# Abstract Categorial Grammars as a Model of the Syntax-Semantics Interface for TAG

Sylvain Pogodalla

# Abstract Categorial Grammars as a Model of the Syntax-Semantics Interface for TAG

Sylvain Pogodalla

sylvain.pogodalla@inria.fr
LORIA/INRIA Nancy–Grand Est, Nancy, France

FSMLNP 2017 & TAG+13
Umeå, Sweden
September 4, 2017

# Overview

1. Introduction
   - Motivations
   - Mathematical Notions
2. Abstract Categorial Grammars
   - Definitions and Notations
   - Properties
   - ACG as a Model for the Syntax-Semantics Interface
3. TAG as ACG
   - Substitution and Adjunction as Function Application
   - Generalized Derivation Trees
   - Semantic Construction
   - TAG Encoding
4. Discussion

# Abstract Categorial Grammars (de Groote 2001)

## Main Features

- ACG is a (grammatical) framework
- An ACG $\mathscr{G}$ generates two languages:
  - The abstract language $\mathcal{A}(\mathscr{G})$
  - The object language $\mathcal{O}(\mathscr{G})$

  Abstract language: Admissible *structures*
  Object language: *Realizations* of the admissible structures
- Both languages are the same objects: sets of (linear) $\lambda$-terms

### Remark (Examples)

Most of the examples of this presentation can be run with the he ACG toolkit from the example files (`http://calligramme.loria.fr/acg#Software` and `http://hal.inria.fr/hal-01583962/file/examples.zip`)

# Affiliations (I)

## A type-theoretic view on grammars and grammatical composition

- Finite set of atomic types (e.g., np, s, n, etc.)
- Type formation rules (e.g., $\mathrm{np} \rightarrow \mathrm{s}$ is a type)
- Grammatical composition follows typing rules (e.g., $\mathrm{np}, \mathrm{np} \rightarrow \mathrm{s} \vdash \mathrm{s}$, $\mathrm{np} \leftarrow \mathrm{n}, \mathrm{n} \vdash \mathrm{np}$)

## Applications

Grammatical formalisms:

- AB and classical categorial grammars (Ajdukiewicz 1935; Bar-Hillel 1953): directed application
- Lambek grammars (Lambek 1958): hypothetical reasonning
- Formulas-as-types and proofs-as-terms (Howard 1980; Benthem 1986) to provide Lambek Grammars with Montague semantics

# Types and Signatures

## Definition (Implicative types)

*Implicative types* built upon $A$ are $\mathscr{T}_A ::= A \mid \mathscr{T}_A \multimap \mathscr{T}_A \mid \mathscr{T}_A \rightarrow \mathscr{T}_A$

## Definition (Higher-Order Signatures)

A *higher-order signature* $\Sigma$ 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;
- $\tau : C \rightarrow \mathscr{T}_A$ is a function assigning types to constants.

## Example

- $A = \{\mathrm{np}, \mathrm{n}, \mathrm{s}\}$
- $C = \{c_{John}, c_{sleeps}, \ldots\}$
- $\tau(c_{John}) = \mathrm{np}$, $\tau(c_{sleeps}) = \mathrm{np} \multimap \mathrm{s}$, ...

# Terms

## Definition ($\lambda$-Terms $\Lambda(\Sigma)$)

$X$ an infinite countable set of $\lambda$-variables and $\Sigma = \langle A, C, \tau \rangle$ a signature. $\Lambda(\Sigma)$ consists of:

- $c \in C$
- $x \in X$
- $\lambda^{\circ} x.t$ ($x$ occurs free in $t$ exactly once);
- $\lambda x.t$
- $(t\ u)$

## $\beta$-conversion

$$(\lambda^{\circ} x.t)\ u \to_{\beta} t[u/x]$$

# Type System

## Definition (Typing Judgment)

$\Gamma;\Delta \vdash_\Sigma t : \alpha$ where:

- $\Gamma$ is a finite *set* of non-linear variable typing declarations of the form $x : \beta$;
- $\Delta$ is a finite *multi-set* of linear variable typing declarations of the form $x : \beta$.

## Typing rules

$$\frac{}{\Gamma; \vdash_\Sigma c : \tau(c)}\ \text{(const.)}$$

$$\frac{}{\Gamma; x : \alpha \vdash_\Sigma x : \alpha}\ \text{(lin. var.)} \qquad \frac{}{\Gamma, x : \alpha; \vdash_\Sigma x : \alpha}\ \text{(var.)}$$

$$\frac{\Gamma; \Delta, x : \alpha \vdash_\Sigma t : \beta}{\Gamma; \Delta \vdash_\Sigma \lambda^o x.t : \alpha \multimap \beta}\ \text{(lin. abs.)} \qquad \frac{\Gamma; \Delta_1 \vdash_\Sigma t : \alpha \multimap \beta \quad \Gamma; \Delta_2 \vdash_\Sigma u : \alpha}{\Gamma; \Delta_1, \Delta_2 \vdash_\Sigma (t\, u) : \beta}\ \text{(lin. app.)}$$

$$\frac{\Gamma, x : \alpha; \Delta \vdash_\Sigma t : \beta}{\Gamma; \Delta \vdash_\Sigma \lambda x.t : \alpha \to \beta}\ \text{(abs.)} \qquad \frac{\Gamma; \Delta \vdash_\Sigma t : \alpha \to \beta \quad \Gamma; \vdash_\Sigma u : \alpha}{\Gamma; \Delta \vdash_\Sigma (t\, u) : \beta}\ \text{(app.)}$$

# Notations

- $\lambda x.\lambda y.t = \lambda x\ y.t$
- $((t\ u)\ v)\ w) = t\ u\ v\ w$
- $\alpha \to (\beta \to (\gamma \to \delta)) = \alpha \to \beta \to \gamma \to \delta$
- infix notation: if $+$ is a constant, $(+\ f)\ g = f + g$

# Example: The Vocabulary of Strings $\Sigma_{strings}$

## $\Sigma_{strings}$

Types
- $o$ is the unique atomic type
- $\sigma = o \multimap o$ the type of strings (defined)

Constants
- $John, sleeps, a, b, \ldots : \sigma$
- $\# : o$
- we define
  - an infix operator $+ = \lambda^{o} f\ g.\lambda^{o} z.f(g\ z) : \sigma \multimap \sigma \multimap \sigma$
  - $\epsilon = \lambda^{o} x.x : \sigma$

## Example ($\epsilon$ is neutral for $+$)

$$\begin{aligned}
a + \epsilon &= +\ a\ \epsilon \\
&= (\lambda^{o} f\ g.\lambda^{o} z.f(g\ z))\ a\ \epsilon \\
&\rightarrow_\beta (\lambda^{o} g.\lambda^{o} z.a(g\ z))\ \epsilon \\
&\rightarrow_\beta \lambda^{o} z.a\ (\epsilon\ z) \\
&\rightarrow_\beta \lambda^{o} z.a\ z \\
&= a
\end{aligned}$$

# Example

## Remark

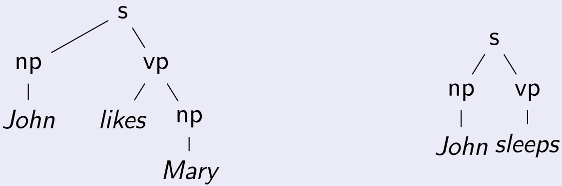
$$
\begin{aligned}
(t+u)\ z & = ((\lambda^{\circ} f\ g.\lambda^{\circ} z'.f\ (g\ z'))\ t\ u)\ z \\
& \rightarrow_{\beta} (\lambda^{\circ} z'.t\ (u\ z'))\ z \\
& \rightarrow_{\beta} t\ (u\ z)
\end{aligned}
$$

## Example (Associativity of +)

$$
\begin{aligned}
f+(g+h) & = (\lambda^{\circ} f'\ g'.\lambda^{\circ} z.f'\ (g'\ z))\ f\ (g+h) \\
& \rightarrow_{\beta} \lambda^{\circ} z.f\ ((g+h)\ z) \\
& \rightarrow_{\beta} \lambda^{\circ} z.f\ (g\ (h\ z)) \\
(f+g)+h & = (\lambda^{\circ} f'\ g'.\lambda^{\circ} z.f'\ (g'\ z))\ (f+g)\ h \\
& \rightarrow_{\beta} \lambda^{\circ} z.(f+g)(h\ z) \\
& = \lambda^{\circ} z.f(g(h\ z))
\end{aligned}
$$

# Example: The Vocabulary of Trees $\Sigma_{trees}$

## Trees Built on a Ranked Alphabet




$s_2$ ($np_1$ *John*) ($vp_2$ *likes* ($np_1$ *Mary*))   $s_2$ ($np_1$ *John*) ($vp_1$ *sleeps*)

## Alphabet

- s of arity 2 (non-terminal)
- np of arity 1 (non-terminal)
- vp? $vp_1$ of arity 1 and $vp_2$ of arity 2 (non-terminals)
- *John* of arity 0 (terminal)

## (Higher-Order) Signature

- $s_2 : \tau \multimap \tau \multimap \tau$
- $np_1 : \tau \multimap \tau$
- $vp_1 : \tau \multimap \tau$, $vp_2 : \tau \multimap \tau \multimap \tau$
- $John : \tau$

# Affiliations (II)

## Two levels for the grammatical architecture (Curry 1961)

- Tectogrammatics: "the study of [abstract] grammatical structure in itself" (e.g., the derivations)
- Phenogrammatics: "the study of the way [abstract] phrases are represented by expressions" (e.g., the yields)

The combinatorial properties of words (e.g., transitive verbs expect two np) are not described at the same place as their combinations are (e.g., SVO, SOV, etc.). The functorial types can be undirected.

