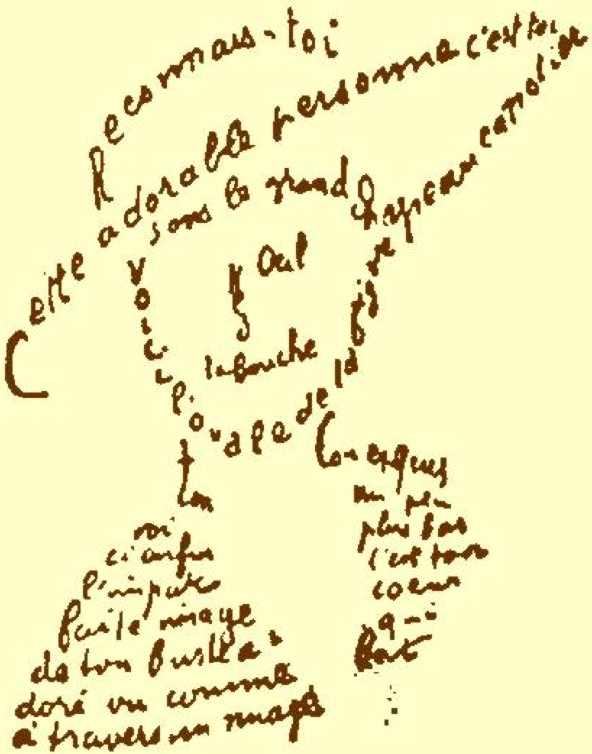

Reconnais-toi
Cette adorable personne c'est toi
sous le grand chapeau canotier
Oeil
la bouche
Voici l'ovale de ta figure
ton cou exquis
un peu plus bas c'est ton coeur qui bat
Voici enfin l'imparfaite image de ton buste adoré vu comme à travers un nuage

Abstract Categorial Grammars

Philippe de Groote

LORIA & Inria-Lorraine

Content

Content

Content

Content

Content

Content

Context and Motivations

Categorial Grammars:

Categorial Grammars:

A type-theoretic view of grammars and grammatical composition.

Categorial Grammars:

A type-theoretic view of grammars and grammatical composition.

Examples: AB-grammars, Lambek-grammars

$$
\begin{array}{rcl}
\text{Pierre} & : & NP \\
\text{une} & : & NP \,/\, N \\
\text{pomme} & : & N \\
\text{mange} & : & (NP \setminus S) \,/\, NP
\end{array}
$$

Categorial Grammars:

A type-theoretic view of grammars and grammatical composition.

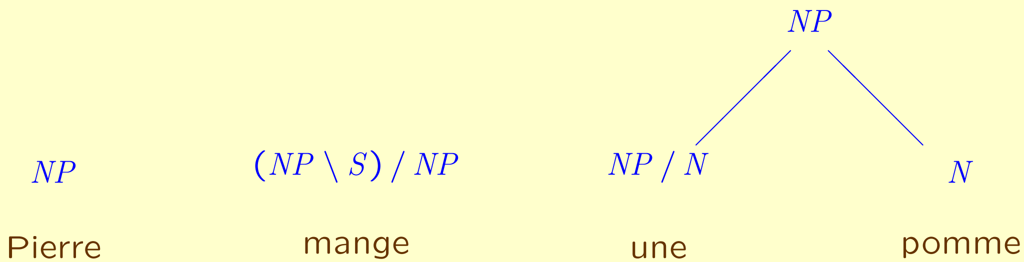
Examples: AB-grammars, Lambek-grammars

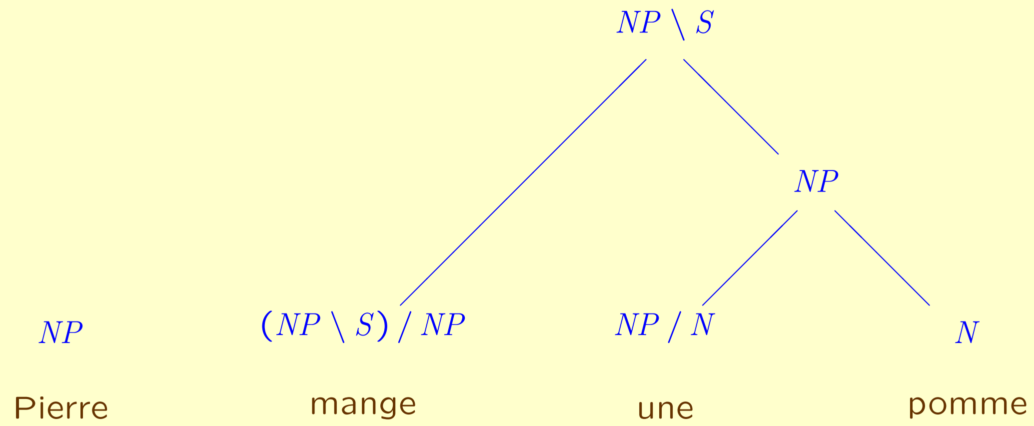
$$\begin{array}{rcl} \text{Pierre} & : & NP \\ \text{une} & : & NP \,/\, N \\ \text{pomme} & : & N \\ \text{mange} & : & (NP \setminus S) \,/\, NP \end{array}$$

