Course Material

At http://www.loria.fr/equipes/calligramme/acg/

- Slides and handout (1st week and 2nd week)
- Lecture Notes (2nd week, moving version)
- Software and examples (2nd week)

Advances in Abstract Categorial Grammars: Language Theory and Linguistic Modeling

Formal-Language-Theoretic Properties of 2nd Order ACG[1]
Syntax-Semantics Interface: an ACG Perspective[2]

Makoto Kanazawa[1] Sylvain Pogodalla[2]

[1] kanazawa@nii.ac.jp
NII Tokyo
Japan
[2] sylvain.pogodalla@loria.fr
LORIA/INRIA Nancy–Grand Est
France

21st ESSLLI – Bordeaux, France
July 27–31, 2009

Outline
First Lecture

Outline (cont'd)

Second Lecture

The Tectogrammatical and Phenogrammatical Distinction

On the Grammar Architecture [Curry(1961)]

- Tectogrammatical: abstract combinatorial structure of the grammar
- Phenogrammatical: concrete operations on syntactic data structures (strings, trees, descriptions)
- Contrary to the view that:
  - Syntactic objects are the main objects
  - Semantics (and phonology, and . . .) are by-products

Related Works

[Montague(1974)], [Dowty(1982)],[Ranta(1994)],[Oehrle(1994), Oehrle(1995)], [Muskens(2001)], [Muskens(2003)], [Kracht(2003)], [Pollard(2004)], [Pollard(2008)]. . .

Some Observations on Various Grammatical Formalisms

Syntactic Objects (trees, proofs, f-structures) are somehow prior and semantics must be parasitic on those syntactic objects

[Muskens(2001)]

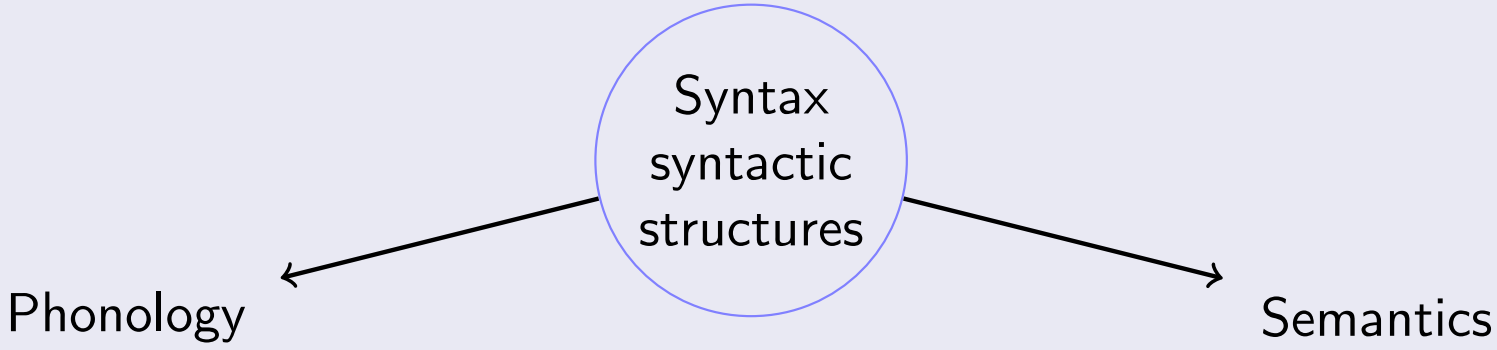
Changing the syntactic analysis to simplify one mapping makes the other mapping more complex. A third possibility is to keep both correspondences simple by localizing the complexity in the syntactic component itself.(...) [T]here is a mismatch between phonology and meaning, which has to be encoded somewhere in the mapping among the levels of structure. If this mismatch is eliminated at one point in the system, it pops up elsewhere.

[Jackendoff(2002), p.15]

Mainstream Architectures

On the Place of the Syntactic Component

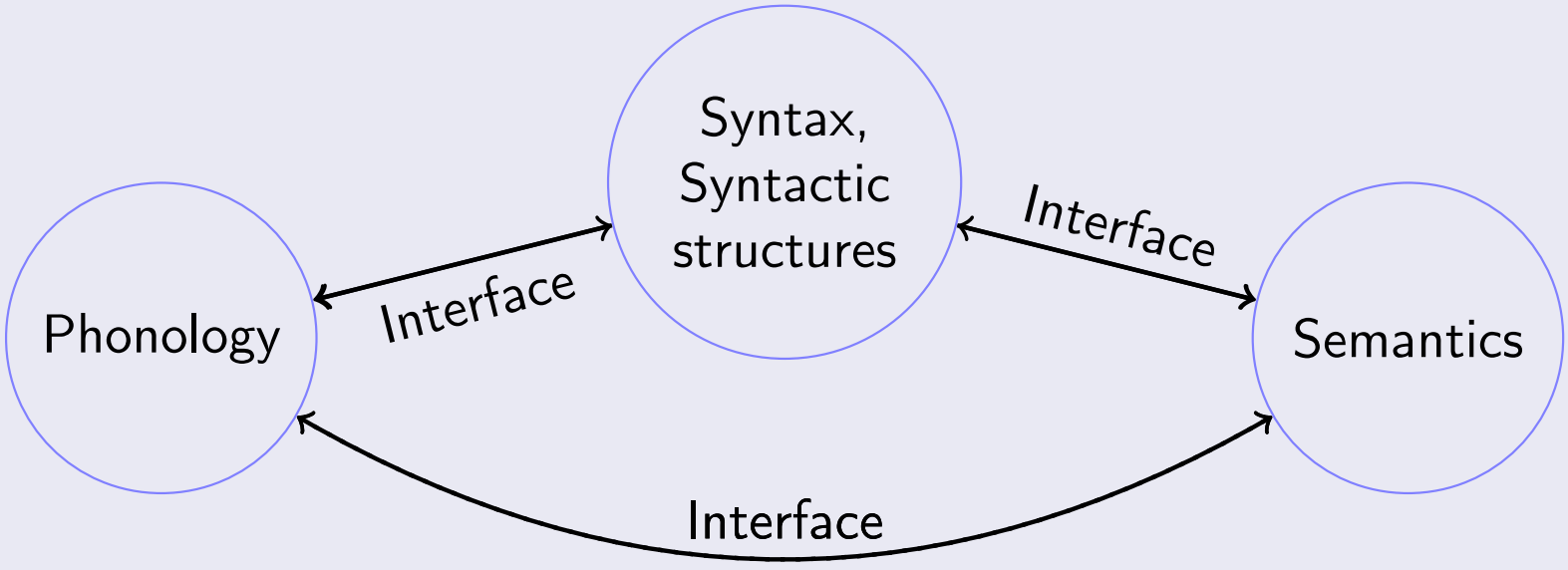
Three Components



- Generative theory: "Free combinatoriality of language is due to a single source, localized in syntactic structure"
- Syntactocentric formalisms = function from the syntactic component to the other ones

A Tripartite Parallel Architecture

Three Components



Weakly Syntactocentric formalisms = relation between the syntactic component and the other ones

Language comprises a number of independent combinatorial systems which are aligned with each other by means of a collection of interface systems. Syntax is among the combinatorial systems, but far from the only one. [Jackendoff(2002)]

Grammatical Formalisms
A Classification by Architecture

General Classification

	Cascaded	Parallel (Curryesque)
Syntactocentric	GB, MG	CFG, TAG (?), ACG (?)
(Weakly) syntactocentric		HPSG, CVG

The Syntax-Semantics Interface

The combinatorial principles of syntax and semantics are independent; there is no "rue-to-rule" homomorphism. (...) [T]he mapping between syntactic and semantic combinatoriality is many-to-many.

[Culicover and Jackendoff(2005)]

In-situ Operators

- Quantified noun phrases, interrogative wh-expressions, topicalization...
- Their semantic scope is under-determined by their syntactic position

λ-terms in the Syntax... and Everywhere

What are λ-terms useful for?

- Montague-like semantics
- Generalization of trees and strings
- Any kind of signatures (atomic types and typed constants): FOL propositions, descriptions (LFG f-structures, URL), other logics
- Very well studied generative system
- Variable binding system

Not that New in Syntax

- [Ranta(1994)],[Oehrle(1994), Oehrle(1995)], [Muskens(2001)], [Muskens(2003)], [Kracht(2003)], [Pollard(2004)], [Pollard(2008)]...
- Movements in GB/MG to get S-structures.
- Index Transfer syntactic rule in Binding Theory
- TAG $\rightarrow$ MCTAG

ACG: a Grammatical Framework

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* (as in syntactic structures)
  Object language: *Realizations* of the admissible structures
- Both languages are the same objects: sets of (linear) λ-terms

ACG Definition $\mathscr{G} = \langle \Sigma_a, \Sigma_o, \mathscr{L}, s \rangle$

Σ_a

np, s : *type*

Chris : np

met : $np \multimap np \multimap s$

$\mathscr{L}(np) = \sigma$
$\mathscr{L}(s) = \sigma$
$\mathscr{L}(\text{Chris}) = /Chris/$
$\mathscr{L}(\text{met}) = \lambda os.s + /met/ + o$

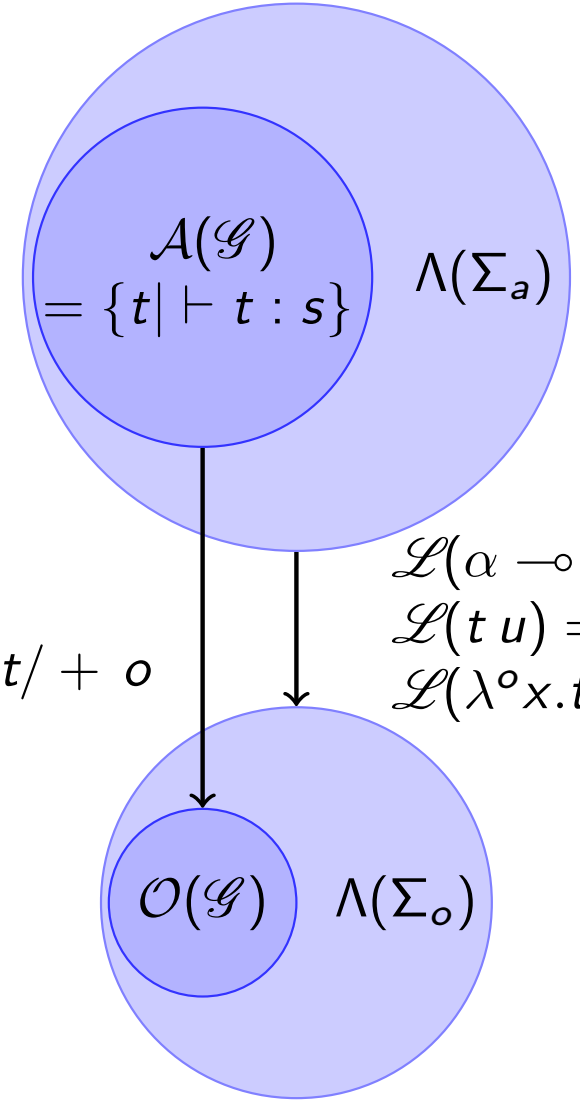
Σ_o

σ : *type*

$/Chris/$: σ

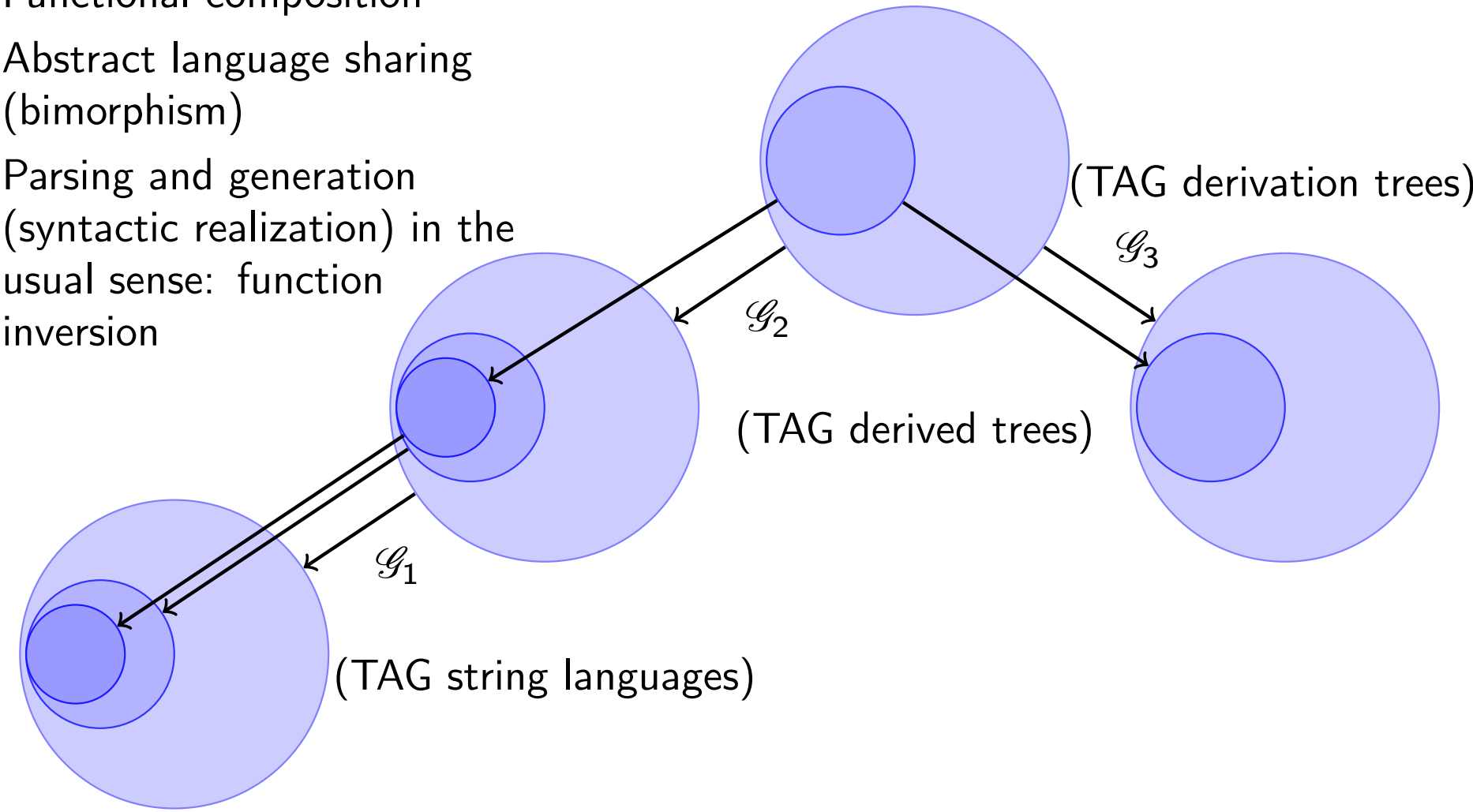
$/met/$: σ

$+$: $\sigma \multimap \sigma \multimap \sigma$



ACG Architecture
Composition Ability

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



About Word Order

My First "Chris met Sandy" ACG Program

Σ_a :

np, s : $type$

Chris, Sandy : np

met : $np \multimap np \multimap s$

$np := \sigma$

$s := \sigma$

Chris := $/Chris/$

Sandy := $/Sandy/$

met := $\lambda^{o} os.s + /met/ + o$

Σ_o :

σ : $type$

$+$: $\sigma \multimap \sigma \multimap \sigma$

$/Chris/, /Sandy/$: σ

$/met/$: σ

met Sandy Chris : s

$\mathscr{L}$(met Sandy Chris)

$= \mathscr{L}$(met) $\mathscr{L}$(Sandy) $\mathscr{L}$(Chris)

