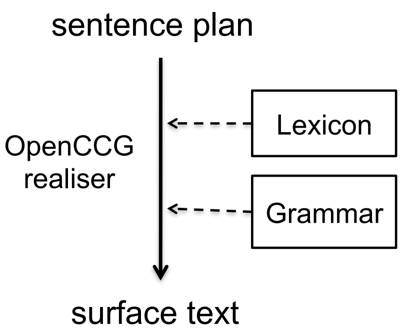
Combinatory Categorial Grammar

Constraining surface realisation in OpenCCG

Recommended Reading

- Michael White. 2006. Efficient Realization of Coordinate Structures in Combinatory Categorial Grammar. *Research on Language and Computation*, 4(1):39–75.
- Mark Steedman and Jason Baldridge. Combinatory Categorial Grammar. To appear in Robert Borsley and Kersti Borjars (eds.) *Constraint-based approaches to grammar: alternatives to transformational syntax*. Oxford: Blackwell. PDF (Will appear in February 2011.)

OpenCCG surface realisation



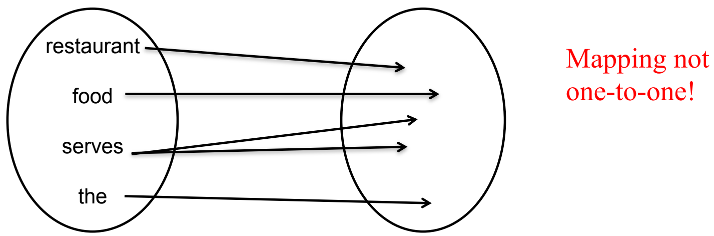
Sentence plans are hybrid logic dependency structures

What do the grammar and lexicon look like?

Categorial Grammar

Categorial grammars are **lexicalised** grammars

- a grammar is just a "dictionary"
- there are no language-specific grammar rules
- a grammar is a **mapping** from words to structures



Lexicalised grammars

Many kinds of lexicalised grammar

- Categorial grammars (including CCGs)
- Lexicalised Tree Adjoining Grammars (LTAGs)
- CFGs in Greibach Normal Form

Lexicalised grammars are more efficient than arbitrary CFGs for NLG

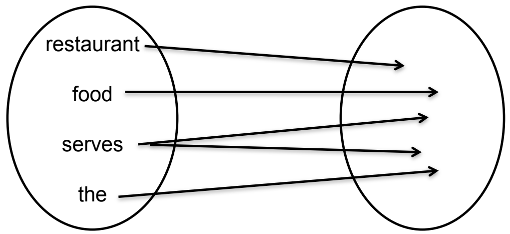
- search space is simpler (Koller & Striegnitz, 2002)

Categorial grammars (CGs)

A CG is a mapping from words to categories

- i.e. a set of word-category pairs

What do categories look like?



Categories

Two kinds of category

- “atomic” categories
- “complex” categories

Atomic categories

Each CG is built around a finite set of atomic categories

- simple, non-composite, atomic symbols
- similar to the symbols of a CFG

Examples:

- S – sentence/clause
- NP – noun phrase
- N - noun
- PP – preposition phrase

Atomic categories in XML

Use `atomcat` elements with a `type` attribute

```
<atomcat type="S"/>

<atomcat type="NP"/>
```

Complex categories

- Complex categories are built up from atomic category symbols
- From any ***finite*** set of atomic categories, can construct an ***infinite*** set of complex categories using two operators
  - directional slash operators: / and \

Traditional arithmetic notation is a useful analogy

Arithmetic notation

Arithmetic notation gives us a finite set of digits

- 0, 1, 2, . . ., 9

And a small set of operators for describing an infinite set of numbers: e.g.,

- concatenation: 23, 456, 92789
- addition: 2+7, 7+23, 456+65
- subtraction: 45 - 6, (2+6) - (67- 34)

Recursive definition

Categories are defined recursively

Atomic categories constitute the "base"

- every atomic category is also a category

The recursion involves the slash operators

- if X and Y are both categories, then so is (X/Y)
- if X and Y are both categories, then so is (X\Y)

Simple examples

category	meaning
(S\NP)	verb phrase, intransitive verb
(NP/N)	determiner
(N\N)	noun post-modifier, relative clause
(PP/NP)	preposition
(PP\NP)	postposition

Embedded examples

category	meaning
((S\NP)/NP)	transitive verb
((S\NP)/NP)/NP	ditransitive verb
((N\N)/NP)	post-nominal preposition
((S\NP)\(S\NP))	adverb
((S\NP)\((S\NP)/NP))	reflexive pronoun
((N\N)/(S\NP))	relative pronoun

Notational conveniences

Drop outermost parentheses

- (S\NP) ⇒ S\NP
- ((N\N)/(S\NP)) ⇒ (N\N)/(S\NP)

Assume left associativity of / and \

- ((S\NP)/NP)/NP ⇒ S\NP/NP/NP
- (N\N)/(S\NP) ⇒ N\N/(S\NP)

