Context-freeness, automata and denotational semantics

Sylvain Salvati

INRIA, LaBRI, université de Bordeaux

FSMNLP 2015, Düsseldorf

Context-freeness and algebras

$S \to NP\ VP$
$VP \to V\ NP$
$NP \to Det\ N$
$N \to N\ CP$
$CP \to C\ VP$
$CP \to C\ S/NP$
$S/NP \to NP\ V$
$C \to$ that

$Det \to$ a
$N \to$ dog
$N \to$ cat
$N \to$ rat
$N \to$ cheese
$V \to$ saw
$V \to$ chased
$V \to$ ate

Context-freeness and algebras

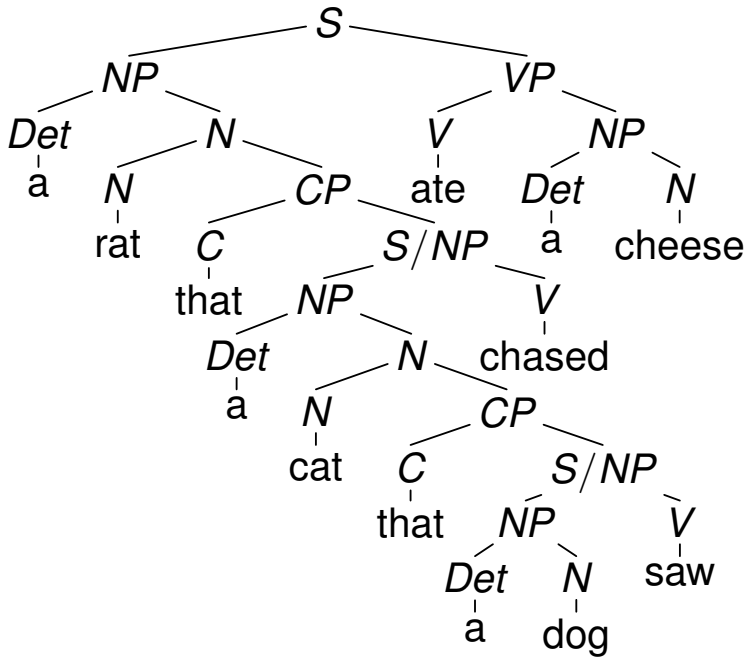
SENTENCE :	$S \to NP\ VP$	A :	$Det \to$ a
VERBP :	$VP \to V\ NP$	DOG :	$N \to$ dog
NOUNP :	$NP \to Det\ N$	CAT :	$N \to$ cat
NADJ :	$N \to N\ CP$	RAT :	$N \to$ rat
CPADJ :	$CP \to C\ VP$	CHEESE :	$N \to$ cheese
CPREL :	$CP \to C\ S/NP$	SAW :	$V \to$ saw
REL :	$S/NP \to NP\ V$	CHASED :	$V \to$ chased
THAT :	$C \to$ that	ATE :	$V \to$ ate

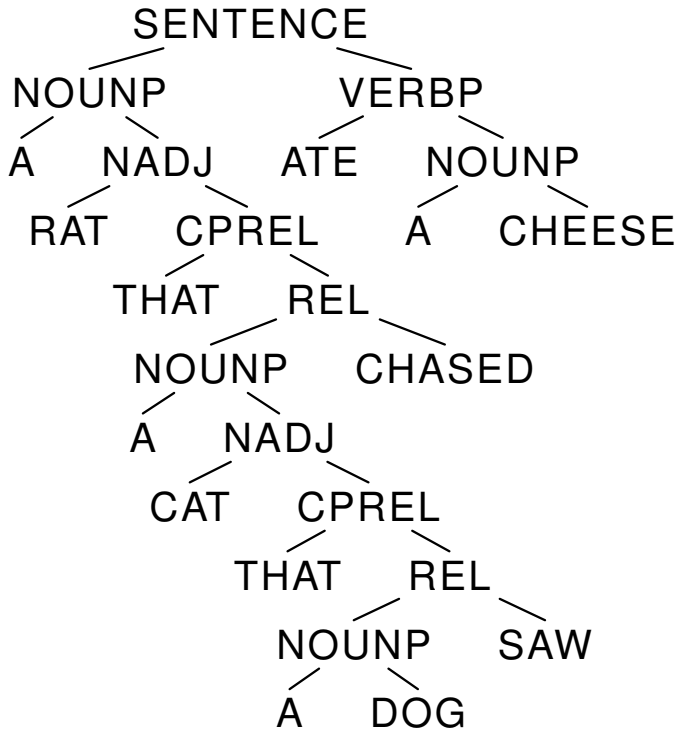
Context-freeness and algebras

SENTENCE :	$NP \to VP \to S$	A :	Det
VERBP :	$V \to NP \to VP$	DOG :	N
NOUNP :	$Det \to N \to NP$	CAT :	N
NADJ :	$CP \to N \to N$	RAT :	N
CPADJ :	$C \to VP \to CP$	CHEESE :	N
CPREL :	$C \to S/NP \to CP$	SAW :	V
REL :	$NP \to V \to S/NP$	CHASED :	V
THAT :	C	ATE :	V

$\text{SENTENCE}\, x\, y = \text{VERBP}\, x\, y = \text{NOUNP}\, x\, y = \text{NADJ}\, x\, y =$
$\text{CPADJ}\, x\, y = \text{CPREL}\, x\, y = \text{REL}\, x\, y = x + y$
$\text{A} = a$, $\text{DOG} = dog$, ...

The rule tree





Context-freeness and algebras

$$
\left|\begin{array}{lcl} S & \rightarrow & SS \\ S & \rightarrow & aSb \\ S & \rightarrow & \epsilon \end{array}\right.
$$

Context-freeness and algebras

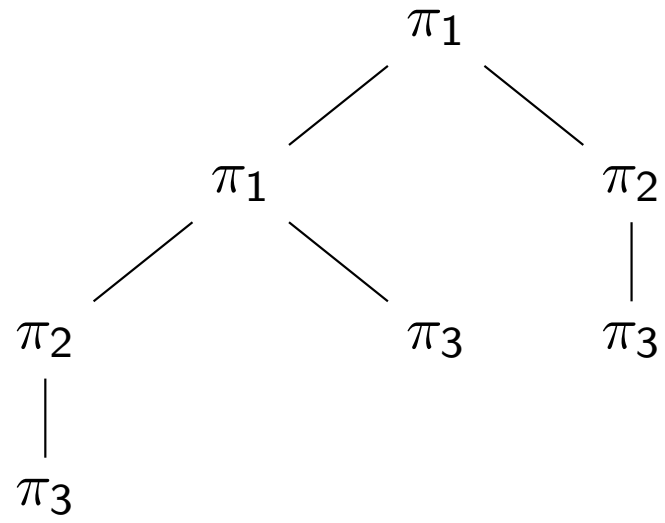
$$
\begin{array}{rl|lcl}
\pi_1: & & S & \to & SS \\
 & \pi_2: & S & \to & aSb \\
 & & S & \to & \epsilon
\end{array}
$$

Context-freeness and algebras

$$\begin{array}{r|lcl} \pi_1 : S \to S \to S & S & \to & SS \\ \pi_2 : S \to S & S & \to & aSb \\ \pi_3 : S & S & \to & \epsilon \end{array}$$

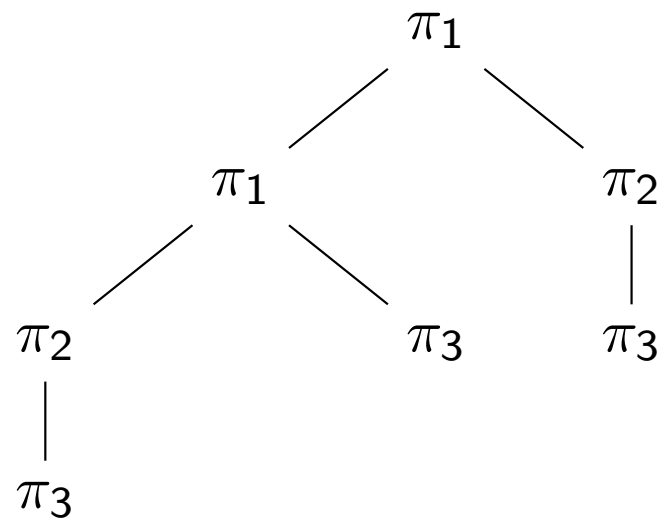
Context-freeness and algebras

$$\begin{array}{rl|lcl} \pi_1 : S \to S \to S & & S & \to & SS \\ \pi_2 : S \to S & & S & \to & aSb \\ \pi_3 : S & & S & \to & \epsilon \end{array}$$



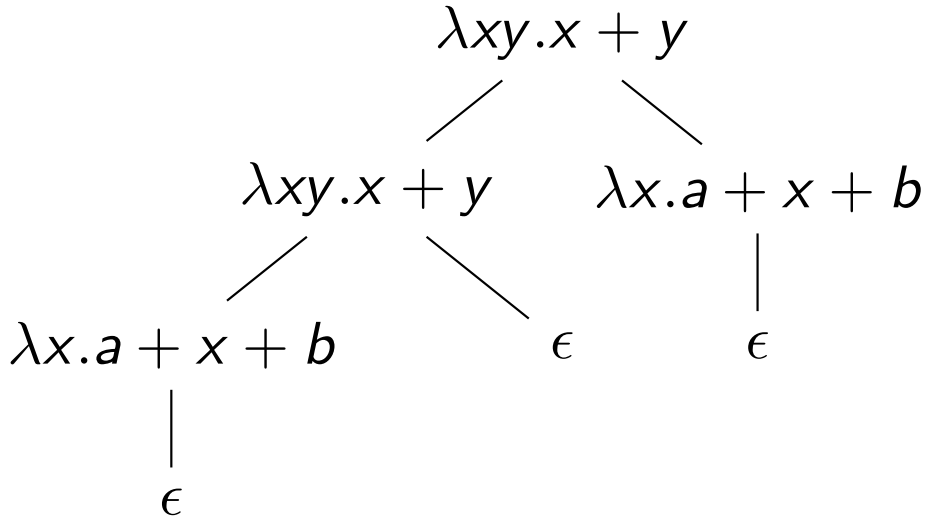
Context-freeness and algebras

$$
\begin{array}{r|lcll|rcl}
\pi_1 : S \to S \to S & S & \to & SS & \mathcal{Y}(\pi_1) & = & \lambda xy.x + y \\
\pi_2 : S \to S & S & \to & aSb & \mathcal{Y}(\pi_2) & = & \lambda x.a + x + b \\
\pi_3 : S & S & \to & \epsilon & \mathcal{Y}(\pi_3) & = & \epsilon
\end{array}
$$



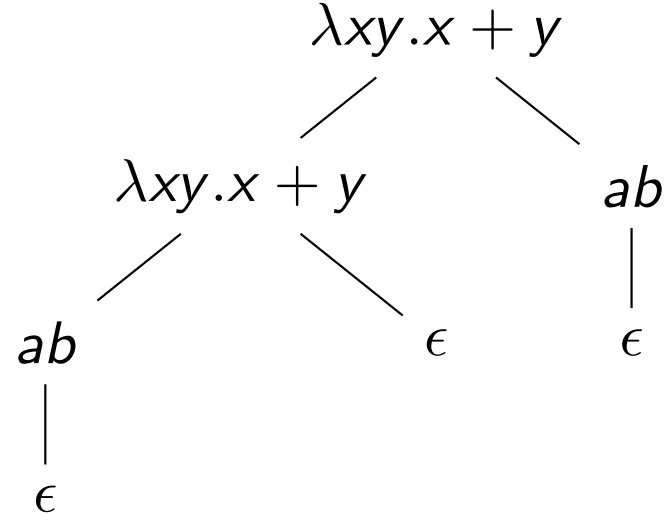
Context-freeness and algebras

$$
\begin{array}{rl|lcll}
\pi_1 : S \to S \to S & & S & \to & SS & \mathcal{Y}(\pi_1) = \lambda xy.x + y \\
\pi_2 : S \to S & & S & \to & aSb & \mathcal{Y}(\pi_2) = \lambda x.a + x + b \\
\pi_3 : S & & S & \to & \epsilon & \mathcal{Y}(\pi_3) = \epsilon
\end{array}
$$



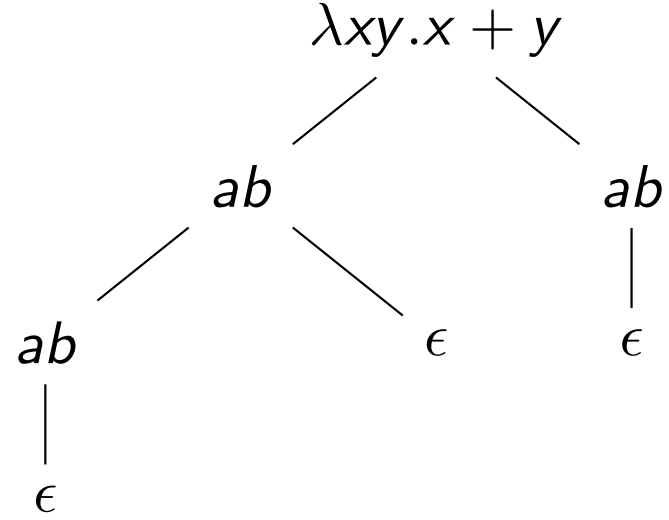
Context-freeness and algebras

$$
\begin{array}{rl|lcll}
\pi_1 : S \to S \to S & & S & \to & SS & \mathcal{Y}(\pi_1) = \lambda xy.x + y \\
\pi_2 : S \to S & & S & \to & aSb & \mathcal{Y}(\pi_2) = \lambda x.a + x + b \\
\pi_3 : S & & S & \to & \epsilon & \mathcal{Y}(\pi_3) = \epsilon
\end{array}
$$



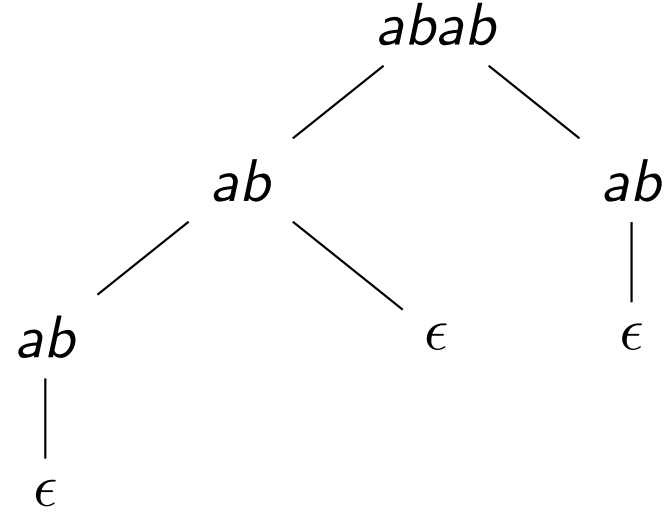
Context-freeness and algebras

$$
\begin{array}{r|lcll|lcl}
\pi_1 : S \to S \to S & S & \to & SS & \mathcal{Y}(\pi_1) & = & \lambda xy.x + y \\
\pi_2 : S \to S & S & \to & aSb & \mathcal{Y}(\pi_2) & = & \lambda x.a + x + b \\
\pi_3 : S & S & \to & \epsilon & \mathcal{Y}(\pi_3) & = & \epsilon
\end{array}
$$