$= (\lambda^{o} os.s + /met/ + o)(/Sandy/)(/Chris/)$

$= (\lambda^{o} s.s + /met/ + /Sandy/)(/Chris/)$

$= /Chris/ + /met/ + /Sandy/$

About Word Order

Getting Higher-Order Exercise

See `example-01.acg`

Σ_a :

np, s :	$type$
Chris, Sandy :	np
met :	$np \multimap np \multimap s$
that :	$(np \multimap s) \multimap np \multimap np$

$\text{that}\,(\lambda^{\circ} t.\text{met}\; t\; \text{Chris})\;\text{Sandy} : np$

Chris :=	$/Chris/$
Sandy :=	$/Sandy/$
met :=	$\lambda^{\circ} os.s + /met/ + o$
that :=	$\lambda^{\circ} Pn.n + /that/ + (P\,\epsilon)$

Σ_o :

$/Chris/$:	σ
$/Sandy/$:	σ
$/met/$:	σ
$/that/, \epsilon$:	σ

$$
\begin{aligned}
&\mathscr{L}(\text{that}\,(\lambda^{\circ} t.\text{met}\; t\; \text{Chris})\;\text{Sandy}) \\
&= \mathscr{L}(\text{that})\,(\lambda^{\circ} t.\mathscr{L}(\text{met})\; t\; \mathscr{L}(\text{Chris}))\;\mathscr{L}(\text{Sandy}) \\
&= \mathscr{L}(\text{that})\,(\lambda^{\circ} t.(\lambda^{\circ} os.s + /met/ + o)\; t\; /Chris/)\;/Sandy/ \\
&= \mathscr{L}(\text{that})\,(\lambda^{\circ} t.(\lambda^{\circ} s.s + /met/ + t)\; /Chris/)\;/Sandy/ \\
&= \mathscr{L}(\text{that})\,(\lambda^{\circ} t./Chris/ + /met/ + t)\;/Sandy/ \\
&= (\lambda^{\circ} Pn.n + /that/ + (P\,\epsilon))\,(\lambda^{\circ} t./Chris/ + /met/ + t)\;/Sandy \\
&= (\lambda^{\circ} n.n + /that/ + ((\lambda^{\circ} t./Chris/ + /met/ + t)\,\epsilon))\;/Sandy/ \\
&= (\lambda^{\circ} n.n + /that/ + (/Chris/ + /met/ + \epsilon))\;/Sandy/ \\
&= /Sandy/ + /that/ + (/Chris/ + /met/ + \epsilon) \\
&= /Sandy/ + /that/ + /Chris/ + /met/
\end{aligned}
$$

About Word Order
Medial Extraction

$$\Sigma_a :$$

$$\begin{array}{ll}
\text{Chris, Sandy} : & np \\
\text{met} : & np \multimap np \multimap s \\
\text{that} : & (np \multimap s) \multimap np \multimap np \\
\text{yesterday} : & s \multimap s
\end{array}$$

$$\text{that}\ (\lambda^{o} t.\text{yesterday}\ (\text{met}\ t\ \text{Chris}))\ \text{Sandy} : np$$

$$\begin{array}{ll}
\text{Chris} := & /Chris/ \\
\text{Sandy} := & /Sandy/ \\
\text{met} := & \lambda^{o} os.s + /met/ + o \\
\text{that} := & \lambda^{o} Pn.n + /that/ + (P\,\epsilon) \\
\text{yesterday} & := \lambda^{o} s.s + /yesterday/
\end{array}$$

$$\Sigma_o :\quad \mathscr{L}(\text{that}\ (\lambda^{o} t.\text{yesterday}\ (\text{met}\ t\ \text{Chris}))\ \text{Sandy})$$

$$\begin{array}{lll}
/Chris/ : & \sigma & = \mathscr{L}(\text{that})\ (\lambda^{o} t.(\lambda s.s + /yesterday/)\ (/Chris/ + /met/ + t))\ /Sandy/ \\
/Sandy/ : & \sigma & = \mathscr{L}(\text{that})\ (\lambda^{o} t./Chris/ + /met/ + t + /yesterday/)\ /Sandy/ \\
/met/ : & \sigma & = (\lambda^{o} n.n + /that/ + (/Chris/ + /met/ + \epsilon + /yesterday/))\ /Sandy/ \\
/that/, \epsilon : & \sigma & = /Sandy/ + /that/ + (/Chris/ + /met/ + \epsilon + /yesterday/) \\
/yesterday/ : & \sigma & = /Sandy/ + /that/ + /Chris/ + /met/ + /yesterday/
\end{array}$$

Intermediate Conclusion

So far...

- Discussion on possible architectures of grammatical formalisms
- Discussion on function-compositional properties of ACG and *modularity*
- Word order in ACG:
  - Not in the logic of combinatorics, rather at the object level

  $\Rightarrow$ (Possibly straightforward) Encoding of various formalisms

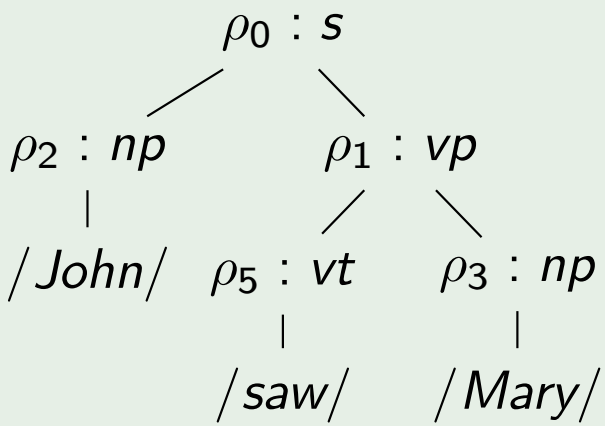
Next to come: Examples!

- Modularity:
  - CFG: at the syntax-semantics interface
  - TAG: at the syntactic level
  - TAG: at the syntax-semantics interface
- Parallel architecture:
  - CG: What do we call *syntax*?
  - CVG: What do we call *interface*?

CFG into ACG Encoding

Example (CFG)

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



CFG as ACG

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

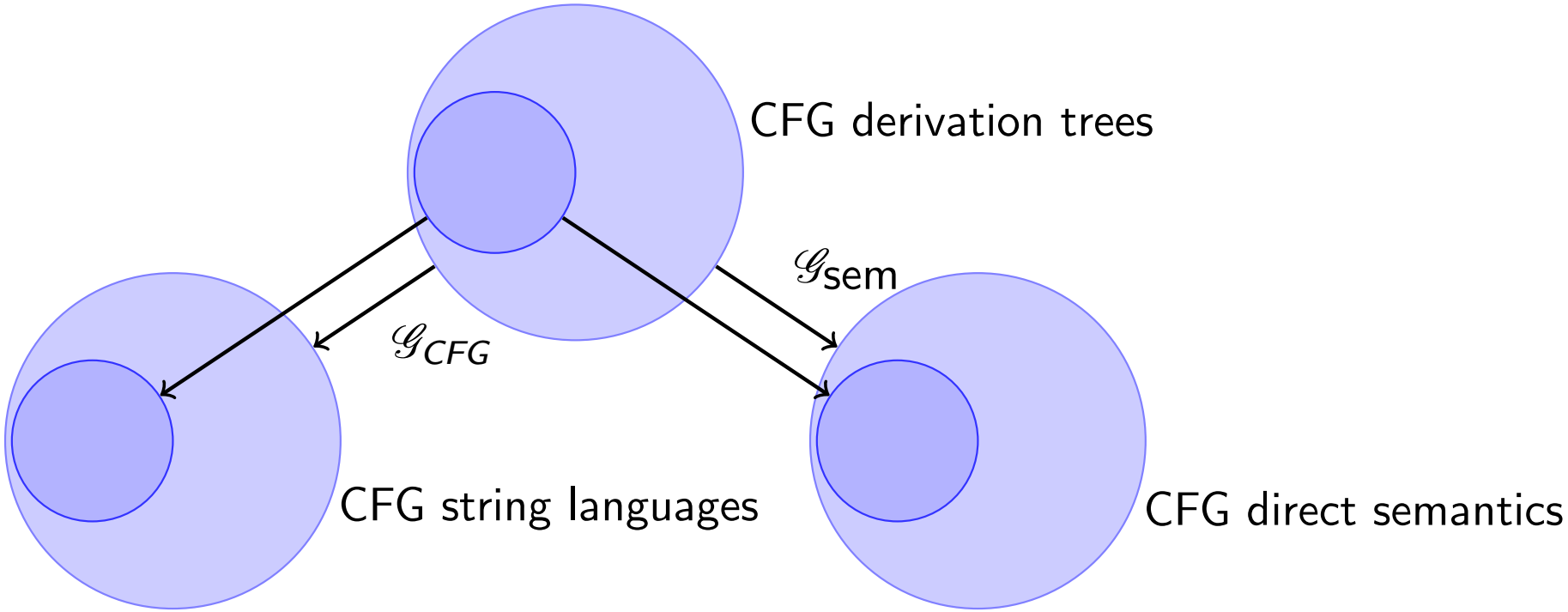
CFG into ACG Encoding (cont'd)

CFG as ACG

	Σ_{Rules}	$\mathscr{L}_{\text{CFG}}$	$\Sigma_{Strings}$
ρ_0	$: np \multimap vp \multimap s$	$:=$	$\lambda xy.x + y \quad : \sigma \multimap \sigma \multimap \sigma$
ρ_1	$: vt \multimap np \multimap vp$	$:=$	$\lambda xy.x + y \quad : \sigma \multimap \sigma \multimap \sigma$
ρ_2	$: np$	$:=$	$/John/ \quad : \sigma$
ρ_3	$: np$	$:=$	$/Mary/ \quad : \sigma$
ρ_4	$: vp$	$:=$	$/left/ \quad : \sigma$
ρ_5	$: vt$	$:=$	$/saw/ \quad : \sigma$

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

CFG Encoding



A Direct Semantics

Sharing Abstract Languages

CFG syntax as ACG

	Σ_{Rules}		$\mathscr{L}_{\text{CFG}}$	$\Sigma_{Strings}$	
ρ_0	$: np \multimap vp \multimap s$	$:=$	$\lambda xy.x + y$	$: \sigma \multimap \sigma \multimap \sigma$
ρ_1	$: vt \multimap np \multimap vp$	$:=$	$\lambda xy.x + y$	$: \sigma \multimap \sigma \multimap \sigma$
ρ_2	$: np$	$:=$	$/John/$	$: \sigma$
ρ_3	$: np$	$:=$	$/Mary/$	$: \sigma$
ρ_4	$: vp$	$:=$	$/left/$	$: \sigma$
ρ_5	$: vt$	$:=$	$/saw/$	$: \sigma$

CFG (direct) semantics as ACG

	Σ_{Rules}		$\mathscr{L}_{\text{sem}}$	Σ_{Log}
ρ_0	$: np \multimap vp \multimap s$	$:=$	$\lambda sP.P\,s$	$: e \multimap (e \multimap t) \multimap t$
ρ_1	$: vt \multimap np \multimap vp$	$:=$	$\lambda Pos.P\,s\,o$	$: (e \multimap e \multimap t) \multimap e \multimap e \multimap t$
ρ_2	$: np$	$:=$	John	$: e$
ρ_3	$: np$	$:=$	Mary	$: e$
ρ_4	$: vp$	$:=$	left	$: e \multimap t$
ρ_5	$: vt$	$:=$	saw	$: e \multimap e \multimap t$

A Direct Semantics (cont'd)

CFG (direct) semantics as ACG

	Σ_{Rules}	$\mathscr{L}_{\text{sem}}$	Σ_{Log}	
ρ_0	$: np \multimap vp \multimap s$	$:=$	$\lambda sP.P\,s$	$: e \multimap (e \multimap t) \multimap t$
ρ_1	$: vt \multimap np \multimap vp$	$:=$	$\lambda Pos.P\,s\,o$	$: (e \multimap e \multimap t) \multimap e \multimap e \multimap t$
ρ_2	$: np$	$:=$	John	$: e$
ρ_3	$: np$	$:=$	Mary	$: e$
ρ_4	$: vp$	$:=$	left	$: e \multimap t$
ρ_5	$: vt$	$:=$	saw	$: e \multimap e \multimap t$

$$
\begin{aligned}
\mathscr{L}_{\text{sem}}(\rho_0\ \rho_2\ (\rho_1\ \rho_5 \rho_3) : s) &= (\lambda sP.P\,s)\text{John}((\lambda Pos.P\,s\,o)\text{saw Mary}) \\
&\to_\beta (\lambda P.P\ /John/)((\lambda os.\text{saw}\,s\,o)\text{Mary}) \\
&\to_\beta (\lambda P.P\ \text{John})(\lambda^\circ s.\text{saw}\,s\,\text{Mary}) \\
&\to_\beta (\lambda^\circ s.\text{saw}\,s\,/Mary/)\text{John} \\
&\to_\beta \text{saw John Mary}
\end{aligned}
$$

See `example-02.acg`.

A Continuized (Higher–Order) Semantics

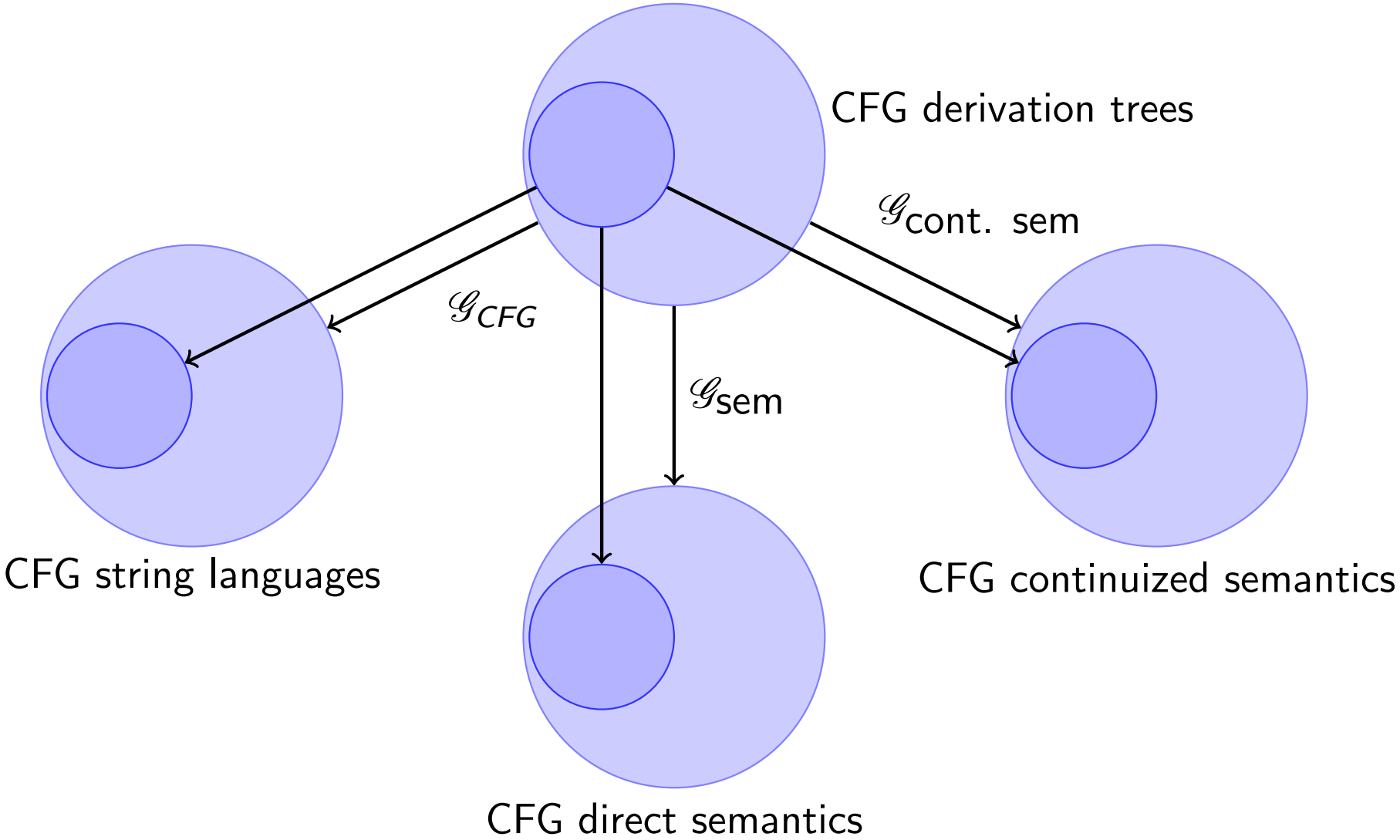
CFG continuized semantics as ACG [Barker(2002)]