## Applications

- Type theoretical grammar and GF (Ranta 1994; Ranta 2004)
- Labeled categorial systems (Oehrle 1994; Oehrle 1995),
- $\lambda$-grammars (Muskens 2001; Muskens 2003)
- etc.

# Relating $\Lambda(\Sigma_{trees})$ to $\Sigma_{strings}$ with $\mathscr{L}_{yield}$

$\Sigma_{trees}$ :

| | |
|---|---|
| $\tau$ | : type |
| *John*, *sleeps* | : $\tau$ |
| $\mathrm{np}_1, \ldots$ | : $\tau \multimap \tau$ |
| $\mathrm{vp}_2, \ldots$ | : $\tau \multimap \tau \multimap \tau$ |

$\Lambda(\Sigma_{trees})$

$$
\begin{array}{rl}
\mathrm{np}_1\ John & : \tau \\
\lambda^{o} x.\mathrm{np}_1\ x & : \tau \multimap \tau \\
\lambda^{o} P.P\ \mathrm{np}_1 & : ((\tau \multimap \tau) \multimap \alpha) \multimap \alpha
\end{array}
$$

$$
\begin{array}{l}
\mathscr{L}_{yield}(\tau) = \sigma \\
\mathscr{L}_{yield}(John) = John \\
\mathscr{L}_{yield}(\mathrm{np}_1) = \lambda x.x \\
\mathscr{L}_{yield}(\mathrm{vp}_2) = \lambda x\ y.x + y
\end{array}
$$

$$
\begin{array}{l}
\mathscr{L}_{yield}(\alpha \multimap \beta) = \mathscr{L}_{yield}(\alpha) \multimap \mathscr{L}_{yield}(\beta) \\
\mathscr{L}_{yield}(t\ u) = \mathscr{L}_{yield}(t)\ \mathscr{L}_{yield}(u) \\
\mathscr{L}_{yield}(\lambda^{o} x.t) = \lambda^{o} x.\mathscr{L}_{yield}(t) \\
\mathscr{L}_{yield}(\lambda x.t) = \lambda x.\mathscr{L}_{yield}(t) \\
\mathscr{L}_{yield}(\alpha \rightarrow \beta) = \mathscr{L}_{yield}(\alpha) \rightarrow \mathscr{L}_{yield}(\beta)
\end{array}
$$

$\Sigma_{strings}$ :

| | |
|---|---|
| $o$ | : type |
| $\sigma = o \multimap o$ | : type |
| *John*, *sleeps* | : $\sigma$ |
| $+$ | : $\sigma \multimap \sigma \multimap \sigma$ |

$$
\begin{array}{rl}
John & : \sigma \\
\lambda^{o} P.P(John + sleeps) & : (\sigma \multimap \sigma) \multimap \sigma
\end{array}
$$

$\Lambda(\Sigma_{strings})$

# Abstract Categorial Grammars

## Definition (Lexicon)

$\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ and $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$ are two higher-order signatures.
A *lexicon* $\mathscr{L} = \langle F, G \rangle$ from $\Sigma_1$ to $\Sigma_2$ is such that:

- $F : \mathscr{T}_{A_1} \rightarrow \mathscr{T}_{A_2}$ is an homomorphism
- $G : \Lambda(\Sigma_1) \rightarrow \Lambda(\Sigma_2)$ is an homomorphism
- $F$ and $G$ are such that for all $c \in C_1$, $\vdash_{\Sigma_2} G(c) : F(\tau_1(c))$ is provable.

Notation: We use $\mathscr{L}$ instead of $F$ or $G$.

# Abstract Categorial Grammars

(de Groote 2001)

## Definition (Abstract Categorial Grammar)

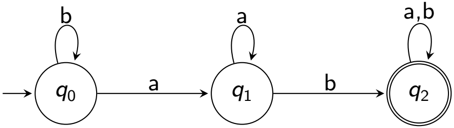
An *abstract categorial grammar* is a quadruple $\mathscr{G} = \langle \Sigma_1, \Sigma_2, \mathscr{L}, \mathsf{s} \rangle$ where:

- $\Sigma_1$ and $\Sigma_2$ are higher-order signatures
- $\mathscr{L} : \Sigma_1 \to \Sigma_2$ is a lexicon.
- $\mathsf{s} \in \mathscr{T}_{A_1}$ is the *distinguished type* of the grammar.

Notations:

$$\left.\begin{array}{ll} \mathscr{L}(\alpha) & = \beta \\ \mathscr{G}(\alpha) & = \beta \\ \alpha & := \beta \\ [\![\alpha]\!] & = \beta \end{array}\right\} \text{all denote that the interpretation of } \alpha \text{ is } \beta$$

# FSA Example




## A signature $\Sigma_{FSA}$ for transitions to $q_2$

**Atomic Types** $q_0$, $q_1$, and $q_2$

**Constants**

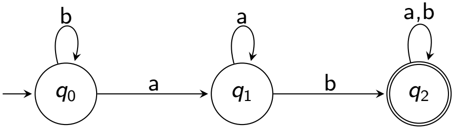
$$
\begin{array}{llllll}
\delta_{q_0,a} & : q_1 \multimap q_0 & \delta_{q_1,a} & : q_1 \multimap q_1 & \delta_{q_2,a} & : q_2 \multimap q_2 \\
\delta_{q_0,b} & : q_0 \multimap q_0 & \delta_{q_1,b} & : q_2 \multimap q_1 & \delta_{q_2,b} & : q_2 \multimap q_2 \\
F & : q_2 & & & &
\end{array}
$$

$$\delta_{q_0,a}\ \overbrace{(\delta_{q_1,a}\ \overbrace{(\delta_{q_1,b}\ F)}^{q_1})}^{q_1} : q_0$$

## Remark

$q_2$ is reachable from $q_i$ iff there exists $t : q_i \in \Lambda(\Sigma_{FSA})$

# FSA Example (cont'd)




$$\Sigma_{FSA}: \quad \begin{array}{lll} \delta_{q_0,a}: q_1 \multimap q_0 & \delta_{q_1,a}: q_1 \multimap q_1 & \delta_{q_2,a}: q_2 \multimap q_2 \\ \delta_{q_0,b}: q_0 \multimap q_0 & \delta_{q_1,b}: q_2 \multimap q_1 & \delta_{q_2,b}: q_2 \multimap q_2 \\ F: q_2 & & \end{array}$$

$$\Sigma_{strings}: \quad \begin{array}{ll} a, b & : o \multimap o \\ \# & : o \end{array}$$

## The $\mathscr{G}_{FSA}$ ACG

Type interpretation $q_0, q_1, q_2 :=_{FSA} o$

Constant interpretation
$$\begin{array}{llllll} \delta_{q_0,a} & :=_{FSA} \lambda z.a\ z & \delta_{q_1,a} & :=_{FSA} a & \delta_{q_2,a} & :=_{FSA} a \\ \delta_{q_0,b} & :=_{FSA} b & \delta_{q_1,b} & :=_{FSA} b & \delta_{q_2,b} & :=_{FSA} b \\ F & :=_{FSA} \# & & & & \end{array}$$

Distinguished type $q_0$

$$\begin{aligned} [\![\delta_{q_1,b}\ F]\!]_{FSA} &= [\![\delta_{q_1,b}]\!]_{FSA}\ [\![F]\!]_{FSA} \\ &= b\ \# \end{aligned}$$

$$\begin{aligned} \delta_{q_1,a}(\delta_{q_1,b}\ F)) &:=_{FSA} a(b\ \#) \\ \delta_{q_0,a}\ (\delta_{q_1,a}(\delta_{q_1,b}\ F)) &:=_{FSA} a(a(b\ \#)) \end{aligned}$$

# Languages of an ACG

## Definition (Abstract and Object Languages)

Let $\mathscr{G} = \langle \Sigma_1, \Sigma_2, \mathscr{L}, \mathrm{s} \rangle$ be an ACG.
The *abstract language* is defined by

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

The *object language* is defined by

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

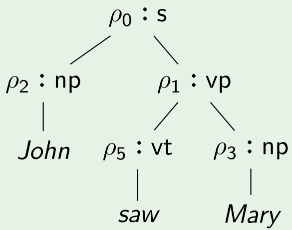
## ACG parsing

- $u \in \Lambda(\Sigma_2)$
- Does $t \in \Lambda(\Sigma_1)$ exists such that $\mathscr{L}(t) = u$?