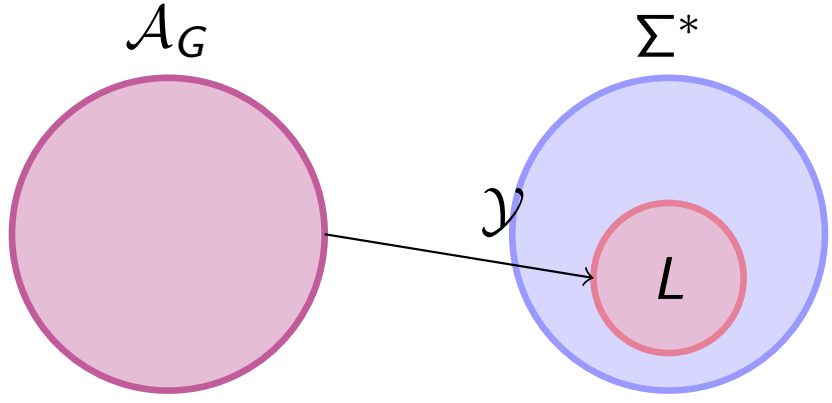
Context-freeness and algebras

$$
\begin{array}{rl|lcll}
\pi_1 : S \to S \to S & S & \to & SS & \mathcal{Y}(\pi_1) = & \lambda xy.x + y \\
\pi_2 : S \to S & S & \to & aSb & \mathcal{Y}(\pi_2) = & \lambda x.a + x + b \\
\pi_3 : S & S & \to & \epsilon & \mathcal{Y}(\pi_3) = & \epsilon
\end{array}
$$



Context-freeness and algebras

$$
\begin{array}{rl|lcl|lcl}
\pi_1 : S \to S \to S & & S & \to & SS & \mathcal{Y}(\pi_1) & = & \lambda xy.x + y \\
\pi_2 : S \to S & & S & \to & aSb & \mathcal{Y}(\pi_2) & = & \lambda x.a + x + b \\
\pi_3 : S & & S & \to & \epsilon & \mathcal{Y}(\pi_3) & = & \epsilon
\end{array}
$$



Why going beyond context free languages? (1)

Gazdar, Pullum 1982:

Whether non-context-free characteristics can be found in the stringset of some natural language remains an open question, just as it was a quarter century ago.

Why going beyond context free languages? (2)

Verb-object cross-serial dependencies in subordinate clauses

English:
that Charles lets Mary help Peter teach John to swim

German:
daß der Karl die Maria dem Peter den Hans schwimmen lehren helfen läßt

Dutch:
dat Karel Marie Piet Jan laat helpen leren zwemmen

Swiss German:
dass de Karl d'Maria em Peter de Hans laat hälfe lärne schwüme

Why going beyond context free languages? (2)

Verb-object cross-serial dependencies in subordinate clauses

English:

that Charles lets Mary help Peter teach John to swim

German:

daß der Karl die Maria dem Peter den Hans schwimmen lehren helfen läßt

Dutch:

dat Karel Marie Piet Jan laat helpen leren zwemmen

Swiss German:

dass de Karl d'Maria em Peter de Hans laat hälfe lärne schwüme

Why going beyond context free languages? (2)

Verb-object cross-serial dependencies in subordinate clauses

English:

that Charles lets Mary help Peter teach John to swim

German:

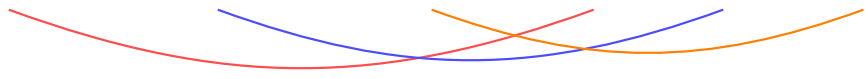
daß der Karl die Maria dem Peter den Hans schwimmen lehren helfen läßt

Dutch:

dat Karel Marie Piet Jan laat helpen leren zwemmen

Swiss German:

dass de Karl d'Maria em Peter de Hans laat hälfe lärne schwüme



Non-context-freeness of natural languages

Shieber 1985:
This Swiss German sentence is grammatical if and only if $k = m$ and $l = n$:

De Jan säit, dass mer (d'chind)^k (em Hans)^l es huus haend wele laa^m hälfe^n aastriiche.
Jan said that we have wanted (to let the children)^m (help Hans)^n paint the house.

$$\{a^m b^n c^m d^n | m, n \geq 1\} \notin CFL$$

How to go beyond context-free grammars?

Context-free grammars have very nice properties:

- simple and efficient parsing algorithm,
- good formal language properties (closure under intersection with regular sets, good probabilistic extensions, *etc. . .*)

Claim:

> *Most of those properties are a consequence of the algebraic background of CFLs.*

Extending CFLs mainly consists in allowing richer interpretations of algebras.

λ-definable interpretations

The richest well-behaved interpretations are the one definable with simply typed λ-calculus.

Simple Types:

$$\mathbb{T} : \quad \alpha, \beta, \gamma ::= o \mid \alpha \to \beta$$

λ-definable interpretations

The richest well-behaved interpretations are the one definable with simply typed λ-calculus.

Simple Types:

$$\mathbb{T} : \quad \alpha, \beta, \gamma ::= o \mid \alpha \to \beta$$

$$\mathrm{order}(o) = 1 \quad \mathrm{order}(\alpha \to \beta) = \mathsf{max}\{\mathrm{order}(\alpha + 1), \mathrm{order}(\beta)\}$$

λ-definable interpretations

The richest well-behaved interpretations are the one definable with simply typed λ-calculus.

Simple Types:

$$\mathbb{T}: \quad \alpha, \beta, \gamma ::= o \mid \alpha \to \beta$$
$$\mathrm{order}(o) = 1 \quad \mathrm{order}(\alpha \to \beta) = \mathsf{max}\{\mathrm{order}(\alpha + 1), \mathrm{order}(\beta)\}$$

Simply typed λ-calculus:

$$\Lambda^{\alpha}: \quad M^{\alpha}, N^{\alpha} ::= x^{\alpha} \mid c^{\alpha} \mid \lambda x^{\alpha}.M^{\beta} \mid M^{\beta \to \alpha} N^{\beta}$$

$$(\beta) \quad (\lambda x.M)N = M[N/x]$$

$$(\eta) \quad \lambda x.Mx = M \text{ where } x \notin fv(M)$$

Simply typed λ-calculus generalizes trees

The ranked alphabet $\{e; g; f\}$ where $\mathrm{rank}(e) = 0$, $\mathrm{rank}(g) = 1$, $\mathrm{rank}(f) = 2$ can be represented by the following second order constants:

$$e : o,\ g : o \to o,\ f : o \to o \to o$$

Simply typed λ-calculus generalizes trees

The ranked alphabet $\{e; g; f\}$ where $\mathrm{rank}(e) = 0$, $\mathrm{rank}(g) = 1$, $\mathrm{rank}(f) = 2$ can be represented by the following second order constants:

$$e : o,\ g : o \to o,\ f : o \to o \to o$$

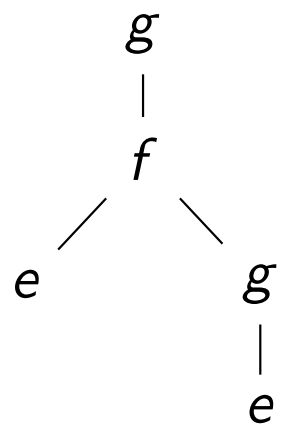
the term $g(f(e, g(e)))$ is represented by the λ-term $g(f\ e\ (g\ e))$

Simply typed λ-calculus generalizes trees

The ranked alphabet $\{e; g; f\}$ where $\mathrm{rank}(e) = 0$, $\mathrm{rank}(g) = 1$, $\mathrm{rank}(f) = 2$ can be represented by the following second order constants:

$$e : o,\ g : o \to o,\ f : o \to o \to o$$

the term $g(f(e, g(e)))$ is represented by the λ-term $g(f\ e\ (g\ e))$
The Böhm tree of the λ-term is the same as the graphic representation of the term:



Simply typed λ-calculus generalizes strings

The elements of $\{a; b\}^*$ can be represented with the constants:

$$a : o \to o,\ b : o \to o$$

Strings are represented by terms of type $o \to o$:

the string aba is represented by $/aba/ = \lambda x^o.a(b(a\,x^o))$

Simply typed λ-calculus generalizes strings

The elements of $\{a; b\}^*$ can be represented with the constants:

$$a : o \to o,\ b : o \to o$$

Strings are represented by terms of type $o \to o$:

the string aba is represented by $/aba/ = \lambda x^o.a(b(a\,x^o))$

Concatenation is then $s_1 + s_2 = \lambda x^o.s_1(s_2(x^o))$:

$$\begin{array}{rcl} /ab/ + /bb/ & = & \lambda x^o.a(b(x^o)) + \lambda x^o.b(b(x^o)) \\ & = & \lambda x^o.(\lambda y^o.a(b\,y^o))((\lambda z^o.b(b\,z^o))x^o) \\ & =_{\beta\eta} & \lambda x^o.a(b(b(b\,z^o))) \end{array}$$

and the empty string is $\lambda x^o.x^o$

Cross-serial dependencies again

Idealized versions of cross-serial dependencies:

- English: that we (let Mary)$^{n+1}$ (help John)$^{p+1}$ to swim

Cross-serial dependencies again

Idealized versions of cross-serial dependencies:

- English: that we (let Mary)$^{n+1}$ (help John)$^{p+1}$ to swim
- German: daß wir (der Maria)$^{n+1}$ (dem Peter)$^{p+1}$ schwimmen (helfen)$^{p+1}$ (lassen)$^{n+1}$

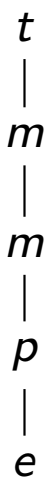
Cross-serial dependencies again

Idealized versions of cross-serial dependencies:

- English: that we (let Mary)$^{n+1}$ (help John)$^{p+1}$ to swim
- German: daß wir (der Maria)$^{n+1}$ (dem Peter)$^{p+1}$ schwimmen (helfen)$^{p+1}$ (lassen)$^{n+1}$
- Dutch: dat we (Marie)$^{n+1}$ (Piet)$^{p+1}$ (laten)$^{n+1}$ (helpen)$^{p+1}$ zwemmen

A (too) simple grammar: the case of English

$t : l \to s$, $m : l \to l$, $p : l \to l$, $e : l$.
English: that we (let Mary)$^{n+1}$ (help John)$^{p+1}$ to swim
$\mathcal{E}(t) = \lambda x$.that we let Mary x
$\mathcal{E}(m) = \lambda x$.let Mary x
$\mathcal{E}(p) = \lambda x$.help John x
$\mathcal{E}(e) =$ help John to swim



A (too) simple grammar: the case of English

$t : l \to s$, $m : l \to l$, $p : l \to l$, $e : l$.
English: that we (let Mary)$^{n+1}$ (help John)$^{p+1}$ to swim
$\mathcal{E}(t) = \lambda x$.that we let Mary x
$\mathcal{E}(m) = \lambda x$.let Mary x
$\mathcal{E}(p) = \lambda x$.help John x
$\mathcal{E}(e) =$ help John to swim

λx.that we let Mary x
|
λx.let Mary x
|
λx.let Mary x
|
λx.help John x
|
help John to swim

A (too) simple grammar: the case of English

$t : l \to s$, $m : l \to l$, $p : l \to l$, $e : l$.
English: that we (let Mary)$^{n+1}$ (help John)$^{p+1}$ to swim
$\mathcal{E}(t) = \lambda x$.that we let Mary x
$\mathcal{E}(m) = \lambda x$.let Mary x
$\mathcal{E}(p) = \lambda x$.help John x
$\mathcal{E}(e) =$ help John to swim

λx.that we let Mary x
|
λx.let Mary x
|
λx.let Mary x
|
help John help John to swim
|
help John to swim

A (too) simple grammar: the case of English

$t : l \to s$, $m : l \to l$, $p : l \to l$, $e : l$.
English: that we (let Mary)$^{n+1}$ (help John)$^{p+1}$ to swim
$\mathcal{E}(t) = \lambda x$.that we let Mary x
$\mathcal{E}(m) = \lambda x$.let Mary x
$\mathcal{E}(p) = \lambda x$.help John x
$\mathcal{E}(e) =$ help John to swim

λx.that we let Mary x
|
λx.let Mary x
|
let Mary help John help John to swim
|
help John help John to swim
|
help John to swim

A (too) simple grammar: the case of English

$t : l \to s$, $m : l \to l$, $p : l \to l$, $e : l$.
English: that we (let Mary)$^{n+1}$ (help John)$^{p+1}$ to swim
$\mathcal{E}(t) = \lambda x$.that we let Mary x
$\mathcal{E}(m) = \lambda x$.let Mary x
$\mathcal{E}(p) = \lambda x$.help John x
$\mathcal{E}(e) =$ help John to swim

λx.that we let Mary x

|

let Mary let Mary help John help John to swim

|

let Mary help John help John to swim

|

help John help John to swim

|

help John to swim

A (too) simple grammar: the case of English

$t : l \to s$, $m : l \to l$, $p : l \to l$, $e : l$.
English: that we (let Mary)$^{n+1}$ (help John)$^{p+1}$ to swim
$\mathcal{E}(t) = \lambda x$.that we let Mary x
$\mathcal{E}(m) = \lambda x$.let Mary x
$\mathcal{E}(p) = \lambda x$.help John x
$\mathcal{E}(e) =$ help John to swim

that we let Mary let Mary let Mary help John help John to swim

|

let Mary let Mary help John help John to swim

|

let Mary help John help John to swim

|

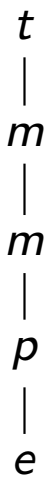
help John help John to swim

|

help John to swim

A (too) simple grammar: the case of German

$t : l \to s$, $m : l \to l$, $p : l \to l$, $e : l$.
German: daß wir (der Maria)$^{n+1}$ (dem Peter)$^{p+1}$ schwimmen (helfen)$^{p+1}$ (lassen)$^{n+1}$
$\mathcal{G}(t) = \lambda x$.daß wir der Maria x dem Peter helfen lassen
$\mathcal{G}(m) = \lambda x$.der Maria x lassen
$\mathcal{G}(p) = \lambda x$.dem Peter x helfen
$\mathcal{G}(e) =$ dem Peter schwimmen helfen



A (too) simple grammar: the case of German

$t : l \to s$, $m : l \to l$, $p : l \to l$, $e : l$.
German: daß wir (der Maria)$^{n+1}$ (dem Peter)$^{p+1}$ schwimmen (helfen)$^{p+1}$ (lassen)$^{n+1}$
$\mathcal{G}(t) = \lambda x$.daß wir der Maria x dem Peter helfen lassen
$\mathcal{G}(m) = \lambda x$.der Maria x lassen
$\mathcal{G}(p) = \lambda x$.dem Peter x helfen
$\mathcal{G}(e) =$ dem Peter schwimmen helfen

λx.daß wir der Maria x lassen
|
λx.der Maria x lassen
|
λx.der Maria x lassen
|
λx.dem Peter x helfen
|
dem Peter schwimmen helfen

A (too) simple grammar: the case of German

$t : l \to s$, $m : l \to l$, $p : l \to l$, $e : l$.
German: daß wir (der Maria)$^{n+1}$ (dem Peter)$^{p+1}$ schwimmen (helfen)$^{p+1}$ (lassen)$^{n+1}$
$\mathcal{G}(t) = \lambda x$.daß wir der Maria x dem Peter helfen lassen
$\mathcal{G}(m) = \lambda x$.der Maria x lassen
$\mathcal{G}(p) = \lambda x$.dem Peter x helfen
$\mathcal{G}(e) =$ dem Peter schwimmen helfen

