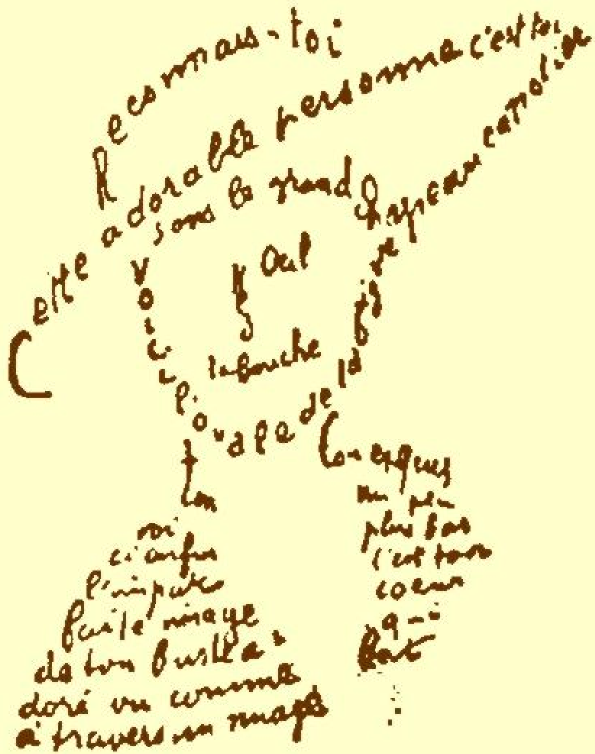
Reconnais-toi
Cette adorable personne c'est toi
sous le grand chapeau canotier
Oeil
la bouche
voici l'ovale de ta figure
ton cou exquis
voici enfin l'imparfaite image de ton buste adoré vu comme à travers un nuage
un peu plus bas c'est ton coeur qui bat

Abstract Categorial Grammar Parsing

the general case

in Honor of Gérard Huet

Philippe de Groote

Inria-Lorraine

Content

Content

Content

Content

Content

Definition

Motivations

Motivations

- To provide a type-theoretic notion of grammar, taking advantages of ideas by Curry and Lambek.

Motivations

- To provide a type-theoretic notion of grammar, taking advantages of ideas by Curry and Lambek.
- To provide a grammatical framework in which other existing grammatical models may be encoded.

Motivations

- To provide a type-theoretic notion of grammar, taking advantages of ideas by Curry and Lambek.
- To provide a grammatical framework in which other existing grammatical models may be encoded.
- To see the parse-structures as first-class citizen.

Motivations

- To provide a type-theoretic notion of grammar, taking advantages of ideas by Curry and Lambek.
- To provide a grammatical framework in which other existing grammatical models may be encoded.
- To see the parse-structures as first-class citizen.
- To allow the user to define grammatical composition combinators.

Motivations

- To provide a type-theoretic notion of grammar, taking advantages of ideas by Curry and Lambek.
- To provide a grammatical framework in which other existing grammatical models may be encoded.
- To see the parse-structures as first-class citizen.
- To allow the user to define grammatical composition combinators.
- To base the formalism on a small set of mathematical primitives that combine via simple composition rules.

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 $\boldsymbol{\Sigma} = \langle A, C, \tau \rangle$, 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 $\mathbf{\Sigma} = \langle A, C, \tau \rangle$, 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 $\boldsymbol{\Sigma} = \langle A, C, \tau \rangle$, 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;

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

$\Lambda(\Sigma)$ denotes the set of linear λ-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 $\mathcal{L} = \langle \eta, \theta \rangle$ from $\mathbf{\Sigma}_1$ to $\mathbf{\Sigma}_2$ is made of two functions:

$\eta : A_1 \to \mathcal{T}(A_2)$,

$\theta : C_1 \to \Lambda(\Sigma_2)$,

such that

$$\vdash_{\Sigma_2} \theta(c) : \eta(\tau_1(c)).$$

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

$$\mathcal{G} = \langle \mathbf{\Sigma}_1, \mathbf{\Sigma}_2, \mathcal{L}, s \rangle$$

where :

