Combinatory Categorial Grammar: a (gentle) introduction

Claudio Delli Bovi

Dipartimento di Informatica

Sapienza Università di Roma

The story so far: syntax

Context-free grammars:

- Terminal symbols (**lexicon**)
- Non-terminal symbols (**phrases/clauses**)
- A set of rules (**productions**)

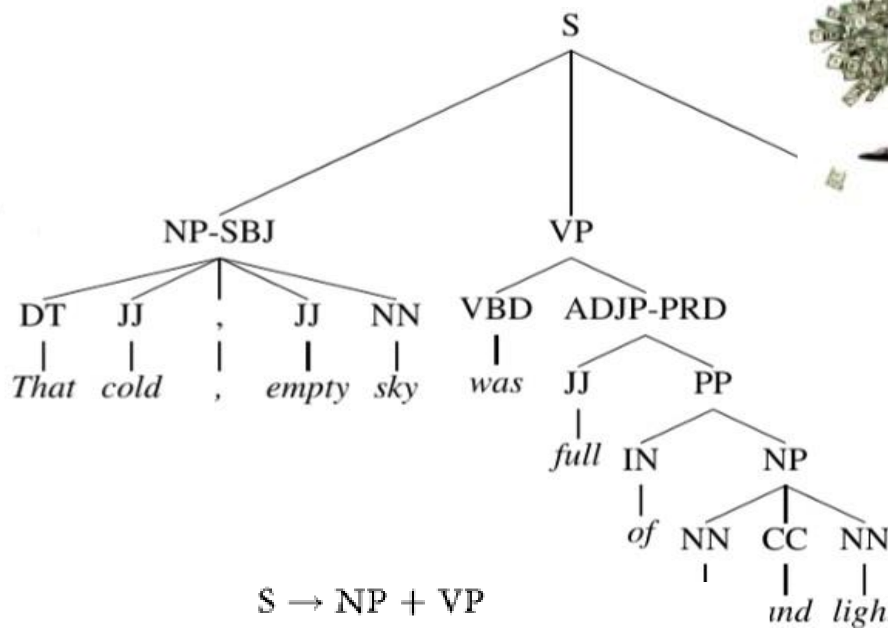
+

Dependency grammars

Treebanks

Parsing: CKY, Earley

...



$S \rightarrow NP + VP$
$NP \rightarrow$ Proper Noun
$VP \rightarrow V + NP$
$VP \rightarrow V + NP + ADJ$
Proper Noun $\rightarrow$ Jack | Jill | . . .
$V \rightarrow$ finds | believes | imagines . . .

The story so far: semantics

Representing concepts and meanings (**senses**):

First Order Logic

λ-calculus formalism

+

Lexical semantics:

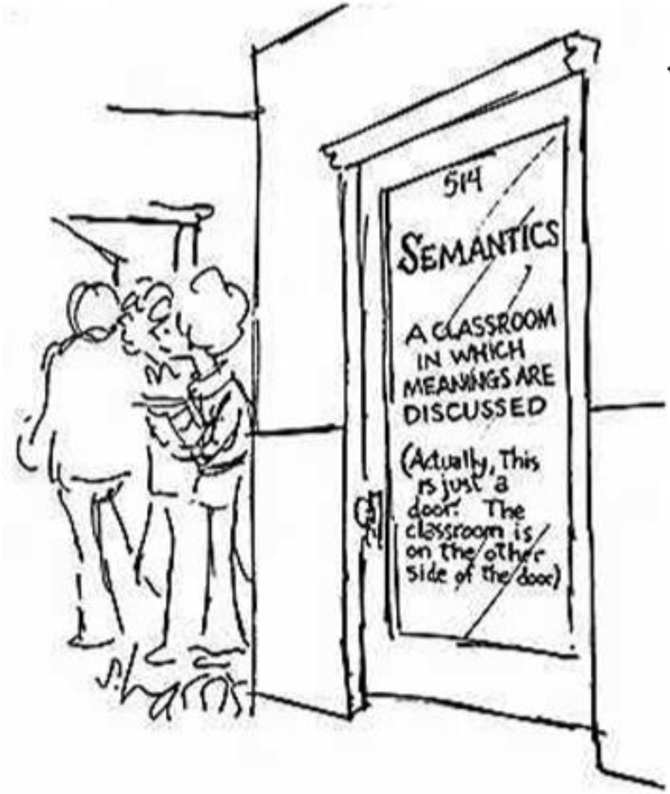
Word senses

Semantic roles

Taxonomies and semantic networks

(**WordNet**, **BabelNet**)

...