λx.daß wir der Maria x lassen
|
λx.der Maria x lassen
|
λx.der Maria x lassen
|
dem Peter dem Peter schwimmen helfen helfen
|
dem Peter schwimmen helfen

A (too) simple grammar: the case of German

$t : l \to s$, $m : l \to l$, $p : l \to l$, $e : l$.
German: daß wir (der Maria)$^{n+1}$ (dem Peter)$^{p+1}$ schwimmen (helfen)$^{p+1}$ (lassen)$^{n+1}$
$\mathcal{G}(t) = \lambda x$.daß wir der Maria x dem Peter helfen lassen
$\mathcal{G}(m) = \lambda x$.der Maria x lassen
$\mathcal{G}(p) = \lambda x$.dem Peter x helfen
$\mathcal{G}(e) =$ dem Peter schwimmen helfen

λx.daß wir der Maria x lassen
|
λx.der Maria x lassen
|
der Maria dem Peter dem Peter schwimmen helfen helfen lassen
|
dem Peter dem Peter schwimmen helfen helfen
|
dem Peter schwimmen helfen

A (too) simple grammar: the case of German

$t : l \to s$, $m : l \to l$, $p : l \to l$, $e : l$.
German: daß wir (der Maria)$^{n+1}$ (dem Peter)$^{p+1}$ schwimmen (helfen)$^{p+1}$ (lassen)$^{n+1}$
$\mathcal{G}(t) = \lambda x.$daß wir der Maria x dem Peter helfen lassen
$\mathcal{G}(m) = \lambda x.$der Maria x lassen
$\mathcal{G}(p) = \lambda x.$dem Peter x helfen
$\mathcal{G}(e) =$ dem Peter schwimmen helfen

$\lambda x.$daß wir der Maria x lassen
|
der Maria der Maria dem Peter dem Peter schwimmen helfen helfen lassen lassen
|
der Maria dem Peter dem Peter schwimmen helfen helfen lassen
|
dem Peter dem Peter schwimmen helfen helfen
|
dem Peter schwimmen helfen

A (too) simple grammar: the case of German

$t : l \to s$, $m : l \to l$, $p : l \to l$, $e : l$.
German: daß wir (der Maria)$^{n+1}$ (dem Peter)$^{p+1}$ schwimmen (helfen)$^{p+1}$ (lassen)$^{n+1}$
$\mathcal{G}(t) = \lambda x$.daß wir der Maria x dem Peter helfen lassen
$\mathcal{G}(m) = \lambda x$.der Maria x lassen
$\mathcal{G}(p) = \lambda x$.dem Peter x helfen
$\mathcal{G}(e) =$ dem Peter schwimmen helfen

daß wir (der Maria)^3 dem Peter dem Peter schwimmen helfen helfen (lassen)^3

|

der Maria der Maria dem Peter dem Peter schwimmen helfen helfen lassen lassen

|

der Maria dem Peter dem Peter schwimmen helfen helfen lassen

|

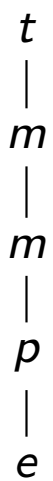
dem Peter dem Peter schwimmen helfen helfen

|

dem Peter schwimmen helfen

A (too) simple grammar: the case of Dutch

$t : l \to s$, $m : l \to l$, $p : l \to l$, $e : l$.
Dutch: dat we (Marie)$^{n+1}$ (Piet)$^{p+1}$ (laten)$^{n+1}$ (helpen)$^{p+1}$ zwemmen
$\mathcal{D}(t) = \lambda P.P(\lambda xy.\text{dat we Marie } x \text{ laten } y)$
$\mathcal{D}(m) = \lambda Pf.P(\lambda xy.f(\text{Mary } x)(\text{laten } y))$
$\mathcal{D}(p) = \lambda Pf.P(\lambda xy.f(\text{Piet } x)(\text{helpen } y))$
$\mathcal{D}(e) = \lambda f.f\,(\text{Piet})(\text{ helpen zwemmen})$



A (too) simple grammar: the case of Dutch

$t : l \to s$, $m : l \to l$, $p : l \to l$, $e : l$.
Dutch: dat we (Marie)$^{n+1}$ (Piet)$^{p+1}$ (laten)$^{n+1}$ (helpen)$^{p+1}$ zwemmen
$\mathcal{D}(t) = \lambda P.P(\lambda xy.\text{dat we Marie } x \text{ laten } y)$
$\mathcal{D}(m) = \lambda Pf.P(\lambda xy.f(\text{Mary } x)(\text{laten } y))$
$\mathcal{D}(p) = \lambda Pf.P(\lambda xy.f(\text{Piet } x)(\text{helpen } y))$
$\mathcal{D}(e) = \lambda f.f\,(\text{Piet})\,(\text{ helpen zwemmen})$

$$\lambda P.P(\lambda xy.\text{dat we Marie } x \text{ laten } y)$$
$$\vdots$$
$$\lambda Pf.P(\lambda xy.f(\text{Mary } x)(\text{laten } y))$$
$$\vdots$$
$$\lambda Pf.P(\lambda xy.f(\text{Mary } x)(\text{laten } y))$$
$$\vdots$$
$$\lambda Pf.P(\lambda xy.f(\text{Piet } x)(\text{helpen } y))$$
$$\vdots$$
$$\lambda f.f\,(\text{Piet})\,(\text{helpen zwemmen})$$

$$\lambda Pf.P(\lambda xy.f(\text{Piet } x)(\text{helpen } y))(\lambda f.f\,(\text{Piet})\,(\text{ helpen zwemmen}))$$

A (too) simple grammar: the case of Dutch

$t : l \to s$, $m : l \to l$, $p : l \to l$, $e : l$.
Dutch: dat we (Marie)$^{n+1}$ (Piet)$^{p+1}$ (laten)$^{n+1}$ (helpen)$^{p+1}$ zwemmen
$\mathcal{D}(t) = \lambda P.P(\lambda xy.\text{dat we Marie } x \text{ laten } y)$
$\mathcal{D}(m) = \lambda Pf.P(\lambda xy.f(\text{Mary } x)(\text{laten } y))$
$\mathcal{D}(p) = \lambda Pf.P(\lambda xy.f(\text{Piet } x)(\text{helpen } y))$
$\mathcal{D}(e) = \lambda f.f\,(\text{Piet})\,(\text{ helpen zwemmen})$

$$\lambda P.P(\lambda xy.\text{dat we Marie } x \text{ laten } y)$$
$$|$$
$$\lambda Pf.P(\lambda xy.f(\text{Mary } x)(\text{laten } y))$$
$$|$$
$$\lambda Pf.P(\lambda xy.f(\text{Mary } x)(\text{laten } y))$$
$$|$$
$$\lambda Pf.P(\lambda xy.f(\text{Piet } x)(\text{helpen } y))$$
$$|$$
$$\lambda f.f\,(\text{Piet})\,(\text{helpen zwemmen})$$

$\lambda Pf.P(\lambda xy.f(\text{Piet } x)(\text{helpen } y))(\lambda f.f\,(\text{Piet})\,(\text{ helpen zwemmen}))$

A (too) simple grammar: the case of Dutch

$t : l \to s$, $m : l \to l$, $p : l \to l$, $e : l$.
Dutch: dat we (Marie)$^{n+1}$ (Piet)$^{p+1}$ (laten)$^{n+1}$ (helpen)$^{p+1}$ zwemmen
$\mathcal{D}(t) = \lambda P.P(\lambda xy.\text{dat we Marie } x \text{ laten } y)$
$\mathcal{D}(m) = \lambda Pf.P(\lambda xy.f(\text{Mary } x)(\text{laten } y))$
$\mathcal{D}(p) = \lambda Pf.P(\lambda xy.f(\text{Piet } x)(\text{helpen } y))$
$\mathcal{D}(e) = \lambda f.f\,(\text{Piet})\,(\text{ helpen zwemmen})$

$$\lambda P.P(\lambda xy.\text{dat we Marie } x \text{ laten } y)$$
$$\vdots$$
$$\lambda Pf.P(\lambda xy.f(\text{Mary } x)(\text{laten } y))$$
$$\vdots$$
$$\lambda Pf.P(\lambda xy.f(\text{Mary } x)(\text{laten } y))$$
$$\vdots$$
$$\lambda Pf.P(\lambda xy.f(\text{Piet } x)(\text{helpen } y))$$
$$\vdots$$
$$\lambda f.f\,(\text{Piet})\,(\text{helpen zwemmen})$$

$\lambda f.(\lambda f.f\,(\text{Piet})\,(\text{ helpen}))(\lambda xy.f(\text{Piet } x)(\text{helpen zwemmen } y))$

A (too) simple grammar: the case of Dutch

$t : l \to s$, $m : l \to l$, $p : l \to l$, $e : l$.
Dutch: dat we (Marie)$^{n+1}$ (Piet)$^{p+1}$ (laten)$^{n+1}$ (helpen)$^{p+1}$ zwemmen
$\mathcal{D}(t) = \lambda P.P(\lambda xy.\text{dat we Marie } x \text{ laten } y)$
$\mathcal{D}(m) = \lambda Pf.P(\lambda xy.f(\text{Mary } x)(\text{laten } y))$
$\mathcal{D}(p) = \lambda Pf.P(\lambda xy.f(\text{Piet } x)(\text{helpen } y))$
$\mathcal{D}(e) = \lambda f.f\,(\text{Piet})\,(\text{helpen zwemmen})$

$$\lambda P.P(\lambda xy.\text{dat we Marie } x \text{ laten } y)$$
$$\vdots$$
$$\lambda Pf.P(\lambda xy.f(\text{Mary } x)(\text{laten } y))$$
$$\vdots$$
$$\lambda Pf.P(\lambda xy.f(\text{Mary } x)(\text{laten } y))$$
$$\vdots$$
$$\lambda Pf.P(\lambda xy.f(\text{Piet } x)(\text{helpen } y))$$
$$\vdots$$
$$\lambda f.f\,(\text{Piet})\,(\text{helpen zwemmen})$$

$\lambda f.(\lambda f.f\,(\text{Piet})\,(\text{helpen}))(\lambda xy.f(\text{Piet } x)(\text{helpen zwemmen } y))$

A (too) simple grammar: the case of Dutch

$t : l \to s$, $m : l \to l$, $p : l \to l$, $e : l$.
Dutch: dat we (Marie)$^{n+1}$ (Piet)$^{p+1}$ (laten)$^{n+1}$ (helpen)$^{p+1}$ zwemmen
$\mathcal{D}(t) = \lambda P.P(\lambda xy.\text{dat we Marie } x \text{ laten } y)$
$\mathcal{D}(m) = \lambda Pf.P(\lambda xy.f(\text{Mary } x)(\text{laten } y))$
$\mathcal{D}(p) = \lambda Pf.P(\lambda xy.f(\text{Piet } x)(\text{helpen } y))$
$\mathcal{D}(e) = \lambda f.f\,(\text{Piet})\,(\text{helpen zwemmen})$

$$\lambda P.P(\lambda xy.\text{dat we Marie } x \text{ laten } y)$$
$$\vdash$$
$$\lambda Pf.P(\lambda xy.f(\text{Mary } x)(\text{laten } y))$$
$$\vdash$$
$$\lambda Pf.P(\lambda xy.f(\text{Mary } x)(\text{laten } y))$$
$$\vdash$$
$$\lambda Pf.P(\lambda xy.f(\text{Piet } x)(\text{helpen } y))$$
$$\vdash$$
$$\lambda f.f\,(\text{Piet})\,(\text{helpen zwemmen})$$

$\lambda f.(\lambda xy.f(\text{Piet } x)(\text{helpen } y))\,(\text{Piet})\,(\text{helpen zwemmen}))$

A (too) simple grammar: the case of Dutch

$t : l \to s$, $m : l \to l$, $p : l \to l$, $e : l$.
Dutch: dat we (Marie)$^{n+1}$ (Piet)$^{p+1}$ (laten)$^{n+1}$ (helpen)$^{p+1}$ zwemmen
$\mathcal{D}(t) = \lambda P.P(\lambda xy.\text{dat we Marie } x \text{ laten } y)$
$\mathcal{D}(m) = \lambda Pf.P(\lambda xy.f(\text{Mary } x)(\text{laten } y))$
$\mathcal{D}(p) = \lambda Pf.P(\lambda xy.f(\text{Piet } x)(\text{helpen } y))$
$\mathcal{D}(e) = \lambda f.f\,(\text{Piet})\,(\text{helpen zwemmen})$

$$\lambda P.P(\lambda xy.\text{dat we Marie } x \text{ laten } y)$$
$$\vdots$$
$$\lambda Pf.P(\lambda xy.f(\text{Mary } x)(\text{laten } y))$$
$$\vdots$$
$$\lambda Pf.P(\lambda xy.f(\text{Mary } x)(\text{laten } y))$$
$$\vdots$$
$$\lambda Pf.P(\lambda xy.f(\text{Piet } x)(\text{helpen } y))$$
$$\vdots$$
$$\lambda f.f\,(\text{Piet})\,(\text{helpen zwemmen})$$

$\lambda f.(\lambda xy.f(\text{Piet } x)(\text{helpen } y))\,(\text{Piet})\,(\text{helpen zwemmen}))$

A (too) simple grammar: the case of Dutch

$t : l \to s$, $m : l \to l$, $p : l \to l$, $e : l$.
Dutch: dat we (Marie)$^{n+1}$ (Piet)$^{p+1}$ (laten)$^{n+1}$ (helpen)$^{p+1}$ zwemmen
$\mathcal{D}(t) = \lambda P.P(\lambda xy.\text{dat we Marie } x \text{ laten } y)$
$\mathcal{D}(m) = \lambda Pf.P(\lambda xy.f(\text{Mary } x)(\text{laten } y))$
$\mathcal{D}(p) = \lambda Pf.P(\lambda xy.f(\text{Piet } x)(\text{helpen } y))$
$\mathcal{D}(e) = \lambda f.f\,(\text{Piet})\,(\text{ helpen zwemmen})$

$$\lambda P.P(\lambda xy.\text{dat we Marie } x \text{ laten } y)$$
$$\vdots$$
$$\lambda Pf.P(\lambda xy.f(\text{Mary } x)(\text{laten } y))$$
$$\vdots$$
$$\lambda Pf.P(\lambda xy.f(\text{Mary } x)(\text{laten } y))$$
$$\vdots$$
$$\lambda Pf.P(\lambda xy.f(\text{Piet } x)(\text{helpen } y))$$
$$\vdots$$
$$\lambda f.f\,(\text{Piet})\,(\text{helpen zwemmen})$$

$\lambda f.(\lambda y.f(\text{Piet Piet})(\text{helpen } y))\ (\text{ helpen zwemmen})$