$\mathbf{\Sigma}_1 = \langle A_1, C_1, \tau_1 \rangle$ and $\mathbf{\Sigma}_2 = \langle A_2, C_2, \tau_2 \rangle$ are two higher-order linear signatures; $\mathbf{\Sigma}_1$ is called the abstract vocabulary and $\mathbf{\Sigma}_2$ is called the object vocabulary;

Definition

An abstract categorial grammar is a quadruple

$$\mathcal{G} = \langle \mathbf{\Sigma}_1, \mathbf{\Sigma}_2, \mathcal{L}, s \rangle$$

where :

- $\mathbf{\Sigma}_1 = \langle A_1, C_1, \tau_1 \rangle$ and $\mathbf{\Sigma}_2 = \langle A_2, C_2, \tau_2 \rangle$ are two higher-order linear signatures; $\mathbf{\Sigma}_1$ is called the abstract vocabulary and $\mathbf{\Sigma}_2$ is called the object vocabulary;

- $\mathcal{L} : \mathbf{\Sigma}_1 \rightarrow \mathbf{\Sigma}_2$ is a lexicon from the abstract vocabulary to the object vocabulary;

Definition

An abstract categorial grammar is a quadruple

$$\mathcal{G} = \langle \mathbf{\Sigma}_1, \mathbf{\Sigma}_2, \mathcal{L}, s \rangle$$

where :

- $\mathbf{\Sigma}_1 = \langle A_1, C_1, \tau_1 \rangle$ and $\mathbf{\Sigma}_2 = \langle A_2, C_2, \tau_2 \rangle$ are two higher-order linear signatures; $\mathbf{\Sigma}_1$ is called the abstract vocabulary and $\mathbf{\Sigma}_2$ is called the object vocabulary;

- $\mathcal{L} : \mathbf{\Sigma}_1 \to \mathbf{\Sigma}_2$ 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:

$$\mathcal{A}(\mathcal{G}) = \{t \in \Lambda(\mathbf{\Sigma}_1) \mid \vdash_{\Sigma_1} t : s \text{ is derivable}\}$$

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 $\mathcal{L}$:

$$\mathcal{O}(\mathcal{G}) = \{t \in \Lambda(\mathbf{\Sigma}_2) \mid \exists u \in \mathcal{A}(\mathcal{G}).\ t = \mathcal{L}(u)\}$$

Some properties

Some properties

- Membership is decidable if and only if Multiplicative Exponential Linear Logic is decidable.

Some properties

- Membership is decidable if and only if Multiplicative Exponential Linear Logic is decidable.
- Membership for lexicalized ACGs is NP-complete.

Some properties

- Membership is decidable if and only if Multiplicative Exponential Linear Logic is decidable.
- Membership for lexicalized ACGs is NP-complete.
- Membership for second-order ACGs is polynomial.

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:

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

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:

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

Signatures

Signatures

$$\Sigma_0: \quad \begin{array}{rcl} N, NP, S & : & \mathsf{type} \\ \mathsf{J} & : & NP \\ \mathsf{U} & : & N \\ \mathsf{A} & : & N \multimap ((NP \multimap S) \multimap S) \\ \mathsf{S} & : & ((NP \multimap S) \multimap S) \multimap (NP \multimap S) \end{array}$$

Signatures

$$
\Sigma_0: \quad \begin{array}{rcl}
N, NP, S & : & \text{type} \\
\mathsf{J} & : & NP \\
\mathsf{U} & : & N \\
\mathsf{A} & : & N \multimap ((NP \multimap S) \multimap S) \\
\mathsf{S} & : & ((NP \multimap S) \multimap S) \multimap (NP \multimap S)
\end{array}
$$

$$
\Sigma_1: \quad \text{a, John, seeks, unicorn} \; : \; STRING
$$

Signatures

Σ_0:

$$\begin{array}{rcl} N, NP, S & : & \text{type} \\ \mathsf{J} & : & NP \\ \mathsf{U} & : & N \\ \mathsf{A} & : & N \multimap ((NP \multimap S) \multimap S) \\ \mathsf{S} & : & ((NP \multimap S) \multimap S) \multimap (NP \multimap S) \end{array}$$