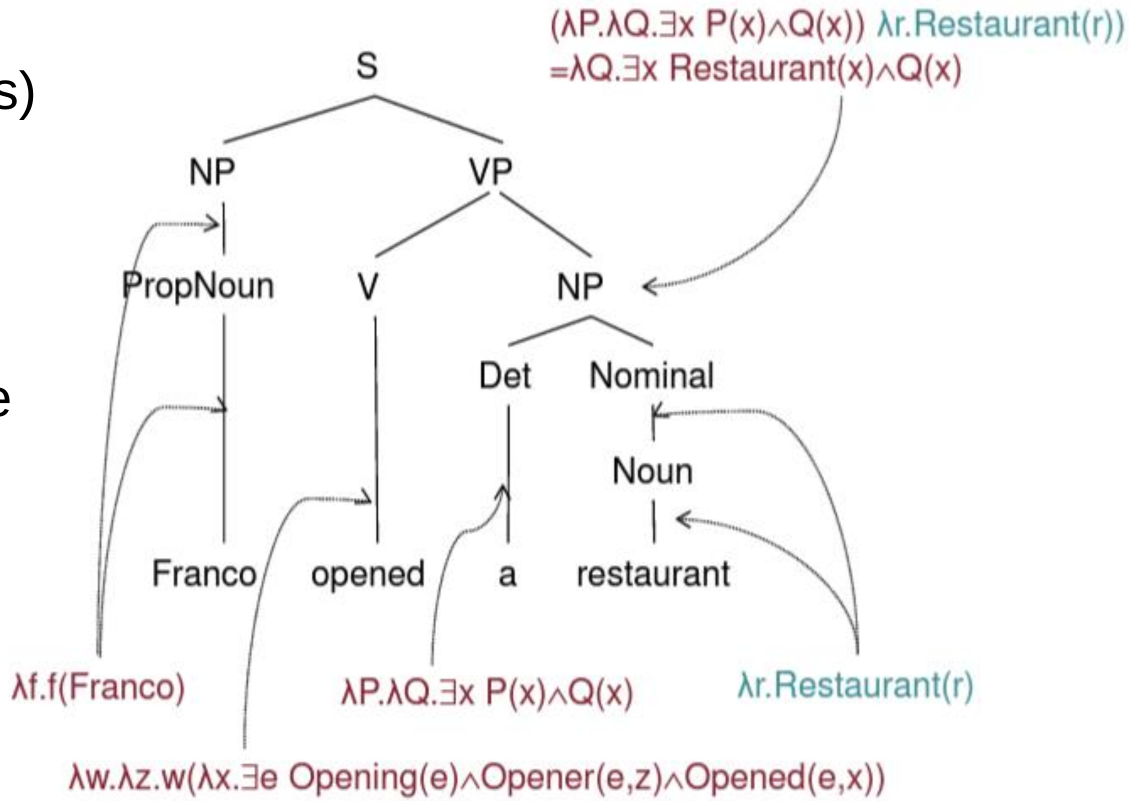
$$\lambda f.(\lambda x.f\ (x\ x))\ (\lambda x.f\ (x\ x))$$

The story so far: semantics

Fine! We have plenty of *formalisms* (FOL, λ-calculus) and a convenient way of representing *word senses* and *lexical relations*.

But how do we work out the meaning of a *sentence*?