# CFG into ACG Encoding

## Example (CFG)

$$
\begin{array}{lll}
\rho_0 : & s & \rightarrow np\ vp \\
\rho_1 : & vp & \rightarrow vt\ np \\
\rho_2 : & np & \rightarrow \mathit{John} \\
\rho_3 : & np & \rightarrow \mathit{Mary} \\
\rho_4 : & vp & \rightarrow \mathit{left} \\
\rho_5 : & vt & \rightarrow \mathit{saw}
\end{array}
$$




$$
\begin{array}{rlcrl}
 & \Sigma_{Rules} & & \Sigma_{Strings} & \\
np, vp, vt, s & : \text{atomic types} & :=_{CFG} & \sigma & \\
\rho_0 & : np \multimap vp \multimap s & :=_{CFG} & \lambda xy.x + y & : \sigma \multimap \sigma \multimap \sigma \\
\rho_1 & : vt \multimap np \multimap vp & :=_{CFG} & \lambda xy.x + y & : \sigma \multimap \sigma \multimap \sigma \\
\rho_2 & : np & :=_{CFG} & \mathit{John} & : \sigma \\
\rho_3 & : np & :=_{CFG} & \mathit{Mary} & : \sigma \\
\rho_4 & : vp & :=_{CFG} & \mathit{left} & : \sigma \\
\rho_5 & : vt & :=_{CFG} & \mathit{saw} & : \sigma
\end{array}
$$

# CFG into ACG Encoding (cont'd)

## CFG as ACG

| | | $\Sigma_{Rules}$ | | $\Sigma_{Strings}$ | |
|---|---|---|---|---|---|
| np, vp, vt, s | : | atomic types | $:=_{CFG}$ | $\sigma$ | |
| $\rho_0$ | : | $\mathrm{np} \multimap \mathrm{vp} \multimap \mathrm{s}$ | $:=_{CFG}$ | $\lambda xy.x + y$ | $: \sigma \multimap \sigma \multimap \sigma$ |
| $\rho_1$ | : | $\mathrm{vt} \multimap \mathrm{np} \multimap \mathrm{vp}$ | $:=_{CFG}$ | $\lambda xy.x + y$ | $: \sigma \multimap \sigma \multimap \sigma$ |
| $\rho_2$ | : | np | $:=_{CFG}$ | *John* | $: \sigma$ |
| $\rho_3$ | : | np | $:=_{CFG}$ | *Mary* | $: \sigma$ |
| $\rho_4$ | : | vp | $:=_{CFG}$ | *left* | $: \sigma$ |
| $\rho_5$ | : | vt | $:=_{CFG}$ | *saw* | $: \sigma$ |

$$
\begin{aligned}
\mathscr{L}_{CFG}(\rho_0\,\rho_2\,(\rho_1\,\rho_5\rho_3) : \mathrm{s}) &= (\lambda xy.x + y) John((\lambda xy.x + y) saw\, Mary)\\
&\rightarrow_\beta (\lambda y.John + y)((\lambda y.saw + y) Mary)\\
&\rightarrow_\beta (\lambda y.John + y)(saw + Mary)\\
&\rightarrow_\beta John + (saw + Mary)
\end{aligned}
$$

# About Linearity

Can we generate $\{\omega\omega | \omega \in L\}$? Yes

## Example ($L$ is the langage of any sequence of $a$'s and $b$'s)

$$
\begin{array}{llll}
\mathsf{s}' & : \mathit{type} & :=_{\mathit{Dup}} & \sigma \\
\mathsf{s} & : \mathit{type} & :=_{\mathit{Dup}} & (\sigma \multimap \sigma \multimap \sigma) \multimap \sigma \\
\varepsilon & : \mathsf{s} & :=_{\mathit{Dup}} & \lambda f.f\ \epsilon\ \epsilon \\
A & : \mathsf{s} \multimap \mathsf{s} & :=_{\mathit{Dup}} & \lambda^{o} f.\lambda^{o} g.f(\lambda^{o} x\ y.g\ (a+x)\ (a+y)) \\
B & : \mathsf{s} \multimap \mathsf{s} & :=_{\mathit{Dup}} & \lambda^{o} f.\lambda^{o} g.f(\lambda^{o} x\ y.g\ (b+x)\ (b+y)) \\
C & : \mathsf{s} \multimap \mathsf{s}' & :=_{\mathit{Dup}} & \lambda^{o} f.f(\lambda^{o} x\ y.x+y)
\end{array}
$$

$$
C\ (A\ (B\ (B\ \varepsilon))) : \mathsf{s}' \quad :=_{\mathit{Dup}} \quad a+b+b+a+b+b : \sigma
$$

# ACG Hierarchy

## Definition (Order)

The order $\mathrm{ord}(\tau)$ of a type $\tau \in \mathscr{T}_A$ is inductively defined as:

- $\mathrm{ord}(a) = 1$ if $a \in A$
- $\mathrm{ord}(\alpha \multimap \beta) = \mathrm{ord}(\alpha \rightarrow \beta) = \max(1 + \mathrm{ord}(\alpha), \mathrm{ord}(\beta))$ otherwise

Ex:

- $\mathrm{ord}(\mathrm{np} \multimap \mathrm{s}) = 2$
- $\mathrm{ord}((\sigma \multimap \sigma \multimap \sigma) \multimap \sigma) = 4$ ($\sigma = o \multimap o$ is not atomic!)

## Definition (Order and complexity of an ACG; ACG hierarchy)

- The *order* of an ACG is the maximum of the orders of its abstract constants.
- The *complexity* of an ACG is the maximum of the orders of the realizations of its atomic types.

# ACG: Formal Properties

(Groote and Pogodalla 2004; Salvati 2006; Kanazawa and Salvati 2007; Kanazawa 2009), (Salvati 2007; Kanazawa 2007; Kanazawa 2008; Kanazawa 2017)

## Generative Power

| | String language | Tree language |
|---|---|---|
| $\mathrm{ACG}_{(1,n)}$ | finite | finite |
| $\mathrm{ACG}_{(2,1)}$ | regular | regular |
| $\mathrm{ACG}_{(2,2)}$ | context-free | linear contex-free |
| $\mathrm{ACG}_{(2,3)}$ | non-duplicating macro<br>well-nested multiple context-free | $\subset$ 1-visit attribute grammar |
| $\mathrm{ACG}_{(2,4)}$ | mildly context-sensitive<br>(multiple context-free) | hyperedge replacement gram. |
| $\mathrm{ACG}_{(2,4+n)}$ | $\mathrm{ACG}_{(2,4)}$ | $\mathrm{ACG}_{(2,4)}$ |
| $\mathrm{ACG}_{(3,n)}$ | MELL decidability | MELL decidability |

## Complexity

- $\mathrm{ACG}_{(2,n)}$ parsing is polynomial, equivalent to datalog querying
- Reduces to best cases with standard techniques (magic set rewriting) with correct prefix Earley algorithms

# Higher Order Logic as an Object Language

## The $\Sigma_{logic}$ vocabulary

**Types** $e$, $t$

**Constants** Logical constants

| | | | |
|---|---|---|---|
| $\land$ | $: t \multimap t \multimap t$ | $\lor$ | $: t \multimap t \multimap t$ |
| $\Rightarrow$ | $: t \multimap t \multimap t$ | $\neg$ | $: t \multimap t$ |
| $\exists$ | $: (e \to t) \multimap t$ | $\forall$ | $: (e \to t) \multimap t$ |

Non-logical constants

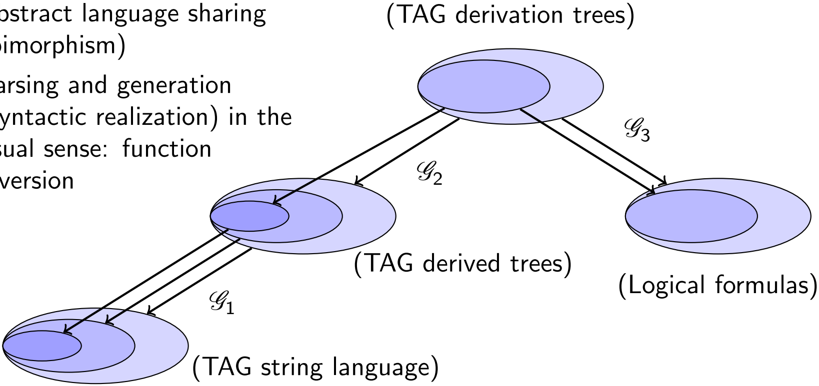
| | | | |
|---|---|---|---|
| john | $: e$ | love, chase | $: e \multimap e \multimap t$ |
| sleep | $: e \multimap t$ | seemingly, usually | $: t \multimap t$ |
| seem | $: e \multimap (e \multimap t) \multimap t$ | claim, think | $: e \multimap t \multimap t$ |
| WHO | $: (e \multimap t) \multimap t$ | big, black, dog, cat | $: e \multimap t$ |