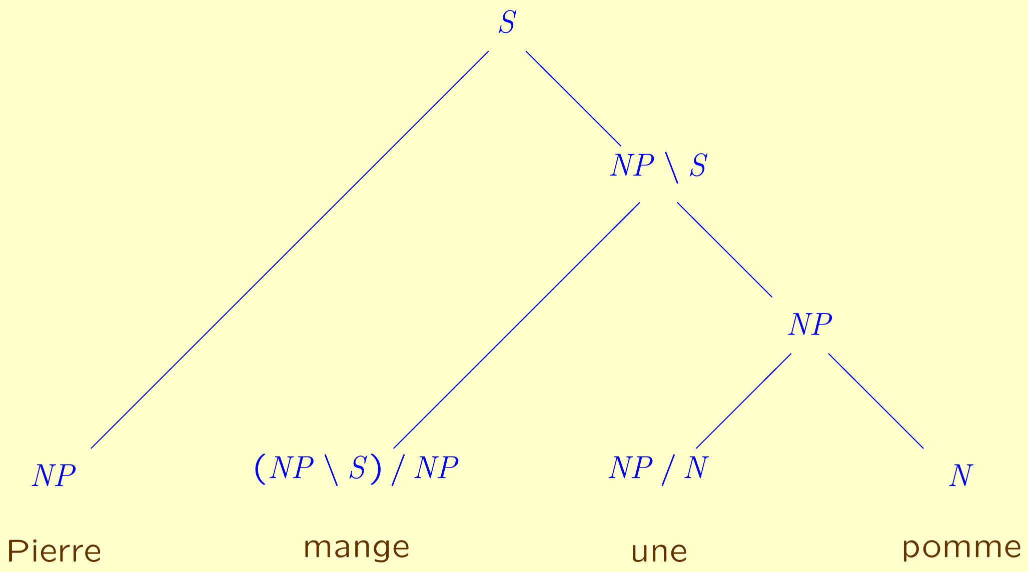
Syntax/semantics interface based on the Curry-Howard isomorphism.

Pierre mange une pomme

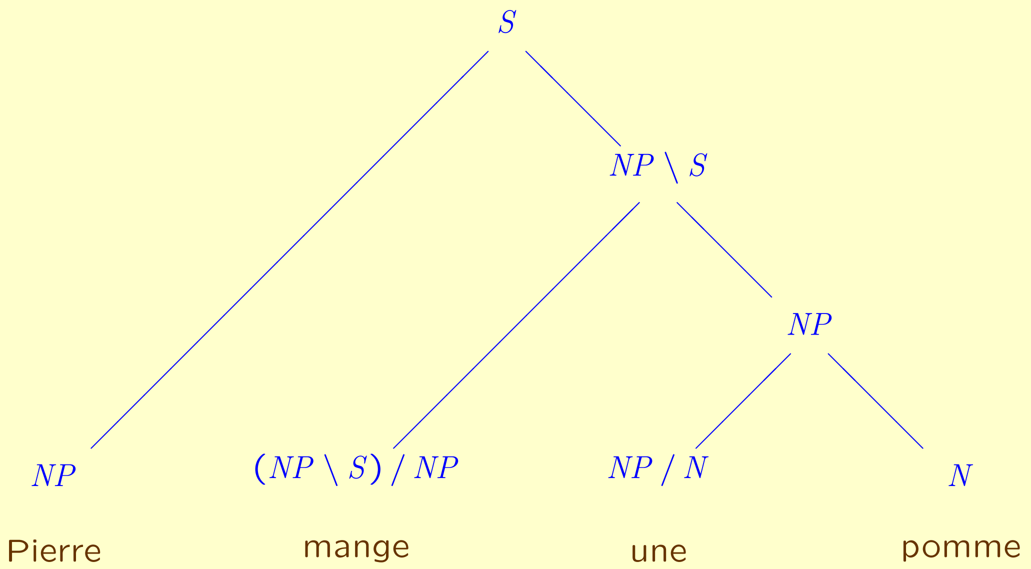
NP	$(NP \setminus S) / NP$	NP / N	N
Pierre	mange	une	pomme