- *Parse* the sentence
- Get the semantics for each word
- Proceed *bottom-up*

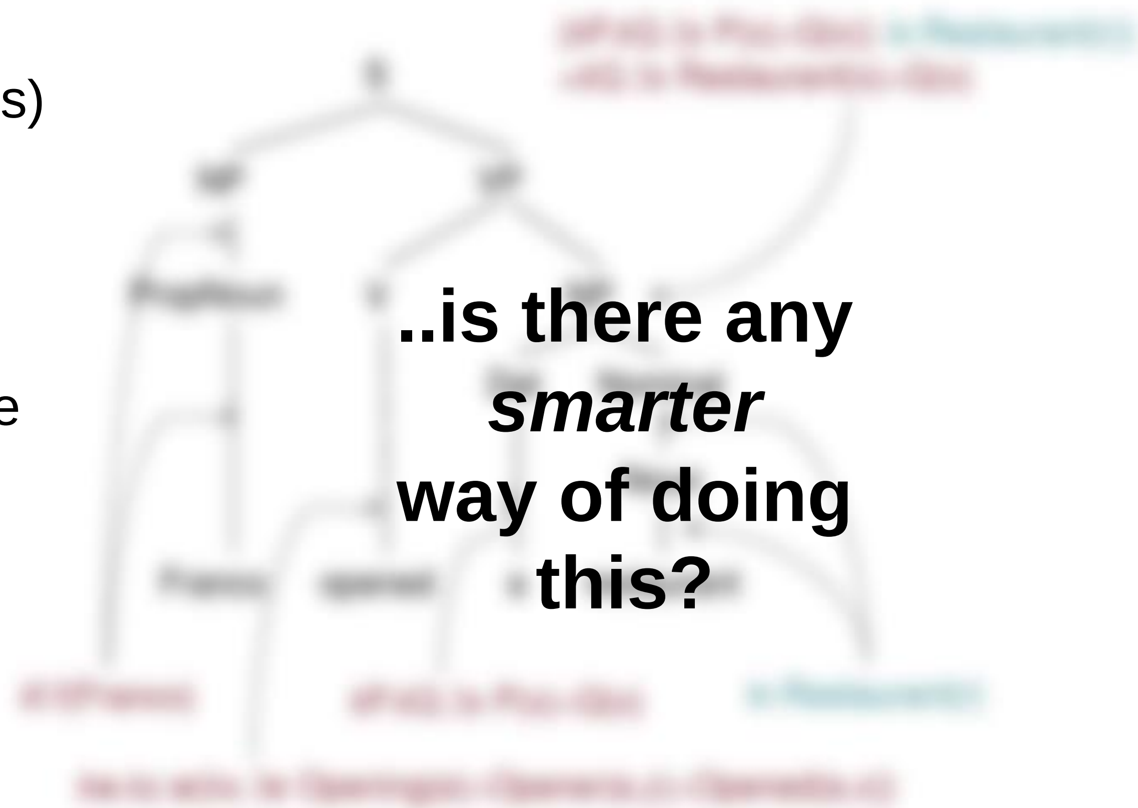


The story so far: semantics

Fine! We have plenty of *formalisms* (FOL, λ-calculus) and a convenient way of representing *word senses* and *lexical relations*.

But how do we work out the meaning of a *sentence*?

- *Parse* the sentence
- Get the semantics for each word
- Proceed *bottom-up*



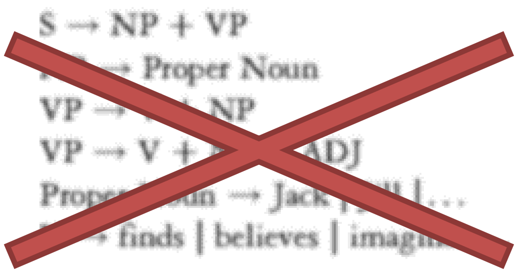
..is there any *smarter* way of doing this?

Categorial Grammar

The term **Categorial Grammar** names a group of theories of natural language syntax and semantics in which the main responsibility for defining syntactic form is borne by the lexicon. (M. Steedman)

Categorial Grammars (CGs) developed as an alternative approach to CFGs.

They capture the same information by associating a *functional type*, or **category**, with all grammatical entities.



