Structures Informatiques et Logiques pour la Modélisation Linguistique (MPRI 2.27.1 - second part)

Philippe de Groote

Inria

2021-2022

Syntax/semantics Interface

Syntax/semantics Interface

Compositionality

Compositionality principle

- The meaning of a complex expression is determined by the meanings of its constituents and by the formation rules used to combine them.

Montague's homomorphism requirement

- Semantics must be obtained as a homomorphic image of syntax.

Contextuality principle

- The meaning of an expression is determined by the meanings of the complex expressions of which it is a constituent.

Compositionality

Compositionality principle

- The meaning of a complex expression is determined by the meanings of its constituents and by the formation rules used to combine them.

Montague's homomorphism requirement

- Semantics must be obtained as a homomorphic image of syntax.

Contextuality principle

- The meaning of an expression is determined by the meanings of the complex expressions of which it is a constituent.

Compositionality

Compositionality principle

- The meaning of a complex expression is determined by the meanings of its constituents and by the formation rules used to combine them.

Montague's homomorphism requirement

- Semantics must be obtained as a homomorphic image of syntax.

Contextuality principle

- The meaning of an expression is determined by the meanings of the complex expressions of which it is a constituent.

Compositionality

Compositionality principle

- The meaning of a complex expression is determined by the meanings of its constituents and by the formation rules used to combine them.

Montague's homomorphism requirement

- Semantics must be obtained as a homomorphic image of syntax.

Contextuality principle

- The meaning of an expression is determined by the meanings of the complex expressions of which it is a constituent.

Compositionality

Compositionality principle

- The meaning of a complex expression is determined by the meanings of its constituents and by the formation rules used to combine them.

Montague's homomorphism requirement

- Semantics must be obtained as a homomorphic image of syntax.

Contextuality principle

- The meaning of an expression is determined by the meanings of the complex expressions of which it is a constituent.

Compositionality

Compositionality principle

- The meaning of a complex expression is determined by the meanings of its constituents and by the formation rules used to combine them.

Montague's homomorphism requirement

- Semantics must be obtained as a homomorphic image of syntax.

Contextuality principle

- The meaning of an expression is determined by the meanings of the complex expressions of which it is a constituent.

Compositionality

Compositionality principle

- The meaning of a complex expression is determined by the meanings of its constituents and by the formation rules used to combine them.

Montague's homomorphism requirement

- Semantics must be obtained as a homomorphic image of syntax.

Contextuality principle

- The meaning of an expression is determined by the meanings of the complex expressions of which it is a constituent.

Syntax/semantics Interface

Syntax/semantics Interface

Rule to rule semantics

Context free grammar:

S → NP VP
VP → tV NP
tV → loves
NP → John
NP → somebody

Semantic rules:

$$[\![\mathrm{S}]\!] = [\![\mathrm{NP}]\!]\,[\![\mathrm{VP}]\!]$$
$$[\![\mathrm{VP}]\!] = \lambda x.\,[\![\mathrm{NP}]\!]\,(\lambda y.\,[\![\mathrm{tV}]\!]\,y\,x)$$
$$[\![\mathrm{tV}]\!] = \lambda y.\,\lambda x.\,\mathbf{love}\,x\,y$$
$$[\![\mathrm{NP}]\!] = \lambda k.\,k\,\mathbf{j}$$
$$[\![\mathrm{NP}]\!] = \lambda k.\,\exists y.\,(\mathbf{human}\,y) \land (k\,y)$$

Rule to rule semantics

Context free grammar:

S → NP VP
VP → tV NP
tV → loves
NP → John
NP → somebody

Semantic rules:

$$[\![\mathrm{S}]\!] = [\![\mathrm{NP}]\!]\,[\![\mathrm{VP}]\!]$$
$$[\![\mathrm{VP}]\!] = \lambda x.\,[\![\mathrm{NP}]\!]\,(\lambda y.\,[\![\mathrm{tV}]\!]\,y\,x)$$
$$[\![\mathrm{tV}]\!] = \lambda y.\,\lambda x.\,\mathbf{love}\,x\,y$$
$$[\![\mathrm{NP}]\!] = \lambda k.\,k\,\mathbf{j}$$
$$[\![\mathrm{NP}]\!] = \lambda k.\,\exists y.\,(\mathbf{human}\,y) \wedge (k\,y)$$

Rule to rule semantics

Context free grammar:

$$
\begin{aligned}
S &\rightarrow NP\ VP \\
VP &\rightarrow tV\ NP \\
tV &\rightarrow \text{loves} \\
NP &\rightarrow \text{John} \\
NP &\rightarrow \text{somebody}
\end{aligned}
$$

Semantic rules:

$$
\begin{aligned}
[\![S]\!] &= [\![NP]\!]\,[\![VP]\!] \\
[\![VP]\!] &= \lambda x.\,[\![NP]\!]\,(\lambda y.\,[\![tV]\!]\,y\,x) \\
[\![tV]\!] &= \lambda y.\,\lambda x.\,\mathbf{love}\,x\,y \\
[\![NP]\!] &= \lambda k.\,k\,\mathbf{j} \\
[\![NP]\!] &= \lambda k.\,\exists y.\,(\mathbf{human}\,y) \wedge (k\,y)
\end{aligned}
$$

Rule to rule semantics

Context free grammar:

$$\text{S} \to \text{NP VP}$$
$$\text{VP} \to \text{tV NP}$$
$$\text{tV} \to \text{loves}$$
$$\text{NP} \to \text{John}$$
$$\text{NP} \to \text{somebody}$$

Semantic rules:

$$[\![\text{S}]\!] = [\![\text{NP}]\!]\,[\![\text{VP}]\!]$$
$$[\![\text{VP}]\!] = \lambda x.\,[\![\text{NP}]\!]\,(\lambda y.\,[\![\text{tV}]\!]\,y\,x)$$
$$[\![\text{tV}]\!] = \lambda y.\,\lambda x.\,\mathbf{love}\,x\,y$$
$$[\![\text{NP}]\!] = \lambda k.\,k\,\mathbf{j}$$
$$[\![\text{NP}]\!] = \lambda k.\,\exists y.\,(\mathbf{human}\,y) \wedge (k\,y)$$

Rule to rule semantics

Context free grammar:

S → NP VP
VP → tV NP
tV → loves
NP → John
NP → somebody

Semantic rules:

$$[\![\mathrm{S}]\!] = [\![\mathrm{NP}]\!]\,[\![\mathrm{VP}]\!]$$
$$[\![\mathrm{VP}]\!] = \lambda x.\,[\![\mathrm{NP}]\!]\,(\lambda y.\,[\![\mathrm{tV}]\!]\,y\,x)$$
$$[\![\mathrm{tV}]\!] = \lambda y.\,\lambda x.\,\mathbf{love}\,x\,y$$
$$[\![\mathrm{NP}]\!] = \lambda k.\,k\,\mathbf{j}$$
$$[\![\mathrm{NP}]\!] = \lambda k.\,\exists y.\,(\mathbf{human}\,y) \wedge (k\,y)$$