A (too) simple grammar: the case of Dutch

$t : l \to s$, $m : l \to l$, $p : l \to l$, $e : l$.
Dutch: dat we (Marie)$^{n+1}$ (Piet)$^{p+1}$ (laten)$^{n+1}$ (helpen)$^{p+1}$ zwemmen
$\mathcal{D}(t) = \lambda P.P(\lambda xy.\text{dat we Marie } x \text{ laten } y)$
$\mathcal{D}(m) = \lambda Pf.P(\lambda xy.f(\text{Mary } x)(\text{laten } y))$
$\mathcal{D}(p) = \lambda Pf.P(\lambda xy.f(\text{Piet } x)(\text{helpen } y))$
$\mathcal{D}(e) = \lambda f.f\,(\text{Piet})\,(\text{ helpen zwemmen})$

$$\lambda P.P(\lambda xy.\text{dat we Marie } x \text{ laten } y)$$
$$\vdots$$
$$\lambda Pf.P(\lambda xy.f(\text{Mary } x)(\text{laten } y))$$
$$\vdots$$
$$\lambda Pf.P(\lambda xy.f(\text{Mary } x)(\text{laten } y))$$
$$\vdots$$
$$\lambda Pf.P(\lambda xy.f(\text{Piet } x)(\text{helpen } y))$$
$$\vdots$$
$$\lambda f.f\,(\text{Piet})\,(\text{helpen zwemmen})$$

$\lambda f.(\lambda y.f(\text{Piet Piet})(\text{helpen } y))$ (helpen zwemmen)

A (too) simple grammar: the case of Dutch

$t : l \to s$, $m : l \to l$, $p : l \to l$, $e : l$.
Dutch: dat we (Marie)$^{n+1}$ (Piet)$^{p+1}$ (laten)$^{n+1}$ (helpen)$^{p+1}$ zwemmen
$\mathcal{D}(t) = \lambda P.P(\lambda xy.\text{dat we Marie } x \text{ laten } y)$
$\mathcal{D}(m) = \lambda Pf.P(\lambda xy.f(\text{Mary } x)(\text{laten } y))$
$\mathcal{D}(p) = \lambda Pf.P(\lambda xy.f(\text{Piet } x)(\text{helpen } y))$
$\mathcal{D}(e) = \lambda f.f\,(\text{Piet})\,(\text{ helpen zwemmen})$

$$\lambda P.P(\lambda xy.\text{dat we Marie } x \text{ laten } y)$$
$$\vdots$$
$$\lambda Pf.P(\lambda xy.f(\text{Mary } x)(\text{laten } y))$$
$$\vdots$$
$$\lambda Pf.P(\lambda xy.f(\text{Mary } x)(\text{laten } y))$$
$$\vdots$$
$$\lambda Pf.P(\lambda xy.f(\text{Piet } x)(\text{helpen } y))$$
$$\vdots$$
$$\lambda f.f\,(\text{Piet})\,(\text{helpen zwemmen})$$

$\lambda f.f(\text{Piet Piet})(\text{helpen helpen zwemmen})$

A (too) simple grammar: the case of Dutch

$t : l \to s$, $m : l \to l$, $p : l \to l$, $e : l$.
Dutch: dat we (Marie)$^{n+1}$ (Piet)$^{p+1}$ (laten)$^{n+1}$ (helpen)$^{p+1}$ zwemmen
$\mathcal{D}(t) = \lambda P.P(\lambda xy.\text{dat we Marie } x \text{ laten } y)$
$\mathcal{D}(m) = \lambda Pf.P(\lambda xy.f(\text{Mary } x)(\text{laten } y))$
$\mathcal{D}(p) = \lambda Pf.P(\lambda xy.f(\text{Piet } x)(\text{helpen } y))$
$\mathcal{D}(e) = \lambda f.f\,(\text{Piet})\,(\text{ helpen zwemmen})$

$$\lambda P.P(\lambda xy.\text{dat we Marie } x \text{ laten } y)$$
$$\downarrow$$
$$\lambda Pf.P(\lambda xy.f(\text{Mary } x)(\text{laten } y))$$
$$\downarrow$$
$$\lambda Pf.P(\lambda xy.f(\text{Mary } x)(\text{laten } y))$$
$$\downarrow$$
$$\lambda f.f(\text{Piet Piet})(\text{helpen helpen zwemmen})$$
$$\downarrow$$
$$\lambda f.f\,(\text{Piet})\,(\text{helpen zwemmen})$$

A (too) simple grammar: the case of Dutch

$t : l \to s$, $m : l \to l$, $p : l \to l$, $e : l$.
Dutch: dat we (Marie)$^{n+1}$ (Piet)$^{p+1}$ (laten)$^{n+1}$ (helpen)$^{p+1}$ zwemmen
$\mathcal{D}(t) = \lambda P.P(\lambda xy.\text{dat we Marie } x \text{ laten } y)$
$\mathcal{D}(m) = \lambda Pf.P(\lambda xy.f(\text{Mary } x)(\text{laten } y))$
$\mathcal{D}(p) = \lambda Pf.P(\lambda xy.f(\text{Piet } x)(\text{helpen } y))$
$\mathcal{D}(e) = \lambda f.f\,(\text{Piet})\,(\text{ helpen zwemmen})$

$$\lambda P.P(\lambda xy.\text{dat we Marie } x \text{ laten } y)$$
$$\vdots$$
$$\lambda Pf.P(\lambda xy.f(\text{Mary } x)(\text{laten } y))$$
$$\vdots$$
$$\lambda f.f(\text{Mary Piet Piet})(\text{laten helpen helpen zwemmen})$$
$$\vdots$$
$$\lambda f.f(\text{Piet Piet})(\text{helpen helpen zwemmen})$$
$$\vdots$$
$$\lambda f.f\,(\text{Piet})\,(\text{helpen zwemmen})$$

A (too) simple grammar: the case of Dutch

$t : l \to s$, $m : l \to l$, $p : l \to l$, $e : l$.
Dutch: dat we (Marie)$^{n+1}$ (Piet)$^{p+1}$ (laten)$^{n+1}$ (helpen)$^{p+1}$ zwemmen
$\mathcal{D}(t) = \lambda P.P(\lambda xy.\text{dat we Marie } x \text{ laten } y)$
$\mathcal{D}(m) = \lambda Pf.P(\lambda xy.f(\text{Mary } x)(\text{laten } y))$
$\mathcal{D}(p) = \lambda Pf.P(\lambda xy.f(\text{Piet } x)(\text{helpen } y))$
$\mathcal{D}(e) = \lambda f.f\,(\text{Piet})\,(\text{ helpen zwemmen})$

$$\lambda P.P(\lambda xy.\text{dat we Marie } x \text{ laten } y)$$
$$\vdots$$
$$\lambda f.f(\text{Mary Mary Piet Piet})(\text{laten laten helpen helpen zwemmen})$$
$$\vdots$$
$$\lambda f.f(\text{Mary Piet Piet})(\text{laten helpen helpen zwemmen})$$
$$\vdots$$
$$\lambda f.f(\text{Piet Piet})(\text{helpen helpen zwemmen})$$
$$\vdots$$
$$\lambda f.f\,(\text{Piet})\,(\text{helpen zwemmen})$$

A (too) simple grammar: the case of Dutch

$t : l \to s$, $m : l \to l$, $p : l \to l$, $e : l$.
Dutch: dat we (Marie)$^{n+1}$ (Piet)$^{p+1}$ (laten)$^{n+1}$ (helpen)$^{p+1}$ zwemmen
$\mathcal{D}(t) = \lambda P.P(\lambda xy.\text{dat we Marie } x \text{ laten } y)$
$\mathcal{D}(m) = \lambda Pf.P(\lambda xy.f(\text{Mary } x)(\text{laten } y))$
$\mathcal{D}(p) = \lambda Pf.P(\lambda xy.f(\text{Piet } x)(\text{helpen } y))$
$\mathcal{D}(e) = \lambda f.f\,(\text{Piet})\,(\text{ helpen zwemmen})$

dat we Marie Marie Marie Piet Piet laten laten laten helpen helpen zwemmen

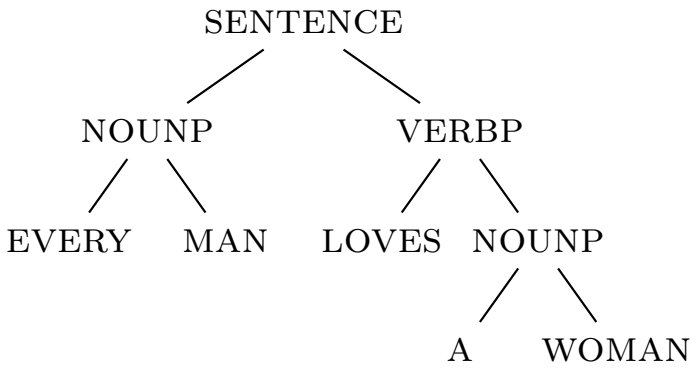
$\lambda f.f(\text{Mary Mary Piet Piet})(\text{laten laten helpen helpen zwemmen})$

$\lambda f.f(\text{Mary Piet Piet})(\text{laten helpen helpen zwemmen})$

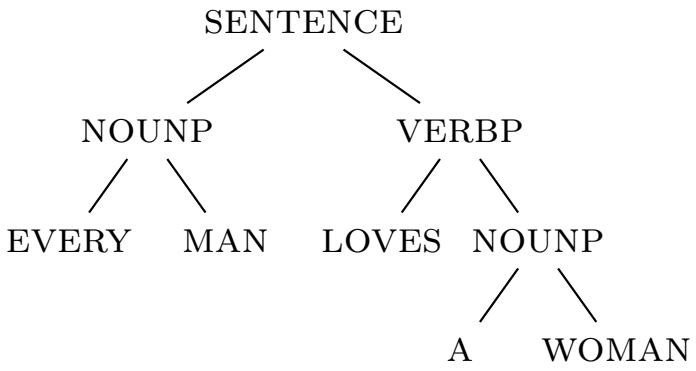
$\lambda f.f(\text{Piet Piet})(\text{helpen helpen zwemmen})$

$\lambda f.f\,(\text{Piet})\,(\text{helpen zwemmen})$

Montague semantics

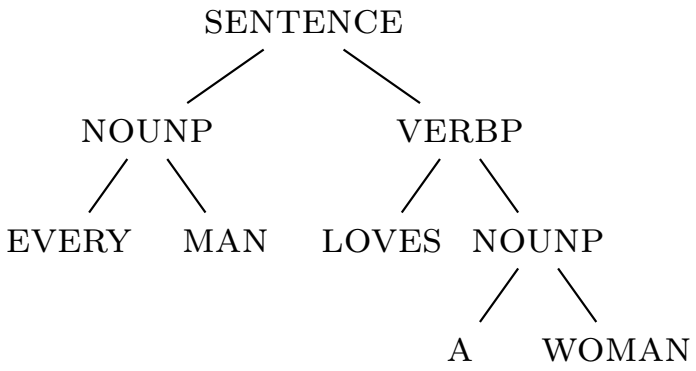


Montague semantics



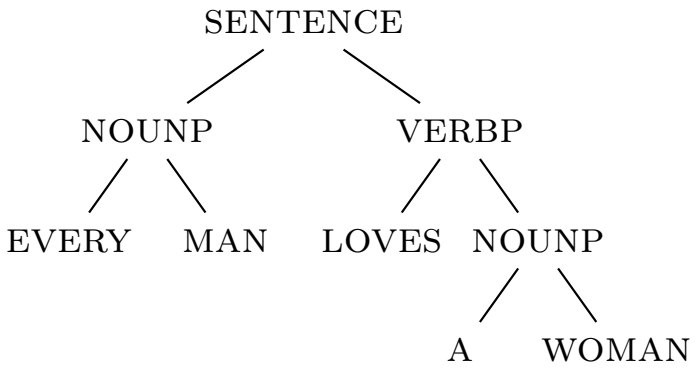
$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$

Montague semantics



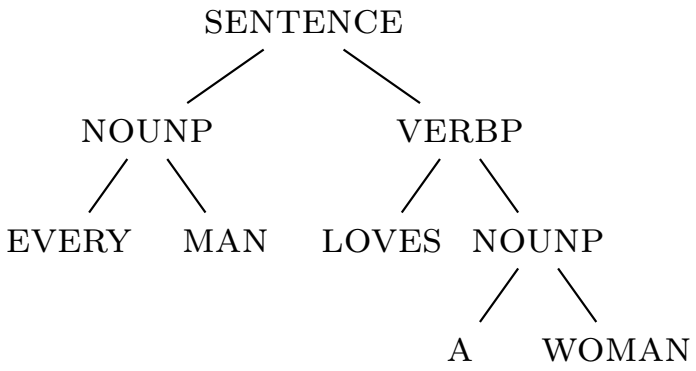
$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$
$\lambda P\ Q.\exists(\lambda z.P\ z \wedge Q\ z)(\lambda x.woman\ x)$

Montague semantics



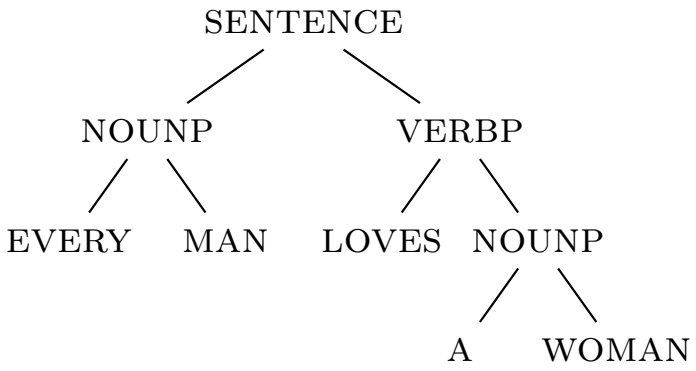
$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$
$\lambda P\, Q.\exists(\lambda z.P\, z \wedge Q\, z)(\lambda x.woman\, x)$

Montague semantics



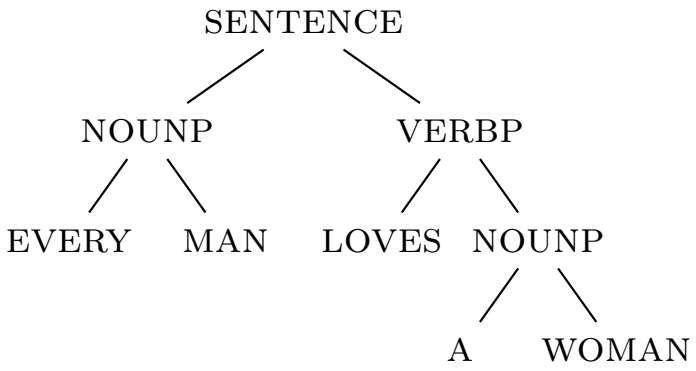
$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$
$\lambda Q.\exists(\lambda z.(\lambda x.\mathit{woman}\, x)\, z \wedge Q\, z)$

Montague semantics



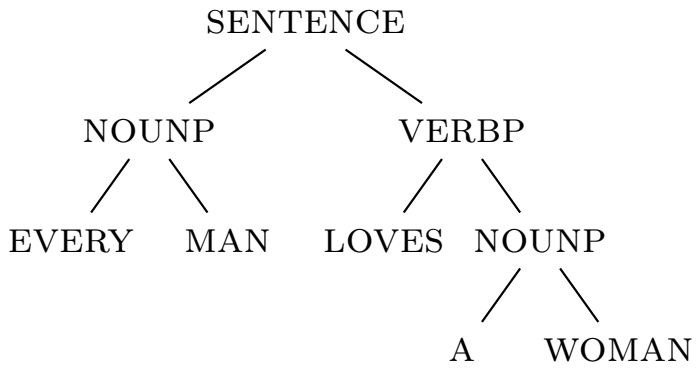
$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$
$\lambda Q.\exists(\lambda z.(\lambda x.\mathit{woman}\, x)\, z \wedge Q\, z)$