Σ_1: a, John, seeks, unicorn : $STRING$

Σ_2:

$$\begin{array}{rcl} \iota, o & : & \text{type} \\ \wedge & : & o \multimap (o \multimap o) \\ \exists & : & (\iota \rightarrow o) \multimap o \\ \mathbf{j} & : & \iota \\ \mathbf{unicorn} & : & \iota \multimap o \\ \mathbf{find} & : & \iota \multimap (\iota \multimap o) \\ \mathbf{try} & : & \iota \multimap ((\iota \multimap o) \multimap o) \end{array}$$

Lexicons

Lexicons

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

Lexicons

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

$$\begin{array}{rcl} N, NP, S & := & STRING \\ \mathsf{J} & := & /\mathsf{John}/ \\ \mathsf{U} & := & /\mathsf{unicorn}/ \\ \mathsf{A} & := & \lambda x.\, \lambda p.\, p\,(/\mathsf{a}/ + x) \\ \mathsf{S} & := & \lambda p.\, \lambda x.\, p\,(\lambda y.\, x + /\mathsf{seeks}/ + y) \end{array}$$

Lexicons

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

$$
\begin{array}{rcl}
N, NP, S & := & STRING \\
\mathsf{J} & := & /\mathsf{John}/ \\
\mathsf{U} & := & /\mathsf{unicorn}/ \\
\mathsf{A} & := & \lambda x.\, \lambda p.\, p\,(/\mathsf{a}/ + x) \\
\mathsf{S} & := & \lambda p.\, \lambda x.\, p\,(\lambda y.\, x + /\mathsf{seeks}/ + y)
\end{array}
$$

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

Lexicons

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

$$
\begin{array}{rcl}
N, NP, S & := & STRING \\
\mathsf{J} & := & /\mathsf{John}/ \\
\mathsf{U} & := & /\mathsf{unicorn}/ \\
\mathsf{A} & := & \lambda x.\, \lambda p.\, p\,(/\mathsf{a}/ + x) \\
\mathsf{S} & := & \lambda p.\, \lambda x.\, p\,(\lambda y.\, x + /\mathsf{seeks}/ + y)
\end{array}
$$

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

$$
\begin{array}{rcl}
N & := & i \rightarrow o \\
NP & := & i \\
S & := & o \\
\mathsf{J} & := & \mathbf{j} \\
\mathsf{U} & := & \lambda x.\, \mathbf{unicorn}\, x \\
\mathsf{A} & := & \lambda p.\, \lambda q.\, \exists x.\, p\, x \land q\, x \\
\mathsf{S} & := & \lambda p.\, \lambda x.\, \mathbf{try}\, x\, (\lambda y.\, p\, (\lambda z.\, \mathbf{find}\, y\, z))
\end{array}
$$

We have that:

We have that:

$$\mathcal{L}_1(\mathsf{S}\,(\mathsf{A}\,\mathsf{U})\,\mathsf{J}) = /\mathsf{John}/ + /\mathsf{seeks}/ + /\mathsf{a}/ + /\mathsf{unicorn}/$$

We have that:

$$\mathcal{L}_1(\mathsf{S}\,(\mathsf{A}\,\mathsf{U})\,\mathsf{J}) = /\mathsf{John}/ + /\mathsf{seeks}/ + /\mathsf{a}/ + /\mathsf{unicorn}/$$

$$\mathcal{L}_2(\mathsf{S}\,(\mathsf{A}\,\mathsf{U})\,\mathsf{J}) = \mathbf{try}\,\mathbf{j}\,(\lambda x.\,\exists y.\,\mathbf{unicorn}\,y \wedge \mathbf{find}\,x\,y)$$

