

m-Linear Context-Free Rewriting Systems as Abstract Categorical Grammars

Philippe de Groote

Sylvain Pogodalla

June 20th, 2003

*Reconnais-toi
Celle adorable personne c'est toi
je suis le rind
v. Oul
i. i. bouche
j. v. a. p. de la
la
ci. an. fu
P. in. p. u. e
P. u. i. t. e. n. i. a. g. e
de ton buste
dore un comme
à travers un magi*



Overview

- ACGs
 - Principles
 - Properties
- m -LCFRS
 - Description
 - Properties
- Modelling of m -LCFRS with ACGs
- Example and conclusion



ACGs

- Parallelizing term construction (making compositionality explicit)
- Terms of the typed linear λ -calculus
 - data-structures: int, strings, trees...
 - Ex: the empty string: $\lambda x.x$, the string a : $\lambda x.a(x)$ of type $\sigma = * \multimap *$,
the $+$: $\lambda f.\lambda g.\lambda x.f(g(x))$
 - resource consciousness

A *lexicon* preserving type coherence:

$$\begin{array}{ll} \text{Abstract language } L_1 & \longrightarrow \text{Object language } L_2 \\ c & \longrightarrow t' \\ \lambda x.u & \longrightarrow \lambda x.u' \\ u(v) & \longrightarrow u'(v') \end{array}$$

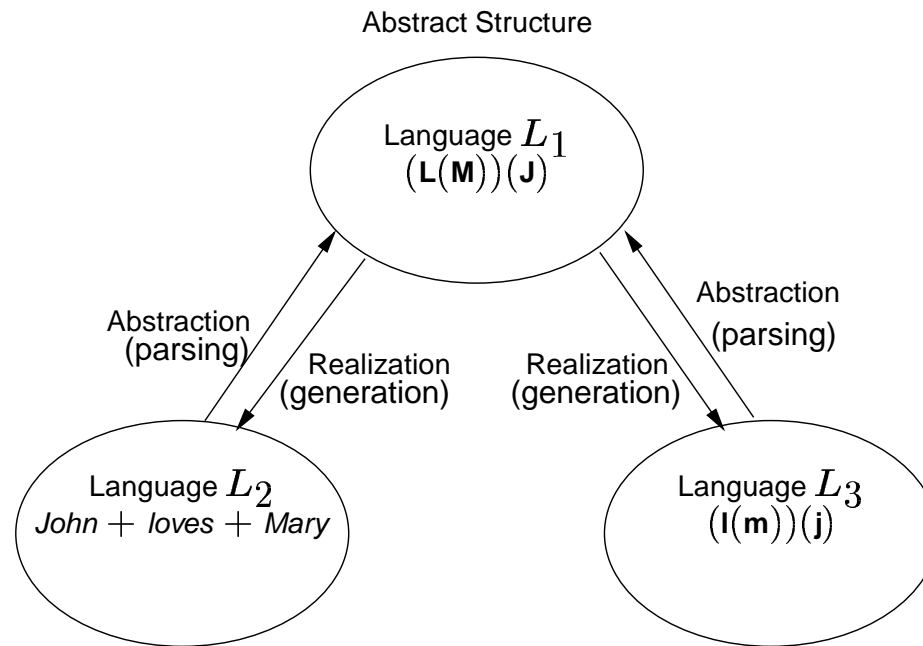


$$\begin{array}{ll}
\text{Language } L_1 & \longrightarrow \text{Language } L_2 \\
\mathbf{J} : NP & \longrightarrow \textit{John} : \sigma \\
\text{Ex.: } \mathbf{L} : NP \multimap NP \multimap S & \longrightarrow \lambda y. \lambda x. x + \textit{loves} + y : \sigma \multimap \sigma \multimap \sigma \\
\mathbf{M} : NP & \longrightarrow \textit{Mary} : \sigma \\
(\mathbf{L}(\mathbf{M}))(\mathbf{J}) : S & \longrightarrow \textit{John} + \textit{loves} + \textit{Mary} : \sigma
\end{array}$$

