Combinatory Logic

Lecture 7

Syntactic formalisms for natural language parsing

FI MU autumn 2011

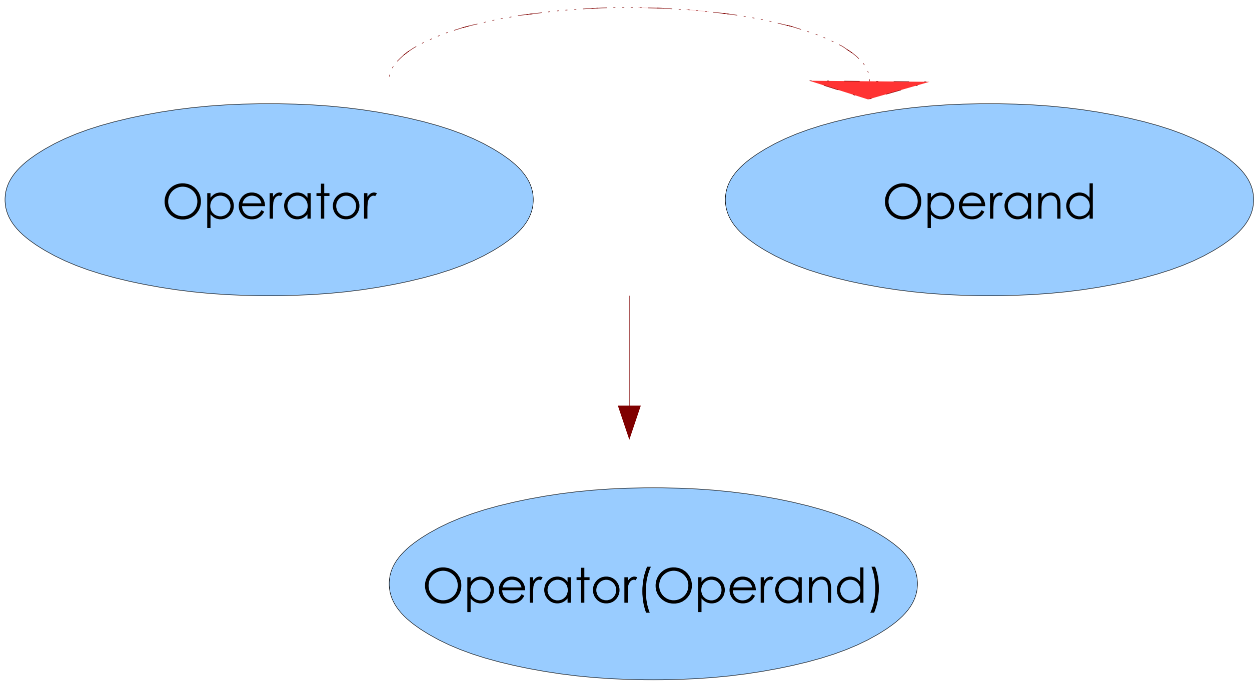
Outline

- Applicative system
- Combinators
- Combinators vs. λ-expressions
- Application to natural language parsing
- Combinators used in CCG

Applicative system

- CL (Curry & Feys, 1958, 1972) as an applicative system

CL is an applicative system because the basic unique operation in CL is the application of an **<u>operator</u>** to an **<u>operand</u>**



Combinators

CL defines general operators, called Combinators.

- Each combinator composes between them the elementary combinators and defines the complexe combinators.

- Certains combinators are considered as the basic combinators to define the other combinators.

- **Elementary combinators**

$\mathbf{I}$	$=_{\text{def}}$	$\lambda x.x$	*(identificator)*
$\mathbf{K}$	$=_{\text{def}}$	$\lambda x.\lambda y.x$	*(cancellator)*
$\mathbf{W}$	$=_{\text{def}}$	$\lambda x.\lambda y.xyy$	*(duplicator)*
$\mathbf{C}$	$=_{\text{def}}$	$\lambda x.\lambda y.\lambda z.xzy$	*(permutator)*
$\mathbf{B}$	$=_{\text{def}}$	$\lambda x.\lambda y.\lambda z.x(yz)$	*(compositor)*

$$\mathrm{S} =_{\mathrm{def}} \lambda x.\lambda y.\lambda z.xz(yz) \quad \textit{(substitution)}$$

$$\Phi =_{\mathrm{def}} \lambda x.\lambda y.\lambda z.\lambda u.x(yu)(zu) \quad \textit{(distribution)}$$

$$\Psi =_{\mathrm{def}} \lambda x.\lambda y.\lambda z.\lambda u.x(yz)(yu) \quad \textit{(distribution)}$$