Syntax/semantics Interface

Signature associated to a CFG

Context free grammar:

$$
\begin{aligned}
\mathsf{S} &\to \mathsf{NP}\ \mathsf{VP} && (p_1)\\
\mathsf{VP} &\to \mathsf{tV}\ \mathsf{NP} && (p_2)\\
\mathsf{tV} &\to \mathsf{loves} && (p_3)\\
\mathsf{NP} &\to \mathsf{John} && (p_4)\\
\mathsf{NP} &\to \mathsf{somebody} && (p_5)
\end{aligned}
$$

Associate a *sort* to each non-terminal, and an *operator* to each production rule:

$$
\begin{aligned}
p_1 &: NP \times VP \to S\\
p_2 &: tV \times NP \to VP\\
p_3 &: \mathsf{tV}\\
p_4 &: \mathsf{NP}\\
p_5 &: \mathsf{NP}
\end{aligned}
$$

Signature associated to a CFG

Context free grammar:

$$
\begin{array}{ll}
\mathsf{S} \rightarrow \mathsf{NP}\ \mathsf{VP} & (\mathrm{p}_1) \\
\mathsf{VP} \rightarrow \mathsf{tV}\ \mathsf{NP} & (\mathrm{p}_2) \\
\mathsf{tV} \rightarrow \mathsf{loves} & (\mathrm{p}_3) \\
\mathsf{NP} \rightarrow \mathsf{John} & (\mathrm{p}_4) \\
\mathsf{NP} \rightarrow \mathsf{somebody} & (\mathrm{p}_5)
\end{array}
$$

Associate a *sort* to each non-terminal, and an *operator* to each production rule:

$$
\begin{array}{l}
\mathrm{p}_1 : \mathrm{NP} \times \mathrm{VP} \rightarrow \mathrm{S} \\
\mathrm{p}_2 : \mathrm{tV} \times \mathrm{NP} \rightarrow \mathrm{VP} \\
\mathrm{p}_3 : \mathrm{tV} \\
\mathrm{p}_4 : \mathrm{NP} \\
\mathrm{p}_5 : \mathrm{NP}
\end{array}
$$

Signature associated to a CFG

Context free grammar:

$$
\begin{array}{lr}
\mathsf{S} \to \mathsf{NP}\ \mathsf{VP} & (\mathrm{p}_1) \\
\mathsf{VP} \to \mathsf{tV}\ \mathsf{NP} & (\mathrm{p}_2) \\
\mathsf{tV} \to \mathsf{loves} & (\mathrm{p}_3) \\
\mathsf{NP} \to \mathsf{John} & (\mathrm{p}_4) \\
\mathsf{NP} \to \mathsf{somebody} & (\mathrm{p}_5)
\end{array}
$$

Associate a *sort* to each non-terminal, and an *operator* to each production rule:

$$
\begin{array}{l}
\mathrm{p}_1 : \mathrm{NP} \times \mathrm{VP} \to \mathrm{S} \\
\mathrm{p}_2 : \mathrm{tV} \times \mathrm{NP} \to \mathrm{VP} \\
\mathrm{p}_3 : \mathrm{tV} \\
\mathrm{p}_4 : \mathrm{NP} \\
\mathrm{p}_5 : \mathrm{NP}
\end{array}
$$

Signature associated to a CFG

Context free grammar:

$$
\begin{array}{ll}
\mathsf{S} \rightarrow \mathsf{NP}\ \mathsf{VP} & (\mathrm{p}_1) \\
\mathsf{VP} \rightarrow \mathsf{tV}\ \mathsf{NP} & (\mathrm{p}_2) \\
\mathsf{tV} \rightarrow \mathsf{loves} & (\mathrm{p}_3) \\
\mathsf{NP} \rightarrow \mathsf{John} & (\mathrm{p}_4) \\
\mathsf{NP} \rightarrow \mathsf{somebody} & (\mathrm{p}_5)
\end{array}
$$

Associate a *sort* to each non-terminal, and an *operator* to each production rule:

$$
\begin{array}{l}
\mathrm{p}_1 : \mathrm{NP} \times \mathrm{VP} \rightarrow \mathrm{S} \\
\mathrm{p}_2 : \mathrm{tV} \times \mathrm{NP} \rightarrow \mathrm{VP} \\
\mathrm{p}_3 : \mathsf{tV} \\
\mathrm{p}_4 : \mathsf{NP} \\
\mathrm{p}_5 : \mathsf{NP}
\end{array}
$$

Syntax/semantics Interface

Syntactic and semantic algebras

Syntactic algebra:

$$
\begin{aligned}
\mathrm{p}_1 &: \mathrm{NP} \times \mathrm{VP} \to \mathrm{S} \\
\mathrm{p}_2 &: \mathrm{tV} \times \mathrm{NP} \to \mathrm{VP} \\
\mathrm{p}_3 &: \mathsf{tV} \\
\mathrm{p}_4 &: \mathsf{NP} \\
\mathrm{p}_5 &: \mathsf{NP}
\end{aligned}
$$

Semantic algebra:

$$
\begin{aligned}
\mathrm{f}_1(a, b) = a\, b \quad &: \mathrm{NP}^* \times \mathrm{VP}^* \to \mathrm{S}^* \\
\mathrm{f}_2(a, b) = \lambda x.\, b\, (\lambda y.\, a\, y\, x) \quad &: \mathrm{tV}^* \times \mathrm{NP}^* \to \mathrm{VP}^* \\
\mathrm{f}_3 = \lambda y.\, \lambda x.\, \mathbf{love}\, x\, y \quad &: \mathrm{tV}^* \\
\mathrm{f}_4 = \lambda k.\, k\, \mathbf{j} \quad &: \mathrm{NP}^* \\
\mathrm{f}_5 = \lambda k.\, \exists y.\, (\mathbf{human}\, y) \wedge (k\, y) \quad &: \mathrm{NP}^*
\end{aligned}
$$

Syntactic and semantic algebras

Syntactic algebra:

$$
\begin{aligned}
\mathrm{p}_1 &: \mathrm{NP} \times \mathrm{VP} \rightarrow \mathrm{S} \\
\mathrm{p}_2 &: \mathrm{tV} \times \mathrm{NP} \rightarrow \mathrm{VP} \\
\mathrm{p}_3 &: \mathsf{tV} \\
\mathrm{p}_4 &: \mathsf{NP} \\
\mathrm{p}_5 &: \mathsf{NP}
\end{aligned}
$$