We have that:

$$\mathcal{L}_1(\mathsf{S}\,(\mathsf{A}\,\mathsf{U})\,\mathsf{J}) = /\mathsf{John}/ + /\mathsf{seeks}/ + /\mathsf{a}/ + /\mathsf{unicorn}/$$

$$\mathcal{L}_2(\mathsf{S}\,(\mathsf{A}\,\mathsf{U})\,\mathsf{J}) = \mathbf{try}\,\mathbf{j}\,(\lambda x.\,\exists y.\,\mathbf{unicorn}\,y \wedge \mathbf{find}\,x\,y)$$

$$\mathcal{L}_1(\mathsf{A}\,\mathsf{U}\,(\lambda x.\,\mathsf{S}\,(\lambda k.\,k\,x)\,\mathsf{J})) = /\mathsf{John}/ + /\mathsf{seeks}/ + /\mathsf{a}/ + /\mathsf{unicorn}/$$

We have that:

$$\mathcal{L}_1(\mathsf{S}\,(\mathsf{A}\,\mathsf{U})\,\mathsf{J}) = /\mathsf{John}/ + /\mathsf{seeks}/ + /\mathsf{a}/ + /\mathsf{unicorn}/$$

$$\mathcal{L}_2(\mathsf{S}\,(\mathsf{A}\,\mathsf{U})\,\mathsf{J}) = \mathbf{try}\,\mathbf{j}\,(\lambda x.\,\exists y.\,\mathbf{unicorn}\,y \wedge \mathbf{find}\,x\,y)$$

$$\mathcal{L}_1(\mathsf{A}\,\mathsf{U}\,(\lambda x.\,\mathsf{S}\,(\lambda k.\,k\,x)\,\mathsf{J})) = /\mathsf{John}/ + /\mathsf{seeks}/ + /\mathsf{a}/ + /\mathsf{unicorn}/$$

$$\mathcal{L}_2(\mathsf{A}\,\mathsf{U}\,(\lambda x.\,\mathsf{S}\,(\lambda k.\,k\,x)\,\mathsf{J})) = \exists y.\,\mathbf{unicorn}\,y \wedge \mathbf{try}\,\mathbf{j}\,(\lambda x.\,\mathbf{find}\,x\,y)$$

A language-theoretic example

A language-theoretic example

Abstract vocabulary:

$$
\begin{array}{rcl}
A, L, S & : & \text{type} \\
H & : & (A \multimap A \multimap A \multimap S) \multimap S \\
I & : & L \multimap S \\
E & : & L \\
C & : & A \multimap L \multimap L
\end{array}
$$

A language-theoretic example

Abstract vocabulary:

$$
\begin{array}{rcl}
A, L, S & : & \text{type} \\
H & : & (A \multimap A \multimap A \multimap S) \multimap S \\
I & : & L \multimap S \\
E & : & L \\
C & : & A \multimap L \multimap L
\end{array}
$$

Lexicon:

$$
\begin{array}{rcl}
A, L, S & := & \mathit{string} \\
H & := & \lambda f.\, f\, /a/\ /b/\ /c/ \\
I & := & \lambda f.\, \lambda x.\, f\, x \\
E & := & \epsilon \\
C & := & \lambda x.\, \lambda y.\, x + y
\end{array}
$$

A language-theoretic example

Abstract vocabulary:

$$\begin{array}{rcl} A, L, S & : & \text{type} \\ H & : & (A \multimap A \multimap A \multimap S) \multimap S \\ I & : & L \multimap S \\ E & : & L \\ C & : & A \multimap L \multimap L \end{array}$$