NP
NP
Pierre
(NP \ S) / NP
mange
NP / N
une
N
pomme


$NP \setminus S$
NP
NP
$(NP \setminus S) / NP$
NP / N
N
Pierre
mange
une
pomme



S
$NP \setminus S$
NP
NP
$(NP \setminus S) / NP$
NP / N
N
Pierre
mange
une
pomme



$$NP, (NP \setminus S) / NP, NP / N, N \vdash S$$

Current Type-Logical Grammars:

Current Type-Logical Grammars:

SYNTACTIC FORM

terms from some prosodic algebra

Current Type-Logical Grammars:

SYNTACTIC FORM	$\Rightarrow$	**PARSING STRUCTURE**
terms from some prosodic algebra		deduction tree
		proof-net

Current Type-Logical Grammars:

SYNTACTIC FORM	$\Rightarrow$	**PARSING STRUCTURE**	$\Rightarrow$	**LOGICAL FORM**
terms from some prosodic algebra		deduction tree proof-net		logical formulas (first-order) (higher-order) (modal) (intentional)

Definition

Types, signatures and λ-terms:

Types, signatures and λ-terms:

$\mathcal{T}(A)$ is the set of linear implicative types built on the set of atomic types A:

$$\mathcal{T}(A) \;::=\; A \mid (\mathcal{T}(A) \multimap \mathcal{T}(A))$$

Types, signatures and λ-terms:

$\mathcal{T}(A)$ is the set of linear implicative types built on the set of atomic types A:

$$\mathcal{T}(A) \ ::= \ A \mid (\mathcal{T}(A) \multimap \mathcal{T}(A))$$

A higher-order linear signature is a triple $\mathbf{\Sigma} = \langle A, C, \tau \rangle$, where:

Types, signatures and λ-terms:

$\mathcal{T}(A)$ is the set of linear implicative types built on the set of atomic types A:

$$\mathcal{T}(A) \ ::= \ A \mid (\mathcal{T}(A) \multimap \mathcal{T}(A))$$

A higher-order linear signature is a triple $\mathbf{\Sigma} = \langle A, C, \tau \rangle$, where:

A is a finite set of atomic types;

Types, signatures and λ-terms:

$\mathcal{T}(A)$ is the set of linear implicative types built on the set of atomic types A:

$$\mathcal{T}(A) \ ::= \ A \mid (\mathcal{T}(A) \multimap \mathcal{T}(A))$$

A higher-order linear signature is a triple $\boldsymbol{\Sigma} = \langle A, C, \tau \rangle$, where:

A is a finite set of atomic types;

C is a finite set of constants;

Types, signatures and λ-terms:

$\mathcal{T}(A)$ is the set of linear implicative types built on the set of atomic types A:

$$\mathcal{T}(A) \ ::= \ A \mid (\mathcal{T}(A) \multimap \mathcal{T}(A))$$

A higher-order linear signature is a triple $\mathbf{\Sigma} = \langle A, C, \tau \rangle$, where:

A is a finite set of atomic types;

C is a finite set of constants;

$\tau : C \to \mathcal{T}(A)$ is a function that assigns each constant in C with a linear implicative type built on A.

Types, signatures and λ-terms:

$\mathcal{T}(A)$ is the set of linear implicative types built on the set of atomic types A:

$$\mathcal{T}(A) \ ::= \ A \mid (\mathcal{T}(A) \multimap \mathcal{T}(A))$$

A higher-order linear signature is a triple $\mathbf{\Sigma} = \langle A, C, \tau \rangle$, where:

- A is a finite set of atomic types;
- C is a finite set of constants;
- $\tau : C \to \mathcal{T}(A)$ is a function that assigns each constant in C with a linear implicative type built on A.

$\Lambda(\Sigma)$ denotes the set of linearλ-terms built upon a higher-order linear signature Σ.

Vocabularies and Lexicons:

Vocabularies and Lexicons:

A vocabulary is simply defined to be a higher-order linear signature.

Vocabularies and Lexicons:

A vocabulary is simply defined to be a higher-order linear signature.