Semantic algebra:

$$
\begin{aligned}
\mathrm{f}_1(a,b) = a\,b &: \mathrm{NP}^* \times \mathrm{VP}^* \rightarrow \mathrm{S}^* \\
\mathrm{f}_2(a,b) = \lambda x.\, b\,(\lambda y.\, a\,y\,x) &: \mathrm{tV}^* \times \mathrm{NP}^* \rightarrow \mathrm{VP}^* \\
\mathrm{f}_3 = \lambda y.\, \lambda x.\, \mathbf{love}\,x\,y &: \mathrm{tV}^* \\
\mathrm{f}_4 = \lambda k.\, k\,\mathbf{j} &: \mathrm{NP}^* \\
\mathrm{f}_5 = \lambda k.\, \exists y.\, (\mathbf{human}\,y) \wedge (k\,y) &: \mathrm{NP}^*
\end{aligned}
$$

Syntactic and semantic algebras

Syntactic algebra:

$$
\begin{aligned}
\mathrm{p}_1 &: \mathrm{NP} \times \mathrm{VP} \rightarrow \mathrm{S} \\
\mathrm{p}_2 &: \mathrm{tV} \times \mathrm{NP} \rightarrow \mathrm{VP} \\
\mathrm{p}_3 &: \mathrm{tV} \\
\mathrm{p}_4 &: \mathrm{NP} \\
\mathrm{p}_5 &: \mathrm{NP}
\end{aligned}
$$

Semantic algebra:

$$
\begin{aligned}
\mathrm{f}_1(a,b) = a\,b \quad &: \mathrm{NP}^* \times \mathrm{VP}^* \rightarrow \mathrm{S}^* \\
\mathrm{f}_2(a,b) = \lambda x.\, b\,(\lambda y.\, a\,y\,x) \quad &: \mathrm{tV}^* \times \mathrm{NP}^* \rightarrow \mathrm{VP}^* \\
\mathrm{f}_3 = \lambda y.\, \lambda x.\, \mathbf{love}\,x\,y \quad &: \mathrm{tV}^* \\
\mathrm{f}_4 = \lambda k.\, k\,\mathbf{j} \quad &: \mathrm{NP}^* \\
\mathrm{f}_5 = \lambda k.\, \exists y.\, (\mathbf{human}\,y) \wedge (k\,y) \quad &: \mathrm{NP}^*
\end{aligned}
$$

Syntactic and semantic algebras

Syntactic algebra:

$$\begin{aligned}
\mathrm{p}_1 &: \mathrm{NP} \times \mathrm{VP} \rightarrow \mathrm{S} \\
\mathrm{p}_2 &: \mathrm{tV} \times \mathrm{NP} \rightarrow \mathrm{VP} \\
\mathrm{p}_3 &: \mathsf{tV} \\
\mathrm{p}_4 &: \mathsf{NP} \\
\mathrm{p}_5 &: \mathsf{NP}
\end{aligned}$$

Semantic algebra:

$$\begin{aligned}
\mathrm{f}_1(a, b) = a\, b \quad &: \mathrm{NP}^* \times \mathrm{VP}^* \rightarrow \mathrm{S}^* \\
\mathrm{f}_2(a, b) = \lambda x.\, b\, (\lambda y.\, a\, y\, x) \quad &: \mathrm{tV}^* \times \mathrm{NP}^* \rightarrow \mathrm{VP}^* \\
\mathrm{f}_3 = \lambda y.\, \lambda x.\, \mathbf{love}\, x\, y \quad &: \mathrm{tV}^* \\
\mathrm{f}_4 = \lambda k.\, k\, \mathbf{j} \quad &: \mathrm{NP}^* \\
\mathrm{f}_5 = \lambda k.\, \exists y.\, (\mathbf{human}\, y) \wedge (k\, y) \quad &: \mathrm{NP}^*
\end{aligned}$$

Where:

$$
\begin{aligned}
\mathrm{S}^* &= o \\
\mathrm{VP}^* &= \iota \rightarrow o \\
\mathrm{tV}^* &= \iota \rightarrow \iota \rightarrow o \\
\mathrm{NP}^* &= (\iota \rightarrow o) \rightarrow o
\end{aligned}
$$

Syntax/semantics Interface

Syntax/semantics Interface

Definition

Let $\mathcal{T}(A)$ be the set of functional types built on the set of atomic types A, i.e.:

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

A higher-order signature is a triple $\Sigma = \langle A, C, \tau \rangle$, 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 associate to each constant in C a simple type built on A.

We use $\Lambda(\Sigma)$ to denote the set of simply typed λ-terms built upon a higher-order signature Σ.
We use $\Lambda^0(\Sigma)$ to denote the set of linear λ-terms built upon a higher-order signature Σ.

Definition

Let $\mathcal{T}(A)$ be the set of functional types built on the set of atomic types A, i.e.:

$$\mathcal{T}(A) \quad ::= \quad A \mid (\,\mathcal{T}(A) \rightarrow \mathcal{T}(A)\,)$$

A higher-order signature is a triple $\Sigma = \langle A, C, \tau \rangle$, 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 associate to each constant in C a simple type built on A.

We use $\Lambda(\Sigma)$ to denote the set of simply typed λ-terms built upon a higher-order signature Σ.
We use $\Lambda^0(\Sigma)$ to denote the set of linear λ-terms built upon a higher-order signature Σ.

Definition

Let $\mathcal{T}(A)$ be the set of functional types built on the set of atomic types A, i.e.:

$$\mathcal{T}(A) \quad ::= \quad A \mid (\, \mathcal{T}(A) \rightarrow \mathcal{T}(A)\,)$$

A higher-order signature is a triple $\Sigma = \langle A, C, \tau \rangle$, 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 associate to each constant in C a simple type built on A.

We use $\Lambda(\Sigma)$ to denote the set of simply typed λ-terms built upon a higher-order signature Σ.
We use $\Lambda^0(\Sigma)$ to denote the set of linear λ-terms built upon a higher-order signature Σ.

Definition

Let $\mathcal{T}(A)$ be the set of functional types built on the set of atomic types A, i.e.:

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

A higher-order signature is a triple $\Sigma = \langle A, C, \tau \rangle$, 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 associate to each constant in C a simple type built on A.

We use $\Lambda(\Sigma)$ to denote the set of simply typed λ-terms built upon a higher-order signature Σ.
We use $\Lambda^0(\Sigma)$ to denote the set of linear λ-terms built upon a higher-order signature Σ.

Definition

Let $\mathcal{T}(A)$ be the set of functional types built on the set of atomic types A, i.e.:

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

A higher-order signature is a triple $\Sigma = \langle A, C, \tau \rangle$, 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 associate to each constant in C a simple type built on A.

We use $\Lambda(\Sigma)$ to denote the set of simply typed λ-terms built upon a higher-order signature Σ.
We use $\Lambda^0(\Sigma)$ to denote the set of linear λ-terms built upon a higher-order signature Σ.

Definition

Let $\mathcal{T}(A)$ be the set of functional types built on the set of atomic types A, i.e.:

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

A higher-order signature is a triple $\Sigma = \langle A, C, \tau \rangle$, 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 associate to each constant in C a simple type built on A.