Categorial Grammar: Lexicon

In CGs *lexical entries for words contain all language-specific information*. For each word, the associated lexical entry contains:

- a syntactic category, to determine which other categories the word may combine with;
- a semantic interpretation, which defines the related semantics.

For instance, a possible entry in the lexicon could look like:

$$\boxed{\mathit{write}} \vdash \boxed{(S\backslash \mathrm{NP})/\mathrm{NP}\ \ \lambda y.\lambda x.\mathrm{write}'(x, y)}$$

Lexeme Category

Categorial Grammar: Lexicon

Lexeme | | Category |
--- | --- | --- | ---
write | ⊢ | $(S\backslash NP)/NP$ | λy.λx.write‘(x, y)
 | | Syntax | Semantics

The so-called *Lambek notation* (arguments under slash) reads like this:

- **A/B** = "give me a B to my **right**, then I'll give you an A"
- **A\B** = "give me a B to my **left**, then I'll give you an A"

λ-calculus expression paired with the syntactic type: *syntactic* and *semantic* information captured *jointly*

Categorial Grammar: Lexicon

A few examples:

- S\NP : $\lambda x.f(x)$ intransitive verb

Categorial Grammar: Lexicon

A few examples:

- S\NP : $\lambda x.f(x)$ — intransitive verb
- (S\NP)/NP : $\lambda x.\lambda y.f(x, y)$ — transitive verb

Categorial Grammar: Lexicon

A few examples:

- S\NP : $\lambda x.f(x)$ — intransitive verb
- (S\NP)/NP : $\lambda x.\lambda y.f(x, y)$ — transitive verb
- ((S\NP)/NP)/NP : $\lambda x.\lambda y.\lambda z.f(x, y, z)$ — ditransitive verb

Categorial Grammar: Lexicon

A few examples:

- $S\backslash NP \;:\; \lambda x.f(x)$ — intransitive verb
- $(S\backslash NP)/NP : \lambda x.\lambda y.f(x, y)$ — transitive verb
- $((S\backslash NP)/NP)/NP : \lambda x.\lambda y.\lambda z.f(x, y, z)$ — ditransitive verb
- $(S\backslash NP)/(S\backslash NP) : \lambda g.\lambda x.f(g\ x)$ — adverb/modal

Categorial Grammar: Lexicon

A few examples:

- S\NP : $\lambda x.f(x)$ — intransitive verb
- (S\NP)/NP : $\lambda x.\lambda y.f(x, y)$ — transitive verb
- ((S\NP)/NP)/NP : $\lambda x.\lambda y.\lambda z.f(x, y, z)$ — ditransitive verb
- (S\NP)/(S\NP) : $\lambda g.\lambda x.f(g\ x)$ — adverb/modal
- S/(S\NP) : $\lambda f.f(x)$ — subjective pronoun

CG parsing: a toy example

CCG   is   fun

CG parsing: a toy example

$$\frac{\text{CCG}}{\begin{matrix} NP \\ CCG \end{matrix}} \quad \frac{\text{is}}{\begin{matrix} S\backslash NP/ADJ \\ \lambda f.\lambda x.f(x) \end{matrix}} \quad \frac{\text{fun}}{\begin{matrix} ADJ \\ \lambda x.fun(x) \end{matrix}}$$

$$\text{is} \;\vdash (S\backslash NP)/ADJ : \lambda f.\lambda x.f(x)$$

Composite categories

$$\vdots$$

$$\text{fun} \;\vdash ADJ : \lambda x.fun(x)$$

$$\vdots$$

$$\text{CCG} \;\vdash NP : CCG$$

Primitive symbols

First, use the *lexicon* to match words with their categories

CG parsing: a toy example

$$
\begin{array}{ccc}
\text{CCG} & \text{is} & \text{fun} \\
\hline
NP & S\backslash NP/ADJ & ADJ \\
CCG & \lambda f.\lambda x.f(x) & \lambda x.fun(x) \\
 & \multicolumn{2}{c}{\xrightarrow{\hspace{6em}} >} \\
 & \multicolumn{2}{c}{S\backslash NP} \\
 & \multicolumn{2}{c}{\lambda x.fun(x)}
\end{array}
$$

