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

Philippe de Groote

Inria

2012-2013

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.

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:

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

Signature associated to a CFG

Context free grammar:

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

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

$$
\begin{array}{l}
p_1 : NP \times VP \to S \\
p_2 : tV \times NP \to VP \\
p_3 : tV \\
p_4 : NP \\
p_5 : NP
\end{array}
$$

Syntactic and semantic algebras

Syntactic algebra:

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

Semantic algebra:

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

Where:

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

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 assigns each constant in C with a simple type built on A.

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

Trees

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

Strings

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

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

In this setting:

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

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

$$\vdots$$

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

$$\vdots$$

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

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 $\mathcal{L} : \Sigma_1 \rightarrow \Sigma_2$.

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.

Abstract and object language

Let $\Lambda^0(\Sigma)$ be the set of linear λ-terms built upon a higher-order signature Σ. 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 $\mathcal{L}$:

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

Signatures

Σ_0:

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

Σ_1: $/\mathrm{a}/, /\mathrm{John}/, /\mathrm{seeks}/, /\mathrm{unicorn}/ \ : \ STRING$

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

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

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

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

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

Syntax/semantics transfer

We have that:

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

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

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

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

Context-free grammars

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

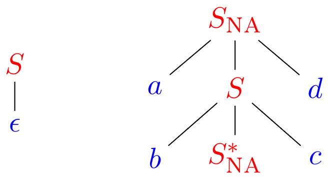
Abstract vocabulary :

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

Lexicon :

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

Tree-adjoining grammars



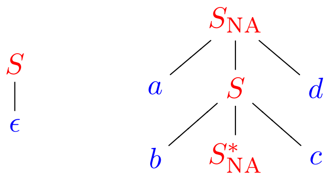
Abstract vocabulary :

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

Lexicon :

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

Tree-adjoining grammars revisited



Abstract vocabulary :

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

Lexicon :

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

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
$\mathcal{L}(2,4+n)$	$\mathcal{L}(2,4)$

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.