- Separates the surface form $\textit{John} + \textit{loves} + \textit{Mary}$ of the term from the rules of its construction $(\mathbf{L}(\mathbf{M}))(\mathbf{J})$
- The latter can be transmitted (composition)

$$\begin{array}{ll}
\text{Language } L_1 & \longrightarrow \text{Language } L_3 \\
\mathbf{J} : NP & \longrightarrow \lambda P. P(\mathbf{j}) : (e \multimap t) \multimap t \\
\mathbf{L} : NP \multimap NP \multimap S & \longrightarrow \lambda PQ. Q(\lambda x. P(\lambda y. (\mathbf{l}(y))(x))) : \\
& \qquad \qquad \qquad ((e \multimap t) \multimap t) \multimap (e \multimap t) \multimap t \multimap t \\
\text{Ex.: } \mathbf{M} : NP & \longrightarrow \lambda P. P(\mathbf{m}) : (e \multimap t) \multimap t \\
(\mathbf{L}(\mathbf{M}))(\mathbf{J}) : S & \longrightarrow (\lambda PQ. Q(\lambda x. P(\lambda y. (\mathbf{l}(y))(x))))(\lambda P. P(\mathbf{m}))(\lambda P. P(\mathbf{j})) \\
& \qquad \qquad \qquad \rightarrow_{\beta} (\lambda Q. Q(\lambda x. (\mathbf{l}(\mathbf{m}))(x)))(\lambda P. P(\mathbf{j})) \\
& \qquad \qquad \qquad \rightarrow_{\beta} (\mathbf{l}(\mathbf{m}))(\mathbf{j}) : t
\end{array}$$





Properties:

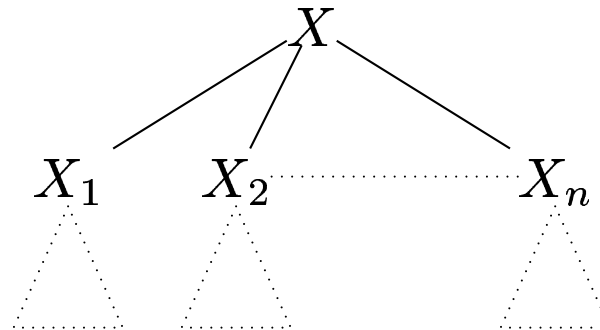
- ACGs can encode CFGs, TAGs and m -LCFRS
- ACGs can generate mildly context-sensitive languages
- What is the exact generative power of ACGs?



m -LCFRS

Description:

- generates mildly context-sensitive languages
- is weakly equivalent to Minimalist Grammars
- CFG-like rules: $X \rightarrow X_1 \cdots X_n$

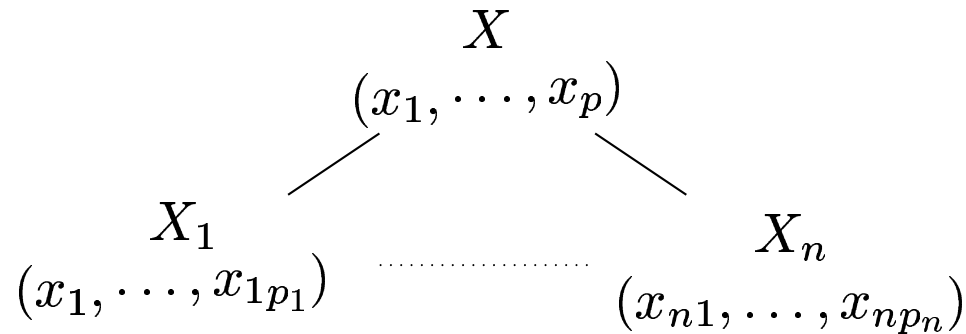


- complex concatenation functions



Annotated rules:

m -LCFRS



such that

$$(x_1, \dots, x_p) = f((x_1, \dots, x_{1p_1}), \dots, (x_{n1}, \dots, x_{np_n}))$$

and:

- every x_i is a linear concatenation of x_{kl} and constants
- if x_{kl} appears in x_i and x_j , then $i = j$