Forward Function Application:

A/B: f **B:** a $\Rightarrow$ **A:** f(a)

CG parsing: a toy example

$$
\begin{array}{ccc}
\text{CCG} & \text{is} & \text{fun} \\
\hline
NP & S\backslash NP/ADJ & ADJ \\
CCG & \lambda f.\lambda x.f(x) & \lambda x.fun(x) \\
\end{array}
$$

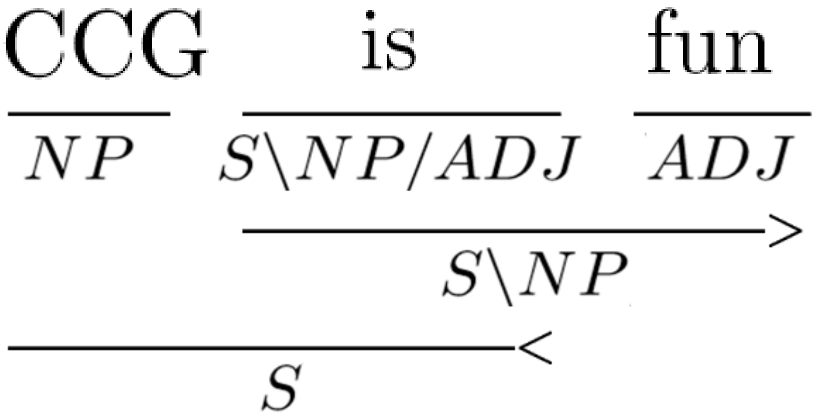
$$
\frac{S\backslash NP/ADJ \quad ADJ}{\begin{array}{c} S\backslash NP \\ \lambda x.fun(x) \end{array}} >
$$

$$
\frac{NP \quad S\backslash NP}{\begin{array}{c} S \\ fun(CCG) \end{array}} <
$$

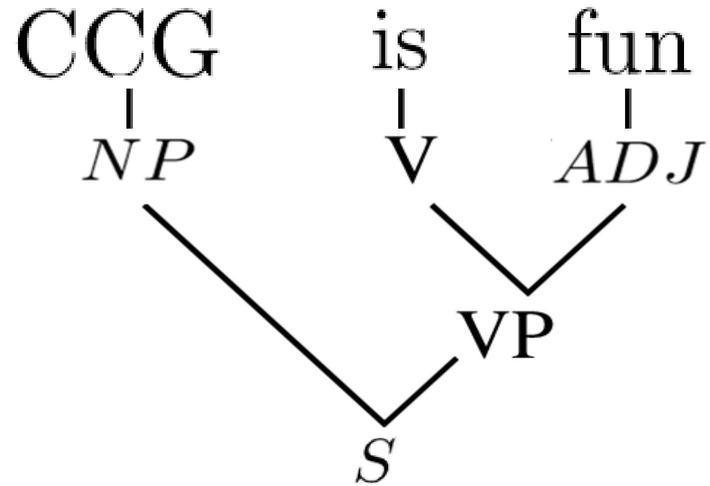
Backward Function Application:

B: a **A\B:** f $\Rightarrow$ **A:** f(a)

CGs vs CFGs



CG parse



CFG parse

[…] “pure” categorial grammar limited to these two rules alone is essentially context-free grammar written in the *accepting*, rather than the *producing*, direction.
(M. Steedman)

CGs vs CFGs

	CFGs	CGs
Combination operations	Many	Few
Parse tree nodes	Non-terminals	Categories
Syntactic symbols	Few dozen	Handful, but can combine
Paired with words	POS tags	Categories

From CGs to CCGs: is this all?

Fair enough: we now have a new fancy way of writing syntax that somehow look more compact and naturally tied up with *semantics*.