Lexicon:

$$\begin{array}{rcl} A, L, S & := & \mathit{string} \\ H & := & \lambda f.\, f\, /a/\, /b/\, /c/ \\ I & := & \lambda f.\, \lambda x.\, f\, x \\ E & := & \epsilon \\ C & := & \lambda x.\, \lambda y.\, x + y \end{array}$$

Typically:

$$H\,(\lambda x_{11} x_{12} x_{13}.\, H\,(\lambda x_{21} x_{22} x_{23}.\, \ldots I\,(C\, x_{ij}\,(C\, x_{kl}\, \ldots (C\, x_{mn}\, E) \ldots)) \ldots)) : S$$

Some Key Properties

Curry-Howard isomorphism

Curry-Howard isomorphism

Coherence theorem

Curry-Howard isomorphism

Coherence theorem

Principal typing

Curry-Howard isomorphism

Coherence theorem

Principal typing

Subject reduction

Curry-Howard isomorphism

Coherence theorem

Principal typing

Subject reduction

Subject expansion

Constructing a Parsing Algorithm

Back to the example

$$
\begin{array}{rcll}
H & := & \lambda f.\, f\,(\lambda z.\, a\, z)\,(\lambda z.\, b\, z)\,(\lambda z.\, c\, z) & : \; (A \multimap A \multimap A \multimap S) \multimap S \\
I & := & \lambda f.\, \lambda x.\, f\, x & : \; L \multimap S \\
E & := & \lambda x.\, x & : \; L \\
C & := & \lambda x.\, \lambda y.\, \lambda z.\, x\,(y\, z) & : \; A \multimap L \multimap L
\end{array}
$$

$$
A, L, S \;:=\; s \multimap s
$$

Back to the example

$$
\begin{array}{rcll}
H & := & \lambda f.\, f\,(\lambda z.\, a\, z)\,(\lambda z.\, b\, z)\,(\lambda z.\, c\, z) & : \; (A \multimap A \multimap A \multimap S) \multimap S \\
I & := & \lambda f.\, \lambda x.\, f\, x & : \; L \multimap S \\
E & := & \lambda x.\, x & : \; L \\
C & := & \lambda x.\, \lambda y.\, \lambda z.\, x\,(y\, z) & : \; A \multimap L \multimap L \\
 & & & \\
A, L, S & := & s \multimap s &
\end{array}
$$

$$\lambda z.\, a\,(c\,(b\,(a\,(b\,(c\, z)))))\ ?$$

A first non deterministic algorithm

A first non deterministic algorithm

1. Try to prove S using the types of the abstract constants as proper axioms.

A first non deterministic algorithm

1. Try to prove S using the types of the abstract constants as proper axioms.

I.e, prove S using $(A \multimap A \multimap A \multimap S) \multimap S$, $L \multimap S$, L, and $A \multimap L \multimap L$.

A first non deterministic algorithm

1. Try to prove S using the types of the abstract constants as proper axioms.

I.e, prove S using $(A \multimap A \multimap A \multimap S) \multimap S$, $L \multimap S$, L, and $A \multimap L \multimap L$.

2. By the Curry-Howard isomorphism, you have constructed a term of the abstract language. Apply the lexicon to this term.

A first non deterministic algorithm

1. Try to prove S using the types of the abstract constants as proper axioms.

I.e, prove S using $(A \multimap A \multimap A \multimap S) \multimap S$, $L \multimap S$, L, and $A \multimap L \multimap L$.

2. By the Curry-Howard isomorphism, you have constructed a term of the abstract language. Apply the lexicon to this term.