	Σ_{Rules}	$\mathscr{L}_{\text{sem}}$	Σ_{Log}
s		$:=$	$(t \multimap t) \multimap t$
np		$:=$	$(e \multimap t) \multimap t$
vp		$:=$	$((e \multimap t) \multimap t) \multimap t$
vt		$:=$	$((e \multimap e \multimap t) \multimap t) \multimap t$
n		$:=$	$((e \multimap t) \multimap t) \multimap t$
det		$:=$	$(((e \multimap t) \multimap t) \multimap t) \multimap (e \multimap t) \multimap t$
ρ_0	$: np \multimap vp \multimap s$	$:=$	$\lambda^o svp.v(\lambda^o P.s(\lambda^o x.p(Px)))$
ρ_1	$: vt \multimap np \multimap vp$	$:=$	$\lambda^o voP.v(\lambda^o R.o(\lambda^o y.P(Ry)))$
ρ_2	$: np$	$:=$	$\lambda^o P.P\,\text{John}$
ρ_5	$: vt$	$:=$	$\lambda^o P.P\,\text{saw}$
ρ_{every}	$: det$	$:=$	$\lambda^o KP.K(\lambda^o Q.\forall x.(Q\,x) \Rightarrow (P\,x))$

See `example-03.acg`:

- Scope ambiguity
- Scope displacement
- NP as a scope island

CFG Encoding



Exercise

- Let $\ell : \mathbb{N} \to \mathbb{N}$ be: $\ell(n) = \begin{cases} k & \text{if } n = 2^k \\ 1 & \text{otherwise} \end{cases}$
- If $w = x_1 \dots x_n$, $w^t = x_n \dots x_1$.
- $|w|_a$ is the number of occurrences of a in w
- $|w|_b$ is the number of occurrences of b in w

Exercise

Can you find $\mathscr{G}_1$ and $\mathscr{G}_2$ such that:

- $\mathcal{O}(\mathscr{G}_1) = \{wcw^t | w \in (a|b)^*\}$;
- for all $t \in \mathcal{A}(\mathscr{G}_1)$ such that $t :=_{\mathscr{G}_1} w$, $t :=_{\mathscr{G}_2} \ell^{|w|_b}(|w|_a)$

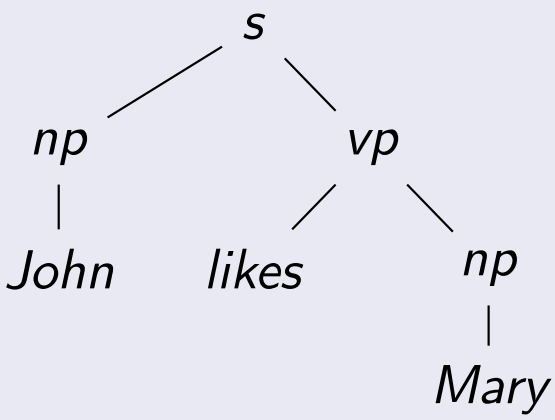
Intermediate Conclusion

Exemplified on CFG

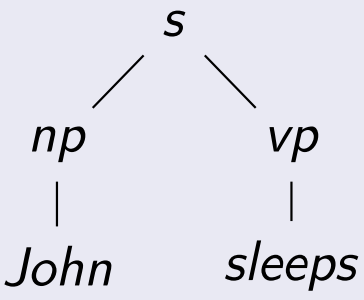
- Abstract structures are mapped to object structures
- Object structures:
  - Strings
  - Simple semantic objects
  - Complex semantic objects:
    - Continuized semantics: $s := (t \multimap t) \multimap t$
    - Results of ACG composition: $s := \mathbb{N} \times \mathbb{N} := (\mathbb{N} \multimap \mathbb{N} \multimap \mathbb{N}) \multimap \mathbb{N}$
    - Dynamic semantics: $s := \gamma \multimap (\gamma \multimap t) \multimap t$ [de Groote(2006)]
    - Underspecified representations
  - Algorithms for parsing and generation (in the usual sense) are essentially the same: ACG parsing : finding the abstract antecedent of an object
- Abstract structures?

Trees as λ-Terms

Trees Build on a Ranked Alphabet



$s_2(np_1\,John)(vp_2\,likes\,(np_1\,Mary))$



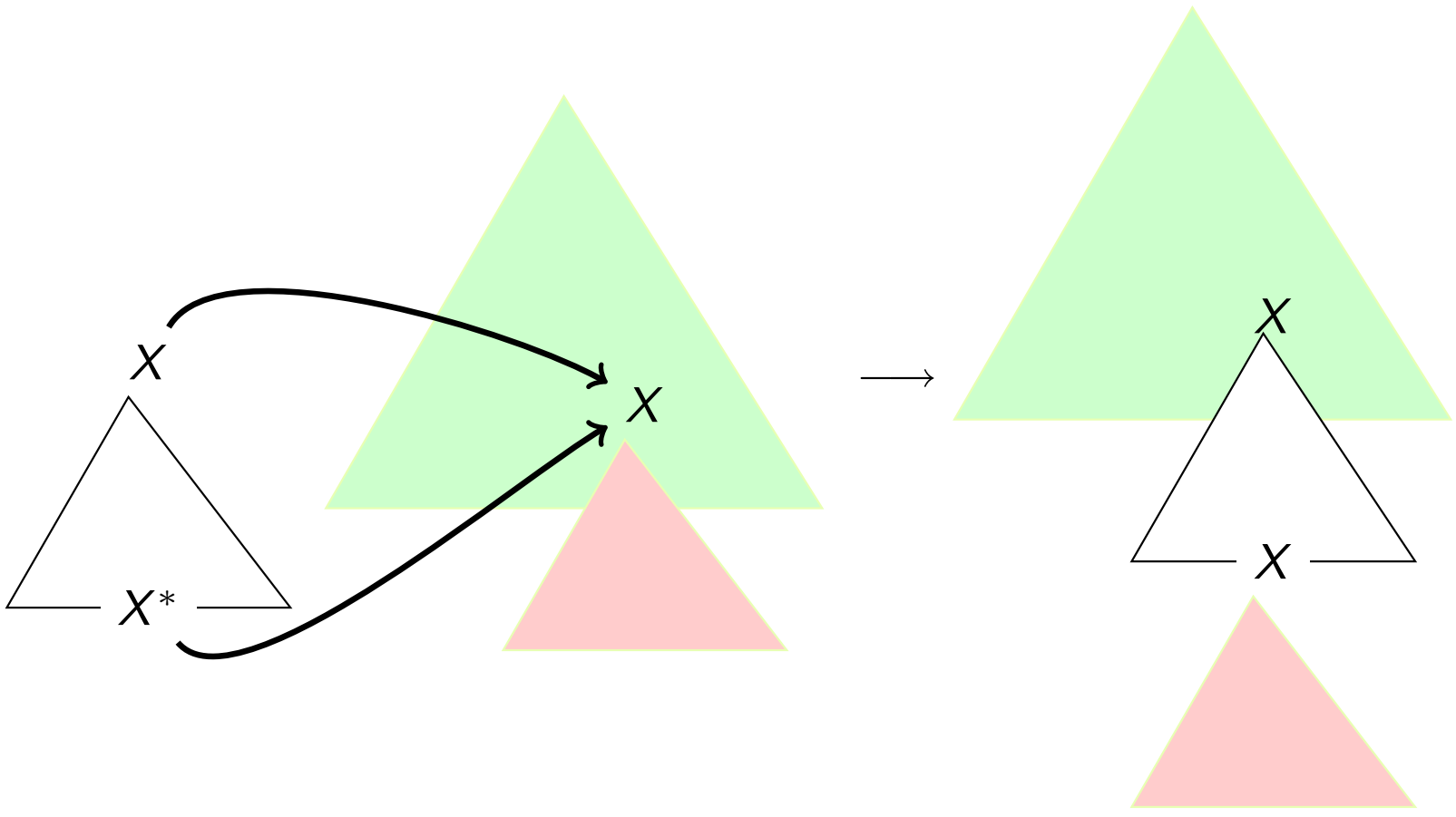
$s_2(np_1\,John)(vp_1\,sleeps)$

- 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)

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

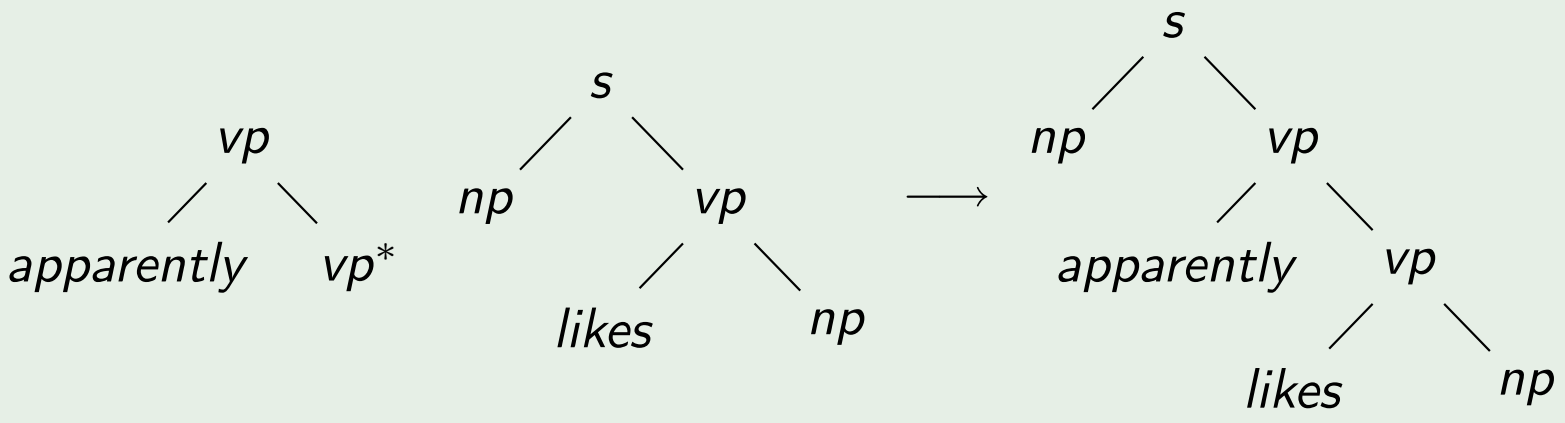
A Functional View on TAG

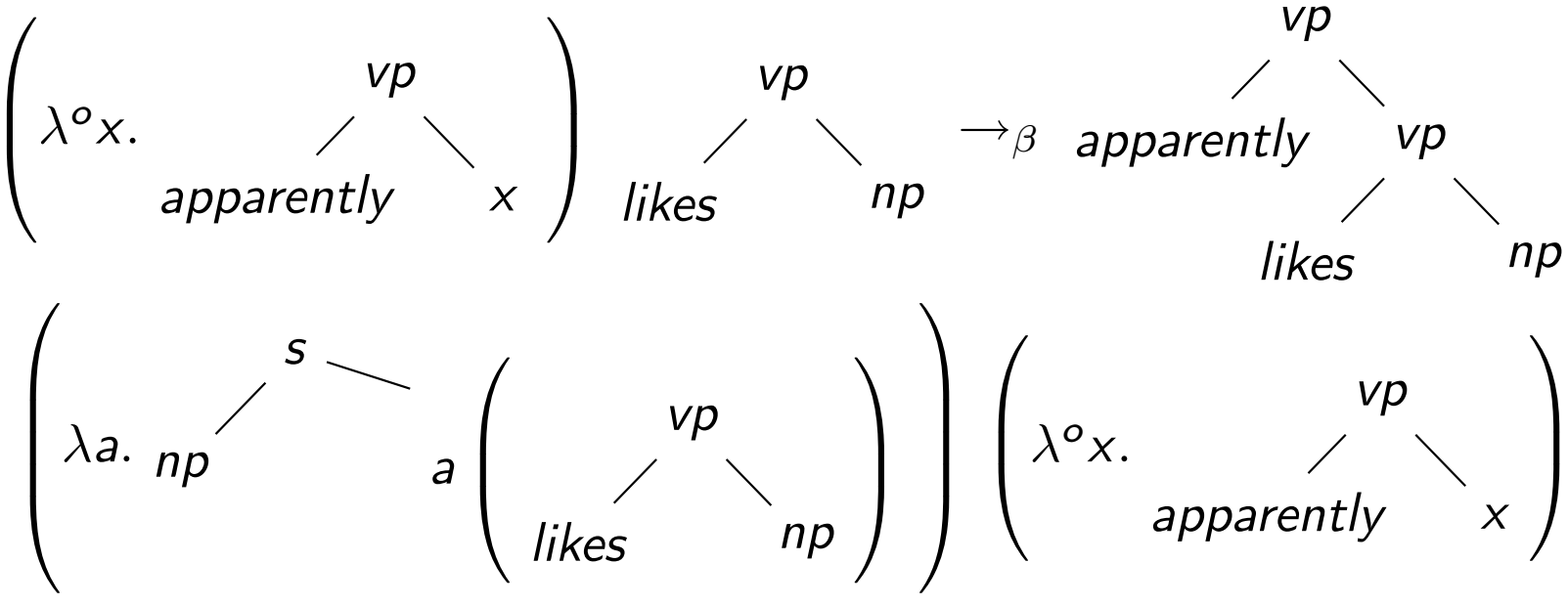
Tree Adjunction:



Auxiliary Trees as Functions

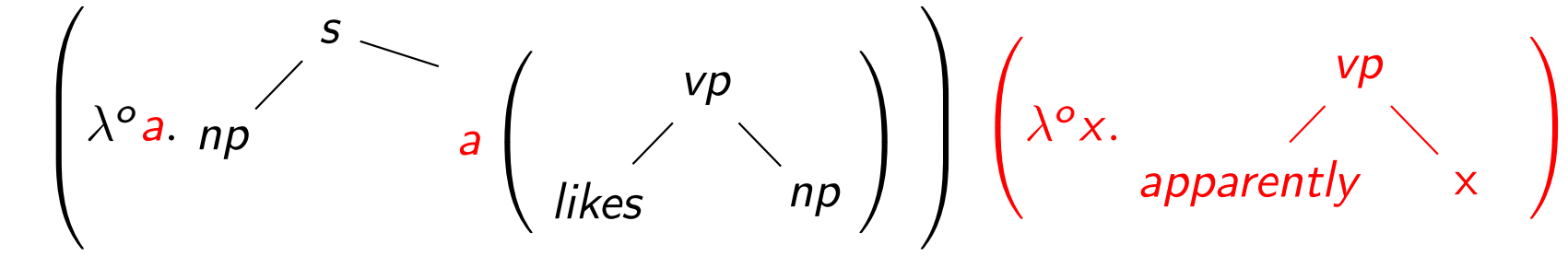
Example

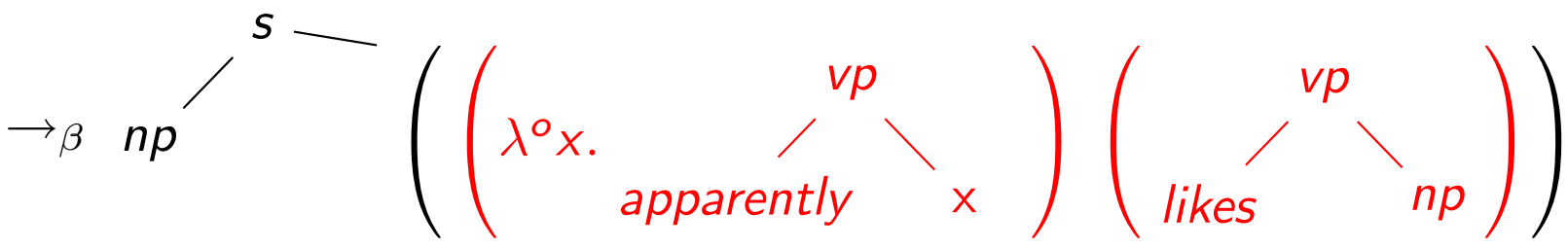


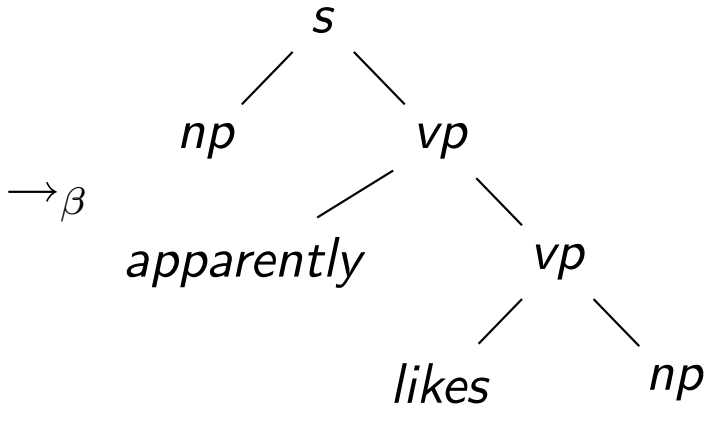


Adjunction as Functional Application

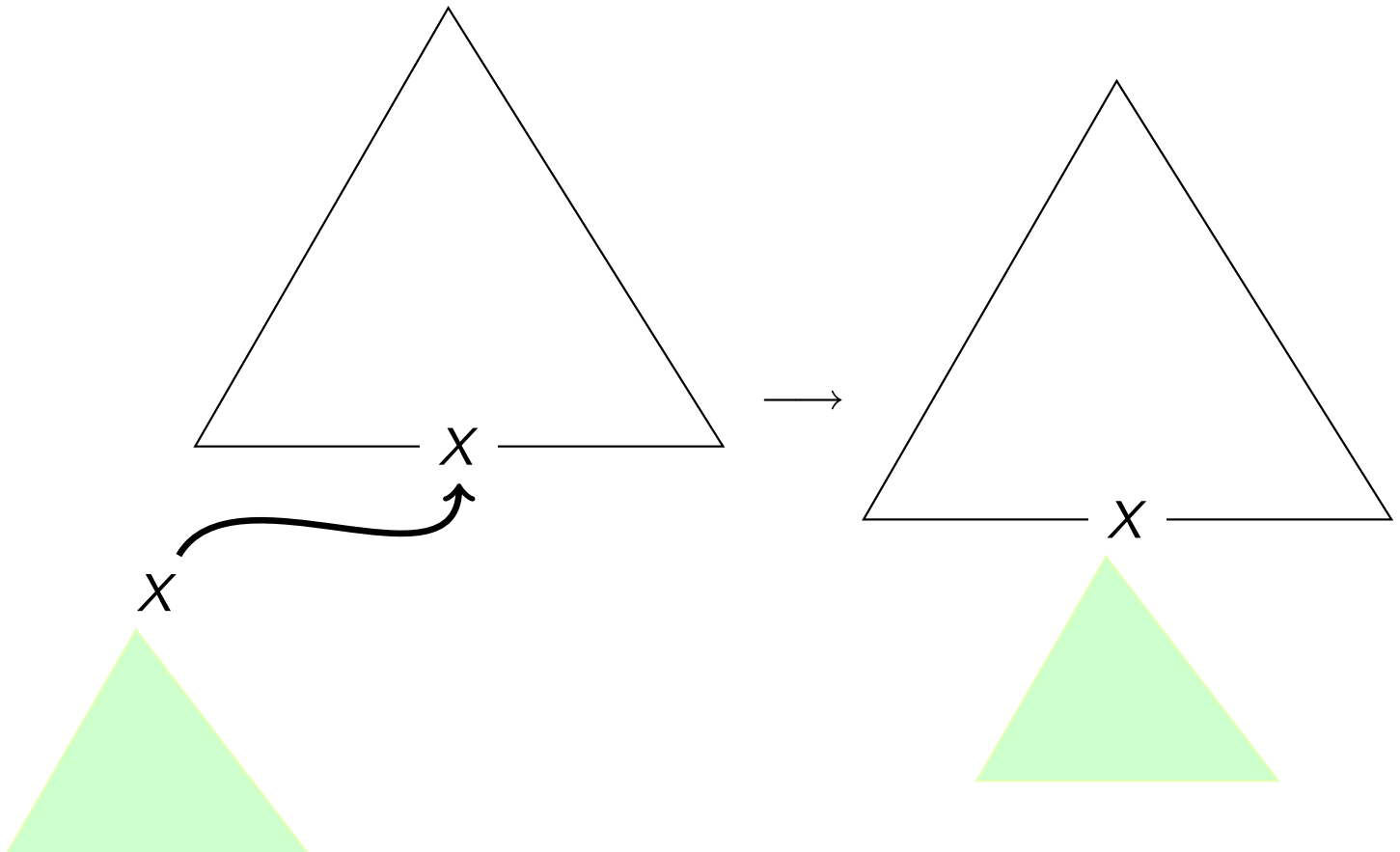
$\gamma'_{likes}\ \gamma'_{apparently} =$





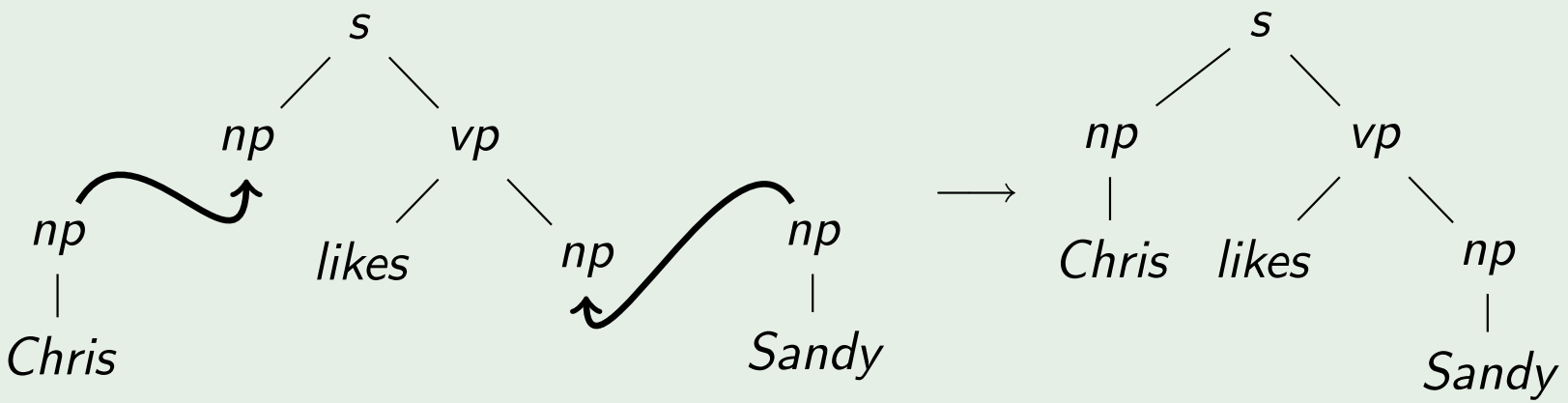


Substitution Operation



Substitution as Functional Application

Example

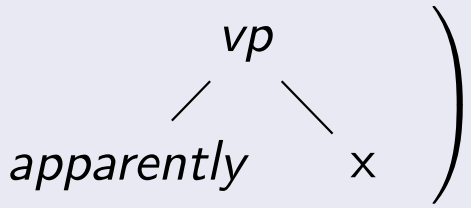


$$\left(\lambda^{o} os.\ \begin{array}{c} s \\ s \quad vp \\ \quad\ likes \quad o \end{array}\right) \left(\begin{array}{c} np \\ | \\ Chris \end{array}\right) \left(\begin{array}{c} np \\ | \\ Sandy \end{array}\right) \rightarrow_{\beta} \begin{array}{c} s \\ np \quad vp \\ | \quad\ \ | \quad\ \ \backslash \\ Chris \quad likes \quad np \\ \qquad\qquad\ \ | \\ \qquad\qquad Sandy \end{array}$$

Putting Everything Together

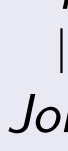
Σ_{trees}:

τ : *type*

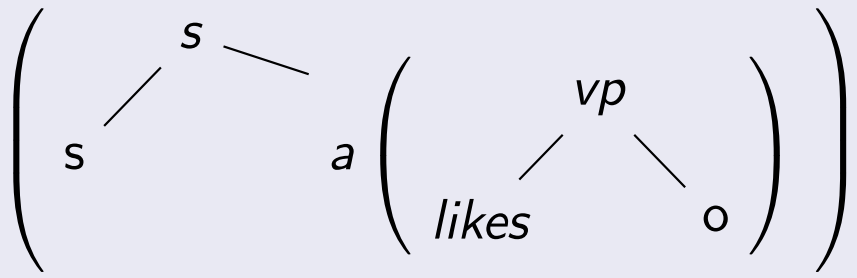
$\gamma_{apparently}$ $= \lambda^{o} ax.a$ 

$: (\tau \multimap \tau) \multimap \tau \multimap \tau$

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

γ_{John} $=$ 

$: \tau$

γ_{likes} $= \lambda^{o} Saso.S$ 

$: (\tau \multimap \tau) \multimap (\tau \multimap \tau) \multimap \tau \multimap \tau \multimap \tau$

TAG Derivation as Term Application

Example

$\gamma_{likes}\, I\, I\, \gamma_{John}\, \gamma_{Mary} =$

$$\left(\lambda^{o} Saso.S\left(\begin{array}{ccc} & s & \\ s & & a\left(\begin{array}{ccc} & vp & \\ likes & & o \end{array}\right) \end{array}\right)\right) (\lambda^{o}x.x)\,(\lambda^{o}x.x) \left(\begin{array}{c} np \\ | \\ John \end{array}\right) \left(\begin{array}{c} np \\ | \\ Mary \end{array}\right)$$

$$\rightarrow_{\beta} \left(\lambda^{o} so.\left(\begin{array}{ccc} & s & \\ s & & \left(\begin{array}{ccc} & vp & \\ likes & & o \end{array}\right) \end{array}\right)\right) \left(\begin{array}{c} np \\ | \\ John \end{array}\right) \left(\begin{array}{c} np \\ | \\ Mary \end{array}\right)$$

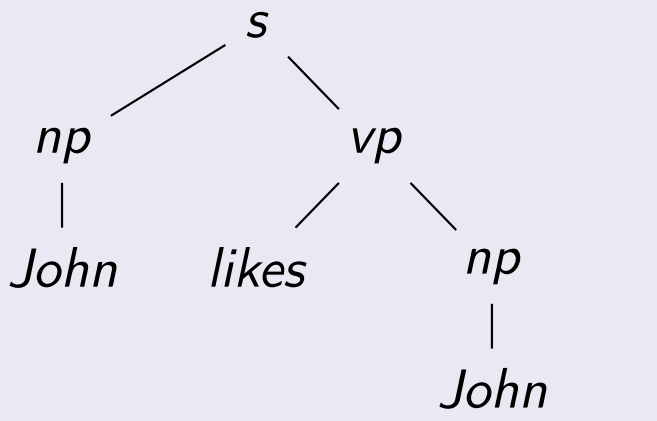
$$\rightarrow_{\beta} \begin{array}{ccccc} & & s & & \\ & np & & vp & \\ & | & & & \\ John & likes & & & np \\ & & & & | \\ & & & & Mary \end{array}$$

See `example-04.acg`

Yield as an ACG

Derived Tree Signature

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



$s_2(np_1\, John)(vp_2\, likes\, (np_1\, Mary))$

String signature (as before):

σ	: type
$/John/, /likes/ \ldots$	: σ

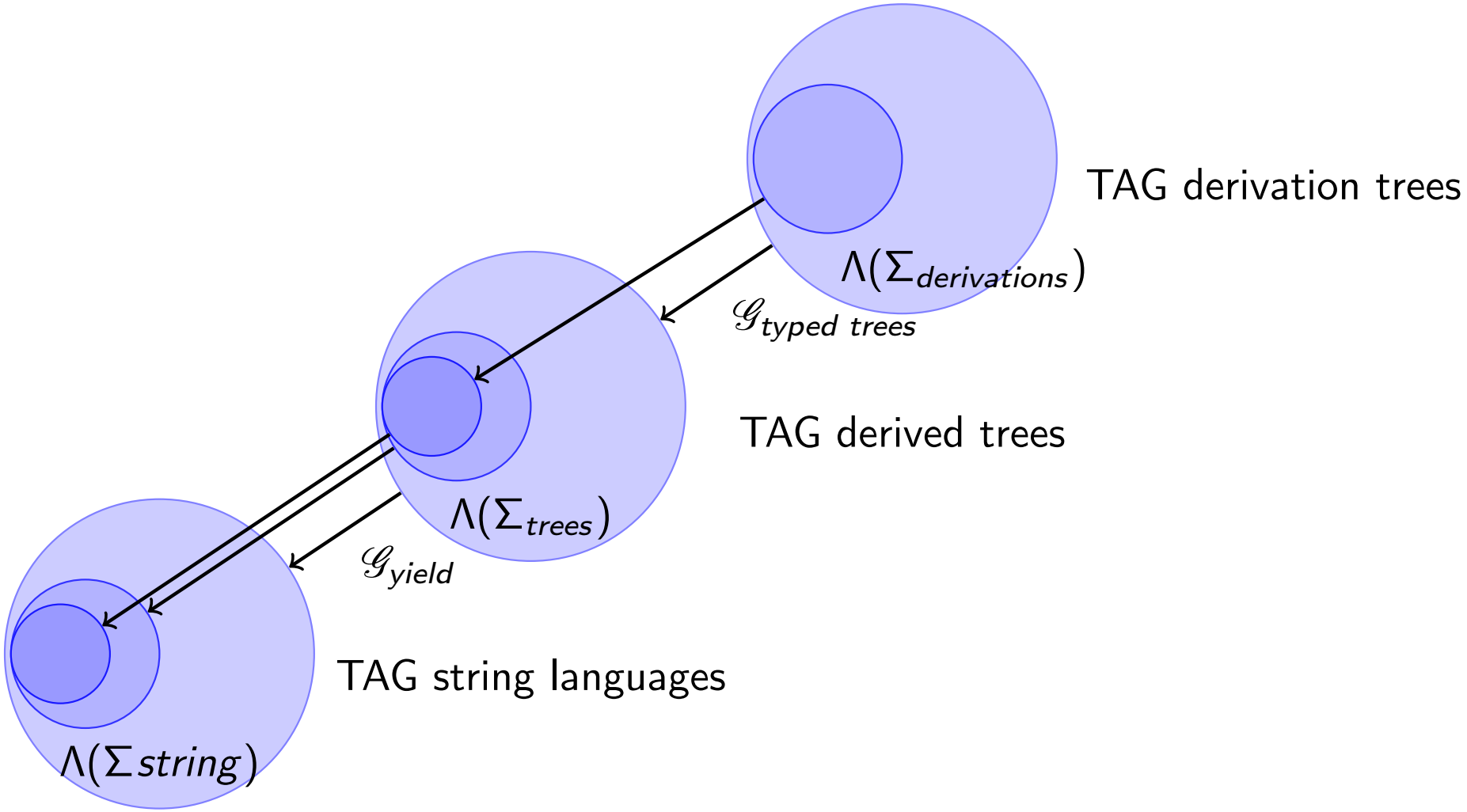
$\mathscr{G}_{Yield}$

τ	$:= \sigma$	$John$	$:= /John/$
X_1	$:= \lambda x.x$	X_2	$:= \lambda xy.x + y$
$\ldots$			

$$s_2(np_1\, John)(vp_2\, likes\, (np_1\, Mary)) := /John/ + (/likes/ + /Mary/)$$

TAG as ACG

The Current Picture



$\mathcal{A}(\mathscr{G}_{yield}) =$ TAG Derived Trees?