ACG parsing works exactly the same, whatever the object language

# ACG Architecture

## Composition Ability

- Functional composition
- Abstract language sharing (bimorphism)
- Parsing and generation (syntactic realization) in the usual sense: function inversion

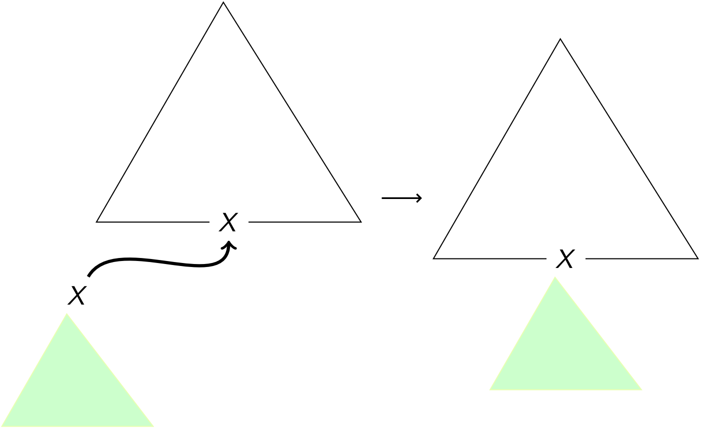


# Tree Adjoining Grammars

Main features

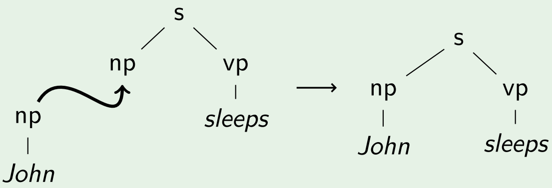
- Tree grammars with two operations
  - Substitution
  - Adjunction
- $L(CFG) \subsetneq L(TAG) \subsetneq L(WnMCFG)$
- Allows for an *extended domain of locality* (Joshi 1994): locally specify (syntactic and semantic) dependencies between parts that can occur arbitrarily far from each other at the surface level at the end of a derivation

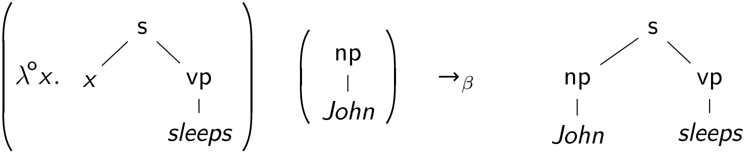
# Substitution



# Substitution as Functional Application

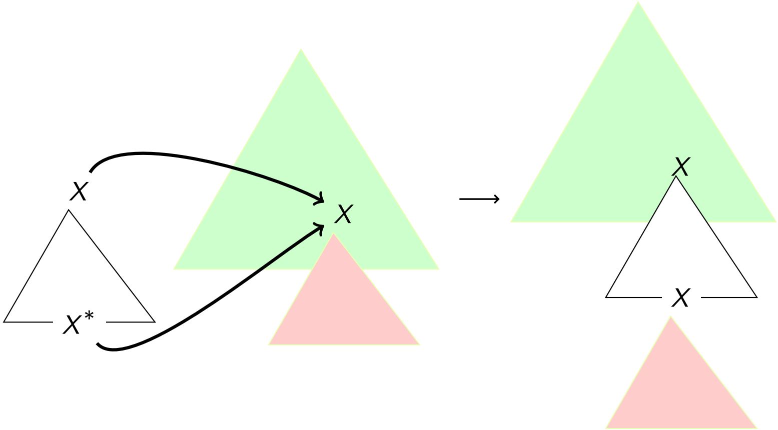
## Example



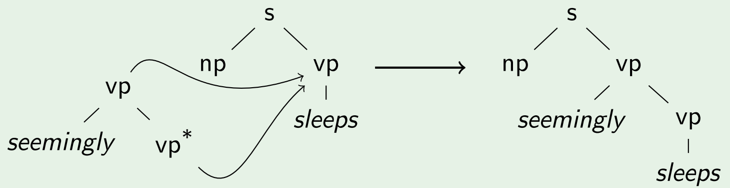

$$(\lambda^{\circ} x.\mathrm{s}_2\ x\ (\mathrm{vp}_1\ \mathit{sleeps}))\quad (\mathrm{np}_1\ \mathit{John})\quad \rightarrow_{\beta}\quad \mathrm{s}_2\ (\mathrm{np}_1\ \mathit{John})\ (\mathrm{vp}_1\,\mathit{sleeps})$$

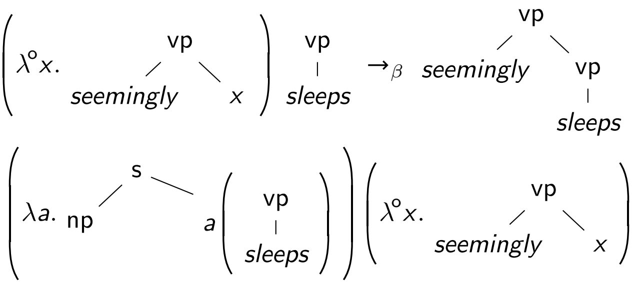
# Adjunction



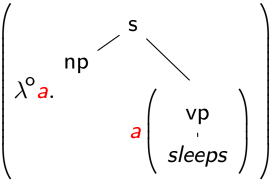
# Auxiliary Trees as Functions

## Example

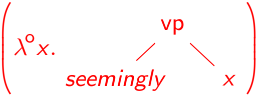


# Adjunction as Functional Application

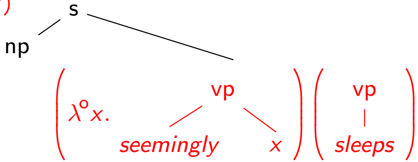



$(\lambda^{o} a.\mathrm{s}_2\ \mathrm{np}\ a(\mathrm{vp}_1\ \textit{sleeps}))$



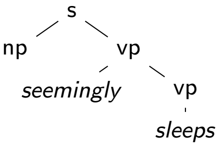

$(\lambda^{o} x.\mathrm{vp}_2\ \textit{seemingly}\ x)$

$\rightarrow_\beta$




$\mathrm{s}_2\ \mathrm{np}\ ((\lambda^{o} x.\mathrm{vp}_2\ \textit{seemingly}\ x)(\mathrm{vp}_1\ \textit{sleeps}))$

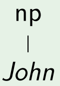
$\rightarrow_\beta$



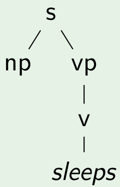

$\mathrm{s}_2\ \mathrm{np}\ (\mathrm{vp}_2\ \textit{seemingly}\ (\mathrm{vp}_1\ \textit{sleeps}))$

# Terms (of $\Lambda(\Sigma_{trees})$) for TAG Operations

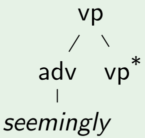
$\gamma_{John} = \mathsf{np}_1\ \mathit{John} : \tau$




$\gamma_{sleeps} = \lambda^{o} S\ a\ s.S\ (\mathsf{s}_2\ s\ (a\ (\mathsf{vp}_1\ (\mathsf{v}_1\ \mathit{sleeps}))))$
$: (\tau \multimap \tau) \multimap (\tau \multimap \tau) \multimap \tau \multimap \tau$




$\gamma_{seemingly} = \lambda^{o} a\ v.a\ (\mathsf{vp}_2\ (\mathsf{adv}_1\ \mathit{seemingly})\ v)$
$: (\tau \multimap \tau) \multimap \tau \multimap \tau$




$I = \lambda^{o} x.x : \tau \multimap \tau$

$\gamma_{sleeps}\ I\ (\gamma_{seemingly}\ I)\ \gamma_{John} \rightarrow_{\beta} \mathsf{s}_2\ (\mathsf{np}_1\ \mathit{John})\ (\mathsf{vp}_2\ (\mathsf{adv}_1\ \mathit{seemingly})\ (\mathsf{vp}_1\ (\mathsf{v}_1\ \mathit{sleeps})))$
$\gamma_{sleeps}\ I\ (\gamma_{seemingly}\ I)\ \gamma_{John} :=_{yield} \mathit{John} + \mathit{seemingly} + \mathit{sleeps}$

# So far...

- A vocabulary for strings $\Sigma_{strings}$
- A vocabulary for trees $\Sigma_{trees}$
- Yield of a tree as an ACG $\mathscr{G}_{yield}$
- TAG operations on trees as applications of terms of $\Lambda(\Sigma_{trees})$

PB: Not only TAG derived trees (e.g., $\gamma_{seemingly}\ I : \tau \multimap \tau$ and $\gamma_{John} : \tau$)