Montague semantics



$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$
$\lambda Q.\exists(\lambda z.woman\, z \wedge Q\, z)$

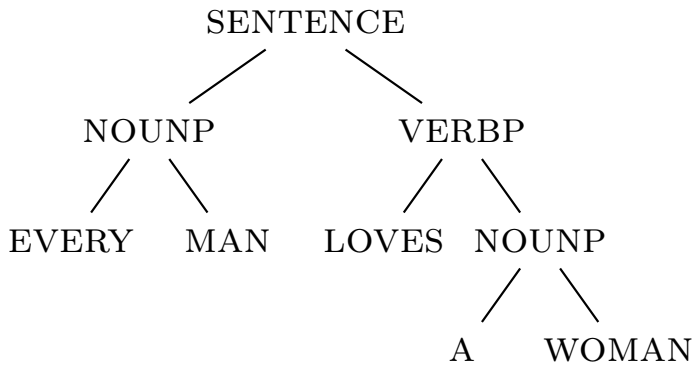
Montague semantics



$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$
$\lambda Q.\exists(\lambda z.woman\, z \wedge Q\, z)$

$\mathcal{L}_{sem}(\text{NOUNP EVERY MAN}) =$
$\lambda Q.\forall(\lambda u.man\, u \Rightarrow Q\, u)$

Montague semantics

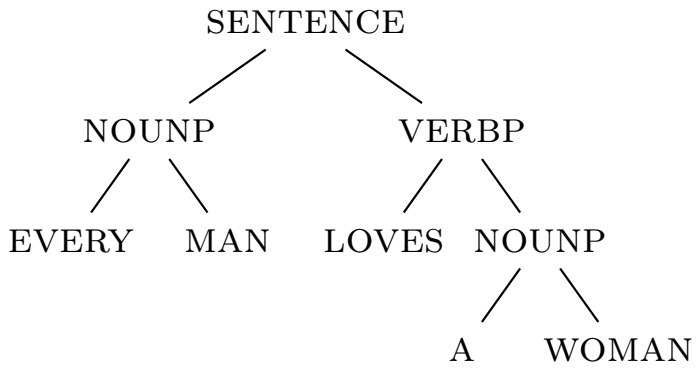


$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$
$\lambda Q.\exists(\lambda z.woman\, z \wedge Q\, z)$

$\mathcal{L}_{sem}(\text{NOUNP EVERY MAN}) =$
$\lambda Q.\forall(\lambda u.man\, u \Rightarrow Q\, u)$

$\mathcal{L}_{sem}(\text{VERBP LOVES } (\text{NOUNP A WOMAN})) =$

Montague semantics

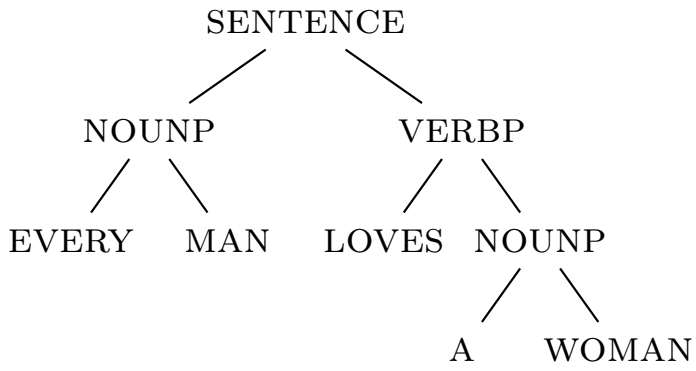


$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$
$\lambda Q.\exists(\lambda z.\mathit{woman}\, z \wedge Q\, z)$

$\mathcal{L}_{sem}(\text{NOUNP EVERY MAN}) =$
$\lambda Q.\forall(\lambda u.\mathit{man}\, u \Rightarrow Q\, u)$

$\mathcal{L}_{sem}(\text{VERBP LOVES (NOUNP A WOMAN)}) =$
$\lambda O\, S.S(\lambda x.O(\lambda y.\mathit{love}\, x\, y))(\lambda P.\exists(\lambda z.\mathit{woman}\, z \wedge P\, z))$

Montague semantics

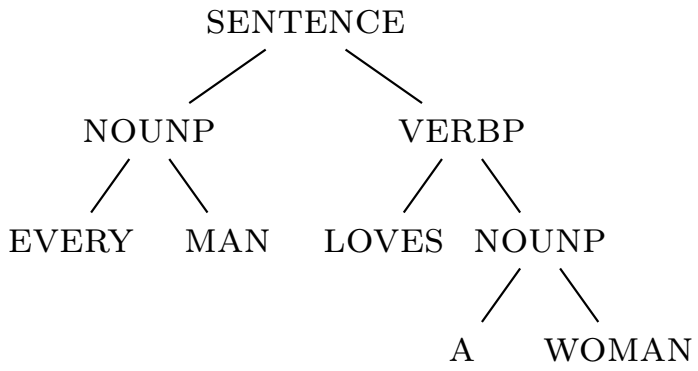


$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$
$\lambda Q.\exists(\lambda z.woman\ z \wedge Q\ z)$

$\mathcal{L}_{sem}(\text{NOUNP EVERY MAN}) =$
$\lambda Q.\forall(\lambda u.man\ u \Rightarrow Q\ u)$

$\mathcal{L}_{sem}(\text{VERBP LOVES (NOUNP A WOMAN)}) =$
$\lambda O\ S.S(\lambda x.O(\lambda y.love\ x\ y))(\lambda P.\exists(\lambda z.woman\ z \wedge P\ z))$

Montague semantics

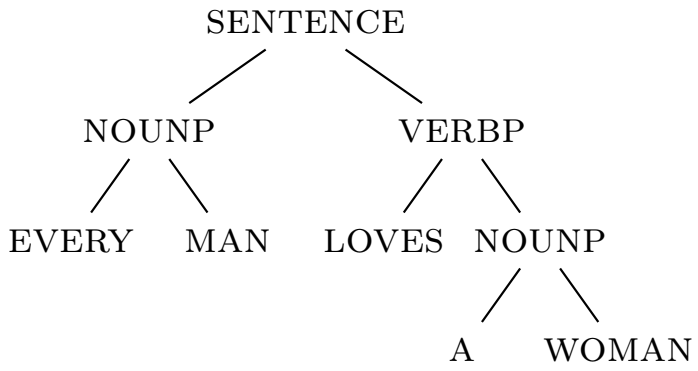


$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$
$\lambda Q.\exists(\lambda z.woman\ z \wedge Q\ z)$

$\mathcal{L}_{sem}(\text{NOUNP EVERY MAN}) =$
$\lambda Q.\forall(\lambda u.man\ u \Rightarrow Q\ u)$

$\mathcal{L}_{sem}(\text{VERBP LOVES } (\text{NOUNP A WOMAN})) =$
$\lambda S.S(\lambda x.(\lambda P.\exists(\lambda z.(\lambda z.woman\ z \wedge P\ z)))(\lambda y.love\ x\ y))$

Montague semantics

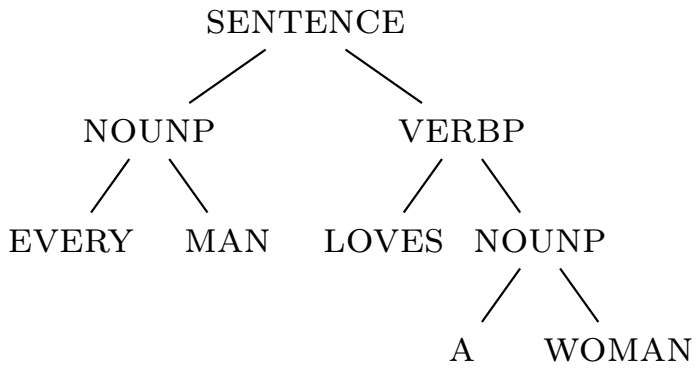


$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$
$\lambda Q.\exists(\lambda z.woman\ z \wedge Q\ z)$

$\mathcal{L}_{sem}(\text{NOUNP EVERY MAN}) =$
$\lambda Q.\forall(\lambda u.man\ u \Rightarrow Q\ u)$

$\mathcal{L}_{sem}(\text{VERBP LOVES (NOUNP A WOMAN)}) =$
$\lambda S.S(\lambda x.(\lambda P.\exists(\lambda z.(\lambda z.woman\ z \wedge P\ z)))(\lambda y.love\ x\ y))$

Montague semantics

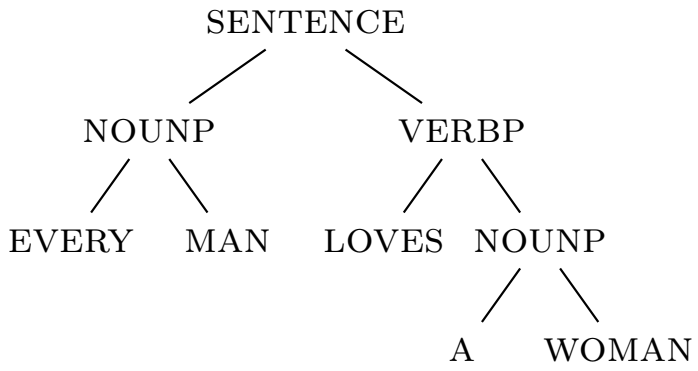


$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$
$\lambda Q.\exists(\lambda z.\mathit{woman}\, z \wedge Q\, z)$

$\mathcal{L}_{sem}(\text{NOUNP EVERY MAN}) =$
$\lambda Q.\forall(\lambda u.\mathit{man}\, u \Rightarrow Q\, u)$

$\mathcal{L}_{sem}(\text{VERBP LOVES } (\text{NOUNP A WOMAN})) =$
$\lambda S.S(\lambda x.\exists(\lambda z.\mathit{woman}\, z \wedge (\lambda y.\mathit{love}\, x\, y)\, z))$

Montague semantics

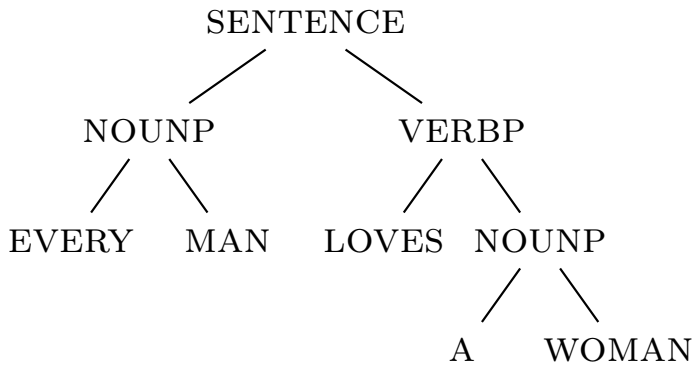


$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$
$\lambda Q.\exists(\lambda z.woman\ z \wedge Q\ z)$

$\mathcal{L}_{sem}(\text{NOUNP EVERY MAN}) =$
$\lambda Q.\forall(\lambda u.man\ u \Rightarrow Q\ u)$

$\mathcal{L}_{sem}(\text{VERBP LOVES (NOUNP A WOMAN)}) =$
$\lambda S.S(\lambda x.\exists(\lambda z.woman\ z \wedge (\lambda y.love\ x\ y)\ z))$

Montague semantics

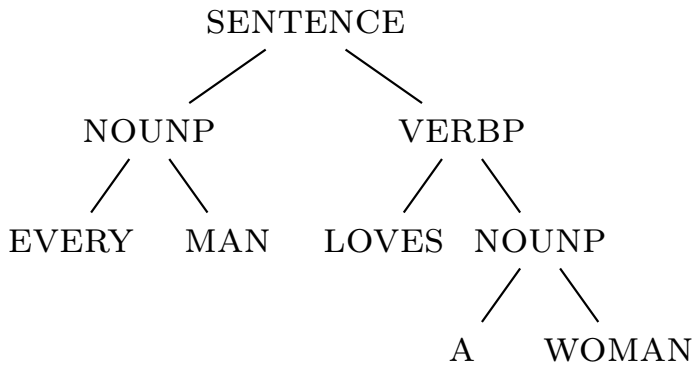


$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$
$\lambda Q.\exists(\lambda z.\mathit{woman}\, z \wedge Q\, z)$

$\mathcal{L}_{sem}(\text{NOUNP EVERY MAN}) =$
$\lambda Q.\forall(\lambda u.\mathit{man}\, u \Rightarrow Q\, u)$

$\mathcal{L}_{sem}(\text{VERBP LOVES } (\text{NOUNP A WOMAN})) =$
$\lambda S.S(\lambda x.\exists(\lambda z.\mathit{woman}\, z \wedge \mathit{love}\, x\, z))$

Montage semantics



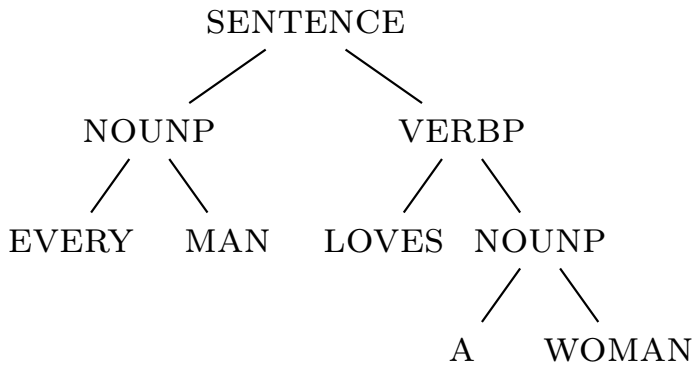
$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$
$\lambda Q.\exists(\lambda z.\mathit{woman}\, z \wedge Q\, z)$

$\mathcal{L}_{sem}(\text{NOUNP EVERY MAN}) =$
$\lambda Q.\forall(\lambda u.\mathit{man}\, u \Rightarrow Q\, u)$

$\mathcal{L}_{sem}(\text{VERBP LOVES }(\text{NOUNP A WOMAN})) =$
$\lambda S.S(\lambda x.\exists(\lambda z.\mathit{woman}\, z \wedge \mathit{love}\, x\, z))$

$\mathcal{L}_{sem}(\text{SENTENCE}(\text{NOUNP EVERY MAN})(\text{VERBP LOVES }(\text{NOUNP A WOMAN}))) =$

Montague semantics



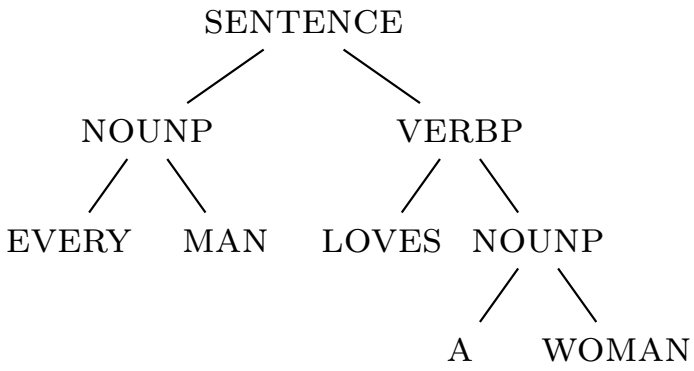
$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$
$\lambda Q.\exists(\lambda z.\mathit{woman}\, z \wedge Q\, z)$