Given two vocabularies $\mathbf{\Sigma}_1 = \langle A_1, C_1, \tau_1 \rangle$ and $\mathbf{\Sigma}_2 = \langle A_2, C_2, \tau_2 \rangle$, a lexicon $\mathcal{L} = \langle F, G \rangle$ from $\mathbf{\Sigma}_1$ to $\mathbf{\Sigma}_2$ is made of two functions:

$F : A_1 \rightarrow \mathcal{T}(A_2)$,

$G : C_1 \rightarrow \Lambda(\mathbf{\Sigma}_2)$,

such that

$$\vdash_{\Sigma_2} G(c) : \widehat{F}(\tau_1(c)).$$

Definition:

Definition:

An abstract categorial grammar is a quadruple

$$\mathcal{G} = \langle \Sigma_1, \Sigma_2, \mathcal{L}, s \rangle$$

where :

Definition:

An abstract categorial grammar is a quadruple

$$\mathcal{G} = \langle \Sigma_1, \Sigma_2, \mathcal{L}, s \rangle$$

where :

$\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ and $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$ are two higher-order linear signatures; Σ_1 is called the abstract vocabulary and Σ_2 is called the object vocabulary;

Definition:

An abstract categorial grammar is a quadruple

$$\mathcal{G} = \langle \Sigma_1, \Sigma_2, \mathcal{L}, s \rangle$$

where :

- $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ and $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$ are two higher-order linear signatures; Σ_1 is called the abstract vocabulary and Σ_2 is called the object vocabulary;

- $\mathcal{L} : \Sigma_1 \rightarrow \Sigma_2$ is a lexicon from the abstract vocabulary to the object vocabulary;

Definition:

An abstract categorial grammar is a quadruple

$$\mathcal{G} = \langle \Sigma_1, \Sigma_2, \mathcal{L}, s \rangle$$

where :

- $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ and $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$ are two higher-order linear signatures; Σ_1 is called the abstract vocabulary and Σ_2 is called the object vocabulary;

- $\mathcal{L} : \Sigma_1 \rightarrow \Sigma_2$ is a lexicon from the abstract vocabulary to the object vocabulary;

- $s \in \mathcal{T}(A_1)$ is a type of the abstract vocabulary; it is called the distinguished type of the grammar.

Languages generated by an ACG:

Languages generated by an ACG:

The abstract language generated by $\mathcal{G}$ ($\mathcal{A}(\mathcal{G})$) is defined as follows:

$$\mathcal{A}(\mathcal{G}) = \{t \in \Lambda(\Sigma_1) \mid\ \vdash_{\Sigma_1} t : s \text{ is derivable}\}$$

Languages generated by an ACG:

The abstract language generated by $\mathcal{G}$ ($\mathcal{A}(\mathcal{G})$) is defined as follows:

$$\mathcal{A}(\mathcal{G}) = \{t \in \Lambda(\Sigma_1) \mid \vdash_{\Sigma_1} t : s \text{ is derivable}\}$$

The object language generated by $\mathcal{G}$ ($\mathcal{O}(\mathcal{G})$) is defined to be the image of the abstract language by the term homomorphism induced by the lexicon $\mathcal{L}$:

$$\mathcal{O}(\mathcal{G}) = \{t \in \Lambda(\Sigma_2) \mid \exists u \in \mathcal{A}(\mathcal{G}).\ t = \mathcal{L}(u)\}$$

Examples

Strings as linear λ-terms

Strings as linear λ-terms

There is a canonical way of representing strings as linear λ-terms. It consists of representing strings as function composition:

$$\text{`}abbac\text{'} = \lambda x.\, a\,(b\,(b\,(a\,(c\,x))))$$

Strings as linear λ-terms

There is a canonical way of representing strings as linear λ-terms. It consists of representing strings as function composition:

$$`abbac' = \lambda x.\, a\,(b\,(b\,(a\,(c\,x))))$$

In this setting:

$$\begin{aligned} \epsilon &\triangleq \lambda x.\, x \\ \alpha + \beta &\triangleq \lambda \alpha.\, \lambda \beta.\, \lambda x.\, \alpha\,(\beta\, x) \end{aligned}$$

Context-free grammars

Context-free grammars

$$S \longrightarrow \epsilon$$
$$S \longrightarrow aSb$$

Context-free grammars

$$S \longrightarrow \epsilon$$
$$S \longrightarrow aSb$$

Abstract vocabulary :

$$S : \text{type}$$
$$A : S$$
$$B : S \multimap S$$

Context-free grammars

$$S \longrightarrow \epsilon$$
$$S \longrightarrow aSb$$

Abstract vocabulary :