We use $\Lambda(\Sigma)$ to denote the set of simply typed λ-terms built upon a higher-order signature Σ.

We use $\Lambda^0(\Sigma)$ to denote the set of linear λ-terms built upon a higher-order signature Σ.

Definition

Let $\mathcal{T}(A)$ be the set of functional types built on the set of atomic types A, i.e.:

$$\mathcal{T}(A) \quad ::= \quad A \mid (\,\mathcal{T}(A) \rightarrow \mathcal{T}(A)\,)$$

A higher-order signature is a triple $\Sigma = \langle A, C, \tau \rangle$, 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 associate to each constant in C a simple type built on A.

We use $\Lambda(\Sigma)$ to denote the set of simply typed λ-terms built upon a higher-order signature Σ.

We use $\Lambda^0(\Sigma)$ to denote the set of linear λ-terms built upon a higher-order signature Σ.

Definition

Let $\mathcal{T}(A)$ be the set of functional types built on the set of atomic types A, i.e.:

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

A higher-order signature is a triple $\Sigma = \langle A, C, \tau \rangle$, 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 associate to each constant in C a simple type built on A.

We use $\Lambda(\Sigma)$ to denote the set of simply typed λ-terms built upon a higher-order signature Σ.
We use $\Lambda^0(\Sigma)$ to denote the set of linear λ-terms built upon a higher-order signature Σ.

Syntax/semantics Interface

Trees

$$\mathrm{p}_1 : \mathrm{NP} \to \mathrm{VP} \to \mathrm{S}$$
$$\mathrm{p}_2 : \mathrm{tV} \to \mathrm{NP} \to \mathrm{VP}$$
$$\mathrm{p}_3 : \mathsf{tV}$$
$$\mathrm{p}_4 : \mathsf{NP}$$
$$\mathrm{p}_5 : \mathsf{NP}$$

Strings

A canonical way of representing strings as λ-terms consists of representing them as function compositions:

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

In this setting:

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

Strings

A canonical way of representing strings as λ-terms consists of representing them as function compositions:

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

In this setting:

$$\epsilon \triangleq \lambda x.\, x$$

$$\alpha + \beta \triangleq \lambda \alpha.\, \lambda \beta.\, \lambda x.\, \alpha\,(\beta\,x)$$

Strings

A canonical way of representing strings as λ-terms consists of representing them as function compositions:

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

In this setting:

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

First-order logic

$$
\begin{aligned}
\text{zero} &: \text{term} \\
\text{succ} &: \text{term} \rightarrow \text{term} \\
\text{add} &: \text{term} \rightarrow \text{term} \rightarrow \text{term} \\
&\vdots \\
\text{eq} &: \text{term} \rightarrow \text{term} \rightarrow \text{prop} \\
\text{not} &: \text{prop} \rightarrow \text{prop} \\
\text{and} &: \text{prop} \rightarrow \text{prop} \rightarrow \text{prop} \\
\text{forall} &: (\text{term} \rightarrow \text{prop}) \rightarrow \text{prop}
\end{aligned}
$$

First-order logic

$$
\begin{aligned}
\text{zero} &: \text{term} \\
\text{succ} &: \text{term} \rightarrow \text{term} \\
\text{add} &: \text{term} \rightarrow \text{term} \rightarrow \text{term} \\
&\vdots \\
\text{eq} &: \text{term} \rightarrow \text{term} \rightarrow \text{prop} \\
\text{not} &: \text{prop} \rightarrow \text{prop} \\
\text{and} &: \text{prop} \rightarrow \text{prop} \rightarrow \text{prop} \\
\text{forall} &: (\text{term} \rightarrow \text{prop}) \rightarrow \text{prop}
\end{aligned}
$$

linguistic example

$$
\begin{array}{rcl}
& \vdots & \\
\text{a} & : & \text{N} \rightarrow \text{NP} \\
\text{wise} & : & \text{N} \rightarrow \text{N} \\
\text{man} & : & \text{N} \\
\text{who} & : & (\text{NP} \rightarrow \text{S}) \rightarrow \text{N} \rightarrow \text{N} \\
\text{loves} & : & \text{NP} \rightarrow \text{NP} \rightarrow \text{S} \\
\text{himself} & : & (\text{NP} \rightarrow \text{NP} \rightarrow \text{S}) \rightarrow \text{NP} \rightarrow \text{S} \\
& \vdots &
\end{array}
$$

linguistic example

$$
\begin{aligned}
&\vdots \\
\text{a} &: \text{N} \to \text{NP} \\
\text{wise} &: \text{N} \to \text{N} \\
\text{man} &: \text{N} \\
\text{who} &: (\text{NP} \to \text{S}) \to \text{N} \to \text{N} \\
\text{loves} &: \text{NP} \to \text{NP} \to \text{S} \\
\text{himself} &: (\text{NP} \to \text{NP} \to \text{S}) \to \text{NP} \to \text{S} \\
&\vdots
\end{aligned}
$$

Syntax/semantics Interface

Definition

Given two higher-order signatures $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ and $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$, a higher-order homomorphism $\mathcal{H} = \langle \eta, \theta \rangle$ from Σ_1 to Σ_2 is generated by two functions:

- $\eta : A_1 \to \mathcal{T}(A_2)$,
- $\theta : C_1 \to \Lambda(\Sigma_2)$,

such that

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

where $\hat{\eta}$ is the homomorphic extension of η, i.e.:

- $\hat{\eta}(a) = \eta(a)$, for $a \in A_1$.
- $\hat{\eta}(\alpha \to \beta) = \hat{\eta}(\alpha) \to \hat{\eta}(\beta)$.

A higher-order homomorphism is said to be linear when $\theta : C_1 \to \Lambda^0(\Sigma_2)$.

Definition

Given two higher-order signatures $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ and $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$, a higher-order homomorphism $\mathcal{H} = \langle \eta, \theta \rangle$ from Σ_1 to Σ_2 is generated by two functions:

- $\eta : A_1 \to \mathcal{T}(A_2)$,
- $\theta : C_1 \to \Lambda(\Sigma_2)$,

such that

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

where $\hat{\eta}$ is the homomorphic extension of η, i.e.:

- $\hat{\eta}(a) = \eta(a)$, for $a \in A_1$.
- $\hat{\eta}(\alpha \to \beta) = \hat{\eta}(\alpha) \to \hat{\eta}(\beta)$.

A higher-order homomorphism is said to be linear when $\theta : C_1 \to \Lambda^0(\Sigma_2)$.

Definition

Given two higher-order signatures $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ and $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$, a higher-order homomorphism $\mathcal{H} = \langle \eta, \theta \rangle$ from Σ_1 to Σ_2 is generated by two functions:

- $\eta : A_1 \to \mathcal{T}(A_2)$,
- $\theta : C_1 \to \Lambda(\Sigma_2)$,

such that

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

where $\hat{\eta}$ is the homomorphic extension of η, i.e.:

- $\hat{\eta}(a) = \eta(a)$, for $a \in A_1$.
- $\hat{\eta}(\alpha \to \beta) = \hat{\eta}(\alpha) \to \hat{\eta}(\beta)$.