$\mathcal{L}_{sem}(\text{NOUNP EVERY MAN}) =$
$\lambda Q.\forall(\lambda u.\mathit{man}\, u \Rightarrow Q\, u)$

$\mathcal{L}_{sem}(\text{VERBP LOVES }(\text{NOUNP A WOMAN})) =$
$\lambda S.S(\lambda x.\exists(\lambda z.\mathit{woman}\, z \wedge \mathit{love}\, x\, z))$

$\mathcal{L}_{sem}(\text{SENTENCE}(\text{NOUNP EVERY MAN})(\text{VERBP LOVES }(\text{NOUNP A WOMAN}))) =$
$(\lambda S.S(\lambda x.\exists(\lambda z.\mathit{woman}\, z \wedge \mathit{love}\, x\, z)))(\lambda Q.\forall(\lambda u.\mathit{man}\, u \Rightarrow Q\, u))$

Montague semantics



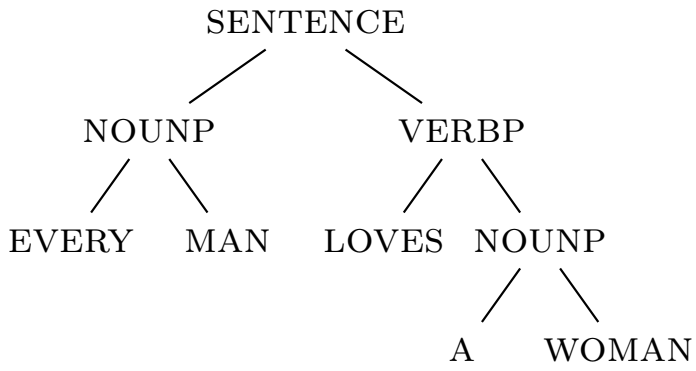
$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$
$\lambda Q.\exists(\lambda z.woman\ z \wedge Q\ z)$

$\mathcal{L}_{sem}(\text{NOUNP EVERY MAN}) =$
$\lambda Q.\forall(\lambda u.man\ u \Rightarrow Q\ u)$

$\mathcal{L}_{sem}(\text{VERBP LOVES (NOUNP A WOMAN)}) =$
$\lambda S.S(\lambda x.\exists(\lambda z.woman\ z \wedge love\ x\ z))$

$\mathcal{L}_{sem}(\text{SENTENCE(NOUNP EVERY MAN)(VERBP LOVES (NOUNP A WOMAN))}) =$
$(\lambda S.S(\lambda x.\exists(\lambda z.woman\ z \wedge love\ x\ z)))(\lambda Q.\forall(\lambda u.man\ u \Rightarrow Q\ u))$

Montague semantics



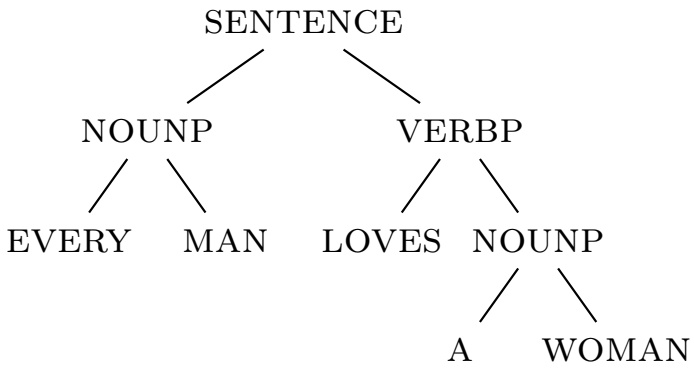
$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$
$\lambda Q.\exists(\lambda z.\mathit{woman}\, z \wedge Q\, z)$

$\mathcal{L}_{sem}(\text{NOUNP EVERY MAN}) =$
$\lambda Q.\forall(\lambda u.\mathit{man}\, u \Rightarrow Q\, u)$

$\mathcal{L}_{sem}(\text{VERBP LOVES }(\text{NOUNP A WOMAN})) =$
$\lambda S.S(\lambda x.\exists(\lambda z.\mathit{woman}\, z \wedge \mathit{love}\, x\, z))$

$\mathcal{L}_{sem}(\text{SENTENCE}(\text{NOUNP EVERY MAN})(\text{VERBP LOVES }(\text{NOUNP A WOMAN}))) =$
$(\lambda Q.\forall(\lambda u.\mathit{man}\, u \Rightarrow Q\, u))(\lambda x.(\exists(\lambda z.\mathit{woman}\, z \wedge \mathit{love}\, x\, z)))$

Montague semantics



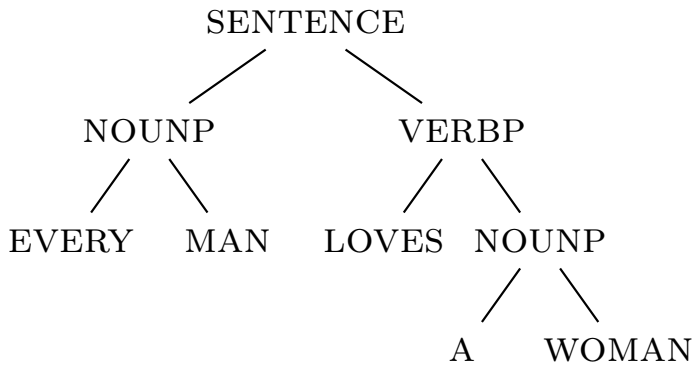
$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$
$\lambda Q.\exists(\lambda z.woman\ z \wedge Q\ z)$

$\mathcal{L}_{sem}(\text{NOUNP EVERY MAN}) =$
$\lambda Q.\forall(\lambda u.man\ u \Rightarrow Q\ u)$

$\mathcal{L}_{sem}(\text{VERBP LOVES (NOUNP A WOMAN)}) =$
$\lambda S.S(\lambda x.\exists(\lambda z.woman\ z \wedge love\ x\ z))$

$\mathcal{L}_{sem}(\text{SENTENCE}(\text{NOUNP EVERY MAN})(\text{VERBP LOVES (NOUNP A WOMAN)})) =$
$(\lambda Q.\forall(\lambda u.man\ u \Rightarrow Q\ u))(\lambda x.(\exists(\lambda z.woman\ z \wedge love\ x\ z)))$

Montage semantics



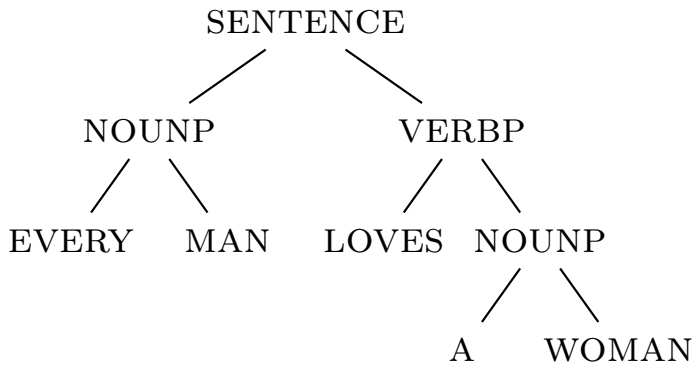
$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$
$\lambda Q.\exists(\lambda z.\mathit{woman}\, z \wedge Q\, z)$

$\mathcal{L}_{sem}(\text{NOUNP EVERY MAN}) =$
$\lambda Q.\forall(\lambda u.\mathit{man}\, u \Rightarrow Q\, u)$

$\mathcal{L}_{sem}(\text{VERBP LOVES }(\text{NOUNP A WOMAN})) =$
$\lambda S.S(\lambda x.\exists(\lambda z.\mathit{woman}\, z \wedge \mathit{love}\, x\, z))$

$\mathcal{L}_{sem}(\text{SENTENCE}(\text{NOUNP EVERY MAN})(\text{VERBP LOVES }(\text{NOUNP A WOMAN}))) =$
$\forall(\lambda u.\mathit{man}\, u \Rightarrow (\lambda x.\exists(\lambda z.\mathit{woman}\, z \wedge \mathit{love}\, x\, z))\, u)$

Montague semantics



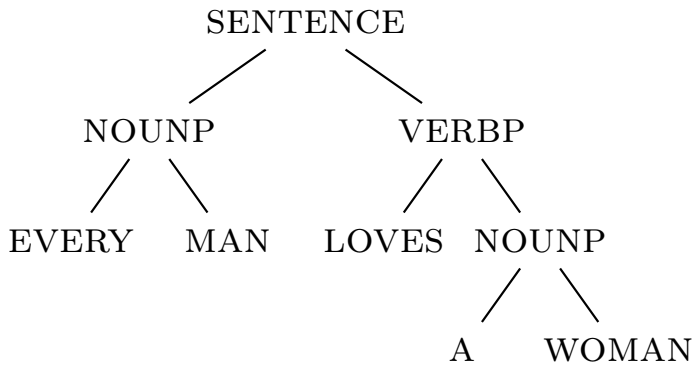
$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$
$\lambda Q.\exists(\lambda z.\mathit{woman}\, z \wedge Q\, z)$

$\mathcal{L}_{sem}(\text{NOUNP EVERY MAN}) =$
$\lambda Q.\forall(\lambda u.\mathit{man}\, u \Rightarrow Q\, u)$

$\mathcal{L}_{sem}(\text{VERBP LOVES }(\text{NOUNP A WOMAN})) =$
$\lambda S.S(\lambda x.\exists(\lambda z.\mathit{woman}\, z \wedge \mathit{love}\, x\, z))$

$\mathcal{L}_{sem}(\text{SENTENCE}(\text{NOUNP EVERY MAN})(\text{VERBP LOVES }(\text{NOUNP A WOMAN}))) =$
$\forall(\lambda u.\mathit{man}\, u \Rightarrow (\lambda x.\exists(\lambda z.\mathit{woman}\, z \wedge \mathit{love}\, x\, z))\, u)$

Montague semantics



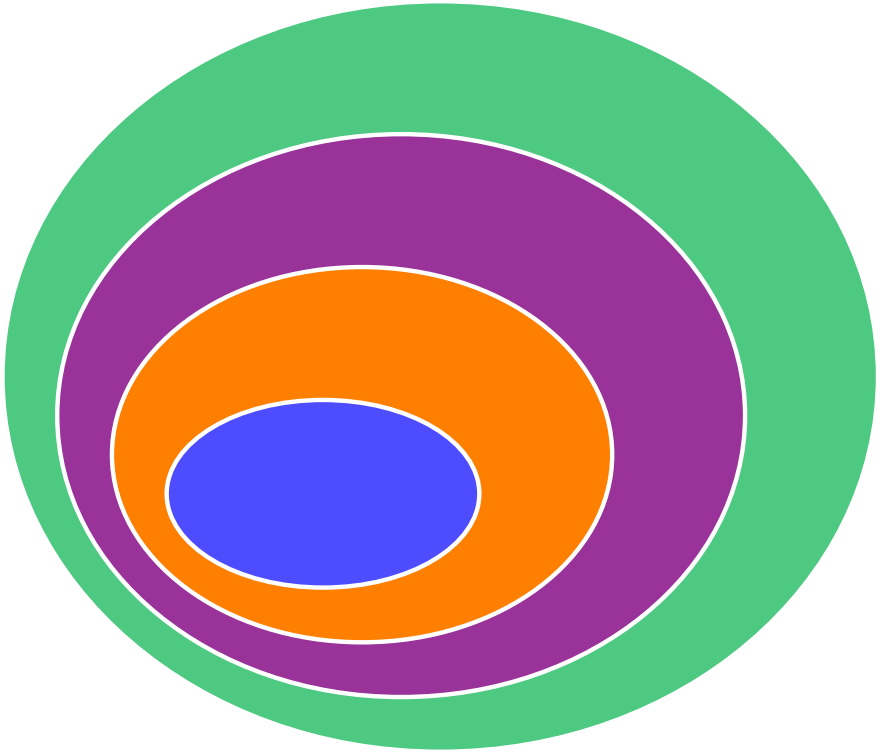
$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$
$\lambda Q.\exists(\lambda z.\mathit{woman}\, z \wedge Q\, z)$

$\mathcal{L}_{sem}(\text{NOUNP EVERY MAN}) =$
$\lambda Q.\forall(\lambda u.\mathit{man}\, u \Rightarrow Q\, u)$

$\mathcal{L}_{sem}(\text{VERBP LOVES }(\text{NOUNP A WOMAN})) =$
$\lambda S.S(\lambda x.\exists(\lambda z.\mathit{woman}\, z \wedge \mathit{love}\, x\, z))$

$\mathcal{L}_{sem}(\text{SENTENCE}(\text{NOUNP EVERY MAN})(\text{VERBP LOVES }(\text{NOUNP A WOMAN}))) =$
$\forall(\lambda u.\mathit{man}\, u \Rightarrow \exists(\lambda z.\mathit{woman}\, z \wedge \mathit{love}\, u\, z))$

Non-duplicating grammars and mildly context sensitive languages



$MCTAG = MCFG = HR$
$= OUT(DTWT)$
$= yDT_{fc}(REGT) = LUSCG = MG$
$= \lambda\text{-CFL}_{lin}(4) = \lambda\text{-CFL}_{lin}$

$MCFG_{wm} = IO_{nd} = OI_{nd}$
$= CCFG = \lambda\text{-CFL}_{lin}(3)$

$TAG = LIG = CCG = HG$

$CFL = \lambda\text{-CFL}_{lin}(2)$

Are good properties preserved?

- parsing decidability?
- closure under intersection with regular sets?
- deletion elimination?
- ...

The not so intuitive answer is yes. For proving it, we appeal to finitary domain theory.

Finitary Scott domains (1)

Applicative structure

- $\mathcal{D}_o$ is a finite lattice (typically a the lattice of subsets of the states of an automaton)
- $\mathcal{D}_{\alpha\to\beta}$ is the set of monotone functions from $\mathcal{D}_\alpha$ to $\mathcal{D}_\beta$ ordered pointwise:
  - $f \in \mathcal{D}_{\alpha\to\beta}$ iff for every $a, b \in \mathcal{D}_\alpha$, $a \leq b$ implies $f(a) \leq f(b)$,
  - $f \leq g$ iff for every a, $f(a) \leq g(a)$.

Finitary Scott domains (1)

Applicative structure

- $\mathcal{D}_o$ is a finite lattice (typically a the lattice of subsets of the states of an automaton)
- $\mathcal{D}_{\alpha\to\beta}$ is the set of monotone functions from $\mathcal{D}_\alpha$ to $\mathcal{D}_\beta$ ordered pointwise:
  - $f \in \mathcal{D}_{\alpha\to\beta}$ iff for every $a, b \in \mathcal{D}_\alpha$, $a \leq b$ implies $f(a) \leq f(b)$,
  - $f \leq g$ iff for every a, $f(a) \leq g(a)$.