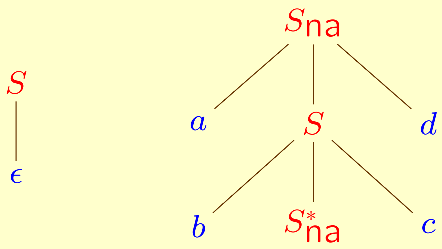
$$S : \text{type}$$
$$A : S$$
$$B : S \multimap S$$

Lexicon :

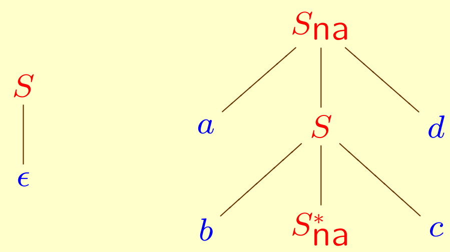
$$S \mapsto string$$
$$A \mapsto \epsilon$$
$$B \mapsto \lambda x.\, a + x + b$$

Tree-adjoining grammars

Tree-adjoining grammars



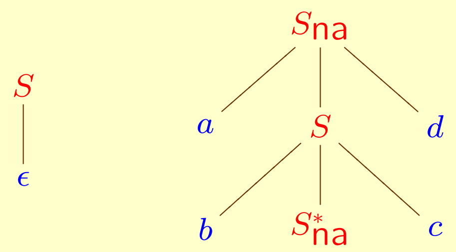
Tree-adjoining grammars



Abstract vocabulary :

$$\begin{array}{l} S, S', S'' : \text{type} \\ A : (S'' \multimap S') \multimap S \\ B : S'' \multimap (S'' \multimap S') \multimap S' \\ C : S'' \multimap S' \end{array}$$

Tree-adjoining grammars



Abstract vocabulary :

$$S, S', S'' : \text{type}$$
$$A : (S'' \multimap S') \multimap S$$
$$B : S'' \multimap (S'' \multimap S') \multimap S'$$
$$C : S'' \multimap S'$$

Lexicon :

$$S, S', S'' \mapsto \mathit{string}$$
$$A \mapsto \lambda f.\, f\, \epsilon$$
$$B \mapsto \lambda x.\, \lambda g.\, a + g\,(b + x + c) + d$$
$$C \mapsto \lambda x.\, x$$

A toy example

A toy example

Pierre lit un article que Marie a écrit

A toy example

Pierre lit un article que Marie a écrit

Σ_0:

$$
\begin{array}{rcl}
N, NP, S & : & \text{type;} \\
\mathsf{P}, \mathsf{M} & : & NP \\
\mathsf{A} & : & N \\
\mathsf{L}, \mathsf{AE} & : & NP \multimap (NP \multimap S) \\
\mathsf{U} & : & N \multimap NP \\
\mathsf{Q} & : & (NP \multimap S) \multimap (N \multimap N)
\end{array}
$$

A toy example

Pierre lit un article que Marie a écrit

Σ_0:

$$\begin{array}{rcl} N, NP, S & : & \text{type;} \\ \mathsf{P}, \mathsf{M} & : & NP \\ \mathsf{A} & : & N \\ \mathsf{L}, \mathsf{AE} & : & NP \multimap (NP \multimap S) \\ \mathsf{U} & : & N \multimap NP \\ \mathsf{Q} & : & (NP \multimap S) \multimap (N \multimap N) \end{array}$$

Σ_1: /Pierre/, /Marie/, /article/, /lit/, /a écrit/, /un/, /que/ : $STRING$

A toy example

Pierre lit un article que Marie a écrit

Σ_0:

$$
\begin{array}{rcl}
N, NP, S & : & \text{type;} \\
\mathsf{P}, \mathsf{M} & : & NP \\
\mathsf{A} & : & N \\
\mathsf{L}, \mathsf{AE} & : & NP \multimap (NP \multimap S) \\
\mathsf{U} & : & N \multimap NP \\
\mathsf{Q} & : & (NP \multimap S) \multimap (N \multimap N)
\end{array}
$$

Σ_1: /Pierre/, /Marie/, /article/, /lit/, /a écrit/, /un/, /que/ : $STRING$

Σ_2:

$$
\begin{array}{rcl}
\iota, o & : & \text{type;} \\
\mathbf{p}, \mathbf{m} & : & \iota \\
\text{article} & : & \iota \multimap o \\
\text{read}, \text{wrote} & : & \iota \multimap (\iota \multimap o) \\
\wedge & : & o \multimap (o \multimap o) \\
\exists & : & (\iota \rightarrow o) \multimap o
\end{array}
$$

