IA165
Combinatory Logic for Computational Semantics

Spring 2012

Juyeon Kang
qkang@fi.muni.cz

B410, Faculty of Informatics, Masaryk University,
Brno, Czech Rep.

Application of the combinators to natural language analysis

Introduction and elimination rule of combinators

```
Elim. Rules            Intro. Rules

If                      f
--- [e-I]             ---[i-I]
f                      If
Kfx                     f
----- [e-K]           ----[i-K]
f                      Kfx
```

```
Elim. Rules            Intro. Rules

C*fx                    xf
--- [e-C*]            ---[i-C*]
xf C*fx
Bfxy                   f(xy)
----- [e-B]           ----[i-B]
f(xy)                  Bfxy
Φfxyz                  f(xz)(yz)
----- [e-Φ]           ----[i-Φ]
f(xz)(yz)              Φfxyz
```

- Bracketing: associativity

$((X \circ Y) \circ Z)x$

$\geq B(BXY)Zx$

$\geq BXY(Zx)$

$\geq X(Y(Zx))$

$\leq X(BYZx)$

$\leq BX(BYZx)$

$= X(Y \circ Z)$

$Bxyz = (Bxy)z$

$xyz = x(yz)$

Equivalence to Church's 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.

$$K \equiv \lambda xy.x$$

$$T \equiv \lambda x.x$$

$$S \equiv \lambda xyz.xz(yz)$$

- I =def λx.x (identificator)
- K =def λx.λy.x (cancellator)
- W =def λx.λy.xyy (duplicator)
- C =def λx.λy.λz.xzy (permutator)
- B =def λx.λy.λz.x(yz)(compositor)
- S =def λx.λy.λz.xz(yz) (substitution)
- Φ =def λx.λy.λz.λu.x(yu)(zu) (distribution)
- ψ =def λx.λy.λz.λu.x(yz)(yu) (distribution)

Application of the CL to NLP

- S. k. Shaumyan (University of Yale, USA)

  → first application of combinators to natural language analysis
- M. Steedman (University of Endinburgh, of Pennsylvania, UK, USA)

  → relation with the extended Categorial Grammar
- J.-P. Desclés (Paris-Sorbonne University, FR)

  → application of combinators to complex phenomena

CCG and CL

- 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, 2000):

a. $\mathbf{B}fg \equiv \lambda x.f(g\ x)$ b. $\mathbf{T}x \equiv \lambda f.f\ x$ c. $\mathbf{S}fg \equiv \lambda x.fx(g\ x)$ d. $\mathbf{\Phi}fgq \equiv \lambda x.fq(g\ q)$

Main idea in CCG 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

- Application of the combinators using syntactic tools: typed combinators

→ CCG (Combinatory Categorial Grammar) offers CCG types and rules
→ β *−reduction* rules of combinators are integrated into the CCG rules

→ "systematic"

- Application of the combinators directly to the natural language sentences

: need to introduce the brackets and decide yourself when you apply the combinators

→ "heuristic"

Example: use the combinators as a logical tool of semantic analysis

a. apply directly the β -*reduction* rules of combinators

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)   [e-**C***]*

4. is-brillant (John)

- Combinator B and C*:

John loves Mary = (loves Mary)John

1. **C***John loves Mary [i-**C***]
2. **C***John (loves Mary) bracketing
3. **B**(C*John) loves Mary [i-**B**]
4. (C*John) (loves Mary) [e-**B**]
5. (loves Mary) John [e-**C***]

John reads a book at home = (at home(reads (a book)))John

1. (**C***John) reads a book at home [i–**C***]
2. (**B**(**C***John)reads) a book at home [i–**B**]
3. (**B**(**B**(**C***John)reads) a) book at home [i–**B**]
4. (**B**(**B**(**B**(**C***John)reads) a) book at home [i–**B**]
5. (**B**(**B**(**B**(**B**(**C***John)reads) a) book) at home [i–**B**]

?6. (**B**(**B**(**B**(**B**(**B**(**C***John)reads) a) book) at) home [i–**B**]

?7. (**B**(**B**(**B**(**B**(**B**(**B**(**C***John)reads) a) book) at) home) [i–**B**]

.....

?. (**C***John)(reads(a book((at home)))) [e–**B**]

?. (reads(a book((at home))))John [e–**C***]

Example: use the combinators as a logical tool of semantic analysis

b. use the CCG types and rules by integrating the β-reduction rules of combinators into the CCG rules

Introduction to CCG types and rules

- **CCG types**

  primitive types: S for sentence, NP for noun phrase, N for noun

  derived types: S/NP, N/N, N\N, (S/NP)/NP, NP/N...

- **Directionality: / (over) and \ (under)**

  a/b: a applies to b, a\b: b is applied to a

  → direction of application of operator to operand

- CCG rules: basic CG rules + combinators

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

Forward(>) and backward (<) functional application rules

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

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

$x/y \quad y \rightarrow x \quad \approx \quad 2/4 * 2 = 4$

$$x/y \quad y \quad \rightarrow x \qquad \approx \qquad 2/4 \ * \ 2 = 4$$

x/y means that an expression e1 of type x/y waits an expression e2 of type y to its left in order to obtain an expression (e1 e2)of type x.

Example

x is an expression e1 of type (S\NP)/NP: *loves*

y is an expression e2 of type NP: *Anna*

(S\NP)/NP:*loves* NP:*Anna* → (S\NP)

loves (Anna)

Function composition (FC) rules with the combinator B

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

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

e1: (x/y) e2:(y/z)

--------------------->(>**B**)

(x/z): ***B*** *e1 e2*

An expression *e1* of type x/y waits an expression *e2* of type y/z to obtain an expression **B***e1 e2* of type x/z by the composition rule.

Example: e1 of type (S\NP)/NP: *likes* and e2 of type (NP/N): *a*

```
(S\NP)/NP:likes   (NP/N):a
-------------------------> (>B)
((S\NP)/N): (B likes a)
```

Type-raising rules with the combinator C*

$$X:a \Rightarrow S/(S\backslash X):\lambda f.fa \quad (>C*)$$

$$X:a \Rightarrow S\backslash(S/X):\lambda f.fa \quad (<C*)$$

e1:x

---------------> (>**C***)

S/(S\x): **C*x**

An expression *e1* of type x are type-raised to transform *e1* into **C***e1* by the type-raising rule.

Example: *e1* of type N*:John

```
      e1:John
     ---------> (>C*)
S/(S\NP):C*John
```

- Syntax-semantic interface using CCG and typed combinators

  → apply the elimination rules of the combinators to the result obtained from the syntactic analysis

  → get the semantic representation in terms of operator and operand

  → make clear two steps at the syntactic and semantic level

- Combinator B and C* in the CCG rules

John likes the puzzle

```
John            likes         the    puzzle
NP              (S\NP)/NP     NP/N    N
S/(S\NP): (C*John)                                    [>C*]
        S/NP: B(C*John) likes                         [>B]
                              NP: (the puzzle)         [>]
S: B(C*John)likes (the puzzle)                        [>]
S: (C*John) (likes (the puzzle))                      [e-B]
S: (likes (the puzzle))John                           [e-C*]
```

Next week...

- Continue about the application of the combinators to natural language analysis: more complex phenomena