A higher-order homomorphism is said to be linear when $\theta : C_1 \to \Lambda^0(\Sigma_2)$.

Definition

Given two higher-order signatures $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ and $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$, a higher-order homomorphism $\mathcal{H} = \langle \eta, \theta \rangle$ from Σ_1 to Σ_2 is generated by two functions:

- $\eta : A_1 \to \mathcal{T}(A_2)$,
- $\theta : C_1 \to \Lambda(\Sigma_2)$,

such that

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

where $\hat{\eta}$ is the homomorphic extension of η, i.e.:

- $\hat{\eta}(a) = \eta(a)$, for $a \in A_1$.
- $\hat{\eta}(\alpha \to \beta) = \hat{\eta}(\alpha) \to \hat{\eta}(\beta)$.

A higher-order homomorphism is said to be linear when $\theta : C_1 \to \Lambda^0(\Sigma_2)$.

Definition

Given two higher-order signatures $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ and $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$, a higher-order homomorphism $\mathcal{H} = \langle \eta, \theta \rangle$ from Σ_1 to Σ_2 is generated by two functions:

- $\eta : A_1 \to \mathcal{T}(A_2)$,
- $\theta : C_1 \to \Lambda(\Sigma_2)$,

such that

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

where $\hat{\eta}$ is the homomorphic extension of η, i.e.:

- $\hat{\eta}(a) = \eta(a)$, for $a \in A_1$.
- $\hat{\eta}(\alpha \to \beta) = \hat{\eta}(\alpha) \to \hat{\eta}(\beta)$.

A higher-order homomorphism is said to be linear when $\theta : C_1 \to \Lambda^0(\Sigma_2)$.

Definition

Given two higher-order signatures $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ and $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$, a higher-order homomorphism $\mathcal{H} = \langle \eta, \theta \rangle$ from Σ_1 to Σ_2 is generated by two functions:

- $\eta : A_1 \rightarrow \mathcal{T}(A_2)$,
- $\theta : C_1 \rightarrow \Lambda(\Sigma_2)$,

such that

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

where $\hat{\eta}$ is the homomorphic extension of η, i.e.:

- $\hat{\eta}(a) = \eta(a)$, for $a \in A_1$.
- $\hat{\eta}(\alpha \rightarrow \beta) = \hat{\eta}(\alpha) \rightarrow \hat{\eta}(\beta)$.

A higher-order homomorphism is said to be linear when $\theta : C_1 \rightarrow \Lambda^0(\Sigma_2)$.

Definition

Given two higher-order signatures $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ and $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$, a higher-order homomorphism $\mathcal{H} = \langle \eta, \theta \rangle$ from Σ_1 to Σ_2 is generated by two functions:

- $\eta : A_1 \rightarrow \mathcal{T}(A_2)$,
- $\theta : C_1 \rightarrow \Lambda(\Sigma_2)$,

such that

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

where $\hat{\eta}$ is the homomorphic extension of η, i.e.:

- $\hat{\eta}(a) = \eta(a)$, for $a \in A_1$.
- $\hat{\eta}(\alpha \rightarrow \beta) = \hat{\eta}(\alpha) \rightarrow \hat{\eta}(\beta)$.

A higher-order homomorphism is said to be linear when $\theta : C_1 \rightarrow \Lambda^0(\Sigma_2)$.

Syntax/semantics Interface

Syntax/semantics Interface

Vocabularies and Lexicon

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

Given two vocabularies Σ_1 and Σ_2, a lexicon $\mathcal{L}$ from Σ_1 to Σ_2 is defined to be a linear higher-order homomorphism $\mathcal{L} : \Sigma_1 \to \Sigma_2$.

Vocabularies and Lexicon

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

Given two vocabularies Σ_1 and Σ_2, a lexicon $\mathcal{L}$ from Σ_1 to Σ_2 is defined to be a linear higher-order homomorphism $\mathcal{L} : \Sigma_1 \to \Sigma_2$.

Vocabularies and Lexicon

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

Given two vocabularies Σ_1 and Σ_2, a lexicon $\mathcal{L}$ from Σ_1 to Σ_2 is defined to be a linear higher-order homomorphism $\mathcal{L} : \Sigma_1 \to \Sigma_2$.

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.

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;
- $\mathcal{L} : \Sigma_1 \to \Sigma_2$ 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.

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;
- $\mathcal{L} : \Sigma_1 \to \Sigma_2$ 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.

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;
- $\mathcal{L} : \Sigma_1 \to \Sigma_2$ 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.

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;
- $\mathcal{L} : \Sigma_1 \to \Sigma_2$ 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.

Syntax/semantics Interface

Abstract and object language

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

$$\mathcal{A}(\mathcal{G}) = \{t \in \Lambda^0(\Sigma_1) \mid \vdash_{\Sigma_1} t : s \text{ is derivable}\}$$

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 $\mathcal{L}$:

$$\mathcal{O}(\mathcal{G}) = \{t \in \Lambda^0(\Sigma_2) \mid \exists u \in \mathcal{A}(\mathcal{G}).\ t = \mathcal{L}(u)\}$$

Abstract and object language

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

$$\mathcal{A}(\mathcal{G}) = \{t \in \Lambda^0(\Sigma_1) \mid \vdash_{\Sigma_1} t : s \text{ is derivable}\}$$

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 $\mathcal{L}$:

$$\mathcal{O}(\mathcal{G}) = \{t \in \Lambda^0(\Sigma_2) \mid \exists u \in \mathcal{A}(\mathcal{G}).\ t = \mathcal{L}(u)\}$$

Abstract and object language

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

$$\mathcal{A}(\mathcal{G}) = \{t \in \Lambda^0(\Sigma_1) \mid \ \vdash_{\Sigma_1} t : s \text{ is derivable}\}$$

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 $\mathcal{L}$:

$$\mathcal{O}(\mathcal{G}) = \{t \in \Lambda^0(\Sigma_2) \mid \exists u \in \mathcal{A}(\mathcal{G}).\ t = \mathcal{L}(u)\}$$

Abstract and object language

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

$$\mathcal{A}(\mathcal{G}) = \{t \in \Lambda^0(\Sigma_1) \mid \ \vdash_{\Sigma_1} t : s \text{ is derivable}\}$$

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 $\mathcal{L}$:

$$\mathcal{O}(\mathcal{G}) = \{t \in \Lambda^0(\Sigma_2) \mid \exists u \in \mathcal{A}(\mathcal{G}).\ t = \mathcal{L}(u)\}$$

Abstract and object language

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

$$\mathcal{A}(\mathcal{G}) = \{t \in \Lambda^0(\Sigma_1) \mid \vdash_{\Sigma_1} t : s \text{ is derivable}\}$$