Example

$$\begin{array}{c}
 S \\
 (x_1 + x_2 + x_3 + x_4 + x_5) \\
 | \\
 A \\
 (x_1, x_2, x_3, x_4, x_5)
 \end{array}$$

$$\begin{array}{c}
 A \\
 (x_1 + a, x_2 + b, x_3 + c, x_4 + d, x_5 + e) \\
 | \\
 A \\
 (x_1, x_2, x_3, x_4, x_5) \\
 | \\
 A \\
 (a, b, c, d, e)
 \end{array}$$

$$\begin{array}{c}
 S \\
 (a^n + b^n + c^n + d^n + e^n) \\
 | \\
 A \\
 (a^n, b^n, c^n, d^n, e^n) \\
 \vdots \\
 A \\
 (a^3, b^3, c^3, d^3, e^3) \\
 | \\
 A \\
 (a + a, b + b, c + c, d + d, e + e) \\
 | \\
 A \\
 (a, b, c, d, e)
 \end{array}$$



Modelling m -LCFRS with ACGs

Elements:

- strings: OK
- concatenation: OK
- tuples?

The type system has \multimap but no conjunct.

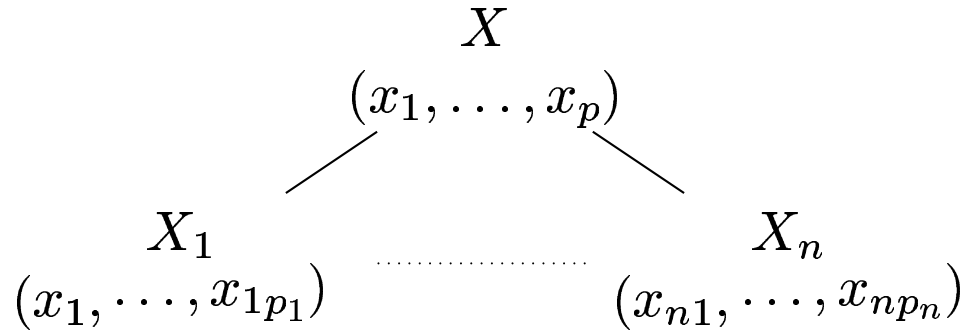
A n -tuple is a higher-order function whose parameter is a function that accepts n parameters: $\lambda g.gx_1 \cdots x_n$

We model $f(x_1, \dots, x_n)$ by currying, then $(\lambda g.gx_1 \cdots x_n)(f) \rightarrow_{\beta} fx_1 \cdots x_n$
(application is the other way around)



Modelling m -LCFRS with ACGs

A rule r :



Language L_1
type X_i

Language L_2
 $\longrightarrow (\underbrace{\sigma \multimap \dots \multimap \sigma}_{p_i \text{ times}} \multimap \sigma) \multimap \sigma$

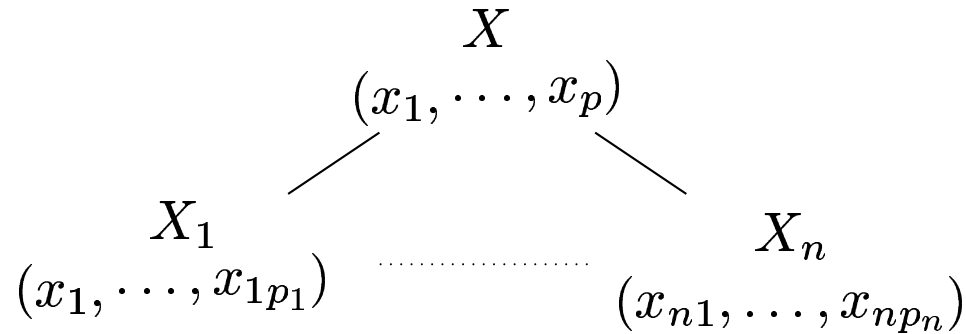
constant $r : X_1 \multimap X_2 \multimap \dots \multimap X \longrightarrow r' ?$

r' is of type

$$(\underbrace{\sigma \multimap \dots \multimap \sigma}_{p_1 \text{ times}} \multimap \sigma) \multimap \sigma \multimap \dots \multimap ((\underbrace{\sigma \multimap \dots \multimap \sigma}_{p_n \text{ times}} \multimap \sigma) \multimap \sigma) \multimap (\underbrace{\sigma \multimap \dots \multimap \sigma}_{p \text{ times}} \multimap \sigma) \multimap \sigma$$