Σ_{trees}:

$$\tau : type$$

$$\gamma_{apparently} = \lambda^{o} ax.a \left(\begin{array}{ccc} & vp & \\ \diagup & & \diagdown \\ apparently & & x \end{array} \right) : (\tau \multimap \tau) \multimap \tau \multimap \tau$$

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

$$\gamma_{John} = \begin{array}{c} np \\ | \\ John \end{array} : \tau$$

$$\gamma_{apparently}\, I\, \gamma_{John} = \begin{array}{ccc} & vp & \\ \diagup & & \diagdown \\ apparently & & np \\ & & | \\ & & John \end{array}$$

TAG as ACG

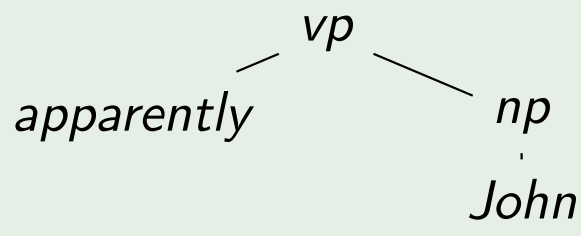
Category Induced Constraints

- 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

$$\begin{array}{lll} & \Sigma_{derivations} & \overset{\mathscr{L}_{\text{typed trees}}}{\longrightarrow} \\ c_{John} & : np & := \\ c_{apparently} & : (vp \multimap vp) \multimap vp \multimap vp & := \\ np, vp, s \ldots & : \text{types} & := \end{array} \quad \begin{array}{ll} \Sigma_{trees} & \\ \gamma_{John} & : \tau \\ \gamma_{apparently} & : (\tau \multimap \tau) \multimap \tau \\ \sigma & \end{array}$$

Example

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



Control on the Derived Trees

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

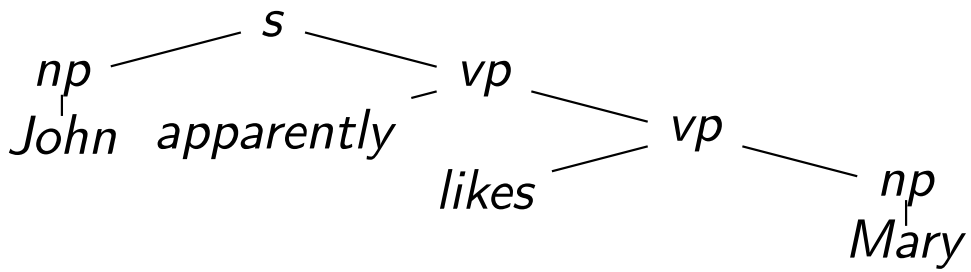
$np, vp, s \ldots := \sigma$

$c_{John} : np := \begin{array}{c} np \\ | \\ John \end{array}$

$c_{apparently} : (vp \multimap vp) \multimap vp \multimap vp := \lambda^{o} ax.a \left(\begin{array}{ccc} & & vp \\ & \diagup & \diagdown \\ apparently & & x \end{array} \right)$

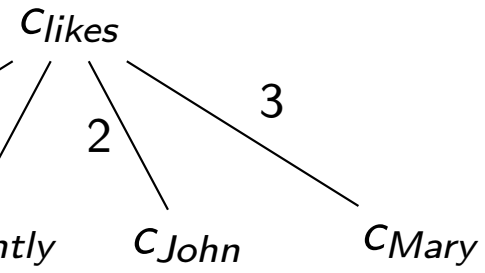
$c_{likes} : (s \multimap s) \multimap (vp \multimap vp) \multimap np \multimap np \multimap s := \lambda^{o} Saso.S \left(\begin{array}{ccccc} & & s & & \\ & \diagup & & \diagdown & \\ s & & & & a \left(\begin{array}{ccc} & & vp \\ & \diagup & \diagdown \\ likes & & o \end{array} \right) \end{array} \right)$

$c_{likes}(c_{apparently}\, I_{vp})\, c_{John}\, c_{Mary} : s =$



TAG Derivation Trees as Abstract Terms

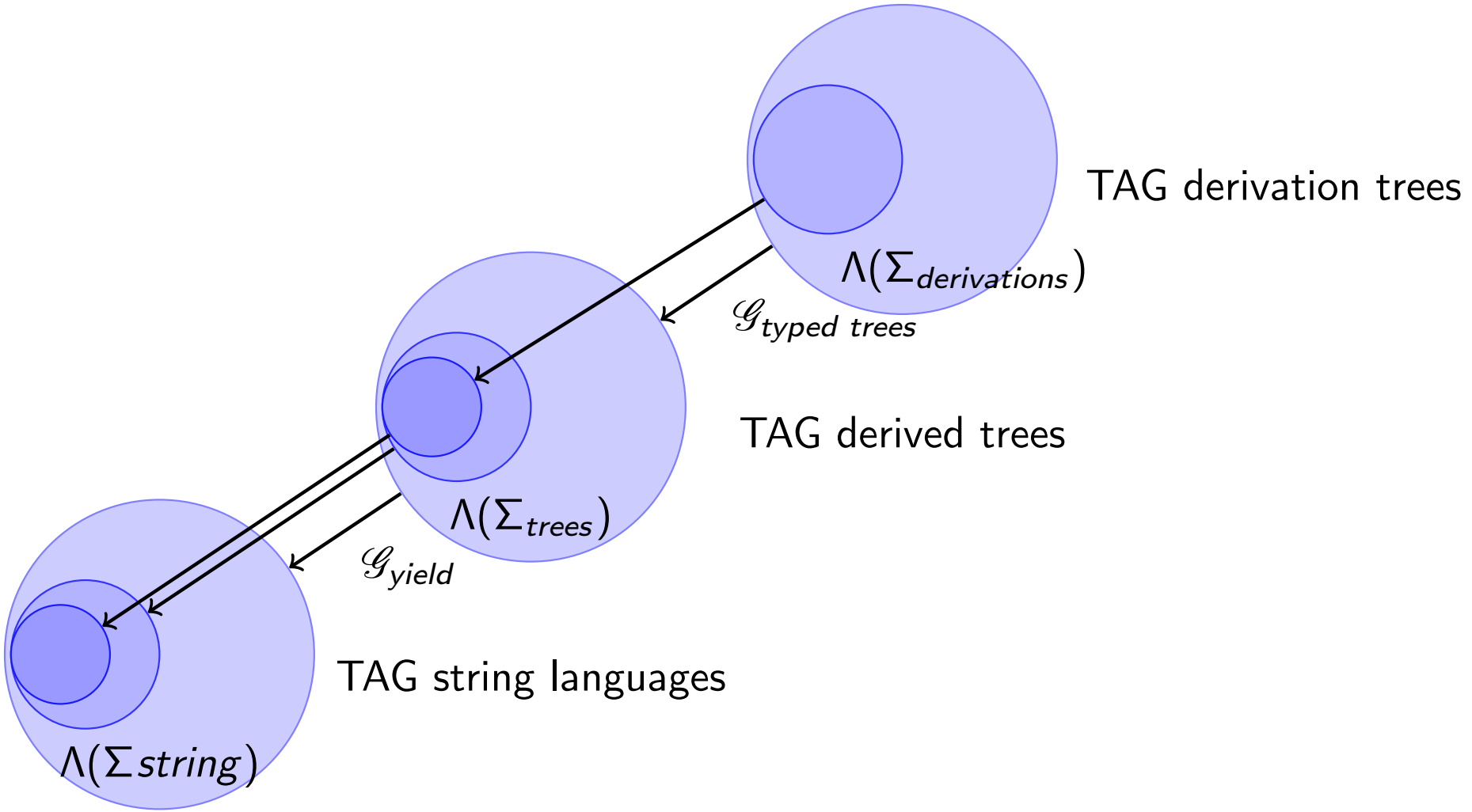
$c_{likes}\ l_s(c_{apparently}\ l_{vp})c_{John}c_{Mary} : s \quad = \quad l_s$



$c_{likes} \quad : \underbrace{(s \multimap s)}_{0} \multimap \underbrace{(vp \multimap vp)}_{1} \multimap \underbrace{np}_{2} \multimap \underbrace{np}_{3} \multimap s$

TAG as ACG

Intermediate Picture



See `example-05.acg`.

Let's Build Some Semantic Representation

Forgetting few seconds about TAG, we have:

- A higher-order signature $\Sigma_{derivations}$:

$$\begin{array}{ll} vp, np, s & : \text{types} \\ c_{john}, c_{mary} & : np \\ c_{apparently} & : vp \multimap vp \\ c_{likes} & (vp \multimap vp) \multimap np \multimap s \end{array}$$

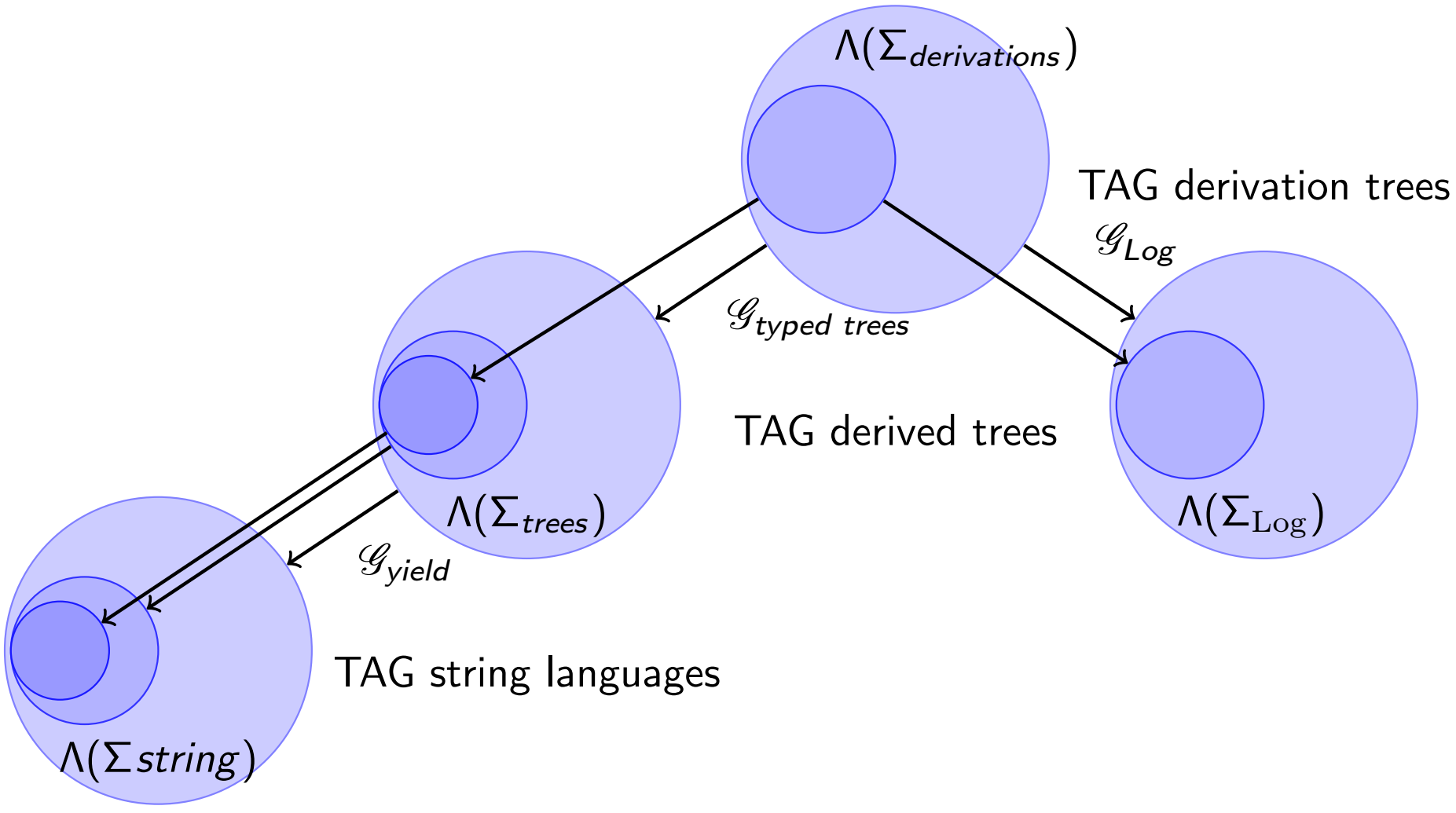
- Some knowledge about Montague-like semantics?

A standard interpretation

$$\begin{array}{llll} s & := t & np & := (e \multimap t) \multimap t \\ vp & := e \multimap t & & \\ c_{john} & := \lambda^o P.P\,\mathrm{j} & c_{apparently} & := \lambda^o aP.a(\lambda x.\mathrm{apparently}(P\,x)) \\ l_{vp} & := \lambda x.x & c_{likes} & := \lambda aos.s(a(\lambda x.o(\lambda y.\mathrm{like}\,x\,y))) \end{array}$$

See `example-06.acg` and `esslli09-tag.acg`.
How to get the object wide scope reading?

TAG with Semantics



Intermediate Conclusion

So far

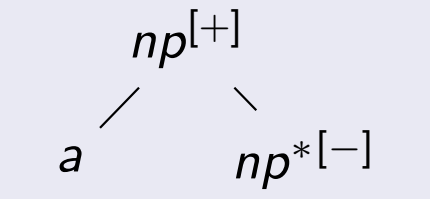
- Trees as λ-terms
- Yield as an ACG
- Typing control: ACG from derivation trees to derived trees
- Some semantics added. Is it a function from syntax?

Questions?

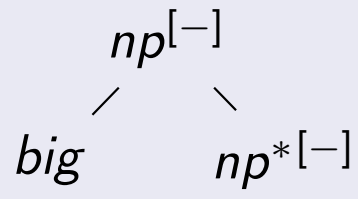
- Any TAG feature missing?
- Order of $\mathscr{G}_{\textit{typed trees}}$? $(c_{\textit{likes}} : (vp \multimap vp) \multimap np \multimap s)$

Features in TAG

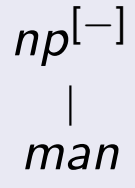
Nouns as NP



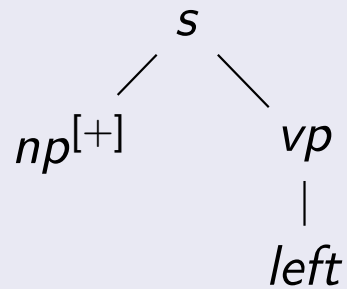
$c_a : np^{[-]} \multimap np^{[+]}$



$c_{big} : (np^{[-]} \multimap np^{[-]}) \multimap np^{[-]} \multimap np^{[-]}$



$c_{man} : (np^{[-]} \multimap np^{[+]}) \multimap (np^{[-]} \multimap np^{[-]}) \multimap np^{[+]}$



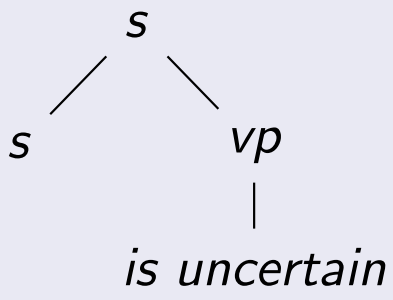
$c_{left} : \ldots \multimap np^{[+]} \multimap s$

- No $I_{np^{[-+]}} : np^{[-]} \multimap np^{[+]}$
- Different categories for the root node and the foot node [Vijay-Shanker(1992)]

TAG Derivation Trees

Enough typing control?