# TAG as ACG (1st part)

## Category Induced Constraints

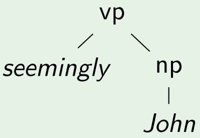
- The site of an adjunction has the same category as the root (and foot) node of the auxiliary tree
- The site of a substitution has the same category as the root node of the substituted tree

| | $\Sigma_{derivations}$ | | | $\Lambda(\Sigma_{trees})$ |
|---|---|---|---|---|
| $c_{John}$ | $:\mathsf{np}$ | $:=_{derived\ trees}$ | $\gamma_{John}$ | $:\tau$ |
| $c_{seemingly}$ | $:(\mathsf{vp} \multimap \mathsf{vp}) \multimap \mathsf{vp} \multimap \mathsf{vp}$ | $:=_{derived\ trees}$ | $\gamma_{seemingly}$ | $:(\tau \multimap \tau) \multimap \tau \multimap \tau$ |
| $\mathsf{np}, \mathsf{vp}, \mathsf{s} \ldots$ | : types | $:=_{derived\ trees}$ | $\tau$ | |
| $I_{\mathsf{vp}}$ | $:\mathsf{vp} \multimap \mathsf{vp}$ | $:=_{derived\ trees}$ | $I$ | $:\tau \multimap \tau$ |

## Example

$c_{seemingly}\ I_{\mathsf{vp}}\ c_{John}$ is not well-typed

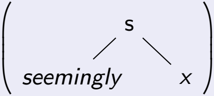
There is no $t : \mathsf{vp} \in \Lambda(\Sigma_{derivations})$ such that $t :=_{derived\ trees}$



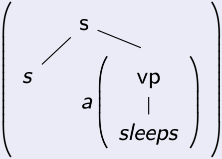
# Control on the Derived Trees

$\mathscr{G}_{derived\ trees} = \langle \Sigma_{derivations}, \Sigma_{trees}, \mathscr{L}_{derived\ trees}, \mathsf{s} \rangle$

$c_{seemingly} : (\mathsf{vp} \multimap \mathsf{vp}) \multimap \mathsf{vp} \multimap \mathsf{vp} \quad :=_{derived\ trees} \lambda^{o} ax.a$

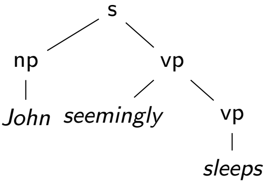



$c_{sleeps} : (\mathsf{s} \multimap \mathsf{s}) \multimap (\mathsf{vp} \multimap \mathsf{vp}) \multimap \mathsf{np} \multimap \mathsf{s} \quad :=_{derived\ trees} \lambda^{o} Sas.S$



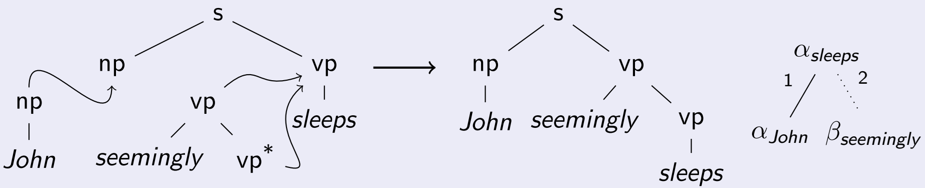

$I_{\mathsf{vp}} : \mathsf{vp} \multimap \mathsf{vp}$

$c_{likes}\ I_{\mathsf{s}} (c_{seemingly}\ I_{\mathsf{vp}})\ c_{John} : \mathsf{s} \quad :=_{derived\ trees}$

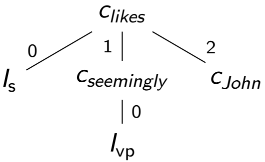


# TAG Derivation Trees as Abstract Terms

## Derivation tree: to record the operations



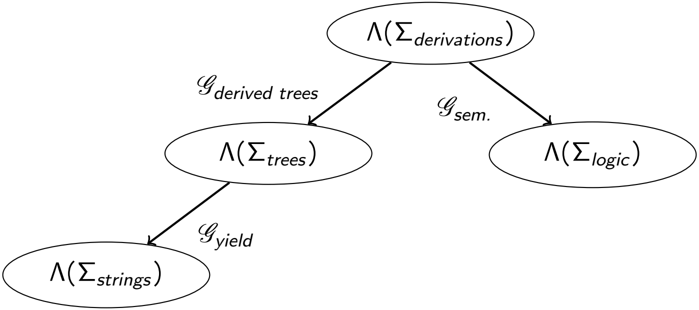

$c_{sleeps}\ I_{\mathsf{s}}\ (c_{seemingly}\ I_{\mathsf{vp}})\ c_{John} =$




$c_{likes}$ $: \underbrace{(\mathsf{s} \multimap \mathsf{s})}_{0} \multimap \underbrace{(\mathsf{vp} \multimap \mathsf{vp})}_{2} \multimap \underbrace{\mathsf{np}}_{1} \multimap \mathsf{s}$

# TAG as ACG

## Intermediate Picture



# Building a Semantic Representation

## $\Sigma_{derivations}$

$$
\begin{array}{ll}
c_{John} & : \mathrm{np} \\
c_{seemingly} & : (\mathrm{vp} \multimap \mathrm{vp}) \multimap \mathrm{vp} \multimap \mathrm{vp} \\
c_{sleeps} & : (\mathrm{s} \multimap \mathrm{s}) \multimap (\mathrm{vp} \multimap \mathrm{vp}) \multimap \mathrm{np} \multimap \mathrm{s} \\
I_{\mathrm{vp}} & : \mathrm{vp} \multimap \mathrm{vp}
\end{array}
$$

## $\Sigma_{logic}$

$$
\begin{array}{ll}
\wedge, \vee, \Rightarrow & : t \multimap t \multimap t \\
\neg & : t \multimap t \\
\exists, \forall & : (e \rightarrow t) \multimap t \\
\mathrm{john} & : e \\
\mathrm{sleep} & : e \multimap t \\
\mathrm{seemingly} & : t \multimap t
\end{array}
$$

## A standard interpretation *à la* Montague

$$
\begin{array}{llll}
\mathrm{s} & :=_{sem.} t & \mathrm{np} & :=_{sem.} (e \rightarrow t) \multimap t \\
\mathrm{vp} & :=_{sem.} e \rightarrow t & \mathrm{n} & :=_{sem.} e \rightarrow t \\
 & & c_{john} & :=_{sem.} \lambda^{o} P.P\,\mathrm{j} \\
I_{\mathrm{s}} & :=_{sem.} \lambda^{o} x.x & c_{seemingly} & :=_{sem.} \lambda^{o} aP.a(\lambda x.\mathrm{seemingly}\ (P\,x)) \\
I_{\mathrm{vp}} & :=_{sem.} \lambda^{o} x.x & c_{likes} & :=_{sem.} \lambda^{o} S\ a\ s.S\ (s\ (a\ (\lambda x.\mathrm{seems}\ x)))
\end{array}
$$

$$c_{sleeps}\ I_{\mathrm{s}}\ (c_{seemingly}\ I_{\mathrm{vp}})\ c_{John} :=_{sem.} \mathrm{seemingly}\ (\mathrm{sleep}\ \mathrm{j})$$

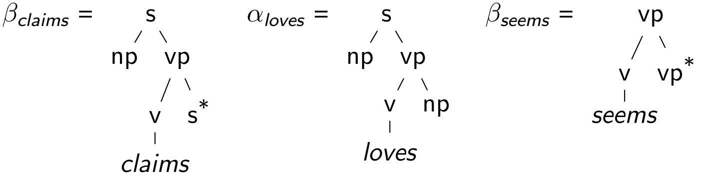
# Long-Distance Dependencies

## Example

(1) Paul claims John loves Mary

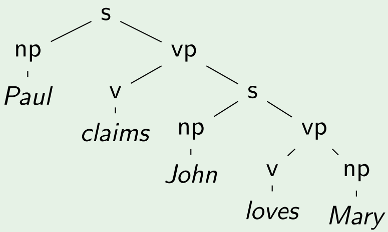
(2) Paul claims John seems to love Mary

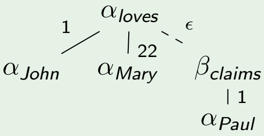
(3) Who does Peter think Paul claims John seems to love



# Long-Distance Dependencies

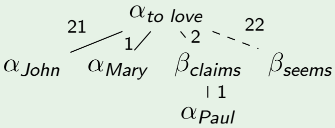
## Example (Paul claims John loves Mary)




claim paul (love john mary)

## Example (Paul claims John seems to love Mary )




claim paul (seem john $(\lambda x.$love $x$ mary$)$)