Complex categories in XML

How to represent S\NP:

```
<complexcat>
  <atomcat type="S"/>
  <slash dir="\"/>
  <atomcat type="NP"/>
</complexcat>
```

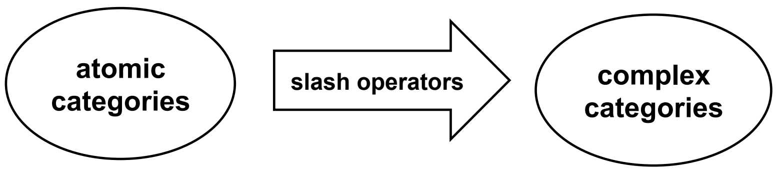
S\NP/NP in XML

```xml
<complexcat>
  <atomcat type="S"/>
  <slash dir="\"/>
  <atomcat type="NP"/>
  <slash dir="/"/>
  <atomcat type="NP"/>
</complexcat>
```

N\N/(S\NP) in XML

```xml
<complexcat>
  <atomcat type="N"/>
  <slash dir="\"/>
  <atomcat type="N"/>
  <slash dir="/"/>
  <complexcat>
    <atomcat type="S"/>
    <slash dir="\"/>
    <atomcat type="NP"/>
  </complexcat>
</complexcat>
```

Categories - summary



What does X/Y mean?

The kind of word or phrase that **combines** with a **following** Y to form an X.

$$\frac{X/Y \qquad Y}{X}\,>$$

This rule is called **forward application**.

Determiners

result

argument

the | restaurant
NP/N | N

NP

Determiner: word that combines with a following N to give an NP, i.e., an NP/N.

Prepositions

in | the restaurant
PP/NP | NP

PP

Preposition: word that combines with a following NP to give a PP, i.e., a PP/NP.

Derivations

in | the | restaurant
PP/NP | NP/N | N

NP

PP

Attributive adjectives

great | food
N/N | N

N

Attributive adjective: word that combines with a following N to give another N, i.e., an N/N.

Adjective stacking

great N/N

Italian N/N

restaurant N

N

N

What does X\Y mean?

The kind of word or phrase that combines with a **preceding** Y to form an X.

Y X\Y

X

This rule is called **backward application**.

Intransitive verbs

Giovanni's NP

rocks S\NP

S

Intransitive verb: word that combines with a preceding NP to give an S, i.e., an S\NP.

Postpositions

one floor NP

above PP\NP

PP

Postposition: word that combines with a preceding NP to give a PP, i.e., a PP\NP.

Transitive verbs

Giovanni's serves pasta
NP S\NP/NP NP

S\NP

S

Transitive verb: word that combines with a following NP to give an intransitive verb, S\NP.

Relative pronouns

restaurant that rocks
N N\N/(S\NP) S\NP

N\N

N

Relative pronoun: word that combines with a following intransitive verb S\NP to give a noun postmodifier N\N.

Adverbs

Giovanni's totally rocks
NP S\NP/(S\NP) S\NP

S\NP

S

Adverb: word that combines with a following intransitive verb S\NP to give another intransitive verb S\NP.

The story so far

- A categorial grammar is a mapping from words to categories
- Categories can be atomic or complex
- Words are combined into phrases by forward and backward application

Our lexicon

Giovanni's :- NP

pasta :- NP

serves :- S\NP/NP

rocks :- S\NP

restaurant :- N

great :- N/N

a : NP/N

that :- N\N/(S\NP)

What does our grammar do?

- It tells us which strings of words are grammatical and which are not.
- It assigns derivational structure to the grammatical strings.
- But what about semantics?

Remember HLDS?

- The input to the OpenCCG realiser is a hybrid logic dependency structure
- So our categorial lexicon needs to include HLDS in some way
- We need to be able to relate the grammatical sentences with their HLDS *(interpretation)*
- And also to relate HLDSs to the grammatical sentences that can realise them *(generation)*

Adding HLDS to our lexicon

Two steps:

1. Add a nominal to each atomic category symbol
2. Add a set of elementary predications of hybrid logic to each lexical category

Then relax and let forward and backward application (i.e. unification) take care of the rest!

Our lexicon again

Giovanni's :- NP

pasta :- NP

serves :- S\NP/NP

rocks :- S\NP

restaurant :- N

great :- N/N

a : NP/N

that :- N\N/(S\NP)

1. Adding nominals to categories

Giovanni's :- NP_x

pasta :- NP_x

serves :- $S_e\backslash NP_x/NP_y$

rocks :- $S_e\backslash NP_x$

restaurant :- N_x

great :- N_x/N_x

a : NP_x/N_x

that :- $N_x\backslash N_x/(S_e\backslash NP_x)$

- Subscripts to atomic category symbols
- Referential indices: unique labels for object or event evoked by the word
- By convention, use x, y, z for *objects*, and e, f, g for *events*
- Coindexed nominals indicate the referent of the argument is the same as referent of result, e.g., "great"

Adding nominals in XML

```
<atomcat type="NP"/>
```

⇩

```
<atomcat type="NP">
  <fs>
    <feat attr="index">
        <lf>
             <nomvar name="X"/>
        </lf>
    </feat>
  </fs>
</atomcat>
```