- $c_{likes} : (s \multimap s) \multimap (vp \multimap vp) \multimap np \multimap np$
- $c_{is\ uncertain} : s \multimap s :=$



- s node: substitution node (*Whether he will actually survice the experience is uncertain*)
- $c_{is\ uncertain} : s \multimap s$: typed as auxiliary trees (with root node of type s)

Adding more control: yet another ACG

- Not all the terms of type s are correct TAG derivations
- Distinguish between $s \multimap s$ (substition node of type s to tree of type s) and $s \multimap s$ (auxiliary tree whose root and foot node are of type s)

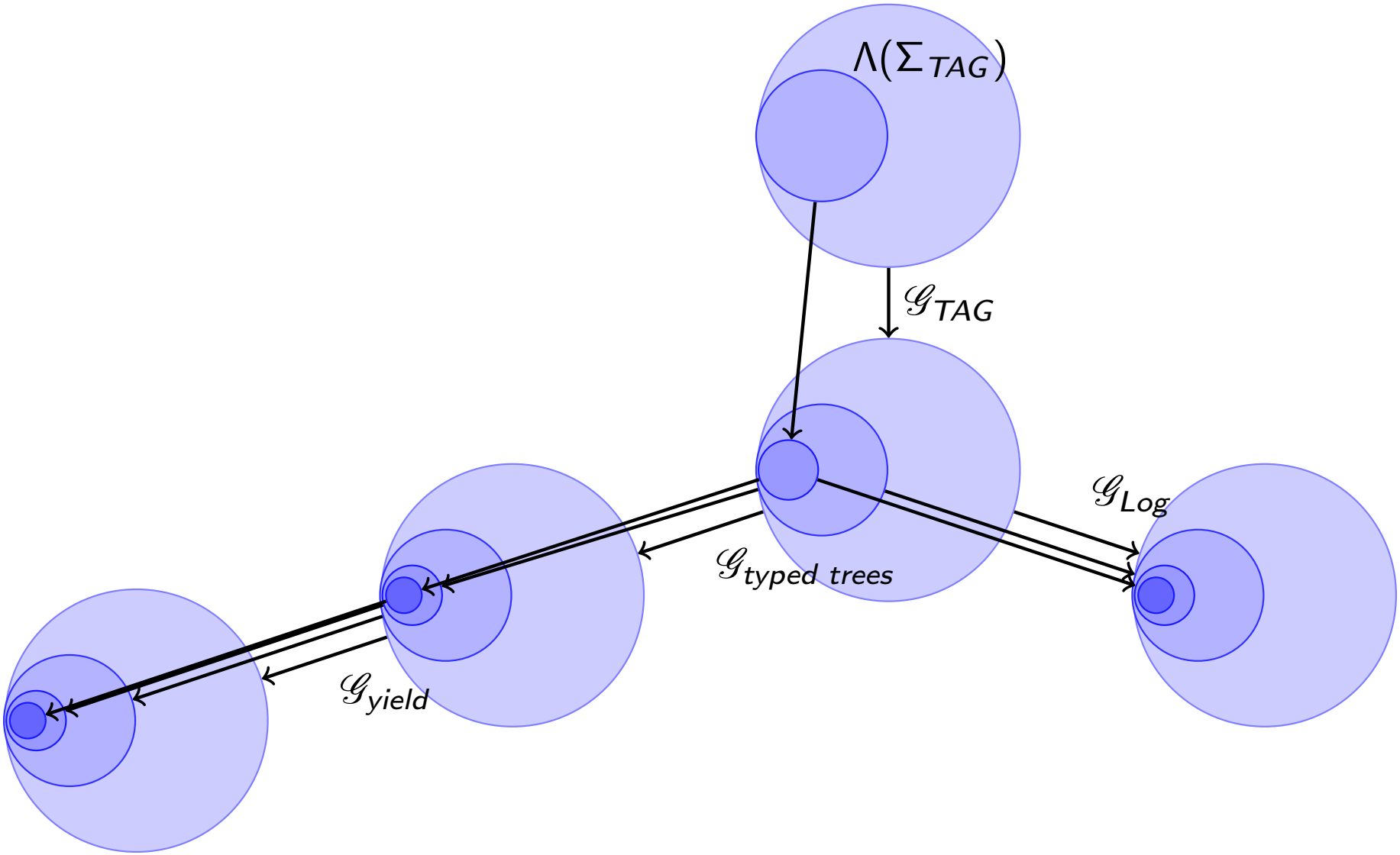
TAG Derivation Trees (cont'd)

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

$$
\begin{array}{lll}
C_{likes} & : s_A \multimap vp_A \multimap np \multimap np & := c_{likes} \\
C_{is\ uncertain} & : s \multimap s & := c_{is\ uncertain} \\
C_{apparently} & : vp_A & := c_{apparently} \\
I_{vp} & : vp_A & := I_{vp}
\end{array}
$$

- Semantics unchanged! ($\mathscr{G}_{Log} = \langle \Sigma_{derivations}, \Sigma_{Log}, \mathscr{L}_{\text{Log}}, s \rangle$)
- See `example-07.acg`.

The Actual Picture



Let's Practice

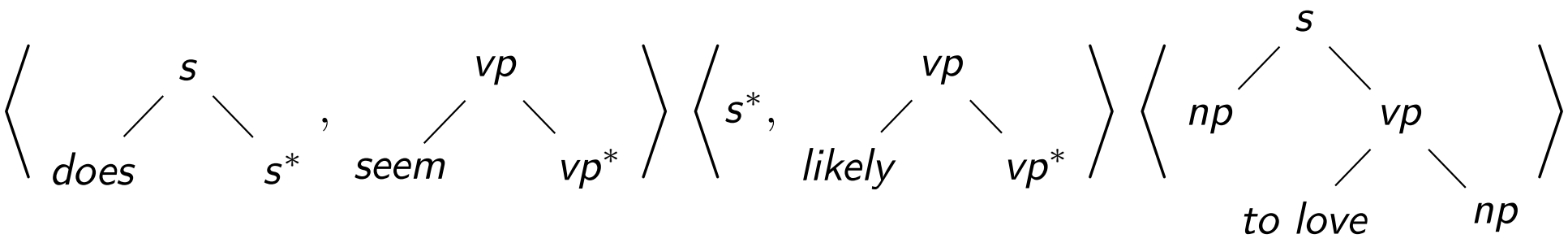
Let $\ell : \mathbb{N} \to \mathbb{N}$ be: $\ell(n) = \begin{cases} k & \text{if } n = 2^k \\ 1 & \text{otherwise} \end{cases}$.

Exercice

Can you find $\mathscr{G}_{TAG}$, $\mathscr{G}_{typed\ trees}$, $\mathscr{G}_{yield}$ and $\mathscr{G}_{Log}$ such that:

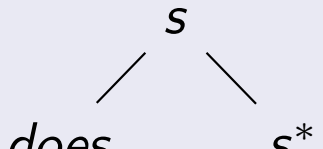
- $\mathcal{O}(\mathscr{G}_{yield} \circ \mathscr{G}_{typed\ trees} \circ \mathscr{G}_{TAG}) = \{a^n | n \in \mathbb{N}\}$;
- for all $n \in \mathbb{N}$ and for all $t \in \mathcal{A}(\mathscr{G}_{TAG})$ such that $\mathscr{L}_{\text{yield}} \circ \mathscr{L}_{\text{typed trees}} \circ \mathscr{L}_{\text{TAG}}(t) = a^n$, $\mathscr{L}_{\text{Log}} \circ \mathscr{L}_{\text{typed trees}} \circ \mathscr{L}_{\text{TAG}}(t) = \ell(n)$.

MCTAG



Extending the current signatures and lexicons

$$C_{does} : s_A \qquad C_s : s_A \multimap s_A$$
$$c_{does} : s \multimap s \qquad c_s : (s \multimap s) \multimap s \multimap s$$
$$\gamma_{does} : \tau \multimap \tau \qquad \gamma_s : (\tau \multimap \tau) \multimap \tau \multimap \tau$$
$$\gamma_{does} = \lambda x.s_2\ does\ x \qquad \gamma_s = \lambda ax.a\,x$$



s^*

$$m_{to\ love} : ((s_A \multimap vp_A \multimap s) \multimap s) \multimap np \multimap np \multimap s := \lambda^o Pso.P(\lambda^o xy.C_{to\ love}\ x\ y\ s\ o)$$
$$m_{does\ seem} : (s_A \multimap vp_A \multimap s) \multimap s := \lambda^o f.f\ C_{does}\ C_{seem}$$
$$m_{likely} : ((s_A \multimap vp_A \multimap s) \multimap s) \multimap (s_A \multimap vp_A \multimap s) \multimap s := \lambda^o Pf.P(\lambda^o xy.f(C_s\,x)\ (C_{likely}\ y))$$

See example-08.acg

Controlling the MCTAG Derivations

- Higher-order types: allows one to mix tuples. Non-local MCTAG

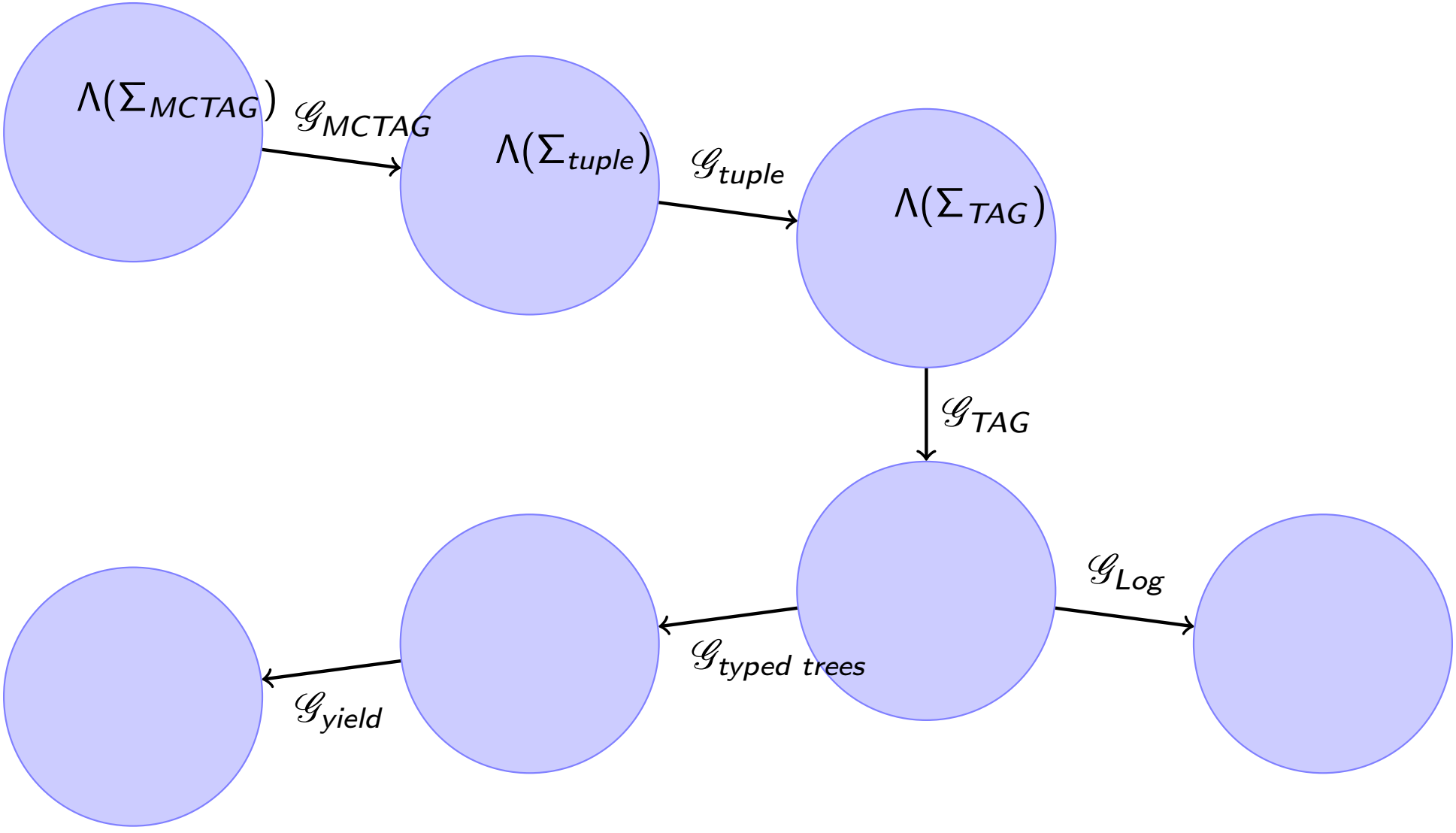
$$m_0 = \left\langle \begin{array}{ccc} & S & \\ A & & B \end{array} \right\rangle \quad m_1 = \left\langle \begin{array}{ccc} & A & \\ C & & A^* \end{array} \right\rangle \quad m_2 = \left\langle \begin{array}{ccc} & B & \\ D & & B^* \end{array} \right\rangle$$

$$m_3 = \langle C^*, D^* \rangle$$

(see `example-09.acg`)

- Add an ACG with atomic types $[s_A, vp_A] := (s_A \multimap vp_A \multimap s) \multimap s$: set-local MCTAG
- 2nd order ACG

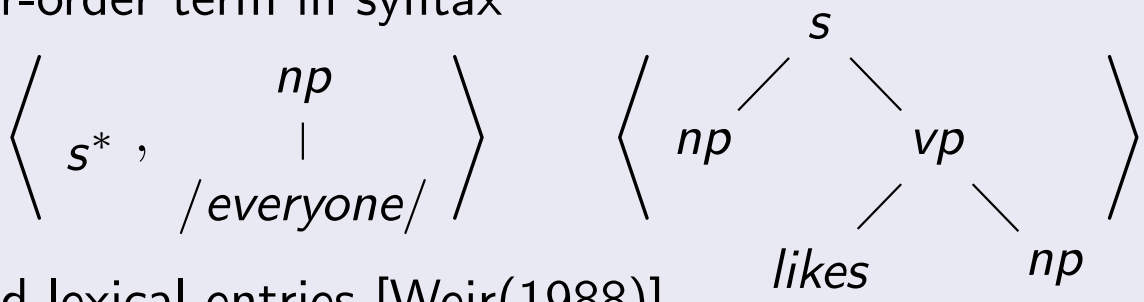
The Final Picture



About Scope Ambiguity in TAG

Main approaches

- In the lexical entries
- Underspecified representation language as the semantics target [Gardent and Kallmeyer(2003), Kallmeyer and Romero(2007)]
- MCTAG: higher-order term in syntax



  Amounts to add lexical entries [Weir(1988)]
- See the next sections

Conclusion

- Architecture: splitting the syntax-semantics interface from the control on admissible structures
- Status of the derivation tree
- s interpreted in various ways (t, $e \multimap t$, k, etc.)

Scope Ambiguities

$$\textit{every man loves some woman} \rightarrow \begin{cases} \forall x.\text{man}\, x \rightarrow (\exists y.\text{woman}\, y \wedge \text{love}\, x\, y) \\ \exists y.\text{woman}\, y \wedge (\forall x.\text{man}\, x \rightarrow \text{love}\, x\, y) \end{cases}$$

Underspecified framework

$$\textit{every man loves some woman} \rightarrow \triangle \rightarrow \boxed{?} \rightarrow \begin{cases} \forall x.\text{man}\, x \rightarrow (\exists y.\text{woman}\, y \wedge \text{love}\, x\, y) \\ \exists y.\text{woman}\, y \wedge (\forall x.\text{man}\, x \rightarrow \text{love}\, x\, y) \end{cases}$$

Type Logical framework

$$\textit{every man loves some woman} \quad \triangle\!\!\triangle \quad \begin{array}{l} \rightarrow \forall x.\text{man}\, x \rightarrow (\exists y.\text{woman}\, y \wedge \text{love}\, x\, y) \\ \rightarrow \exists y.\text{woman}\, y \wedge (\forall x.\text{man}\, x \rightarrow \text{love}\, x\, y) \end{array}$$

Strengths and Weaknesses

Underspecified framework

Pros:

- One syntactic analysis
- Expressivity

Cons:

- Description language
- Ambiguity in the semantic recipe, not in the interface

TL framework

Pros:

- No intermediate language
- Ambiguity handled by the process

Cons:

- Syntactic ambiguity

Question

Is there an ACG way provifing a proof-theoretic approach with only one syntactic structure and no intermediate language?