3. Check whether the resulting object term is equal to the term you have to parse.

The Coherence Theorem comes in

The Coherence Theorem comes in

1. Specialize the object signature by distinguishing between the different occurrences of a same object constant in the term to be parsed:

The Coherence Theorem comes in

1. Specialize the object signature by distinguishing between the different occurrences of a same object constant in the term to be parsed:

$$\begin{array}{rcl} a_1 & : & s_5 \multimap s_6 \\ a_2 & : & s_2 \multimap s_3 \\ b_1 & : & s_3 \multimap s_4 \\ b_2 & : & s_1 \multimap s_2 \\ c_1 & : & s_4 \multimap s_5 \\ c_2 & : & s_0 \multimap s_1 \end{array}$$

$$\lambda z.\, a_1\,(c_1\,(b_1\,(a_2\,(b_2\,(c_2\,z))))) : s_0 \multimap s_6$$

The Coherence Theorem comes in

1. Specialize the object signature by distinguishing between the different occurrences of a same object constant in the term to be parsed:

$$
\begin{array}{lcl}
a_1 & : & s_5 \multimap s_6 \\
a_2 & : & s_2 \multimap s_3 \\
b_1 & : & s_3 \multimap s_4 \\
b_2 & : & s_1 \multimap s_2 \\
c_1 & : & s_4 \multimap s_5 \\
c_2 & : & s_0 \multimap s_1
\end{array}
$$

$$\lambda z.\, a_1\,(c_1\,(b_1\,(a_2\,(b_2\,(c_2\,z))))) : s_0 \multimap s_6$$

2. Specialize the lexical entries accordingly:

$$
\begin{array}{rcl}
\lambda f.\, f\,(\lambda z.\, a_1\, z)\,(\lambda z.\, b_1\, z)\,(\lambda z.\, c_1\, z) & : & \cdots \\
\lambda f.\, f\,(\lambda z.\, a_1\, z)\,(\lambda z.\, b_1\, z)\,(\lambda z.\, c_2\, z) & : & \cdots \\
\ldots & : & \ldots
\end{array}
$$

3. Try to prove $\langle S, s_0 \multimap s_6\rangle$ using:

$$\langle (A \multimap A \multimap A \multimap S) \multimap S,$$
$$((s_5 \multimap s_6) \multimap (s_3 \multimap s_4) \multimap (s_4 \multimap s_5) \multimap (s_0 \multimap s_0)) \multimap (s_0 \multimap s_0)\rangle$$
$$\langle (A \multimap A \multimap A \multimap S) \multimap S,$$
$$((s_5 \multimap s_6) \multimap (s_3 \multimap s_4) \multimap (s_4 \multimap s_5) \multimap (s_0 \multimap s_1)) \multimap (s_0 \multimap s_1)\rangle$$
$$\vdots$$
$$\langle (A \multimap A \multimap A \multimap S) \multimap S,$$
$$((s_5 \multimap s_6) \multimap (s_3 \multimap s_4) \multimap (s_0 \multimap s_1) \multimap (s_0 \multimap s_0)) \multimap (s_0 \multimap s_0)\rangle$$
$$\langle (A \multimap A \multimap A \multimap S) \multimap S,$$
$$((s_5 \multimap s_6) \multimap (s_3 \multimap s_4) \multimap (s_0 \multimap s_1) \multimap (s_0 \multimap s_1)) \multimap (s_0 \multimap s_1)\rangle$$
$$\vdots$$
$$\langle L \multimap S, (s_0 \multimap s_0) \multimap (s_0 \multimap s_0)\rangle$$
$$\langle L \multimap S, (s_0 \multimap s_1) \multimap (s_0 \multimap s_1)\rangle$$
$$\vdots$$

Eliminating redundancies

Consider the following pair:

$$\langle (A \multimap A \multimap A \multimap S) \multimap S, ((s_5 \multimap s_6) \multimap (s_3 \multimap s_4) \multimap (s_4 \multimap s_5) \multimap (s_0 \multimap s_0)) \multimap (s_0 \multimap s_0) \rangle$$

Eliminating redundancies

Consider the following pair:

$$\langle (A \multimap A \multimap A \multimap S) \multimap S, ((s_5 \multimap s_6) \multimap (s_3 \multimap s_4) \multimap (s_4 \multimap s_5) \multimap (s_0 \multimap s_0)) \multimap (s_0 \multimap s_0) \rangle$$

The shape of the specialized object type is completely specified by the grammar. The only relevant information is given by the indices.