Modelling m -LCFRS with ACGs



The result:

is obtained from a function
of x_{n1}, \dots, x_{np_n}

“applied” to a tuple T_n

obtained from a function
of $x_{(n-1)1}, \dots, x_{(n-1)p_{n-1}}$

“applied” to a tuple T_{n-1}

⋮

“applied” to a tuple T_1

is itself a tuple

$$gx_1x_2 \cdots x_p$$

$$\lambda \vec{x}_n. gx_1 \cdots x_p$$

$$T_n(\lambda \vec{x}_n. gx_1 \cdots x_p)$$

$$\lambda \vec{x}_{n-1}. T_n(\lambda \vec{x}_n. gx_1 \cdots x_p)$$

$$T_{n-1}(\lambda \vec{x}_{n-1}. T_n(\lambda \vec{x}_n. gx_1 \cdots x_p))$$

⋮

$$T_1(\lambda \vec{x}_1. T_2(\cdots \lambda \vec{x}_n. gx_1 \cdots x_p) \cdots)$$

$$\lambda g. T_1(\lambda \vec{x}_1. T_2(\cdots \lambda \vec{x}_n. gx_1 \cdots x_p) \cdots)$$

$$\text{Hence } r' = \lambda T_1 T_2 \cdots T_n g. T_1(\lambda \vec{x}_1. T_2(\cdots \lambda \vec{x}_n. gx_1 \cdots x_p) \cdots)$$



Modelling m -LCFRS with ACGs

Theorem. *For every m -LCFRS G , there exists an ACG \mathcal{G}_G such that:*

- *the abstract language $\mathcal{A}(\mathcal{G}_G)$ of normal terms is isomorphic to the set of parse-trees of G ;*
- *the language generated by G coincides with the object language of \mathcal{G}_G .*



Example

The rules :

$$\begin{array}{ll} r_0 : & S' \rightarrow S \quad f_0(x) = x \\ r_1 : & S \rightarrow A \quad f_1(x_1, \dots, x_5) = x_1 + \dots + x_5 \\ r_2 : & A \rightarrow A \quad f_2(x_1, \dots, x_5) = (x_1 + a, \dots, x_5 + e) \\ r_3 : & A \quad f_3() = (a, b, c, d, e) \end{array}$$

The types

$$\begin{array}{l} r_0: S \multimap S' \\ r_1: A \multimap S \\ r_2: A \multimap A \\ r_3: A \end{array}$$



The rules :

$$\begin{aligned}
 r_0 : S' &\rightarrow S & f_0(x) &= x \\
 r_1 : S &\rightarrow A & f_1(x_1, \dots, x_5) &= x_1 + \dots + x_5 \\
 r_2 : A &\rightarrow A & f_2(x_1, \dots, x_5) &= (x_1 + a, \dots, x_5 + e) \\
 r_3 : A & & f_3() &= (a, b, c, d, e)
 \end{aligned}$$

The lexicon:

Language $L_1 \longrightarrow$ Language L_2

$$r_0 \longrightarrow \lambda t.t(\lambda x.x)$$

$$r_1 \longrightarrow \lambda Tg.T(\lambda x_1x_2x_3x_4x_5.g(x_1 + x_2 + x_3 + x_4 + x_5))$$

$$r_2 \longrightarrow \lambda Tg.T(\lambda x_1x_2x_3x_4x_5.g(x_1 + a)(x_2 + b)(x_3 + c)(x_4 + d)(x_5 + e))$$

$$r_3 \longrightarrow \lambda g.gabcde$$

For instance, we can compute:

$$\begin{aligned}
 r_0(r_1(r_2(r_3))) &= r_0(r_1(r_2(r_3))) \\
 &= r_0(r_1(\lambda g.g(a + a)(b + b)(c + c)(d + d)(e + e))) \\
 &= r_0(\lambda g.g(a + a + b + b + c + c + d + d + e + e)) \\
 &= a + a + b + b + c + c + d + d + e + e
 \end{aligned}$$



Conclusion

Result: Encoding of m -LCFRS into ACGs

Perspectives: What the exact expressive power of ACGs?

Is it too powerfull? Computational complexity increases with the order:

- 2nd order = CF languages
- 3rd order \supset CF tree languages
- 4th order \supset m -LCFRS