Scope Ambiguity in Categorial grammars

The standard way

$$\dfrac{/everyone/ \quad \dfrac{/loves/ \quad [np]}{np\backslash s/np \qquad np\backslash s} \qquad /someone/}{\dfrac{s}{s/np} \qquad (np/s)\backslash s}{s}$$

$$C_{someone}\ (\lambda^o y.C_{everyone}\ (\lambda^o x.C_{loves}\ y\ x))$$

$$\dfrac{/everyone/ \qquad \dfrac{/loves/ \quad [np]}{s/np} \qquad /someone/}{\dfrac{s}{np\backslash s} \qquad (np/s)\backslash s}{s}$$

$$C_{everyone}\ (\lambda^o x.C_{someone}\ (\lambda^o y.C_{loves}\ y\ x))$$

The ACG way

- Replace $\backslash$ and $/$ by $\multimap$
- $C_{everyone} : (np \multimap s) \multimap s$

Scope Ambiguity in ACG

$$\begin{array}{lll} & \Sigma_{CG} & \\ C_{loves} & : np \multimap np \multimap s & \longrightarrow \\ C_{everyone} & : (np \multimap s) \multimap s & \\ C_{someone} & : (np \multimap s) \multimap s & \end{array} \qquad \begin{array}{lll} & \Sigma_{string} & \\ /loves/ & & \sigma \\ /everyone/ & : & \sigma \\ /someone/ & : & \sigma \end{array}$$

$$\mathscr{L}_{\text{amb-string}}$$

$$\begin{array}{ll} C_{loves} & := \lambda^{o} os.s + /loves/ + o \\ C_{everyone} & := \lambda^{o} P.P\ /everyone/ \\ C_{someone} & ::= \lambda^{o} P.P\ /someone/ \end{array}$$

$$\begin{aligned} & C_{everyone}(\lambda^{o} x.C_{someone}(\lambda^{o} y.C_{loves}\ x\ y)) \\ & :=_{\text{amb-string}} (\lambda^{o} P.P\ /everyone/)((\lambda^{o} x.\lambda^{o} P.P\ /someone/)(\lambda^{o} y.(\lambda^{o} os.s + /loves/ + \\ & \to_{\beta} (\lambda^{o} P.P\ /everyone/)((\lambda^{o} x.\lambda^{o} P.P\ /someone/)(\lambda^{o} y.x + /loves/ + y)) \\ & \to_{\beta} (\lambda^{o} P.P\ /everyone/)(\lambda^{o} x.(\lambda^{o} y.x + /loves/ + y)\ /someone/) \\ & \to_{\beta} (\lambda^{o} P.P\ /everyone/)(\lambda^{o} x.x + /loves/ + /someone/) \\ & \to_{\beta} (\lambda^{o} x.x + /loves/ + /someone/)\ /everyone/ \\ & \to_{\beta} /everyone/ + /loves/ + /someone/ \end{aligned}$$

Scope Ambiguity in ACG (cont'd)

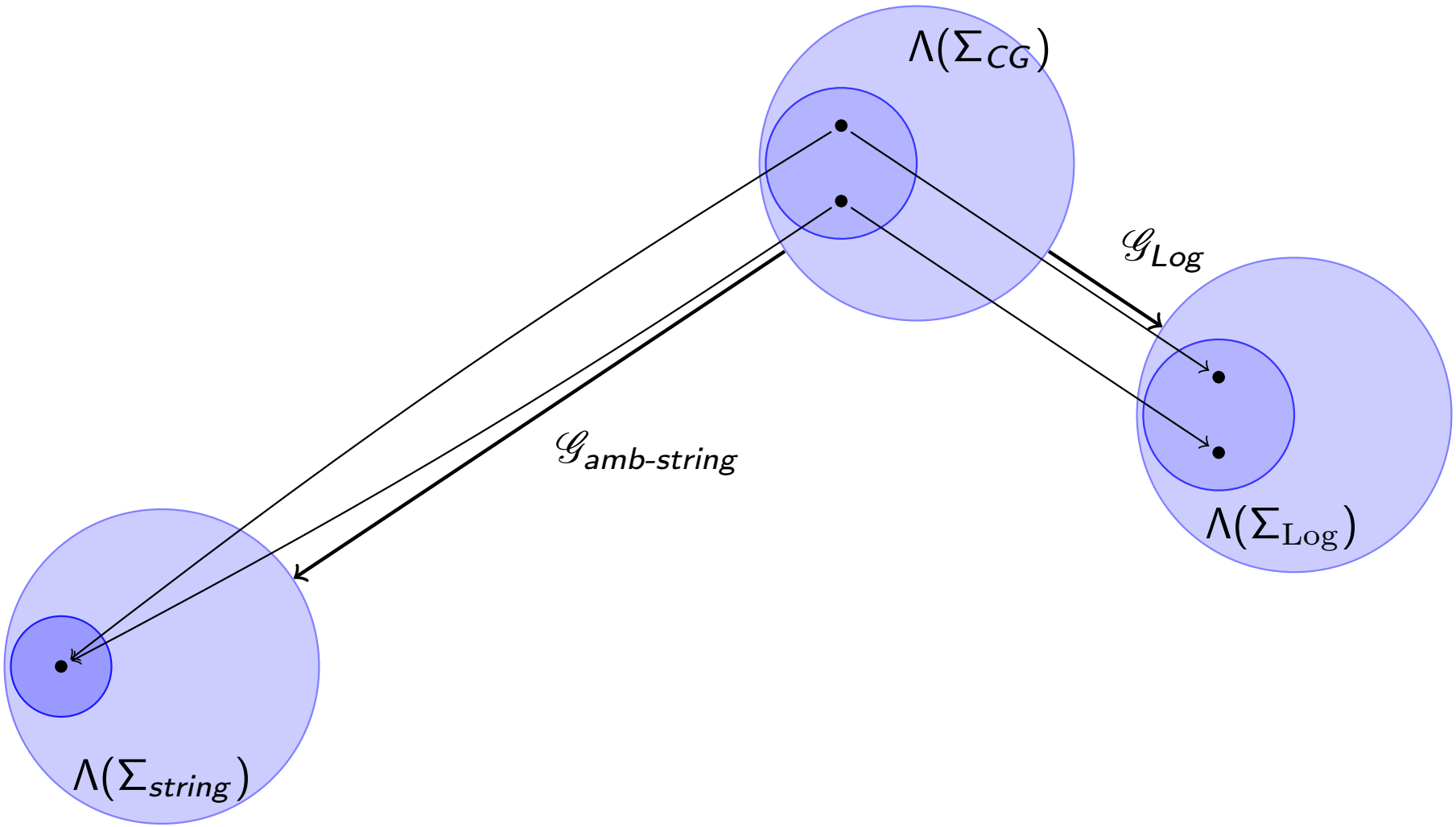
$$\begin{array}{lll} & \Sigma_{CG} & \\ C_{loves} & : np \multimap np \multimap s & \longrightarrow \\ C_{everyone} & : (np \multimap s) \multimap s & \\ C_{someone} & : (np \multimap s) \multimap s & \end{array} \qquad \begin{array}{ll} \Sigma_{string} & \\ /loves/ & \sigma \\ /everyone/ & : \sigma \\ /someone/ & : \sigma \end{array}$$

$$\mathscr{L}_{\text{amb-string}}$$

$$\begin{array}{ll} C_{loves} & := \lambda^o os.s + /loves/ + o \\ C_{everyone} & := \lambda^o P.P\ /everyone/ \\ C_{someone} & ::= \lambda^o P.P\ /someone/ \end{array}$$

$$\begin{aligned} & C_{someone}(\lambda^o y.C_{someone}(\lambda^o x.C_{loves}\ x\ y)) \\ & :=_{\text{amb-string}} (\lambda^o P.P\ /someone/)((\lambda^o y.\lambda^o P.P\ /everyone/)(\lambda^o x.(\lambda^o os.s + /loves/ + \\ & \to_\beta (\lambda^o P.P\ /someone/)((\lambda^o y.\lambda^o P.P\ /everyone/)(\lambda^o x.x + /loves/ + y)) \\ & \to_\beta (\lambda^o P.P\ /someone/)(\lambda^o y.(\lambda^o x.x + /loves/ + y)\ /everyone/) \\ & \to_\beta (\lambda^o P.P\ /someone/)(\lambda^o y./everyone/ + /loves/ + y) \\ & \to_\beta (\lambda^o y./everyone/ + /loves/ + x)\ /someone/ \\ & \to_\beta /everyone/ + /loves/ + /someone/ \end{aligned}$$

Scope Ambiguity
Non Injective Lexicon



Scope Ambiguity in ACG (cont'd)

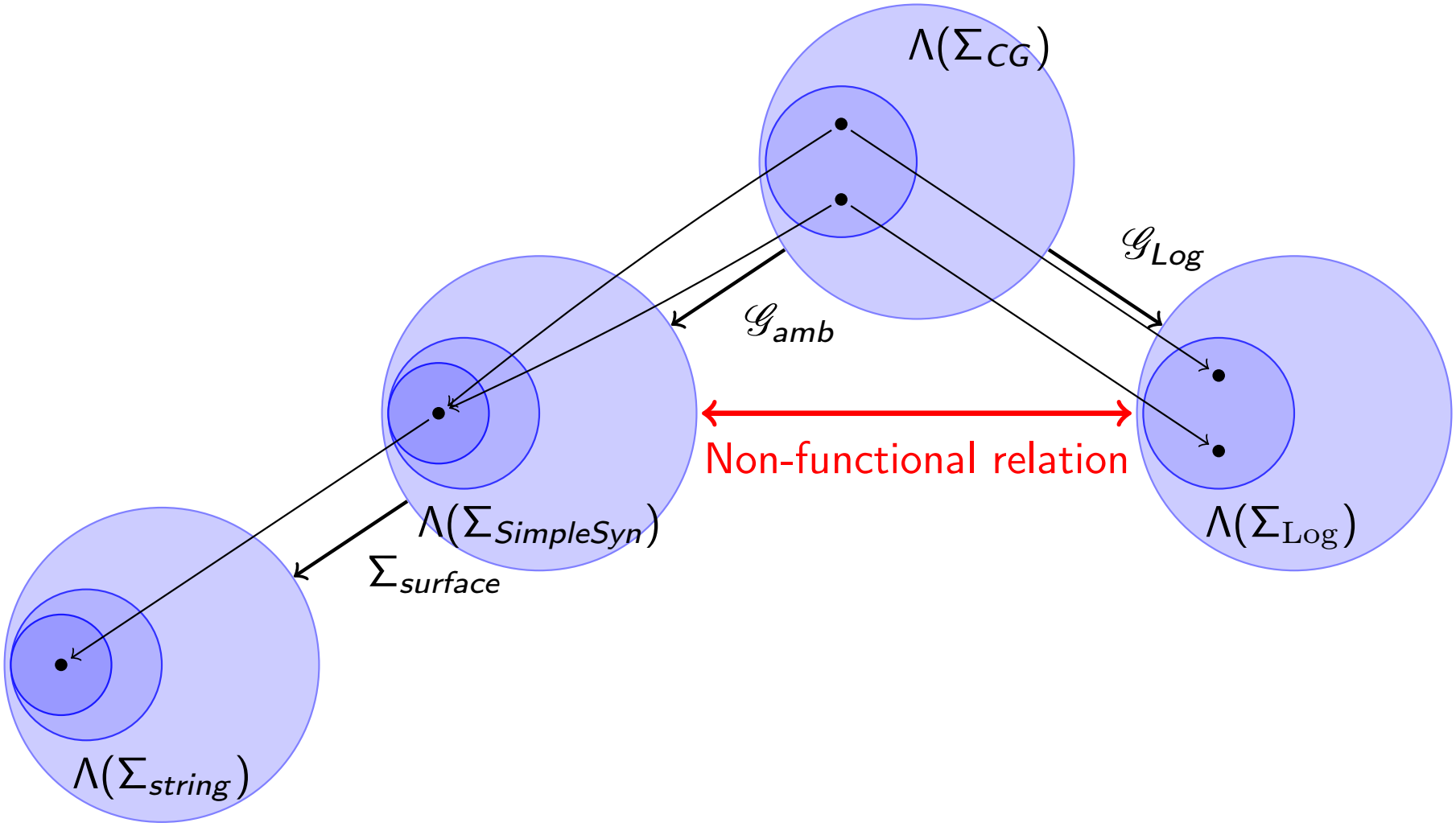
$$\Sigma_{CG} \qquad\qquad \Sigma_{SimpleSyn}$$
$$C_{loves} : np \multimap np \multimap s \qquad c_{loves} : np \multimap np \multimap s$$
$$C_{everyone} : (np \multimap s) \multimap s \longrightarrow c_{everyone} : np$$
$$C_{someone} : (np \multimap s) \multimap s \qquad c_{someone} : np$$

$$\mathscr{L}_{\text{amb}}$$
$$C_{loves} := \lambda^o os./loves/\ o\ s\lambda^o os.c_{loves}\ o\ s$$
$$C_{everyone} := \lambda^o P.P\ /everyone/c_{everyone}$$
$$C_{someone} ::= \lambda^o P.P\ /someone/c_{someone}$$

$$C_{everyone}(\lambda^o x.C_{someone}(\lambda^o y.C_{loves}\ x\ y)) :=_{\text{amb}} c_{loves}\ c_{someone}\ c_{everyone}$$
$$C_{someone}(\lambda^o y.C_{someone}(\lambda^o x.C_{loves}\ x\ y)) :=_{\text{amb}} c_{loves}\ c_{someone}\ c_{everyone}$$

Scope Ambiguity

Non Injective Lexicon



Conjunction

John and every kid ran

$$
\begin{aligned}
C_{and} &:= \lambda^{o} PQR.P(\lambda^{o} x.Q(\lambda^{o} y.R(c_{and}\, x\, y))) \\
C_{run} &:= c_{run} \\
C_{kid} &:= c_{kid} \\
C_{John} &:= c_{John}
\end{aligned}
$$

$$
\begin{aligned}
c_{run} &: np \multimap s \\
c_{kid} &: n \\
c_{John} &: np \\
c_{and} &: np \multimap np \multimap np
\end{aligned}
$$

$c_{run}(c_{and}\, c_{John}(c_{every}\, c_{kid}))$

$$
\begin{aligned}
C_{run} &: np \multimap s \\
C_{kid} &: n \\
C_{John} &: np \\
C_{and} &: ((np \multimap s) \multimap s) \\
&\quad \multimap ((np \multimap s) \multimap s) \\
&\quad \multimap (np \multimap s) \multimap s
\end{aligned}
$$

$C_{and}(\lambda^{o} P.PC_{John})(c_{every}\, c_{kid})C_{run}$

$$
\begin{aligned}
c_{run} &:= \lambda^{o} s.s + /ran/ \\
c_{kid} &:= /kid/ \\
c_{John} &:= /John/ \\
c_{and} &:= \lambda^{o} xy.x + /and/ + y
\end{aligned}
$$

$$
\begin{aligned}
C_{run} &:= \mathrm{run} \\
C_{kid} &:= \mathrm{kid} \\
C_{John} &:= \mathrm{j} \\
C_{and} &:= \lambda^{o} PQ. \\
&\quad \lambda R.(PR) \wedge (QR)
\end{aligned}
$$

Σ_{string}

$(/John/ + /and/ + (/every/ + /kid/)) + /ran/$

Σ_{Log}

$(\mathrm{run}\, \mathrm{j}) \wedge (\forall x.\mathrm{kid}\, x \Rightarrow \mathrm{run}\, x)$

De re and *De dicto* Readings

$$C_{seek} := \lambda^o xP.P(\lambda^o y.c_{seek}\, x\, y)$$
$$C_{book} := c_{book}$$

$c_{seek} : np \multimap np \multimap s$
$c_{book} : n$
$c_{seek}\, c_{John}(c_a\, c_{book})$

$C_{seek} : np \multimap ((np \multimap s) \multimap s) \multimap s$
$C_{book} : n$
$C_{seek}\, C_{John}(C_a\, C_{book})$
$(C_a\, C_{book})(\lambda^o y.C_{seek}\, C_{John}\, (\lambda^o Q.Q\, y))$

$c_{seek} := \lambda^o xy.x + /seeks/ + y$
$c_{book} := /book/$

$C_{seek} := \lambda^o xo.\ \mathrm{try}\, x\, (\lambda^o z.o\ (\lambda^o y.\mathrm{find}\, z\, y))$
$C_{book} := \mathrm{book}$

Σ_{string}
$/John/ + /seeks/ + (/a/ + /book/)$

Σ_{Log}
$\exists y.(\mathrm{book}\, y) \wedge (\mathrm{try}\, \mathrm{j}\, (\lambda^o x.\mathrm{find}\, x\, y))$
$\mathrm{try}\, \mathrm{j}\, (\lambda^o x.\exists y.(\mathrm{book}\, y) \wedge (\mathrm{find}\, x\, y))$

And More...