# Long-Distance Dependencies

## The ACG Way

$$
\begin{array}{l}
c_{claims} :=_{sem.} \lambda^{o} adv_{\mathrm{s}}\ adv_{\mathrm{vp}}\ subj\ comp.adv_{\mathrm{s}}\ (subj\ (adv_{\mathrm{vp}}\ (\lambda x.\mathrm{claim}\ x\ comp)))\\
c_{seems} :=_{sem.} \lambda^{o} mod\ pred.mod\ (\lambda x.\mathrm{seem}\ x\ pred)\\
c_{loves} :=_{sem.} \lambda^{o} adv_{\mathrm{s}}\ adv_{\mathrm{vp}}\ subj\ obj.adv_{\mathrm{s}}\ (subj\ (adv_{\mathrm{vp}}\ (\lambda x.obj\ (\lambda y.\mathrm{love}\ x\ y))))\\
c_{to\ love} :=_{sem.} \lambda^{o} adv_{\mathrm{s}}\ adv_{\mathrm{vp}}\ obj\ subj.adv_{\mathrm{s}}\ (subj\ (adv_{\mathrm{vp}}\ (\lambda x.obj\ (\lambda y.\mathrm{love}\ x\ y))))
\end{array}
$$

See the demo!

$$c_{loves}\ (c_{claims}\ I_{\mathrm{s}}\ I_{\mathrm{vp}}\ c_{Paul})\ I_{\mathrm{vp}}\ c_{John}\ c_{Mary} :=_{sem.} \mathrm{claim\ paul\ (love\ john\ mary)}$$

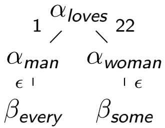
$$c_{to\ love}\ (c_{claims}\ I_{\mathrm{s}}\ I_{\mathrm{vp}}\ c_{Paul})\ (c_{seems}\ I_{\mathrm{vp}})\ c_{Mary}\ c_{John}$$

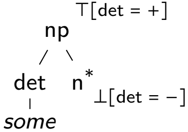
$$:=_{sem.} \mathrm{claim\ paul\ (seem\ john\ (}\lambda x.\mathrm{love}\ x\ \mathrm{mary))}$$

# Quantification

(4) *every man loves some woman*

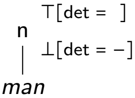
$\forall x.\text{man}(x) \Rightarrow \exists y.\text{woman}(x) \land \text{love}(x, y)$



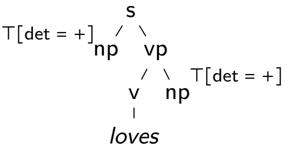

$\text{n}[-] \multimap \text{n}[+]$

$\text{n} \multimap \text{np}$




$(\text{n}[-] \multimap \text{n}[+]) \multimap \text{n}[+]$

$(\text{n} \multimap \text{np}) \multimap \text{np}$




$(\text{s} \multimap \text{s}) \multimap (\text{vp} \multimap \text{vp}) \multimap \text{n}[+] \multimap \text{n}[+] \multimap \text{s}$

$(\text{s} \multimap \text{s}) \multimap (\text{vp} \multimap \text{vp}) \multimap \text{np} \multimap \text{np} \multimap \text{s}$

# A Note on Feature Structures

- The ACG way corresponds to specifying adjunction constraints by *enumeration*
- Vijay-Shanker and Joshi (1988, p. 718):

  *"... if we think of the auxiliary tree as corresponding to functions over feature structures (by $\lambda$-abstracting the variable corresponding to the feature structure for the tree that will appear below the foot node). Adjunction corresponds to applying this function to the feature structure corresponding to the subtree below the node where [it] takes place."*

# Quantification

## The ACG Way

| Constants of $\Sigma_{derivations}$ | | By $\mathscr{G}_{derived\ trees}$ | By $\mathscr{G}_{sem.}$ |
|---|---|---|---|
| $c_{man}$ | $: (\mathrm{n} \multimap \mathrm{np}) \multimap \mathrm{np}$ | $\lambda^{o} d.d\ (\mathrm{n}_1\, man)$ | $\lambda^{o} Q.Q\ \mathrm{man}$ |
| $c_{some}$ | $: \mathrm{n} \multimap \mathrm{np}$ | $\lambda^{o} n.\mathrm{np}_2\ (\mathrm{det}_1\ some)\ n$ | $\lambda^{o} P\ Q.\exists x.(P\ x) \land (Q\ x)$ |
| $c_{every}$ | $: \mathrm{n} \multimap \mathrm{np}$ | $\lambda^{o} n.\mathrm{np}_2\ (\mathrm{det}_1\ every)\ n$ | $\lambda^{o} P\ Q.\forall x.(P\ x) \Rightarrow (Q\ x)$ |

$$c_{loves}\ I_{\mathrm{s}}\ I_{\mathrm{vp}}\ (c_{man}\ c_{every})\ (c_{woman}\ c_{some}) :=_{sem.} \forall x.(\mathrm{man}\ x) \Rightarrow (\exists y.(\mathrm{woman}\ y) \land (\mathrm{love}\ x\ y))$$

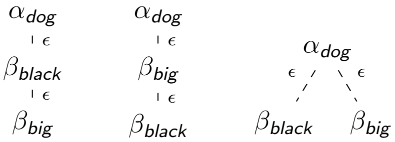
# Multiple Modification

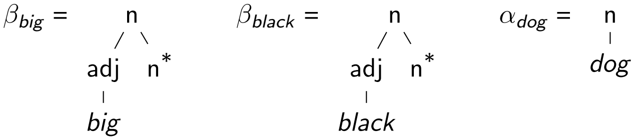
(5) *big black dog*

black$(x) \land$ big$(x) \land$ dog$(x)$

(6) *black big dog*

black$(x) \land$ big$(x) \land$ dog$(x)$




# Multiple Adjunction

## The ACG way

$$
\begin{array}{lll}
c_{big} & : (\mathrm{n} \multimap \mathrm{np}) \multimap \mathrm{n} \multimap \mathrm{np} :=_{sem.} & \lambda^{o} Q\ n.Q\ (\lambda x.(n\ x) \wedge (\mathrm{big}\ x)) \\
c_{black} & : (\mathrm{n} \multimap \mathrm{np}) \multimap \mathrm{n} \multimap \mathrm{np} :=_{sem.} & \lambda^{o} Q\ n.Q\ (\lambda x.(n\ x) \wedge (\mathrm{black}\ x)) \\
c_{dog} & : (\mathrm{n} \multimap \mathrm{np}) \multimap \mathrm{np} \quad :=_{sem.} & \lambda^{o} Q.Q\ \mathrm{dog}
\end{array}
$$

$$
\begin{array}{l}
\lambda^{o} D.c_{dog}\ (c_{black}\ (c_{big}\ D)) :=_{sem.} \lambda^{o} D.D\ (\lambda x.((\mathrm{big}\ x) \wedge (\mathrm{black}\ x)) \wedge (\mathrm{dog}\ x)) \\
\lambda^{o} D.c_{dog}\ (c_{big}\ (c_{black}\ D)) :=_{sem.} \lambda^{o} D.D\ (\lambda x.((\mathrm{black}\ x) \wedge (\mathrm{big}\ x)) \wedge (\mathrm{dog}\ x))
\end{array}
$$

# Etc.

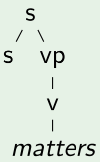
- Control verbs
- Idioms (based on Kobele 2012):
  $c_{kicked\ the\ bucket} :=_{derived\ trees} [\![\lambda^{o} s\ a\ subj.c_{kicked}\ s\ a\ subj\ (c_{bucket}\ c_{the})]\!]_{derived\ trees}$
- Subordinate Conjunctions
- Discourse connectives their syntax-semantics interface (Danlos, Maskharashvili, and Pogodalla 2016; Maskharashvili 2016)

Demo with:

- The ACG toolkit (`http://calligramme.loria.fr/acg#Software`)
- Signatures, lexicons, and scripts available from my publication page as related file to this presentation (`http://hal.inria.fr/hal-01583962/file/examples.zip`)

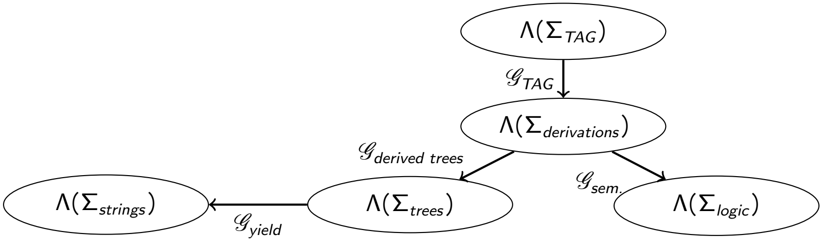
# TAG as ACG: Completing the Encoding

## Example (*To arrive on time matters*)




$c_{matters} : (\text{vp} \multimap \text{vp}) \multimap \text{s} \multimap \text{s}$

- Not 2nd-order!
- $c_{matters}\ I_{\text{vp}} : \text{s} \multimap \text{s}$ has the type of an auxilary tree. Overgeneration



