_{\diamond}Z \quad \Rightarrow \quad X/_{\diamond}Z$$
$$(<\mathbf{B}) \quad Y\backslash_{\diamond}Z \quad X\backslash_{\diamond}Y \quad \Rightarrow \quad X\backslash_{\diamond}Z$$

- Invite you to read the paper "*Multi-Modal CCG"* of (Baldridge and M.Kruijff, 2003)

The positions of several formalisms on the Chomsky hierarchy



Classwork

Exercise of *taggings* and of *categorial calculus*

See the given paper!!