So far

- Coordination of quantified and non-quantified NPs
- VP ellipsis *John saw a kid and so did Bill*
- *de re* and *de dicto* readings
- Quantification and negation (*every kid did'nt run*)

Generalization: the Scoping Constructor

$$\frac{\Gamma \vdash_{\mathsf{TL}} t : \beta \Uparrow \alpha \quad \Delta, x : \beta \vdash_{\mathsf{TL}} u : \alpha}{\Gamma, \Delta \vdash_{\mathsf{TL}} t(\lambda x.u) : \alpha}\,(\mathsf{E}_{\Uparrow}) \qquad \frac{\Gamma \vdash_{\mathsf{TL}} t : \beta}{\Gamma \vdash_{\mathsf{TL}} \lambda x.(x\, t) : \beta \Uparrow \alpha}\,(\mathsf{I}_{\Uparrow})$$

Syntactically behaves as a β and semantically as a $(\beta \multimap \alpha) \multimap \alpha$

Given a lexical entre $w : a \in TL(A)$, we have $c_w : a^{\mathsf{syn}}$ and $C_w : a^{\mathsf{sem}}$ such that:

- if $a \in A$ then $a^{\mathsf{syn}} = a$ and $a^{\mathsf{sem}} = a$
- if $a = \alpha \multimap \beta$ then $a^{\mathsf{syn}} = \alpha^{\mathsf{syn}} \multimap \beta^{\mathsf{syn}}$ and $a^{\mathsf{sem}} = \alpha^{\mathsf{sem}} \multimap \beta^{\mathsf{sem}}$.
- if $a = \alpha \Uparrow \beta$ then $a^{\mathsf{syn}} = \alpha^{\mathsf{syn}}$ and $a^{\mathsf{sem}} = (\alpha^{\mathsf{sem}} \multimap \beta^{\mathsf{sem}}) \multimap \beta^{\mathsf{sem}}$

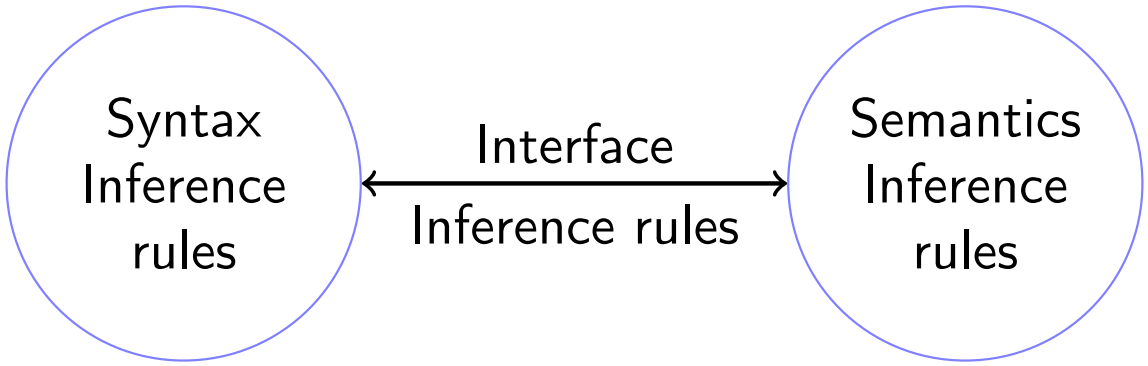
Result

In a commutative and associative TL system, given a suitable translation:

- Theorem: what we can do in TL we can do with ACGs
- Conjecture: what we can do with ACGs we can do with TL

- Also can apply to TAG
- What are the relations with other TL type constructor ($\mathrm{q}(x, y, z)$, $\uparrow$, $\downarrow$)

CVG: a (Weakly) Syntactocentric Formalism



Example (Chris liked Sandy)

$$\dfrac{\dfrac{\dfrac{}{\vdash \text{liked}, \text{like'} : np \multimap_{\text{c}} np \multimap_{\text{s}} s, \iota \multimap \iota \multimap \pi \dashv} \quad \dfrac{}{\vdash \text{Sandy}, \text{Sandy'} : np, \iota \dashv}}{\vdash \left[\text{liked Sandy}^{\,\text{c}}\right], \text{like' Sandy'} np \multimap_{\text{s}} s, \iota \multimap \pi \dashv \vdash \text{Chris}, \text{Chris'} : np, \iota \dashv} \quad \dfrac{}{}}{\vdash \left[^{\text{s}}\text{Chris} \left[\text{liked Sandy}^{\,\text{c}}\right]\right], \text{like' Sandy' Chris'} : s, \pi \dashv}$$

In Situ Operators
The Syntax-Semantics Mismatch

Example (Chris liked **Sandy**)

$$\pi = \dfrac{\dfrac{\vdash \text{top}_{\textbf{in-situ}}, \text{top'} : np \multimap_{\text{a}} np, \iota \multimap \iota^{\pi}_{\pi} \dashv \quad \vdash \text{Sandy}, \text{Sandy'} : np, \iota \dashv}{\vdash \left[\text{Sandy top}_{\textbf{in-situ}}\ ^{\text{a}}\right], \text{top' Sandy'} : np, \iota^{\pi}_{\pi} \dashv} \qquad \vdots}{\dfrac{\begin{array}{cc}\vdots & \\ \pi & \vdash \lambda^{o} t.\left[^{\text{s}}\text{Chris}\ \left[\text{liked}\ t\ ^{\text{c}}\right]\right], \lambda^{o} x.\text{like' } x\ \text{Chris'} : np \multimap s, \iota \multimap \pi \dashv\end{array}}{\vdash \left[^{\text{s}}\text{Chris}\ \left[\text{liked}\ \left[\text{Sandy top}_{\textbf{in-situ}}\ ^{\text{a}}\right]\ ^{\text{c}}\right]\right], (\text{top' Sandy'})\ (\lambda^{o} x.\text{like' } x\ \text{Chris'}) : s, \pi \dashv}}$$

The G Rule

$$\dfrac{\begin{array}{cc}\vdots & \vdots \\ \Gamma \vdash a, b : A, B^{D}_{C} \dashv \Delta & t, x : A, B, \Gamma' \vdash e, c : E, C \dashv \Delta'\end{array}}{\Gamma; \Gamma' \vdash e[t/a], b\,(\lambda^{o} x.c) : E, D \dashv \Delta, \Delta'}\ \text{G}$$

The G Rule Revisited

$$\frac{\vdots \qquad \vdots}{\dfrac{\Gamma \vdash a, b : A, B^D_C \dashv \Delta \quad t, x : A, B, \Gamma' \vdash e, c : E, C \dashv \Delta'}{\Gamma; \Gamma' \vdash e[t/a], b\,(\lambda^o x.c) : E, D \dashv \Delta, \Delta'}\,\mathsf{G}}$$

$$G_{\langle A, B^D_C \rangle} : \langle A, B^D_C \rangle \multimap (\langle A, B \rangle \multimap \langle E, C \rangle) \multimap \langle E, D \rangle$$

$\mathscr{L}_{\mathsf{Syn}}$ $\qquad$ $\mathscr{L}_{\mathsf{Sem}}$

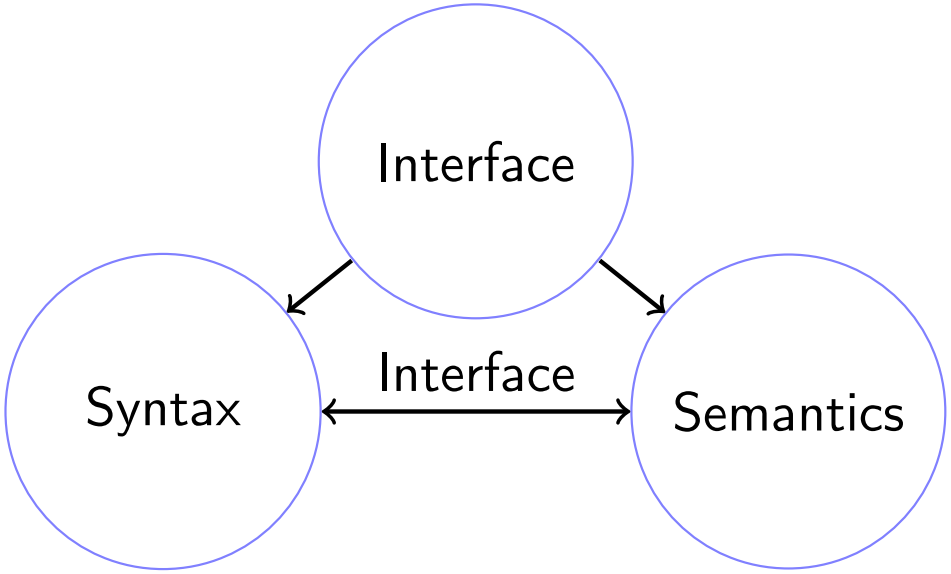
$\langle A, B^D_C \rangle$	$:=_{\mathsf{Syn}} A$	$\langle A, B^D_C \rangle$	$:=_{\mathsf{Sem}} (B \multimap C) \multimap D$
$\langle A, B \rangle \multimap \langle E, C \rangle$	$:=_{\mathsf{Syn}} A \multimap E$	$\langle A, B \rangle \multimap \langle E, C \rangle$	$:=_{\mathsf{Sem}} B \multimap C$
$\langle E, D \rangle$	$:=_{\mathsf{Syn}} E$	$\langle E, D \rangle$	$:=_{\mathsf{Sem}} D$
$G_{\langle A, B^D_C \rangle}$	$:=_{\mathsf{Syn}} \lambda^o tu.u\,t$	$G_{\langle A, B^D_C \rangle}$	$:=_{\mathsf{Sem}} \lambda^o tu.t\,u$

$$\begin{aligned}
\mathscr{L}_{\mathsf{Syn}}(u) &= a \\
\mathscr{L}_{\mathsf{Syn}}(v) &= e \\
\mathscr{L}_{\mathsf{Syn}}(G_{\langle A, B^D_C \rangle}\, u\, (\lambda^o t.v)) &= (\lambda^o tu.u\,t)\, a\, (\lambda^o t.e) \\
&\to_\beta e[t/a]
\end{aligned}$$

A New Perspective on the CVG Architecture

- Architectural motivation:

$$\text{CVG} = \text{ACG}$$



- Overt movement as higher order term in the syntax:

$$\frac{\Gamma \vdash a, d : A_B^C, D_E^F \dashv \Delta \quad t, x : A, D; \Gamma' \vdash b, e : B, E \dashv \Delta'}{\Gamma; \Gamma' \vdash a_t\, b, d_x\, e : C, F \dashv \Delta; \Delta'}\ \mathrm{G}$$

- 2nd order except when trigger by a A_B^C type

Conclusion

- Encoding in a same framework: sharing and comparing analysis
- ACG composition modes: flexible and open architectures
- “Syntax”, “function”, “relation”, “compositionality”, “rule-to-rule” intuitions may be realized by different mathematical notions

C. Barker.
Continuations and the nature of quantification.
Natural Language Semantics, 10:211–242, 2002.

P. W. Culicover and R. Jackendoff.
Simpler Syntax.
Oxford University Press, 2005.

H. B. Curry.
Some logical aspects of grammatical structure.
In R. Jakobson, editor, *Structure of Language and its Mathematical Aspects: Proceedings of the Twelfth Symposium in Applied Mathematics*, pages 56–68.
American Mathematical Society, 1961.

P. de Groote.
Towards a montagovian account of dynamics.
In *Proceedings of Semantics and Linguistic Theory XVI*, 2006.
http://research.nii.ac.jp/salt16/proceedings/degroote.new.pdf.


D. Dowty.
Grammatical relations and montague grammar.
In P. Jacobson and G. K. Pullum, editors, *The Nature of Syntactic Representation*. Reidel, Dordrecht, 1982.


C. Gardent and L. Kallmeyer.
Semantic construction in feature-based tag.
In *Proceedings of the 10th Meeting of the European Chapter of the Association for Computational Linguistics (EACL)*, 2003.


R. Jackendoff.
Foundations of Language: Brain, Meaning, Grammar, Evolution.
Oxford University Press, 2002.


L. Kallmeyer and M. Romero.
Scope and situation binding for ltag.
Research on Language and Computation, 2007.


M. Kracht.
The Mathematics of Language.
Number 63 in Studies in Generative Grammar. Mouton de Gruyter, Berlin, 2003.


R. Montague.
The proper treatment of quantification in ordinary english.
In *Formal Philosophy: Selected Papers of Richard Montague*. Yale University Press, 1974.
Re-edited in "Formal Semantics: The Essential Readings", Paul Portner and Barbara H. Partee, editors. Blackwell Publishers, 2002.


R. Muskens.
Lambda Grammars and the Syntax-Semantics Interface.
In R. van Rooy and M. Stokhof, editors, *Proceedings of the Thirteenth Amsterdam Colloquium*, pages 150–155, Amsterdam, 2001.


R. Muskens.
Lambdas, Language, and Logic.
In G.-J. Kruijff and R. Oehrle, editors, *Resource Sensitivity in Binding and Anaphora*, Studies in Linguistics and Philosophy, pages 23–54. Kluwer, 2003.


D. Oehrle.
Some 3-Dimensional Systems of Labelled Deduction.
Logic Jnl IGPL, 3(2-3):429–448, 1995.
doi: 10.1093/jigpal/3.2-3.429.
URL http://jigpal.oxfordjournals.org.


R. T. Oehrle.
Term-labeled categorial type systems.
Linguistic and Philosophy, 17(6):633–678, December 1994.
doi: 10.1007/BF00985321.


C. Pollard.
High-order categorical grammars.
In *Proceedings of Categorial Grammars 2004, Montpellier*, pages 340–361, June 2004.


C. Pollard.
An introduction to convergent grammar.
Presentation at Calligramme Seminar, 2008.
http://www.ling.ohio-state.edu/~scott/cvg/.


A. Ranta.
Type Theoretical Grammar.
Oxford University Press, 1994.


K. Vijay-Shanker.
Using descriptions of trees in a tree adjoining grammar.
Computational Linguistics, 18(4):481–518, December 1992.


D. J. Weir.
Characterizing Mildly Context-Sensitive Grammar Formalisms.
PhD thesis, University of Pennsylvania, 1988.