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 $\mathcal{L}$:

$$\mathcal{O}(\mathcal{G}) = \{t \in \Lambda^0(\Sigma_2) \mid \exists u \in \mathcal{A}(\mathcal{G}).\ t = \mathcal{L}(u)\}$$

Syntax/semantics Interface

Signatures

Σ_0: N, NP, S : type
J : NP
U : N
A : $N \multimap ((NP \multimap S) \multimap S)$
S : $((NP \multimap S) \multimap S) \multimap (NP \multimap S)$

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

Σ_2: ι, o : type
$\wedge$: $o \multimap (o \multimap o)$
$\exists$: $(\iota \rightarrow o) \multimap o$
j : ι
unicorn : $\iota \multimap o$
find : $\iota \multimap (\iota \multimap o)$
try : $\iota \multimap ((\iota \multimap o) \multimap o)$

Signatures

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

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

$$
\Sigma_2: \quad \begin{array}{rcl}
\iota, o & : & \text{type} \\
\wedge & : & o \multimap (o \multimap o) \\
\exists & : & (\iota \to o) \multimap o \\
\mathbf{j} & : & \iota \\
\mathbf{unicorn} & : & \iota \multimap o \\
\mathbf{find} & : & \iota \multimap (\iota \multimap o) \\
\mathbf{try} & : & \iota \multimap ((\iota \multimap o) \multimap o)
\end{array}
$$

Signatures

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

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

$$
\begin{array}{llcl}
\Sigma_2: & \iota, o & : & \text{type} \\
& \wedge & : & o \multimap (o \multimap o) \\
& \exists & : & (\iota \rightarrow o) \multimap o \\
& \mathbf{j} & : & \iota \\
& \mathbf{unicorn} & : & \iota \multimap o \\
& \mathbf{find} & : & \iota \multimap (\iota \multimap o) \\
& \mathbf{try} & : & \iota \multimap ((\iota \multimap o) \multimap o)
\end{array}
$$

Signatures

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

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

$$\Sigma_2: \quad \begin{array}{rcl} \iota, o & : & \text{type} \\ \wedge & : & o \multimap (o \multimap o) \\ \exists & : & (\iota \rightarrow o) \multimap o \\ \mathbf{j} & : & \iota \\ \mathbf{unicorn} & : & \iota \multimap o \\ \mathbf{find} & : & \iota \multimap (\iota \multimap o) \\ \mathbf{try} & : & \iota \multimap ((\iota \multimap o) \multimap o) \end{array}$$

Lexicons

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

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

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

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

Lexicons

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

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

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

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

Lexicons

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

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

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

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

Lexicons

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

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

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

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

Lexicons

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

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

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

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

Syntax/semantics transfer

We have that:

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

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

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

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

Syntax/semantics transfer

We have that:

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

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

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

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

Syntax/semantics transfer

We have that:

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

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

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

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

Syntax/semantics transfer

We have that:

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

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

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

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

Syntax/semantics transfer

We have that:

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

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

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

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

Syntax/semantics Interface

Context-free grammars

$$S \to \epsilon$$
$$S \to aSb$$

Abstract vocabulary :

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

Lexicon :

$$S := \mathit{string}$$
$$A := \epsilon$$
$$B := \lambda x.\, a + x + b$$

Context-free grammars

$$S \to \epsilon$$
$$S \to aSb$$

Abstract vocabulary :

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

Lexicon :

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

Context-free grammars

$$S \to \epsilon$$
$$S \to aSb$$

Abstract vocabulary :

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

Lexicon :

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

Context-free grammars

$$S \to \epsilon$$
$$S \to aSb$$

Abstract vocabulary :

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

Lexicon :

$$S := \mathit{string}$$
$$A := \epsilon$$
$$B := \lambda x.\, a + x + b$$

Tree-adjoining grammars

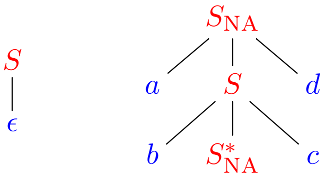
Abstract vocabulary :

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

Lexicon :

$$\begin{array}{l} S, S', S'' := \mathit{string} \\ A := \lambda f.\, f\, \epsilon \\ B := \lambda x.\, \lambda g.\, a + g\,(b + x + c) + d \\ C := \lambda x.\, x \end{array}$$

Tree-adjoining grammars



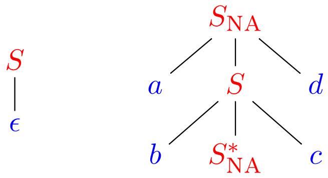
Abstract vocabulary :

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

Lexicon :

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

Tree-adjoining grammars



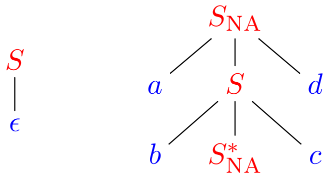
Abstract vocabulary :

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

Lexicon :

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

Tree-adjoining grammars



Abstract vocabulary :

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

Lexicon :

$$\begin{array}{l} S, S', S'' := \mathit{string} \\ A := \lambda f.\, f\, \epsilon \\ B := \lambda x.\, \lambda g.\, a + g\,(b + x + c) + d \\ C := \lambda x.\, x \end{array}$$

Tree-adjoining grammars revisited

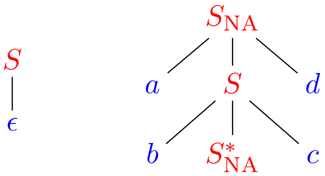
Abstract vocabulary :

$$T, S : \text{type}$$
$$A : T \multimap S$$
$$B : T \multimap T$$
$$C : T$$

Lexicon :

$$S := string$$
$$T := string \multimap string$$
$$A := \lambda f.\, f\, \epsilon$$
$$B := \lambda g.\, \lambda x.\, a + g\,(b + x + c) + d$$
$$C := \lambda x.\, x$$

Tree-adjoining grammars revisited



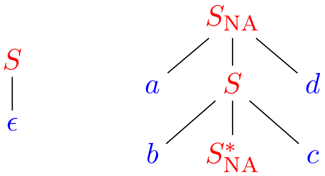
Abstract vocabulary :

$$T, S : \text{type}$$
$$A : T \multimap S$$
$$B : T \multimap T$$
$$C : T$$

Lexicon :

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

Tree-adjoining grammars revisited



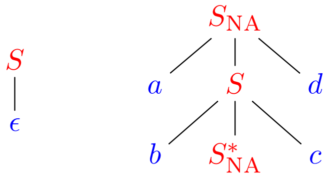
Abstract vocabulary :

$$
\begin{aligned}
T, S &: \text{type} \\
A &: T \multimap S \\
B &: T \multimap T \\
C &: T
\end{aligned}
$$

Lexicon :

$$
\begin{aligned}
S &:= \mathit{string} \\
T &:= \mathit{string} \multimap \mathit{string} \\
A &:= \lambda f.\, f\, \epsilon \\
B &:= \lambda g.\, \lambda x.\, a + g\,(b + x + c) + d \\
C &:= \lambda x.\, x
\end{aligned}
$$