- **ß-reductions**

The combinators are associated with the **ß-reductions** in a canonical form:

ß-reduction relation between X and Y

$$X \geq_{\beta} Y$$

Y was obtained from X by a ß-reduction

$$
\begin{array}{lll}
\mathbf{Ix} & \geq_{\beta} & \mathbf{x} \\
\mathbf{Kxy} & \geq_{\beta} & \mathbf{x} \\
\mathbf{Wxy} & \geq_{\beta} & \mathbf{xyy} \\
\mathbf{Cxyz} & \geq_{\beta} & \mathbf{xzy} \\
\mathbf{Bxyz} & \geq_{\beta} & \mathbf{x(yz)} \\
\mathbf{Sxyz} & \geq_{\beta} & \mathbf{xz(yz)} \\
\mathbf{\Phi xyzu} & \geq_{\beta} & \mathbf{x(yu)(zu)} \\
\boldsymbol{\psi}\mathbf{xyzu} & \geq_{\beta} & \mathbf{x(yz)(yu)}
\end{array}
$$

Each combinator is an operator which has a certain number of arguments (operands); sequences of the arguments which follow the comnator are called "the scope of combinator".

Intuitive interpretations of the elementary combinators are given by the associated **β-reductions**.

- The combinator **I** expresses the identity.
- The combinator **K** expresses the constant function.
- The combinator **W** expresses the diagonalisation or the duplication of an argument.
- The combinator **C** expresses the conversion, that is, the permutation of two arguments of an binary operator.
- The combinator **B** expresses the functional composition of two operators.
- The combinator **S** expresses the functional composition and the duplication of argument.
- The combinator **Φ** expresses the composition in parallel of operators acting on the common data.
- The combinator **ψ** expresses the composition by distribution.

- **Introduction and elimination rules of combinators**

Introduction and elimination rules of combinators can be presented in <u>the style of Gentzen</u> (*natural deduction*).

Elim. Rules	**Intro. Rules**
If	f
--- [e-**I**]	---[i-I]
f	If
Kfx	f
----- [e-**K**]	----[i-**K**]
f	Kfx

Elim. Rules	**Intro. Rules**
Cfx	xf
--- [e-**C**]	---[i-**C**]
xf	Cfx
Bfxy	f(xy)
----- [e-**B**]	----[i-**B**]
f(xy)	**B**fxy
Φfxyz	f(xz)(yz)
----- [e-**Φ**]	----[i-**Φ**]
f(xz)(yz)	**Φ**fxyz

Combinators vs. λ-expressions

The most important difference between the **CL** and **λ-calculus** is the use of the bounded variables.

Every combinator is an λ-expression.

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

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

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

Application to natural language parsing

John is brilliant

- The predicate *is brilliant* is an operator which operate on the operand John to construct the final proposition.
- The applicative representation associated to this analysis is the following:

(is-brillant)John

- We define the operator **John*** as being constructed from the lexicon *John* by

[John* = **C*** John].

1. John* (is-brillant)
2. [John* = **C*** John]
3. **C***John (is-brillant)
4. is-brillant (John)

John is brilliant in λ-term

Operator <u>John*</u> by λ-expression

$$[John^* = \lambda x.x\ (John')]$$

1/ John*(λx.is-brilliant'(x))

2/ [John* = λx.x (John')]

3/ (λx.x(John'))(λx.is-brilliant'(x))

4/ (λx.is-brilliant'(x))(John')

5/ is-brillinat'(John')

Passivisation

Consider the following sentences

a. The man has been killed.

b. One has killed him.