However, we didn't move any further from CFGs in terms of *expressiveness*. Both CGs and CFGs are not powerful enough to capture some linguistic phenomena, such as

- **Object relative clauses:** *[..] the man that Ed saw.*
- **Right-node raising:** *Ed saw and Ned heard Ted.*
- Long-distance relativization, parasitic gaps, argument cluster coordination…

From CGs to CCGs: CCG to the rescue

- **Combinatory Categorial Grammar** (Steedman 1996, 2000)

  CCG is sometimes characterized as the ‘rule-based’ extension of CG’s Lambek system. Roughly speaking, by adding to it more rules that *implicitly* reflect the logical properties of slashes, such as Type Raising or Function Composition, you get CCG.

[Steedman 1996; 2000; 2011; Granroth and Steedman 2012]

CCG: What's new?

- **Composition:**

(>**B**) $\quad$ **A/B:** f $\quad$ **B/C:** g $\quad \Rightarrow \quad$ **A/C:** $\lambda x.f(g(x))$

(<**B**) $\quad$ **B\C:** g $\quad$ **A\B:** f $\quad \Rightarrow \quad$ **A/C:** $\lambda x.f(g(x))$

Equivalent to *function composition*: functional types can compose if the *domain* of one corresponds to the *range* of the other. The result is a new functional type with the range of the first and the domain of the second.

Works in both directions (*forward* and *backward*)

CCG: What's new?

- **Type Raising:**

(>**T**)	**X:** x	$\Rightarrow$	**T/(T\X):** $\lambda f.f(x)$
(<**T**)	**X:** x	$\Rightarrow$	**T\(X/T):** $\lambda f.f(x)$

Used to convert *elementary* types to *functional* types ("*turn arguments into functions over functions-over-such arguments*"), e.g.

birds := **NP** $\Rightarrow$ **S/(S\NP)**

Again, works in both directions (*forward* and *backward*)

CCG: What's new?

Type Raising and Composition rules are often applied together.
For instance:

$$\frac{\text{birds}}{NP} \qquad \frac{\text{like}}{(S\backslash NP)/NP} \qquad \frac{\text{bugs}}{NP}$$

CCG: What's new?

Type Raising and Composition rules are often applied together.
For instance:

$$\dfrac{\dfrac{\text{birds}}{NP}}{S/(S\backslash NP)}{>}\mathbf{T} \qquad \dfrac{\text{like}}{(S\backslash NP)/NP} \qquad \dfrac{\text{bugs}}{NP}$$

Forward Type Raising:

X: x ⇒ **T/(X\T):** λf.f(x)

CCG: What's new?

Type Raising and Composition rules are often applied together.
For instance:

$$\cfrac{\cfrac{\cfrac{\text{birds}}{NP}}{S/(S\backslash NP)}{>}\mathbf{T} \qquad \cfrac{\text{like}}{(S\backslash NP)/NP}}{S/NP}{>}\mathbf{B} \qquad \cfrac{\text{bugs}}{NP}$$

Forward Composition:

A/B: f **B/C:** g $\Rightarrow$ **A/C:** λx.f(g(x))

CCG: What's new?

Type Raising and Composition rules are often applied together.
For instance:

$$\dfrac{\text{birds}}{NP} \qquad \dfrac{\text{like}}{(S\backslash NP)/NP} \qquad \dfrac{\text{bugs}}{NP}$$