$\mathcal{L}_1 : \Sigma_0 \longrightarrow \Sigma_1$

$\mathcal{L}_1 : \Sigma_0 \rightarrow \Sigma_1$

$$N, NP, S \; := \; STRING;$$

P	:=	/Pierre/	:	NP
M	:=	/Marie/	:	NP
A	:=	/article/	:	N
L	:=	$\lambda x. \lambda y. y +$ /lit/ $+ x$	:	$NP \multimap (NP \multimap S)$
AE	:=	$\lambda x. \lambda y. y +$ /a écrit/ $+ x$	:	$NP \multimap (NP \multimap S)$
U	:=	$\lambda x.$ /un/ $+ x$	:	$N \multimap NP$
Q	:=	$\lambda x. \lambda y. y +$ /que/ $+ x\, \epsilon$	:	$(NP \multimap S) \multimap (N \multimap N)$

● ○ |◀ ◀ ❚❚ ▶ ▶|

$\mathcal{L}_1 : \Sigma_0 \rightarrow \Sigma_1$

$$N, NP, S := STRING;$$

$$\begin{array}{rcll}
\mathsf{P} & := & /\text{Pierre}/ & : NP \\
\mathsf{M} & := & /\text{Marie}/ & : NP \\
\mathsf{A} & := & /\text{article}/ & : N \\
\mathsf{L} & := & \lambda x.\, \lambda y.\, y + /\text{lit}/ + x & : NP \multimap (NP \multimap S) \\
\mathsf{AE} & := & \lambda x.\, \lambda y.\, y + /\text{a écrit}/ + x & : NP \multimap (NP \multimap S) \\
\mathsf{U} & := & \lambda x.\, /\text{un}/ + x & : N \multimap NP \\
\mathsf{Q} & := & \lambda x.\, \lambda y.\, y + /\text{que}/ + x\, \epsilon & : (NP \multimap S) \multimap (N \multimap N)
\end{array}$$

Parsing

/Pierre/ + /lit/ + /un/ + /article/ + /que/ + /Marie/ + /a écrit/

$\mathcal{L}_1 : \Sigma_0 \to \Sigma_1$

$$N, NP, S := STRING;$$

$$\begin{array}{rcll}
\mathsf{P} & := & /\mathsf{Pierre}/ & : NP \\
\mathsf{M} & := & /\mathsf{Marie}/ & : NP \\
\mathsf{A} & := & /\mathsf{article}/ & : N \\
\mathsf{L} & := & \lambda x. \lambda y. y + /\mathsf{lit}/ + x & : NP \multimap (NP \multimap S) \\
\mathsf{AE} & := & \lambda x. \lambda y. y + /\mathsf{a\ écrit}/ + x & : NP \multimap (NP \multimap S) \\
\mathsf{U} & := & \lambda x. /\mathsf{un}/ + x & : N \multimap NP \\
\mathsf{Q} & := & \lambda x. \lambda y. y + /\mathsf{que}/ + x\,\epsilon & : (NP \multimap S) \multimap (N \multimap N)
\end{array}$$

Parsing

/Pierre/ + /lit/ + /un/ + /article/ + /que/ + /Marie/ + /a écrit/

yields the following λ-term of type S:

$$\mathsf{L}\,(\mathsf{U}\,(\mathsf{Q}\,(\lambda x.\,\mathsf{AE}\,x\,\mathsf{M})\,\mathsf{A}))\,\mathsf{P}$$

$\mathcal{L}_2 : \Sigma_0 \longrightarrow \Sigma_2$

$\mathcal{L}_2 : \Sigma_0 \to \Sigma_2$

$$
\begin{array}{rcl}
S & := & o; \\
N & := & \iota \multimap o; \\
NP & := & (\iota \multimap o) \multimap o;
\end{array}
$$

$$
\begin{array}{rcll}
\mathsf{P} & := & \lambda k.\, k\, \mathbf{p} & : NP \\
\mathsf{M} & := & \lambda k.\, k\, \mathbf{m} & : NP \\
\mathsf{A} & := & \lambda x.\, \mathsf{article}\, x & : N \\
\mathsf{L} & := & \lambda p.\, \lambda q.\, p\, (\lambda x.\, q\, (\lambda y.\, \mathsf{read}\, y\, x)) & : NP \multimap (NP \multimap S) \\
\mathsf{AE} & := & \lambda p.\, \lambda q.\, p\, (\lambda x.\, q\, (\lambda y.\, \mathsf{wrote}\, y\, x)) & : NP \multimap (NP \multimap S) \\
\mathsf{U} & := & \lambda p.\, \lambda q.\, \exists x.\, (p\, x) \wedge (q\, x) & : N \multimap NP \\
\mathsf{Q} & := & \lambda r.\, \lambda p.\, \lambda x.\, (p\, x) \wedge (r\, (\lambda k.\, k\, x)) & : (NP \multimap P) \multimap (N \multimap N)
\end{array}
$$