Nominal coindexation in XML

$N_x\backslash N_x$ ⇨

```
<complexcat>
  <atomcat type="N">
    <fs>
      <feat attr="index">
        <lf> <nomvar name="X"/> </lf>
      </feat>
    </fs>
  </atomcat>
  <slash dir="\">
 <atomcat type="N">
    <fs>
      <feat attr="index">
        <lf> <nomvar name="X"/> </lf>
      </feat>
    </fs>
  </atomcat>
</complexcat>
```

2. Adding EPs to categories

Giovanni's :- NP_x : @x Giovanni's

pasta :- NP_x : @x pasta

serves :- $S_e\backslash NP_x/NP_y$: @e serve, @e <AGENT> x,
@e <THEME> y

rocks :- $S_e\backslash NP_x$: @e great, @e <THEME> x

restaurant :- N_x : @e restaurant, @e <THEME> x

great :- N_x/N_x : @e great, @e <THEME> x

a : NP_x/N_x :

that :- $N_x\backslash N_x/(S_e\backslash NP_x)$:

Intransitive verbs

Giovanni's

NP_y

@y Giovanni's

rocks

$S_e\backslash NP_x$

@e great, @e <THEME> x

S_e : @e great, @e <THEME> x, @x Giovanni's

Transitive verbs

Giovanni's

NP_w

@w Giovanni's

serves

$S_e\backslash NP_x/NP_y$

@e serve
@e <AGENT> x
@e <THEME> y

pasta

NP_v

@v pasta

$S_e\backslash NP_x$

@e serve, @e <AGENT> x
@e <THEME> y, @y pasta

S_e : @e serve, @e <AGENT> x, @e <THEME> y,
@x Giovanni's, @y pasta

Attributive adjectives

great

N_z/N_z

@g great
@g <THEME> z

Italian

N_y/N_y

@f italian
@f <THEME> y

restaurant

N_x

@e restaurant
@e <THEME> x

N_x

@e restaurant, @e <THEME> x,
@f italian, @f <THEME> x

N_x : @e restaurant, @e <THEME> x,
@f italian, @f <THEME> x, @g great, @g <THEME> x

So where are we?

- We' ve seen how to define a lexicon in CG
- We' ve learned about two important operators in CG, i.e., forward and backward application
- We' ve seen how to combine words both
  - Syntactically (derivations, unification), **and**
  - Semantically (set union of elementary predications)
- But, Combinatory Categorial Grammar gives us much more

From CG to CCG

CCG is an "extension" of CG.

CCG has more rules:
- forward and backward type raising
- forward and backward composition

Everything else remains the same -
- in particular the HLDS representations.

Forward type raising

$$\frac{X}{Y/(Y\backslash X)}\ {\scriptstyle T}$$

$$\cfrac{\cfrac{\text{John}}{NP}}{S/(S\backslash NP)}\ {\scriptstyle T}$$

Type Raising

- CCG includes type-raising rules, which turn arguments into functions over functions over such arguments
- Forward type raising

$$\frac{X}{Y/(Y\backslash X)}\ {\scriptstyle T}$$

- Example:

$$\cfrac{\cfrac{\text{John}}{NP}}{S/(S\backslash NP)}\ {\scriptstyle T}$$

- The rules are order preserving. Here we turn an NP into a rightward looking function over leftward functions, preserving the linear order of constituents

Multiple derivations

Q1: *I know what restaurant serves French food, but what restaurant serves Italian food?*

A1: Babbo serves Italian food.

NP S\NP/NP NP

S\NP (>)

Q2: *I know what kind of food Pierre's serves, but what kind of food does Babbo serve?*

A2: Babbo serves Italian food.

NP S\NP/NP NP

S/(S\NP) (>T)

S/NP (>)

Forward composition

$$\frac{X/Y \quad Y/Z}{X/Z} >B$$

$$\frac{\dfrac{\text{John}}{S/(S\backslash NP)} \quad \dfrac{\text{likes}}{(S\backslash NP)/NP}}{S/NP} >B$$

CCG is more flexible

CCG generates **more** sentences:

- object relative clauses –
  "a restaurant that [John likes]$_{S/NP}$"
- right node raising –
  "[John likes]$_{S/NP}$ but [Charles hates]$_{S/NP}$ Giovanni's"

CCG is more flexible

CCG allows one sentence to be derived in many ways -

- reflecting different intonation patterns
- allowing incremental (i.e. left-branching) derivations from a right-branching lexicon

Further Reading

- Jason Baldridge and Geert-Jan Kruijff. 2003. “Multi-Modal Combinatory Categorial Grammar”. In *Proceedings of EACL 2003.*
- Mike White and Jason Baldridge. 2003. “Adapting Chart Realization to CCG”. In *Proceedings of ENLG 2003.*
- Jason Baldridge and Geert-Jan Kruijff. 2002. “Coupling CCG with Hybrid Logic Dependency Semantics”. In *Proceedings of ACL 2002.*