$$\dfrac{NP}{S/(S\backslash NP)} >\mathbf{T}$$

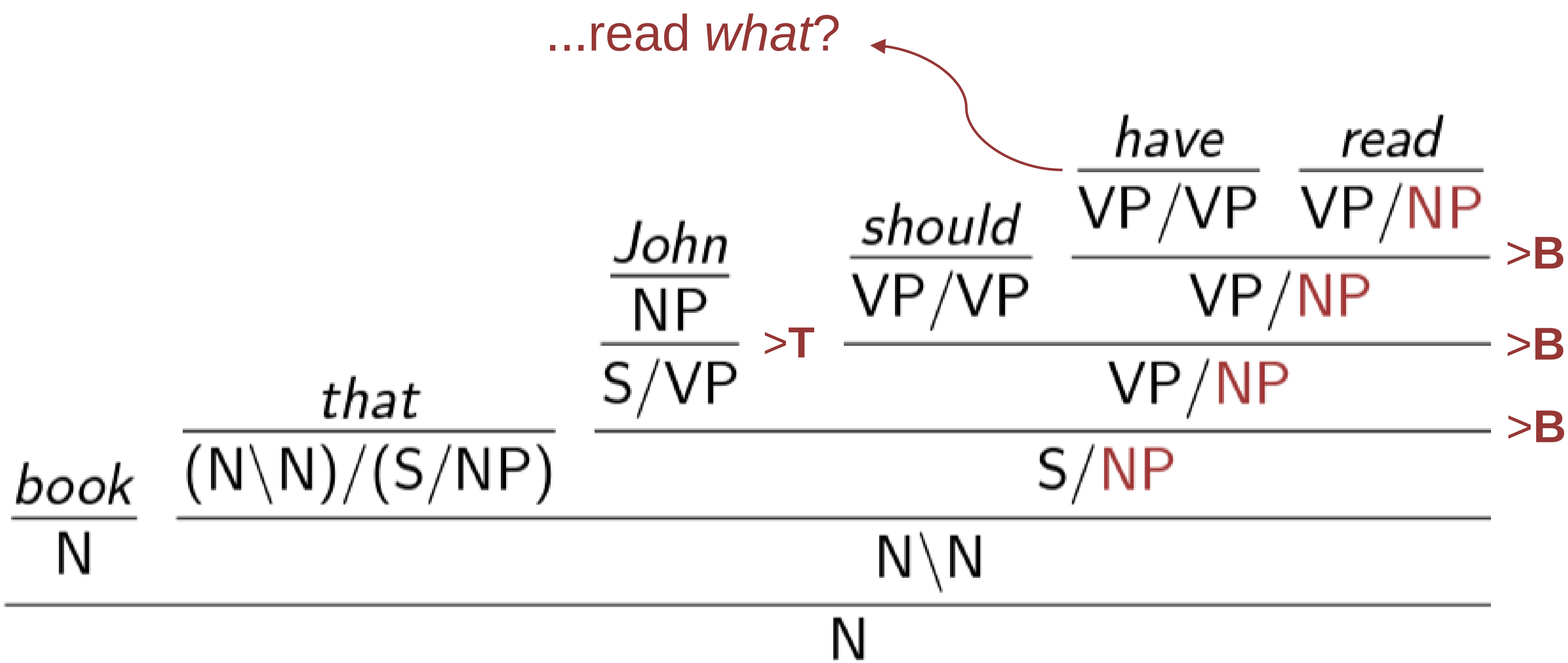
$$\dfrac{S/(S\backslash NP) \qquad (S\backslash NP)/NP}{S/NP} >\mathbf{B}$$

$$\dfrac{S/NP \qquad NP}{S} >$$

Forward Function Application (see previous slides)

CCG: What's new?

Despite the introduction of (even more) ambiguity in the parse, the new rules are useful for dealing with *long-distance dependencies*. Look at this:



CCG: What's new?

- **A (first) special case: coordination**

$$\textit{and} \quad \coloneqq \quad \mathbf{(X\backslash X)/X:} \ \lambda f.\lambda g.\lambda x.(f(x) \wedge g(x))$$

Coordination is handled by specific rules: related operators (e.g. conjunctions) have special lexical entries.

CCG: What's new?

- **A (first) special case: coordination**

and $\coloneqq$ **(X\X)/X:** $\lambda f.\lambda g.\lambda x.(f(x) \wedge g(x))$

Coordination is handled by specific rules: related operators (e.g. conjunctions) have special lexical entries.

- **A (second) special case: quantifiers**

every $\coloneqq$ **(S/(S\NP))/N:** $\lambda f.\lambda g.(\forall x\ f(x) \rightarrow g(x))$

Quantifiers are entered in the lexicon directly in the raised type.

CCG: What's new?

Every girl laughed.