$\mathcal{L}_2 : \Sigma_0 \to \Sigma_2$

$$
\begin{array}{rcl}
S & := & o; \\
N & := & \iota \multimap o; \\
NP & := & (\iota \multimap o) \multimap o;
\end{array}
$$

$$
\begin{array}{rcl@{\quad}c@{\quad}l}
\mathsf{P} & := & \lambda k.\, k\, \mathbf{p} & : & NP \\
\mathsf{M} & := & \lambda k.\, k\, \mathbf{m} & : & NP \\
\mathsf{A} & := & \lambda x.\, \mathsf{article}\, x & : & N \\
\mathsf{L} & := & \lambda p.\, \lambda q.\, p\,(\lambda x.\, q\,(\lambda y.\, \mathsf{read}\, y\, x)) & : & NP \multimap (NP \multimap S) \\
\mathsf{AE} & := & \lambda p.\, \lambda q.\, p\,(\lambda x.\, q\,(\lambda y.\, \mathsf{wrote}\, y\, x)) & : & NP \multimap (NP \multimap S) \\
\mathsf{U} & := & \lambda p.\, \lambda q.\, \exists x.\, (p\, x) \wedge (q\, x) & : & N \multimap NP \\
\mathsf{Q} & := & \lambda r.\, \lambda p.\, \lambda x.\, (p\, x) \wedge (r\,(\lambda k.\, k\, x)) & : & (NP \multimap P) \multimap (N \multimap N)
\end{array}
$$

Applying $\mathcal{L}_2$ to

$$\mathsf{L}\,(\mathsf{U}\,(\mathsf{Q}\,(\lambda x.\, \mathsf{AE}\, x\, \mathsf{M})\, \mathsf{A}))\, \mathsf{P}$$

$\mathcal{L}_2 : \Sigma_0 \to \Sigma_2$

$$
\begin{array}{rcl}
S & := & o; \\
N & := & \iota \multimap o; \\
NP & := & (\iota \multimap o) \multimap o;
\end{array}
$$

$$
\begin{array}{rclcl}
\mathsf{P} & := & \lambda k.\, k\, \mathbf{p} & : & NP \\
\mathsf{M} & := & \lambda k.\, k\, \mathbf{m} & : & NP \\
\mathsf{A} & := & \lambda x.\, \mathsf{article}\, x & : & N \\
\mathsf{L} & := & \lambda p.\, \lambda q.\, p\, (\lambda x.\, q\, (\lambda y.\, \mathsf{read}\, y\, x)) & : & NP \multimap (NP \multimap S) \\
\mathsf{AE} & := & \lambda p.\, \lambda q.\, p\, (\lambda x.\, q\, (\lambda y.\, \mathsf{wrote}\, y\, x)) & : & NP \multimap (NP \multimap S) \\
\mathsf{U} & := & \lambda p.\, \lambda q.\, \exists x.\, (p\, x) \wedge (q\, x) & : & N \multimap NP \\
\mathsf{Q} & := & \lambda r.\, \lambda p.\, \lambda x.\, (p\, x) \wedge (r\, (\lambda k.\, k\, x)) & : & (NP \multimap P) \multimap (N \multimap N)
\end{array}
$$

Applying $\mathcal{L}_2$ to

$$\mathsf{L}\, (\mathsf{U}\, (\mathsf{Q}\, (\lambda x.\, \mathsf{AE}\, x\, \mathsf{M})\, \mathsf{A}))\, \mathsf{P}$$

yields a term that β-reduces to:

$$\exists x.(\mathsf{article}\, x) \wedge (\mathsf{wrote}\, \mathbf{m}\, x) \wedge (\mathsf{read}\, \mathbf{p}\, x)$$

Formal properties

Abstract language membership

Abstract language membership

Straightforward: linear type checking.

Abstract language membership

Straightforward: linear type checking.

Object language membership

Abstract language membership

Straightforward: linear type checking.