Eliminating redundancies

Consider the following pair:

$\langle (A \multimap A \multimap A \multimap S) \multimap S, ((s_5 \multimap s_6) \multimap (s_3 \multimap s_4) \multimap (s_4 \multimap s_5) \multimap (s_0 \multimap s_0)) \multimap (s_0 \multimap s_0) \rangle$

The shape of the specialized object type is completely specified by the grammar. The only relevant information is given by the indices.

Replace the above pair by the following formula:

$$(A[5,6] \multimap A[3,4] \multimap A[4,5] \multimap S[0,0]) \multimap S[0,0]$$

Principal typing

Principal typing

Factorize the several formulas coming from a given lexical entry,

$$
\begin{array}{l}
(A[5,6] \multimap A[3,4] \multimap A[4,5] \multimap S[0,0]) \multimap S[0,0] \\
(A[5,6] \multimap A[3,4] \multimap A[4,5] \multimap S[0,1]) \multimap S[0,1] \\
\quad \vdots \\
(A[5,6] \multimap A[3,4] \multimap A[0,1] \multimap S[0,0]) \multimap S[0,0] \\
(A[5,6] \multimap A[3,4] \multimap A[0,1] \multimap S[0,1]) \multimap S[0,1] \\
\quad \vdots
\end{array}
$$

Principal typing

Factorize the several formulas coming from a given lexical entry,

$$
\begin{array}{l}
(A[5,6] \multimap A[3,4] \multimap A[4,5] \multimap S[0,0]) \multimap S[0,0] \\
(A[5,6] \multimap A[3,4] \multimap A[4,5] \multimap S[0,1]) \multimap S[0,1] \\
\quad\vdots \\
(A[5,6] \multimap A[3,4] \multimap A[0,1] \multimap S[0,0]) \multimap S[0,0] \\
(A[5,6] \multimap A[3,4] \multimap A[0,1] \multimap S[0,1]) \multimap S[0,1] \\
\quad\vdots
\end{array}
$$

as follows:

$$a[i,j], b[k,l], c[m,n] \vdash (A[i,j] \multimap A[k,l] \multimap A[m,n] \multimap S[o,p]) \multimap S[o,p]$$

We end up with the following proof search problem:

We end up with the following proof search problem:

Formulas coming from the lexicon:

$$a[i,j], b[k,l], c[m,n] \vdash (A[i,j] \multimap A[k,l] \multimap A[m,n] \multimap S[o,p]) \multimap S[o,p]$$

$$\vdash L[i,j] \multimap S[i,j]$$

$$\vdash L[i,i]$$

$$\vdash A[i,j] \multimap L[k,i] \multimap L[k,j]$$

Query (coming from the term to be parsed):

$$a[5,6], c[4,5], b[3,4], a[2,3], b[1,2], c[0,1] \vdash S[0,6]$$

Correctness and Completeness

Correctness and Completeness

Correctness : by subject reduction.

Correctness and Completeness

Correctness : by subject reduction.

Completeness : by subject expansion.

The second-order case
— Kanazawa's original construction

The second-order case — Kanazawa's original construction

- Allow the lexicons to be compiled into datalog programs.

The second-order case
— Kanazawa's original construction

- Allow the lexicons to be compiled into datalog programs.
- Polynomiality of 2nd-order ACGs.

The second-order case
— Kanazawa's original construction

- Allow the lexicons to be compiled into datalog programs.
- Polynomiality of 2nd-order ACGs.
- Optimizations techniques are known.

The second-order case
— Kanazawa's original construction

- Allow the lexicons to be compiled into datalog programs.
- Polynomiality of 2nd-order ACGs.
- Optimizations techniques are known.
- CFG, TAG, LCFRS, ... as 2nd-order ACGs.