# $\mathscr{G}_{TAG} = \langle \Sigma_{TAG}, \Sigma_{derivations}, \mathscr{L}_{TAG}, \mathsf{s} \rangle$

(de Groote 2002)

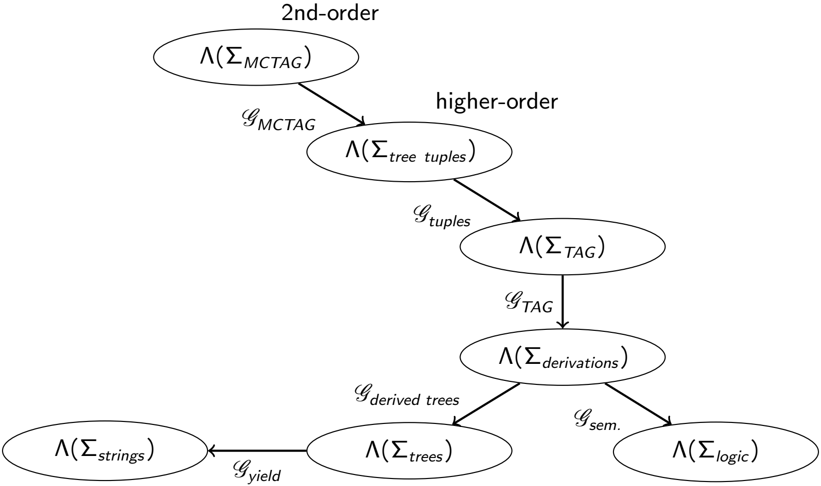
| Types and constants of $\Sigma_{TAG}$ | | Their interpretations in $\Lambda(\Sigma_{derivations})$ | |
|---|---|---|---|
| $\mathsf{np}, \mathsf{s}, \ldots$ | | $\mathsf{np}, \mathsf{s}, \ldots$ | |
| $\mathsf{vp}_A$ | | $\mathsf{vp} \multimap \mathsf{vp}$ | |
| $\mathsf{s}_A$ | | $\mathsf{s} \multimap \mathsf{s}$ | |
| $\mathsf{n}_A$ | | $\mathsf{n} \multimap \mathsf{np}$ | |
| $C_{sleeps}$ | $: \mathsf{s}_A \multimap \mathsf{vp}_A \multimap \mathsf{np} \multimap \mathsf{s}$ | $c_{sleeps}$ | $: (\mathsf{s} \multimap \mathsf{s}) \multimap (\mathsf{vp} \multimap \mathsf{vp}) \multimap \mathsf{np} \multimap \mathsf{s}$ |
| $C_{seemingly}$ | $: \mathsf{vp}_A \multimap \mathsf{vp}_A$ | $c_{seemingly}$ | $: (\mathsf{vp} \multimap \mathsf{vp}) \multimap \mathsf{vp} \multimap \mathsf{vp}$ |
| $I_{\mathsf{s}}$ | $: \mathsf{s}_A$ | $I_{\mathsf{s}}$ | $: \mathsf{s} \multimap \mathsf{s}$ |
| $I_{\mathsf{vp}}$ | $: \mathsf{vp}_A$ | $I_{\mathsf{vp}}$ | $: \mathsf{vp} \multimap \mathsf{vp}$ |
| $C_{matters}$ | $: \mathsf{vp}_A \multimap \mathsf{s} \multimap \mathsf{s}$ | $c_{matters}$ | $: (\mathsf{vp} \multimap \mathsf{vp}) \multimap \mathsf{s} \multimap \mathsf{s}$ |

$\mathscr{G}_{TAG}$ is 2nd-order:

- Parsing results apply
- Both from strings and from logical formulas! See the demo

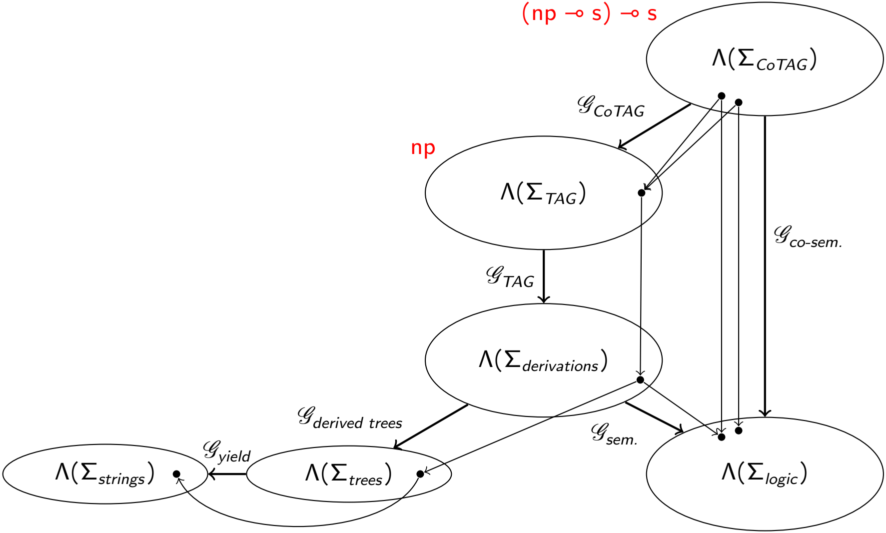
# Playing with the ACG Architecture

## Multi-Component TAG (MCTAG Weir 1988)



# Playing with the ACG Architecture

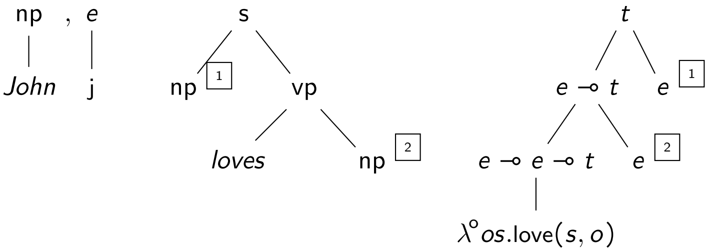
Scope Ambiguity: CoTAG (Barker 2010) as Type-raising (Pogodalla 2007b; Pogodalla 2007a; Kobele and Michaelis 2012)



# Related Approaches

## Synchronous TAG (Nesson and Shieber 2006; Nesson 2009)

Syntactic and semantic derivations are performed synchronously:




- Need to interpret the semantic tree as love(j, m)
- How to go from a logical formula $f$ ($\beta$-reduced) to some possible semantic tree that gets interpreted as $f$?
- Similar with Interpreted Regular Tree Grammars (Koller and Kuhlmann 2011; Koller and Kuhlmann 2012)

# Related Approaches

Semantics by Feature Unification (Gardent and Kallmeyer 2003; Kallmeyer and Romero 2004; Kallmeyer and Romero 2008)

Principles:

- Semantic features are added (typically [Ind = $x$], [P = $l$] where $l$ is the label of some logical formula)
- A representation language specifying the relations between the labels is used to plug together the logical formulas

Limitations:

- No control on whether a feature that is asked at some node will be provided by another node (no static typing)
- Need to handle variable naming on-line (no $\alpha$-conversion)
- The lack of functional application requires using labels (first-order unification)
- How to inverse the process?

# Conclusion

## ACGs

- Flexible framework
- Interesting computational properties
- Modularity by composition, and type-checking
- Provides a unified way to extend other formalisms
- Reversible
- Integrate well with type-theoretic base semantic modelling (discourse dynamics, intensionalization, etc.)

## Work in progress

- Tractable 3rd-order fragments
- Logical equivalence (theoretically and practically)
- Optimization
- Semantics conservativity

# Bibliographie I


Ajdukiewicz, Kazimierz (1935). "Die syntaktische Konnexitat". In: *Studia Philosophica* 1. English translation in Storrs McCall (ed), *Polish Logic 1920-1939*, Oxford University Press, pp. 207-231., pp. 1–27.

Bar-Hillel, Yehoshua (1953). "A Quasi-Arithmetical Notation for Syntactic Description". In: *Language* 29.1, pp. 47–58. ISSN: 00978507, 15350665. DOI: 10.2307/410452.

Barker, Chris (2010). "Cosubstitution, derivational locality, and quantifier scope". In: *Proceedings of the Tenth International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+10)*. New Haven, CT, pp. 135–142. URL: http://semanticsarchive.net/barker/barker-cosubstitution.pdf.

Benthem, Johan van (1986). *Essays in Logical Semantics*. Vol. 39. Studies in Linguistics and Philosophy. Springer Netherlands. DOI: 10.1007/978-94-009-4540-1.

# Bibliographie II


Curry, Haskell Brooks (1961). "Some Logical Aspects of Grammatical Structure". In: *Structure of Language and its Mathematical Aspects: Proceedings of the Twelfth Symposium in Applied Mathematics*. Ed. by Roman Jakobson. American Mathematical Society, pp. 56–68.