→ Invariant of meaning

→ Relation between two sentences

: a. unary passive predicate (*has-been-killed*)

: b. active transitive predicate (*have-killed*)

Definition of the operator of passivisation 'PASS'

$$[\mathrm{PASS} = \mathbf{B} \textstyle\sum \mathbf{C} = \sum \circ \mathbf{C}]$$

where B and C are the combinator of composition and of conversion and where Σ is the existential quantificator which, by applying to a binary predicate, transforms it into the unary predicate.

$$[\mathrm{PASS} = \mathbf{B}\sum\mathbf{C} = \sum \circ \mathbf{C}]$$

1/ has-been-killed (the-man)	*hypothesis*
2/ [has-been-killed=PASS(has killed)]	*passive lexical predicate*
3/ PASS (has-killed)(the-man)	*repl.2.,1.*
4/ [PASS = **B** ∑ **C**]	*definition of 'PASS'*
5/ **B** ∑ **C** (has-killed)(the-man)	*repl.4.,3.*
6/ ∑ (**C**(has-killed))(the-man)	[e-**B**]
7/ (**C**(has-killed)) x (the-man)	[e-∑]
8/ (has-killed)(the-main) x	[e-**C**]
9/ [x in the agentive subject position = *one*]	*definition of 'one'*
10/ (has-killed)(the-man)*one*	*repl.9.,8., normal form*

We establish the paraphrastic relation between the passive sentence with expressed agent and its active counterpart:

The man has been killed by the enemy

↓

The enemy has killed the man

Relation between ***give-to*** and ***receive-from***

z *gives* y *to* x

↕

x *receives* y *from* x

The lexical predicate "*give-to*" has a predicate converse associated to "*receive-from*";

[receive-from z y x = give-to x y z]

1/ (receive-from) z y x

2/ **C**((receive-from) z) x y

3/ **BC**(receive-from) z x y

4/ **C**(**BC**(receive-from)) z x y

5/ **C**(**C**(**BC**(receive-from)) x) y z

6/ **BC**(**C**(**BC**(receive-from))) x y z

7/ [give-to=**BC**(**C**(**BC**(receive-from)))]

8/ give-to x y z

Combinators used in CCG

Motivation of applying the combinators

to natural language parsing

- Linguistic: complex phenomena of natural language applicable to the various languages

- Informatics: left to right parsing (LR)

  ex: reduce the spurious-ambiguity

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 in term of *λ-expression*

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')

Semantic representation in term of the *combinators*

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**)

C*I requested and would prefer musicals

3/ S/(S\NP) (S\NP)/NP CONJ (S\NP)/NP NP (>**B**)

C*I requested and **B** would prefer musicals

4/ S/(S\NP) (S\NP)/NP NP (>**Φ)**

C*I **Φ and requested (B would prefer) musicals**

5/ S/NP NP (>**B**)

B((**C***I) (**Φ and requested (B would prefer)))** **musicals**

6/ S (>)

B((**C***I) (**Φ and requested (B would prefer)))** **musicals**

I requested and would prefer musicals

S: B((C*I)(Φ and requested (B would prefer))) musicals

1/ B((C*I)(Φ and requested (B would prefer))) musicals

2/ (C*I)((Φ and requested (B would prefer))) musicals) [e-B]

3/ ((Φ and requested (B would prefer))) musicals) I [e-C*]

4/ (and (requested musicals) ((B would prefer) musicals)) I [e-Φ]

5/ ((and (requested musicals) (would (prefer musicals))) I) [e-B]

Normal form

A **normal form** is a combinatory expression which is irreducible in the sense that it contain any occurrence of a redex.

If a combinatory expression X reduce to a combinatory expression N which is in **normal form**, so N is called the **normal form** of X.

Example

Bxyz is reducible to x(yz).

x(yz) is a normal form of the combinatory expression **B**xyz.

Example

Prove xyz is the normal form of **BBC**xyz.

$$\mathbf{BBC}xyz \rightarrow_{\beta} xyz$$

1/ **BBC**xyz

2/ **C**(**C**x)yz [e-**B**]

3/ **C**xzy [e-**C**]

4/ xyz [e-**C**]

Classwork

Give the semantic representation in term of combinators.
Please refer to the given paper on last lecture on CCG Parsing.