Object language membership

For lexicalized grammars: decidable.

Abstract language membership

Straightforward: linear type checking.

Object language membership

For lexicalized grammars: decidable.

In general: open (implies decidability of MELL).

Abstract language membership

Straightforward: linear type checking.

Object language membership

For lexicalized grammars: decidable.

In general: open (implies decidability of MELL).

Abstract/Object language emptiness

Abstract language membership

Straightforward: linear type checking.

Object language membership

For lexicalized grammars: decidable.

In general: open (implies decidability of MELL).

Abstract/Object language emptiness

open (equivalent to MELL decidability).

Expressive power

Abstract Categorial Hierarchy.

Abstract Categorial Hierarchy.

Define $\mathbf{G}(n, m)$ be the set of grammars $\mathcal{G} = \langle \Sigma_1, \Sigma_2, \mathcal{L}, s \rangle$ such that:

$\Sigma_1 = \langle A, C, \tau \rangle$;

$\Sigma_2 = STRING$;

$\mathcal{L}(s) = \mathit{string}$;

n is the order of Σ_1 (i.e., the maximal order of the types in $\tau(c)$);

m is the order of $\mathcal{L}$ (i.e., the maximal order of the types in $\mathcal{L}(A)$, consisdering string as atomic).

Abstract Categorial Hierarchy.

Define $\mathbf{G}(n,m)$ be the set of grammars $\mathcal{G} = \langle \Sigma_1, \Sigma_2, \mathcal{L}, s \rangle$ such that:

$\Sigma_1 = \langle A, C, \tau \rangle$;

$\Sigma_2 = STRING$;

$\mathcal{L}(s) = string$;

n is the order of Σ_1 (i.e., the maximal order of the types in $\tau(c)$);

m is the order of $\mathcal{L}$ (i.e., the maximal order of the types in $\mathcal{L}(A)$, considering $string$ as atomic).

Define $\mathbf{L}(n,m)$ be the set of object languages generated by the grammars in $\mathbf{G}(n,m)$.

$\mathbf{L}(n, m+1) \subset \mathbf{L}(n+1, m)$

$\mathbf{L}(n, m+1) \subset \mathbf{L}(n+1, m)$

$\mathbf{L}(n+1, m) \subset \mathbf{L}(n, m+1)$, for $n \geq 3$.

$\mathbf{L}(n, m+1) \subset \mathbf{L}(n+1, m)$

$\mathbf{L}(n+1, m) \subset \mathbf{L}(n, m+1)$, for $n \geq 3$.

$\mathbf{L}(1, m) = \mathbf{L}(1, 1) =$ FL.

$\mathbf{L}(n, m+1) \subset \mathbf{L}(n+1, m)$

$\mathbf{L}(n+1, m) \subset \mathbf{L}(n, m+1)$, for $n \geq 3$.

$\mathbf{L}(1, m) = \mathbf{L}(1, 1) =$ FL.

$\mathbf{L}(2, 1) =$ CFSL.

$\mathbf{L}(n, m+1) \subset \mathbf{L}(n+1, m)$

$\mathbf{L}(n+1, m) \subset \mathbf{L}(n, m+1)$, for $n \geq 3$.

$\mathbf{L}(1, m) = \mathbf{L}(1, 1) =$ FL.

$\mathbf{L}(2, 1) =$ CFSL.

$\mathbf{L}(2, 2) \supset$ yields of linear CFTL.

$\mathbf{L}(n, m+1) \subset \mathbf{L}(n+1, m)$

$\mathbf{L}(n+1, m) \subset \mathbf{L}(n, m+1)$, for $n \geq 3$.

$\mathbf{L}(1, m) = \mathbf{L}(1, 1) =$ FL.

$\mathbf{L}(2, 1) =$ CFSL.

$\mathbf{L}(2, 2) \supset$ yields of linear CFTL.

$\mathbf{L}(2, 3) \supset$ LCFRS.

$\mathbf{L}(n, m+1) \subset \mathbf{L}(n+1, m)$

$\mathbf{L}(n+1, m) \subset \mathbf{L}(n, m+1)$, for $n \geq 3$.

$\mathbf{L}(1, m) = \mathbf{L}(1, 1) =$ FL.

$\mathbf{L}(2, 1) =$ CFSL.

$\mathbf{L}(2, 2) \supset$ yields of linear CFTL.

$\mathbf{L}(2, 3) \supset$ LCFRS.

$\mathbf{L}(2, n) \subset$ PTIME.