Danlos, Laurence, Aleksandre Maskharashvili, and Sylvain Pogodalla (2016). "Interfacing Sentential and Discourse TAG-based Grammars". In: *The 12th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+12)*. Proceedings of the 12th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+12). Düsseldorf, Germany.

de Groote, Philippe (2001). "Towards Abstract Categorial Grammars". In: *Association for Computational Linguistics, 39th Annual Meeting and 10th Conference of the European Chapter, Proceedings of the Conference*, pp. 148–155. ACL anthology: P01−1033.

# Bibliographie III


de Groote, Philippe (2002). "Tree-Adjoining Grammars as Abstract Categorial Grammars". In: *Proceedings of the Sixth International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+6)*. Università di Venezia, pp. 145–150. ACL anthology: W02-2220.

Gardent, Claire and Laura Kallmeyer (2003). "Semantic construction in Feature-Based TAG". In: *Proceedings of the 10th Meeting of the European Chapter of the Association for Computational Linguistics (EACL)*, pp. 123–130. ACL anthology: E03-1030.

Groote, Philippe de and Sylvain Pogodalla (2004). "On the expressive power of Abstract Categorial Grammars: Representing context-free formalisms". In: *Journal of Logic, Language and Information* 13.4, pp. 421–438. DOI: 10.1007/s10849-004-2114-x.

Howard, William Alvin (1980). "To H. B. Curry: Essays on combinatory logic, Lambda Calculus and Formalism". In: Academic Press. Chap. The Formulæ-as-Types Notion of Construction, pp. 479–490.

# Bibliographie IV


Joshi, Aravind K. (1994). "Preface". In: *Computational Intelligence* 10.4, pp. VII–XV. ISSN: 1467-8640. DOI: 10.1111/j.1467-8640.1994.tb00002.x.

Kallmeyer, Laura and Maribel Romero (2004). "LTAG Semantics with Semantic Unification". In: *Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms - TAG+7*, pp. 155–162. URL: http://www.sfs.uni-tuebingen.de/~lk/papers/kallmrom-tag+7.pdf.

Kallmeyer, Laura and Maribel Romero (2008). "Scope and Situation Binding for LTAG". In: *Research on Language and Computation* 6.1, pp. 3–52. DOI: 10.1007/s11168-008-9046-6.

Kanazawa, Makoto (2007). "Parsing and Generation as Datalog Queries". In: *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics (ACL)*. Prague, Czech Republic: Association for Computational Linguistics, pp. 176–183. ACL anthology: P07-1023.

# Bibliographie V

Kanazawa, Makoto (2008). "A prefix-correct Earley recognizer for multiple context-free grammars". In: *Proceedings of the Ninth International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+9)*. Tuebingen, Germany, pp. 49–56. URL: http://tagplus9.cs.sfu.ca/papers/Kanazawa.pdf.

Kanazawa, Makoto (2009). "Second-Order Abstract Categorial Grammars as Hyperedge Replacement Grammars". In: *Journal of Logic, Language, and Information* 19.2, pp. 137–161. DOI: 10.1007/s10849-009-9109-6.

Kanazawa, Makoto (2017). "Parsing and Generation as Datalog Query Evaluation". In: *IfCoLog Journal of Logics and their Applications* 4.4. Special Issue Dedicated to the Memory of Grigori Mints, pp. 1103–1211. URL: http://www.collegepublications.co.uk/downloads/ifcolog00013.pdf#page=303.

# Bibliographie VI

Kanazawa, Makoto and Sylvain Salvati (2007). "Generating Control Languages with Abstract Categorial Grammars". In: *Proceedings of The 12th conference on Formal Grammar FG 2007*. Ed. by Gerald Penn. CSLI Publications. URL: http://research.nii.ac.jp/~kanazawa/publications/control.pdf.

Kobele, Gregory M. (2012). "Idioms and Extended Transducers". In: *Proceedings of the 11th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+)*. Paris, France, pp. 153–161. URL: http://home.uchicago.edu/~gkobele/files/Kobele12Idioms.pdf.

Kobele, Gregory M. and Jens Michaelis (2012). "CoTAGs and ACGs". In: *Logical Aspects of Computational Linguistics*. Ed. by Denis Béchet and Alexander Dikovsky. Vol. 7351. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 119–134. ISBN: 978-3-642-31261-8. DOI: 10.1007/978-3-642-31262-5_8.

# Bibliographie VII


Koller, Alexander and Marco Kuhlmann (2011). "A Generalized View on Parsing and Translation". In: *Proceedings of the 12th International Conference on Parsing Technologies*. Dublin, Ireland: Association for Computational Linguistics, pp. 2–13. ACL anthology: W11-2902.

Koller, Alexander and Marco Kuhlmann (2012). "Decomposing TAG Parsing Algorithms Using Simple Algebraizations". In: *Proceedings of the 11th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+)*. Paris, France, pp. 135–143. URL: http://www.ida.liu.se/~marku61/pdf/koller2012decomposition.pdf.

Lambek, Joachim (1958). "The Mathematics of Sentence Structure". In: *American Mathematical Monthly* 65.3, pp. 154–170. DOI: 10.2307/2310058.

Maskharashvili, Aleksandre (2016). "Discourse Modeling with Abstract Categorial Grammars". PhD thesis. Université de Lorraine. HAL open archive: tel-01412765.

# Bibliographie VIII

Muskens, Reinhard (2001). "Lambda Grammars and the Syntax-Semantics Interface". In: *Proceedings of the Thirteenth Amsterdam Colloquium*. Ed. by Robert van Rooy and Martin Stokhof. Amsterdam, pp. 150–155. URL: https://let.uvt.nl/general/people/rmuskens/pubs/amscoll.pdf.

Muskens, Reinhard (2003). "Lambdas, Language, and Logic". In: *Resource Sensitivity in Binding and Anaphora*. Ed. by Geert-Jan Kruijff and Richard T. Oehrle. Studies in Linguistics and Philosophy. Kluwer, pp. 23–54. DOI: 10.1007/978-94-010-0037-6_2. URL: https://let.uvt.nl/general/people/rmuskens/pubs/lll.pdf.

Nesson, Rebecca Nancy (2009). "Synchronous and Multicomponent Tree-Adjoining Grammars: Complexity, Algorithms, and Applications". PhD thesis. Harvard University. URL: http://rebeccanesson.herokuapp.com/thesis.pdf.

Nesson, Rebecca Nancy and Stuart M. Shieber (2006). "Simpler TAG Semantics Through Synchronization". In: *Proceedings of the 11th Conference on Formal Grammar*. Malaga, Spain: CSLI Publications. URL: http://cslipublications.stanford.edu/FG/2006/nesson.pdf.

# Bibliographie IX


Oehrle, Richard T. (1994). "Term-Labeled Categorial Type Systems". In: *Linguistic and Philosophy* 17.6, pp. 633–678. DOI: 10.1007/BF00985321.

Oehrle, Richard T. (1995). "Some 3-Dimensional Systems of Labelled Deduction". In: *Logic Jnl IGPL* 3.2-3, pp. 429–448. DOI: 10.1093/jigpal/3.2-3.429. eprint: http://jigpal.oxfordjournals.org/cgi/reprint/3/2-3/429.pdf.

Pogodalla, Sylvain (2007a). "Ambiguïté de portée et approche fonctionnelle des TAG". In: *Traitement Automatique des Langues Naturelles - TALN 2007*. Toulouse, France, pp. 325–334.

Pogodalla, Sylvain (2007b). "Generalizing a Proof-Theoretic Account of Scope Ambiguity". In: *7th International Workshop on Computational Semantics - IWCS-7*. Tilburg, Netherlands.

Ranta, Aarne (1994). *Type Theoretical Grammar*. Oxford University Press.

Ranta, Aarne (2004). "Grammatical Framework: A Type-Theoretical Grammar Formalism". In: *Journal of Functional Programming* 14.2, pp. 145–189. DOI: 10.1017/S0956796803004738.

# Bibliographie X


- Salvati, Sylvain (2006). "Encoding second order string ACG with Deterministic Tree Walking Transducers". In: *Proceedings of The 11th conference on Formal Grammar FG 2006*. Ed. by Shuly Wintner. FG Online Proceedings. Malaga Espagne: CSLI Publications, pp. 143–156. URL: http://cslipublications.stanford.edu/FG/2006/salvati.pdf.
- Salvati, Sylvain (2007). "On the complexity of Abstract Categorial Grammars". In: *Proceedings of the 10th Conference on Mathematics of Language, MOL 10*. URL: http://wwwhomes.uni-bielefeld.de/mkracht/mol10/abstracts/acg_complexity.pdf.
- Vijay-Shanker, K. and Aravind K. Joshi (1988). "Feature Structures Based Tree Adjoining Grammars". In: *Proceedings of the 12th International Conference on Computational Linguistics (COLING Budapest)*. Ed. by Dénes Vargha. Vol. 2. Budapest, pp. 714–718. ACL anthology: C88-2147.
- Weir, David J. (1988). "Characterizing Mildly Context-Sensitive Grammar Formalisms". PhD thesis. University of Pennsylvania.