Terms interpretation

- $[\![x, \nu]\!] = \nu(x)$, $[\![c, \nu]\!] = \rho(c)$,

Finitary Scott domains (1)

Applicative structure

- $\mathcal{D}_o$ is a finite lattice (typically a the lattice of subsets of the states of an automaton)
- $\mathcal{D}_{\alpha\to\beta}$ is the set of monotone functions from $\mathcal{D}_\alpha$ to $\mathcal{D}_\beta$ ordered pointwise:
  - $f \in \mathcal{D}_{\alpha\to\beta}$ iff for every $a, b \in \mathcal{D}_\alpha$, $a \leq b$ implies $f(a) \leq f(b)$,
  - $f \leq g$ iff for every a, $f(a) \leq g(a)$.

Terms interpretation

- $[\![x, \nu]\!] = \nu(x)$, $[\![c, \nu]\!] = \rho(c)$,
- $[\![MN, \nu]\!] = [\![M, \nu]\!]([\![N, \nu]\!])$,

Finitary Scott domains (1)

Applicative structure

- $\mathcal{D}_o$ is a finite lattice (typically a the lattice of subsets of the states of an automaton)
- $\mathcal{D}_{\alpha\to\beta}$ is the set of monotone functions from $\mathcal{D}_\alpha$ to $\mathcal{D}_\beta$ ordered pointwise:
  - $f \in \mathcal{D}_{\alpha\to\beta}$ iff for every $a, b \in \mathcal{D}_\alpha$, $a \leq b$ implies $f(a) \leq f(b)$,
  - $f \leq g$ iff for every a, $f(a) \leq g(a)$.

Terms interpretation

- $[\![x, \nu]\!] = \nu(x)$, $[\![c, \nu]\!] = \rho(c)$,
- $[\![MN, \nu]\!] = [\![M, \nu]\!]([\![N, \nu]\!])$,
- $[\![\lambda x.M, \nu]\!](f) = [\![M, \nu[f/x]]\!]$.

Finitary Scott domains (2)

Theorem (Correctness)

For every term M and N, if $M =_{\beta\eta} N$, then $[\![M, \nu]\!] = [\![N, \nu]\!]$.

Finitary Scott domains (2)

Theorem (Correctness)

For every term M and N, if $M =_{\beta\eta} N$, then $[\![M, \nu]\!] = [\![N, \nu]\!]$.

Theorem (Finite Completeness (Statman 82), (Kobele, S. 14))

For every term M there is a finitary Scott model, a function f and a valuation ν so that for every N:

$$[\![N, \nu]\!] \geq f \text{ iff } M =_{\beta\eta} N$$

Finitary Scott domains (2)

Theorem (Correctness)

For every term M and N, if $M =_{\beta\eta} N$, then $[\![M, \nu]\!] = [\![N, \nu]\!]$.

Theorem (Finite Completeness (Statman 82), (Kobele, S. 14))

For every term M there is a finitary Scott model, a function f and a valuation ν so that for every N:

$$[\![N, \nu]\!] \geq f \text{ iff } M =_{\beta\eta} N$$

Definition (Recognizability (S. 09))

A language $\mathcal{L}$ of closed terms of type α is recognizable iff there is a finitary Scott domain $(\mathcal{D}_\alpha)_{\alpha \in \text{types}}$ and $R \subseteq \mathcal{D}_\alpha$ so that:

$$\mathcal{L} = \{M \mid M \text{ closed and } [\![M, \emptyset]\!] \in R\}$$

Decidability

Theorem (Loader 00)

Finite λ-definability is undecidable

Decidability

Theorem (Loader 00)

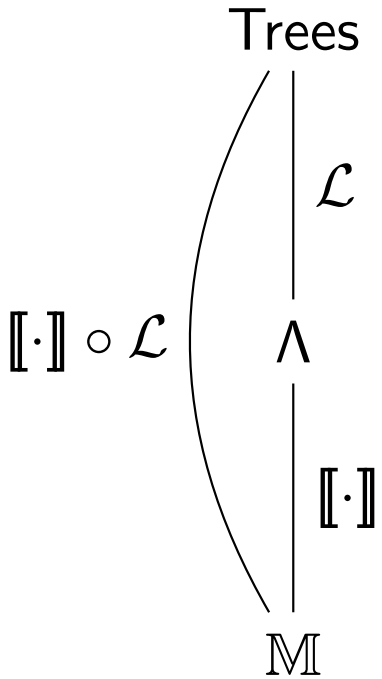
Finite λ-definability is undecidable $\equiv$ emptiness of recognizable sets of λ-terms is undecidable.

Decidability

Theorem (Loader 00)

Finite λ-definability is undecidable $\equiv$ emptiness of recognizable sets of λ-terms is undecidable.

Context-freeness brings decidability back:



$[\![\cdot]\!] \circ \mathcal{L}$ interprets the trees in a finite domain

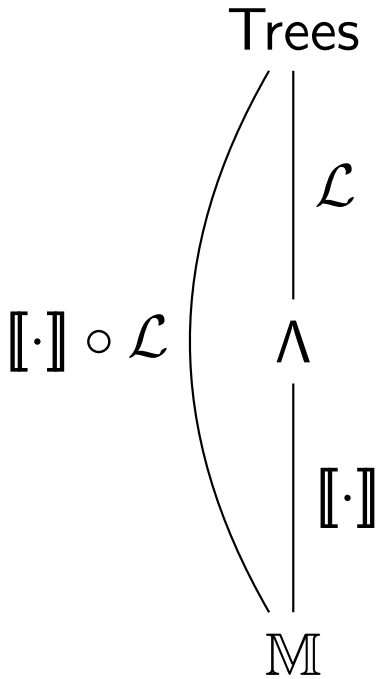
We can effectively compute the set $\{f \in \mathbb{M} \mid \exists t \in \text{Trees}.f = [\![\mathcal{L}(t)]\!]\}$

Decidability

Theorem (Loader 00)

Finite λ-definability is undecidable $\equiv$ emptiness of recognizable sets of λ-terms is undecidable.

Context-freeness brings decidability back:



$[\![\cdot]\!] \circ \mathcal{L}$ interprets the trees in a finite domain

We can effectively compute the set $\{f \in \mathbb{M} \mid \exists t \in \text{Trees}.f = [\![\mathcal{L}(t)]\!]\}$

It yields a finite automaton that recognizes the derivations according to their value in $\mathbb{M}$

Domains in logical form

Step functions

Step functions are a basic block to describe monotone functions, given $a \in \mathcal{D}_1$ and $b \in \mathcal{D}_2$,

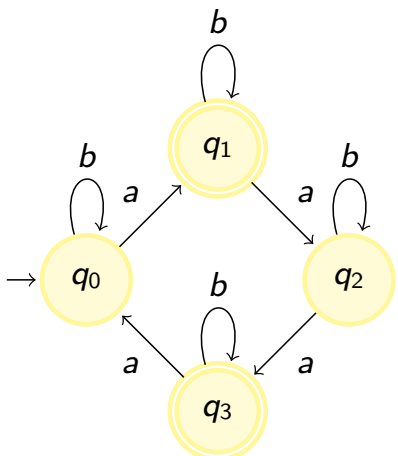
$$(a \mapsto b)(x) = \begin{cases} b \text{ when } a \leq x \\ \bot \text{ otherwise} \end{cases}$$

Domains in logical form

Step functions

Step functions are a basic block to describe monotone functions, given $a \in \mathcal{D}_1$ and $b \in \mathcal{D}_2$,

$$(a \mapsto b)(x) = \begin{cases} b \text{ when } a \leq x \\ \bot \text{ otherwise} \end{cases}$$



$$\dfrac{\dfrac{}{y \geq q_1 \vdash y \geq q_1}}{\vdash \lambda y.a(by) \geq q_1 \mapsto q_0} \qquad \dfrac{\dfrac{}{y \geq q_2 \vdash y \geq q_2}}{\vdash \lambda y.a(by) \geq q_2 \mapsto q_1}$$

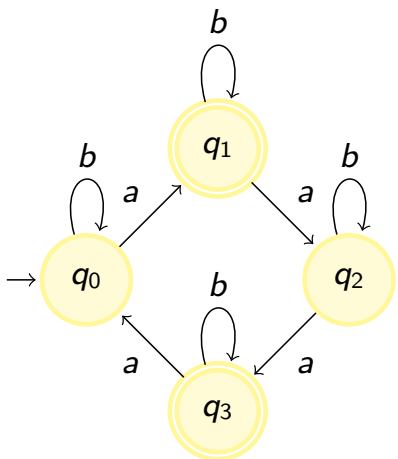
$$\vdash \lambda y.a(by) \geq (q_1 \mapsto q_0) \vee (q_2 \mapsto q_1)$$

Domains in logical form

Step functions

Step functions are a basic block to describe monotone functions, given $a \in \mathcal{D}_1$ and $b \in \mathcal{D}_2$,

$$(a \mapsto b)(x) = \begin{cases} b \text{ when } a \leq x \\ \bot \text{ otherwise} \end{cases}$$



$$\dfrac{\dfrac{\dfrac{}{y \geq q_1 \vdash y \geq q_1}}{\vdash \lambda y.a(by) \geq q_1 \mapsto q_0} \quad \dfrac{\dfrac{}{y \geq q_2 \vdash y \geq q_2}}{\vdash \lambda y.a(by) \geq q_2 \mapsto q_1}}{\vdash \lambda y.a(by) \geq (q_1 \mapsto q_0) \vee (q_2 \mapsto q_1)}$$

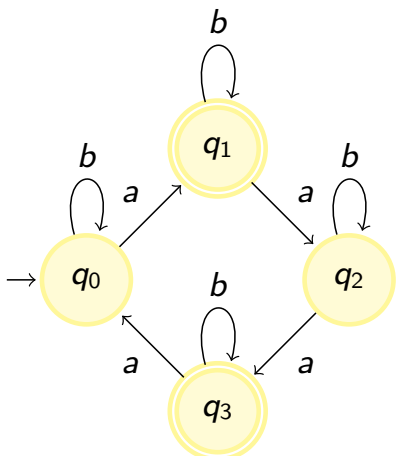
$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{f \geq (q_1 \mapsto q_0) \vee (q_2 \mapsto q_1), x \geq q_2 \vdash x \geq q_2}}{f \geq (q_1 \mapsto q_0) \vee (q_2 \mapsto q_1), x \geq q_2 \vdash (fx) \geq q_1}}{f \geq (q_1 \mapsto q_0) \vee (q_2 \mapsto q_1), x \geq q_2 \vdash b(fx) \geq q_1}}{f \geq (q_1 \mapsto q_0) \vee (q_2 \mapsto q_1), x \geq q_2 \vdash f(b(fx)) \geq q_0}}{\vdash \lambda fx.f(b(fx)) \geq ((q_1 \mapsto q_0) \vee (q_2 \mapsto q_1)) \mapsto q_2 \mapsto q_0}$$

Domains in logical form

Step functions

Step functions are a basic block to describe monotone functions, given $a \in \mathcal{D}_1$ and $b \in \mathcal{D}_2$,

$$(a \mapsto b)(x) = \begin{cases} b \text{ when } a \leq x \\ \bot \text{ otherwise} \end{cases}$$



So we have $(\lambda fx.f(b(fx)))(\lambda y.a(by)) \geq q_2 \mapsto q_0$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{y \geq q_1 \vdash y \geq q_1}}{\vdash \lambda y.a(by) \geq q_1 \mapsto q_0} \quad \dfrac{\dfrac{}{y \geq q_2 \vdash y \geq q_2}}{\vdash \lambda y.a(by) \geq q_2 \mapsto q_1}}{\vdash \lambda y.a(by) \geq (q_1 \mapsto q_0) \vee (q_2 \mapsto q_1)}}{f \geq (q_1 \mapsto q_0) \vee (q_2 \mapsto q_1), x \geq q_2 \vdash x \geq q_2}}{f \geq (q_1 \mapsto q_0) \vee (q_2 \mapsto q_1), x \geq q_2 \vdash (fx) \geq q_1}}{f \geq (q_1 \mapsto q_0) \vee (q_2 \mapsto q_1), x \geq q_2 \vdash b(fx) \geq q_1}}{f \geq (q_1 \mapsto q_0) \vee (q_2 \mapsto q_1), x \geq q_2 \vdash f(b(fx)) \geq q_0}}{\vdash \lambda fx.f(b(fx)) \geq ((q_1 \mapsto q_0) \vee (q_2 \mapsto q_1)) \mapsto q_2 \mapsto q_0}$$

More refined finitary semantics

With more refined modes of λ-calculus we can obtain some further results:

- linear functions (functions that commutes with joins) model non-duplicating λ-terms and allow for state of the art parsing algorithms [Kanazawa 07,11 Bourreau et al. 11,14],

More refined finitary semantics

With more refined modes of λ-calculus we can obtain some further results:

- linear functions (functions that commutes with joins) model non-duplicating λ-terms and allow for state of the art parsing algorithms [Kanazawa 07,11 Bourreau et al. 11,14],
- stable functions [Berry 78]/coherence spaces [Girard 87] allow to remove deletion from grammars [S. 15],

More refined finitary semantics

With more refined modes of λ-calculus we can obtain some further results:

- linear functions (functions that commutes with joins) model non-duplicating λ-terms and allow for state of the art parsing algorithms [Kanazawa 07,11 Bourreau et al. 11,14],
- stable functions [Berry 78]/coherence spaces [Girard 87] allow to remove deletion from grammars [S. 15],
- join-prime functions give extensions of parsing algorithms for OI context-free grammars [Kobele S. 14],

More refined finitary semantics

With more refined modes of λ-calculus we can obtain some further results:

- linear functions (functions that commutes with joins) model non-duplicating λ-terms and allow for state of the art parsing algorithms [Kanazawa 07,11 Bourreau et al. 11,14],
- stable functions [Berry 78]/coherence spaces [Girard 87] allow to remove deletion from grammars [S. 15],
- join-prime functions give extensions of parsing algorithms for OI context-free grammars [Kobele S. 14],

Intentional models like sequential algorithms[Berry Currien 82] or strongly stable functions [Bucciarelli Ehrhard 91] seem to be good candidates to generalize prefix-correct algorithms.

Linguistic modeling with context-free derivations

Algebraic representation of syntactic structures:

- sticks to the hierarchical organization of syntagms,
- rich interpretations allow for precise control of complex word orders,
- a problem is the explosion of the number of non-terminals when dealing with natural language modeling.

Linguistic modeling with context-free derivations

Algebraic representation of syntactic structures:

- sticks to the hierarchical organization of syntagms,
- rich interpretations allow for precise control of complex word orders,
- a problem is the explosion of the number of non-terminals when dealing with natural language modeling.

Some solution have been proposed for dealing with non-terminal explosions:

- feature structures (LFG, HSPG, feature-TAGs, ...),
- meta-grammars (Perrier, Duchier, Parmentier et al.).

Linguistic modeling with context-free derivations

Algebraic representation of syntactic structures:

- sticks to the hierarchical organization of syntagms,
- rich interpretations allow for precise control of complex word orders,
- a problem is the explosion of the number of non-terminals when dealing with natural language modeling.

Some solution have been proposed for dealing with non-terminal explosions:

- feature structures (LFG, HSPG, feature-TAGs, ...),
- meta-grammars (Perrier, Duchier, Parmentier et al.).