Tree-adjoining grammars revisited



Abstract vocabulary :

$$\begin{aligned} T, S &: \text{type} \\ A &: T \multimap S \\ B &: T \multimap T \\ C &: T \end{aligned}$$

Lexicon :

$$\begin{aligned} S &:= \mathit{string} \\ T &:= \mathit{string} \multimap \mathit{string} \\ A &:= \lambda f.\, f\, \epsilon \\ B &:= \lambda g.\, \lambda x.\, a + g\,(b + x + c) + d \\ C &:= \lambda x.\, x \end{aligned}$$

Syntax/semantics Interface

Abstract categorial hierarchy

Let $\mathcal{G} = \langle \Sigma, \Xi, \mathcal{L}, s \rangle$. Define the *order* and the *complexity* of $\mathcal{G}$:

- $\text{ord}(\mathcal{G}) = \max\{\text{ord}(\tau_\Sigma(c)) \mid c \in C_\Sigma\}$.
- $\text{comp}(\mathcal{G}) = \max\{\text{ord}(\mathcal{L}(a)) \mid a \in A_\Sigma\}$.

Define:

- $\mathbf{G}(m,n) = \{\mathcal{G} \mid \text{ord}(\mathcal{G}) \leq m \text{ and } \text{comp}(\mathcal{G}) \leq n\}$
- $\mathcal{L}(m,n) = \{\mathcal{O}(\mathcal{G}) \mid \mathcal{G} \in \mathbf{G}(m,n)\}$

For all $m, n \geq 1$, $\mathcal{L}(m, n+1) \subset \mathcal{L}(m+1, n)$.

For all $m, n \geq 1$, $\mathcal{L}(m+3, n) \subset \mathcal{L}(m+2, n+1)$.

Abstract categorial hierarchy

Let $\mathcal{G} = \langle \Sigma, \Xi, \mathcal{L}, s \rangle$. Define the *order* and the *complexity* of $\mathcal{G}$:

- $\mathsf{ord}(\mathcal{G}) = \max\{\mathsf{ord}(\tau_\Sigma(c)) \mid c \in C_\Sigma\}$.
- $\mathsf{comp}(\mathcal{G}) = \max\{\mathsf{ord}(\mathcal{L}(a)) \mid a \in A_\Sigma\}$.

Define:

- $\mathbf{G}(m, n) = \{\mathcal{G} \mid \mathsf{ord}(\mathcal{G}) \leq m \text{ and } \mathsf{comp}(\mathcal{G}) \leq n\}$
- $\mathcal{L}(m, n) = \{\mathcal{O}(\mathcal{G}) \mid \mathcal{G} \in \mathbf{G}(m, n)\}$

For all $m, n \geq 1$, $\mathcal{L}(m, n+1) \subset \mathcal{L}(m+1, n)$.

For all $m, n \geq 1$, $\mathcal{L}(m+3, n) \subset \mathcal{L}(m+2, n+1)$.

Abstract categorial hierarchy

Let $\mathcal{G} = \langle \Sigma, \Xi, \mathcal{L}, s \rangle$. Define the *order* and the *complexity* of $\mathcal{G}$:

- $\mathsf{ord}(\mathcal{G}) = \max\{\mathsf{ord}(\tau_\Sigma(c)) \mid c \in C_\Sigma\}$.
- $\mathsf{comp}(\mathcal{G}) = \max\{\mathsf{ord}(\mathcal{L}(a)) \mid a \in A_\Sigma\}$.

Define:

- $\mathbf{G}(m, n) = \{\mathcal{G} \mid \mathsf{ord}(\mathcal{G}) \leq m \text{ and } \mathsf{comp}(\mathcal{G}) \leq n\}$
- $\mathcal{L}(m, n) = \{\mathcal{O}(\mathcal{G}) \mid \mathcal{G} \in \mathbf{G}(m, n)\}$

For all $m, n \geq 1$, $\mathcal{L}(m, n+1) \subset \mathcal{L}(m+1, n)$.

For all $m, n \geq 1$, $\mathcal{L}(m+3, n) \subset \mathcal{L}(m+2, n+1)$.

Abstract categorial hierarchy

Let $\mathcal{G} = \langle \Sigma, \Xi, \mathcal{L}, s \rangle$. Define the *order* and the *complexity* of $\mathcal{G}$:

- $\mathsf{ord}(\mathcal{G}) = \max\{\mathsf{ord}(\tau_\Sigma(c)) \,|\, c \in C_\Sigma\}$.
- $\mathsf{comp}(\mathcal{G}) = \max\{\mathsf{ord}(\mathcal{L}(a)) \,|\, a \in A_\Sigma\}$.

Define:

- $\mathbf{G}(m,n) = \{\mathcal{G} \,|\, \mathsf{ord}(\mathcal{G}) \leq m \text{ and } \mathsf{comp}(\mathcal{G}) \leq n\}$
- $\mathcal{L}(m,n) = \{\mathcal{O}(\mathcal{G}) \,|\, \mathcal{G} \in \mathbf{G}(m,n)\}$

For all $m, n \geq 1$, $\mathcal{L}(m, n+1) \subset \mathcal{L}(m+1, n)$.

For all $m, n \geq 1$, $\mathcal{L}(m+3, n) \subset \mathcal{L}(m+2, n+1)$.

Abstract categorial hierarchy

Let $\mathcal{G} = \langle \Sigma, \Xi, \mathcal{L}, s \rangle$. Define the *order* and the *complexity* of $\mathcal{G}$:

- $\mathsf{ord}(\mathcal{G}) = \max\{\mathsf{ord}(\tau_\Sigma(c)) \mid c \in C_\Sigma\}$.
- $\mathsf{comp}(\mathcal{G}) = \max\{\mathsf{ord}(\mathcal{L}(a)) \mid a \in A_\Sigma\}$.

Define:

- $\mathbf{G}(m, n) = \{\mathcal{G} \mid \mathsf{ord}(\mathcal{G}) \leq m \text{ and } \mathsf{comp}(\mathcal{G}) \leq n\}$
- $\mathcal{L}(m, n) = \{\mathcal{O}(\mathcal{G}) \mid \mathcal{G} \in \mathbf{G}(m, n)\}$

For all $m, n \geq 1$, $\mathcal{L}(m, n+1) \subset \mathcal{L}(m+1, n)$.

For all $m, n \geq 1$, $\mathcal{L}(m+3, n) \subset \mathcal{L}(m+2, n+1)$.

Abstract categorial hierarchy

Let $\mathcal{G} = \langle \Sigma, \Xi, \mathcal{L}, s \rangle$. Define the *order* and the *complexity* of $\mathcal{G}$:

- $\mathsf{ord}(\mathcal{G}) = \max\{\mathsf{ord}(\tau_\Sigma(c)) \mid c \in C_\Sigma\}$.
- $\mathsf{comp}(\mathcal{G}) = \max\{\mathsf{ord}(\mathcal{L}(a)) \mid a \in A_\Sigma\}$.