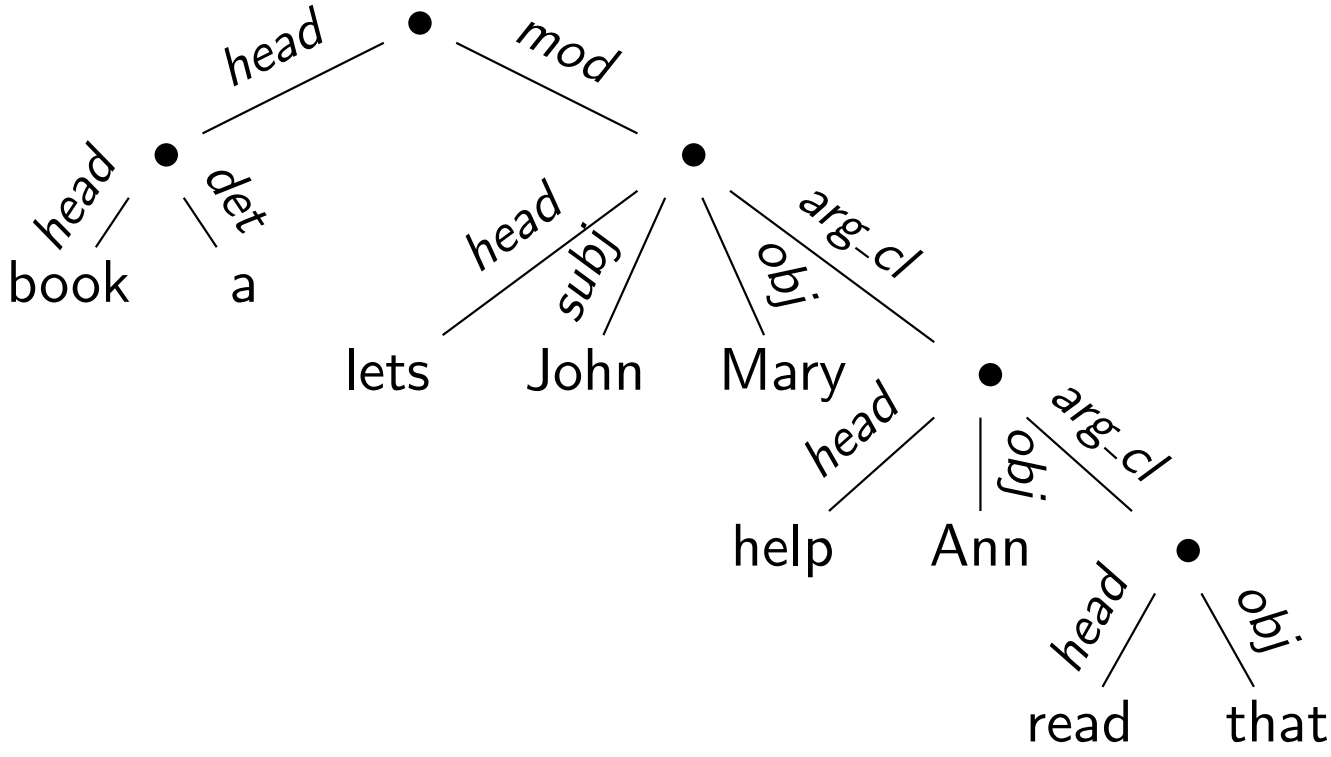
[Clément Kirman S. 15] propose to use the connection between logic and finite state automata, in line with model theoretic syntax [Rogers 96, Moennich et al. 00 ...].

Overview of descriptions

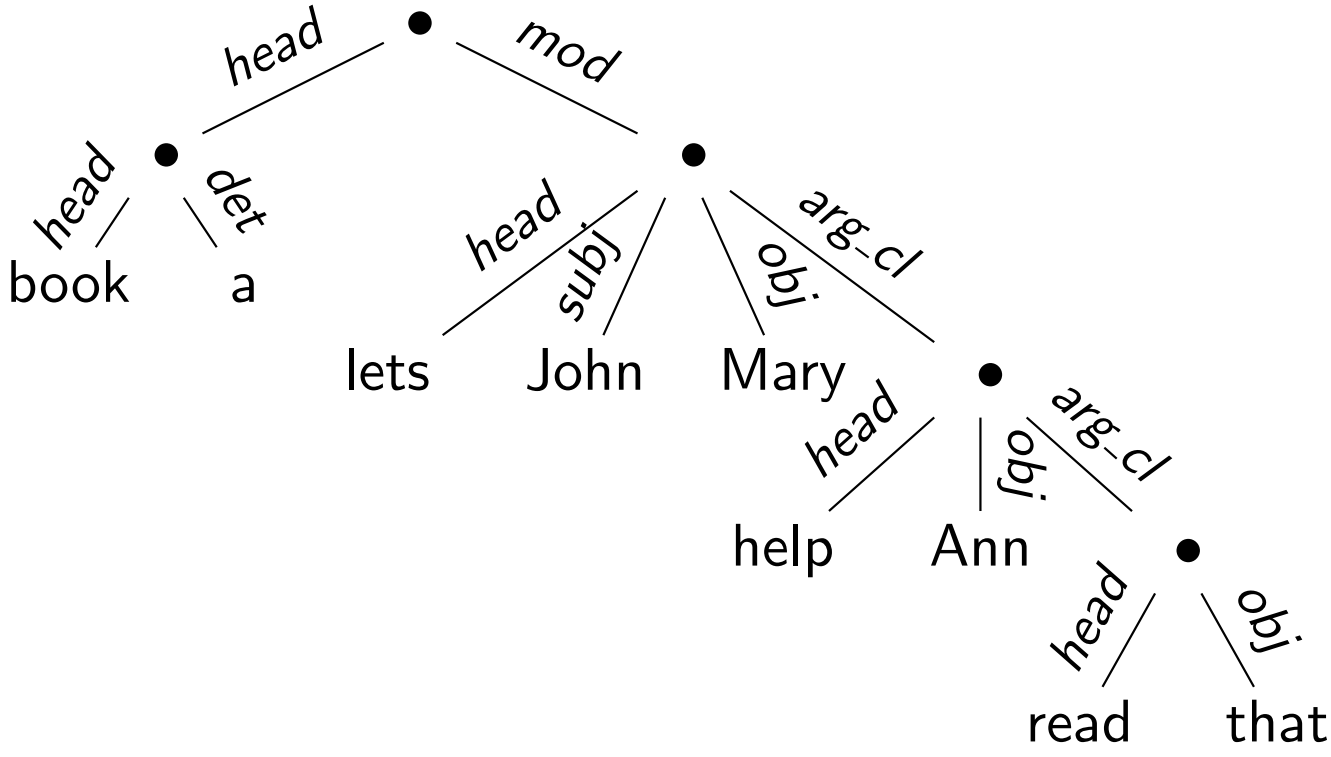
- Definition of syntactic structure:
  - a regular grammar that models the hierarchical structure of syntagms,
  - logical constraints that model relations between words for (agreement, control, wh-movement, island, ...)
- Linearizations of syntactic structures with transduction with logical look-around (to French, Dutch, German, semantics ...)

- The descriptions are concise and modular,
- the compilation of logical constraints to automata guaranty that we do not exceed the generative power of formalisms with context-free derivations.

An example

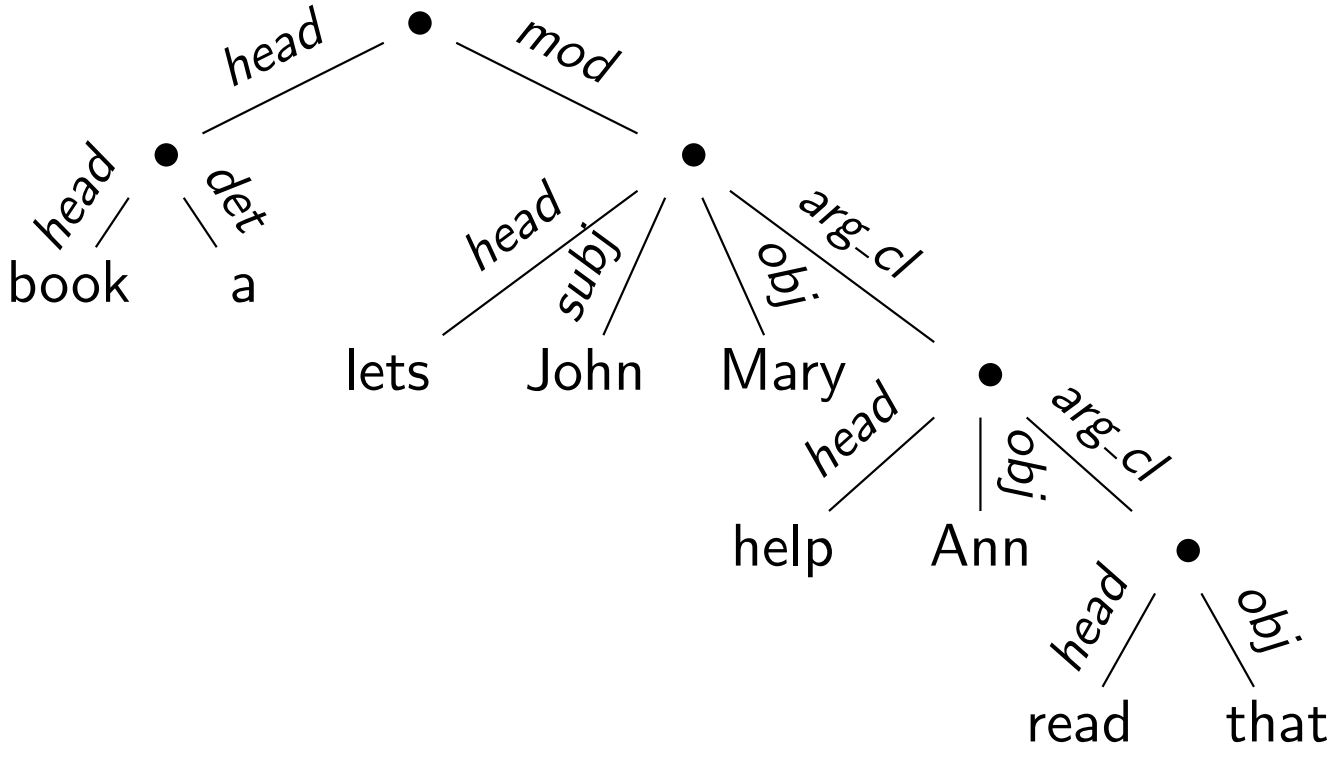

head
mod
head
det
head
subj
obj
arg_cl
head
obj
arg_cl
head
obj
book
a
lets
John
Mary
help
Ann
read
that

An example



$ext\_path(cl, p) := (subj + arg\_cl^* \ obj)(cl, p)$

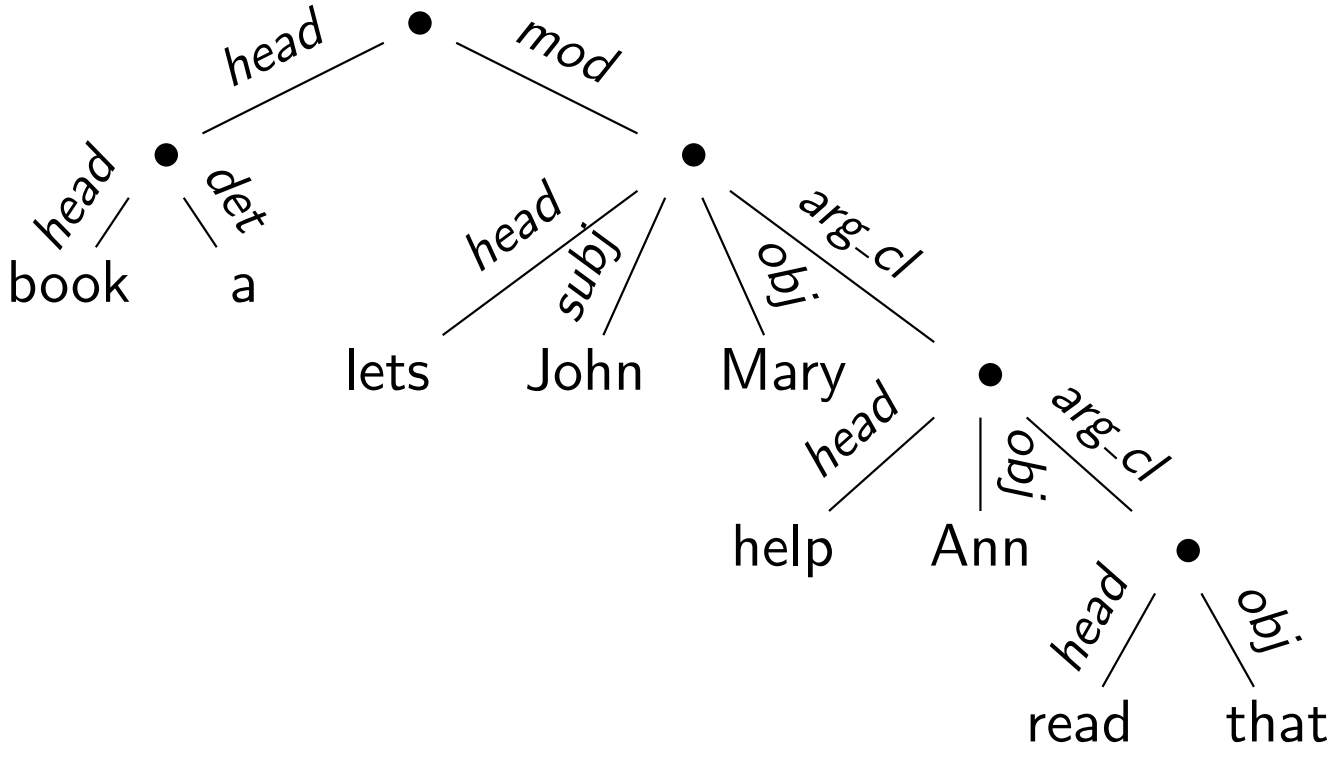
An example



$$ext\_path(cl, p) := (subj + arg\_cl^* \ obj)(cl, p)$$

$$antecedent(ant, pro) := (noun(ant) \vee proper\_noun(ant)) \wedge pro\_rel(pro)$$
$$\wedge \exists rcl.relative(rcl) \wedge head^* \uparrow mod(ant, rcl) \wedge ext\_path(rcl, pro)$$

An example



$ext\_path(cl, p) := (subj + arg\_cl^* \ obj)(cl, p)$

$antecedent(ant, pro) := (noun(ant) \vee proper\_noun(ant)) \wedge pro\_rel(pro)$
$\wedge \exists rcl.relative(rcl) \wedge head^* \uparrow mod(ant, rcl) \wedge ext\_path(rcl, pro)$

`een boek dat Jan Marie Anna laat helpen lezen.`

Thank you

Piece of Bibliography


H. Comon, M. Dauchet, R. Gilleron, D. Lugiez, S. Tison, and M. Tommasi.
Tree Automata Techniques and Applications.
http://www.grappa.univ-lille3.fr/tata/, 1999.


Philippe de Groote.
Towards abstract categorial grammars.
In Association for Computational Linguistic, editor, *Proceedings 39th Annual Meeting and 10th Conference of the European Chapter*, pages 148–155. Morgan Kaufmann Publishers, 2001.


Frank Morawietz.
Two-Step Approaches ot Natural Language Formalisms.
Studies in Generative Grammar. Mouton de Gruyter, Berlin · New York, 2003.


Reinhard 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.


P.-L. Curien R. Amadio.
Domains and lambda-calculi.
Number 46 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1996.