Define:

- $\mathbf{G}(m, n) = \{\mathcal{G} \mid \mathsf{ord}(\mathcal{G}) \leq m \text{ and } \mathsf{comp}(\mathcal{G}) \leq n\}$
- $\mathcal{L}(m, n) = \{\mathcal{O}(\mathcal{G}) \mid \mathcal{G} \in \mathbf{G}(m, n)\}$

For all $m, n \geq 1$, $\mathcal{L}(m, n+1) \subset \mathcal{L}(m+1, n)$.

For all $m, n \geq 1$, $\mathcal{L}(m+3, n) \subset \mathcal{L}(m+2, n+1)$.

Abstract categorial hierarchy

Let $\mathcal{G} = \langle \Sigma, \Xi, \mathcal{L}, s \rangle$. Define the *order* and the *complexity* of $\mathcal{G}$:

- $\mathsf{ord}(\mathcal{G}) = \max\{\mathsf{ord}(\tau_\Sigma(c)) \,|\, c \in C_\Sigma\}$.
- $\mathsf{comp}(\mathcal{G}) = \max\{\mathsf{ord}(\mathcal{L}(a)) \,|\, a \in A_\Sigma\}$.

Define:

- $\mathbf{G}(m, n) = \{\mathcal{G} \,|\, \mathsf{ord}(\mathcal{G}) \leq m \text{ and } \mathsf{comp}(\mathcal{G}) \leq n\}$
- $\mathcal{L}(m, n) = \{\mathcal{O}(\mathcal{G}) \,|\, \mathcal{G} \in \mathbf{G}(m, n)\}$

For all $m, n \geq 1$, $\mathcal{L}(m, n+1) \subset \mathcal{L}(m+1, n)$.

For all $m, n \geq 1$, $\mathcal{L}(m+3, n) \subset \mathcal{L}(m+2, n+1)$.

Abstract categorial hierarchy

Let $\mathcal{G} = \langle \Sigma, \Xi, \mathcal{L}, s \rangle$. Define the *order* and the *complexity* of $\mathcal{G}$:

- $\mathsf{ord}(\mathcal{G}) = \max\{\mathsf{ord}(\tau_\Sigma(c)) \mid c \in C_\Sigma\}$.
- $\mathsf{comp}(\mathcal{G}) = \max\{\mathsf{ord}(\mathcal{L}(a)) \mid a \in A_\Sigma\}$.

Define:

- $\mathbf{G}(m, n) = \{\mathcal{G} \mid \mathsf{ord}(\mathcal{G}) \leq m \text{ and } \mathsf{comp}(\mathcal{G}) \leq n\}$
- $\mathcal{L}(m, n) = \{\mathcal{O}(\mathcal{G}) \mid \mathcal{G} \in \mathbf{G}(m, n)\}$

For all $m, n \geq 1$, $\mathcal{L}(m, n+1) \subset \mathcal{L}(m+1, n)$.

For all $m, n \geq 1$, $\mathcal{L}(m+3, n) \subset \mathcal{L}(m+2, n+1)$.

Abstract categorial hierarchy

Let $\mathcal{G} = \langle \Sigma, \Xi, \mathcal{L}, s \rangle$. Define the *order* and the *complexity* of $\mathcal{G}$:

- $\mathsf{ord}(\mathcal{G}) = \max\{\mathsf{ord}(\tau_\Sigma(c)) \,|\, c \in C_\Sigma\}$.
- $\mathsf{comp}(\mathcal{G}) = \max\{\mathsf{ord}(\mathcal{L}(a)) \,|\, a \in A_\Sigma\}$.

Define:

- $\mathbf{G}(m, n) = \{\mathcal{G} \,|\, \mathsf{ord}(\mathcal{G}) \leq m \text{ and } \mathsf{comp}(\mathcal{G}) \leq n\}$
- $\mathcal{L}(m, n) = \{\mathcal{O}(\mathcal{G}) \,|\, \mathcal{G} \in \mathbf{G}(m, n)\}$

For all $m, n \geq 1$, $\mathcal{L}(m, n+1) \subset \mathcal{L}(m+1, n)$.

For all $m, n \geq 1$, $\mathcal{L}(m+3, n) \subset \mathcal{L}(m+2, n+1)$.

Second-order hierarchy of string languages

$\mathcal{L}(2,1)$	regular languages
$\mathcal{L}(2,2)$	context-free languages
$\mathcal{L}(2,3)$	well-nested mildly context-sensitive languages
$\mathcal{L}(2,4)$	mildly context-sensitive languages
$\mathcal{L}(2,4+n)$	$\mathcal{L}(2,4)$

Second-order hierarchy of string languages

$\mathcal{L}(2,1)$	regular languages
$\mathcal{L}(2,2)$	context-free languages
$\mathcal{L}(2,3)$	well-nested mildly context-sensitive languages
$\mathcal{L}(2,4)$	mildly context-sensitive languages
$\mathcal{L}(2,4+n)$	$\mathcal{L}(2,4)$

Membership

General case

- (Universal) membership is decidable if and only if the multiplicative-exponential fragment of linear logic is decidable.

Lexicalized case

- (Universal) membership is NP-complete.

Second-order case

- Universal membership is NP-complete, and membership is polynomial.

Membership

General case

- (Universal) membership is decidable if and only if the multiplicative-exponential fragment of linear logic is decidable.

Lexicalized case

- (Universal) membership is NP-complete.

Second-order case

- Universal membership is NP-complete, and membership is polynomial.

Membership

General case

- (Universal) membership is decidable if and only if the multiplicative-exponential fragment of linear logic is decidable.

Lexicalized case

- (Universal) membership is NP-complete.

Second-order case

- Universal membership is NP-complete, and membership is polynomial.

Membership

General case

- (Universal) membership is decidable if and only if the multiplicative-exponential fragment of linear logic is decidable.

Lexicalized case

- (Universal) membership is NP-complete.

Second-order case

- Universal membership is NP-complete, and membership is polynomial.

Membership

General case

- (Universal) membership is decidable if and only if the multiplicative-exponential fragment of linear logic is decidable.

Lexicalized case

- (Universal) membership is NP-complete.

Second-order case

- Universal membership is NP-complete, and membership is polynomial.

Membership

General case

- (Universal) membership is decidable if and only if the multiplicative-exponential fragment of linear logic is decidable.

Lexicalized case

- (Universal) membership is NP-complete.

Second-order case

- Universal membership is NP-complete, and membership is polynomial.

Membership

General case

- (Universal) membership is decidable if and only if the multiplicative-exponential fragment of linear logic is decidable.

Lexicalized case

- (Universal) membership is NP-complete.

Second-order case

- Universal membership is NP-complete, and membership is polynomial.