$\lambda f.\lambda g.(\forall x\ f(x) \rightarrow g(x))$

$\lambda x.girl(x)$

$\lambda x.laugh(x)$

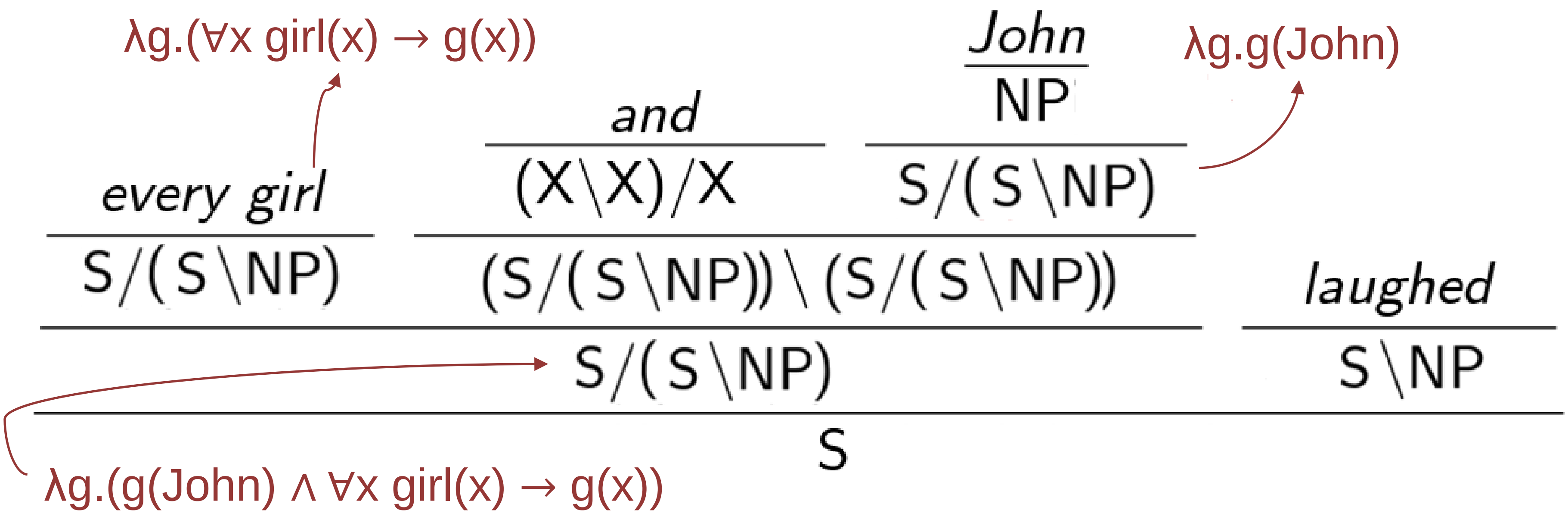
$$\dfrac{\dfrac{\dfrac{\textit{every}}{(S/(S\backslash NP))/N} \quad \dfrac{\textit{girl}}{N}}{S/(S\backslash NP)} \quad \dfrac{\textit{laughed}}{S\backslash NP}}{S}$$

$\lambda g.(\forall x\ girl(x) \rightarrow g(x))$

$\boldsymbol{\forall x\ girl(x) \rightarrow laugh(x)}$

CCG: What's new?

Every girl and John laughed.



laugh(John) ∧ ∀x girl(x) → laugh(x)

Parsing with CCGs: the problem

'Spurious' ambiguity: with our new rules, for each *derived structure* of a sentence, there can be *many* derivations leading to that structure.

Parsing with CCGs: the problem

‘Spurious’ ambiguity: with our new rules, for each *derived structure* of a sentence, there can be *many* derivations leading to that structure.

+

Syntax-only CCG parsing has polynomial time CKY-style algorithms, but parsing with *semantics* requires entire categories as chart signatures (e.g. *fun* := **ADJ λx.fun(x)**).

Parsing with CCGs: the problem

‘Spurious’ ambiguity: with our new rules, for each *derived structure* of a sentence, there can be *many* derivations leading to that structure.

+

Syntax-only CCG parsing has polynomial time CKY-style algorithms, but parsing with *semantics* requires entire categories as chart signatures (e.g. *fun* := **ADJ λx.fun(x)**).

+

Inherent *lexical* and *grammatical* ambiguities of language

Parsing with CCGs: the problem

'Spurious' ambiguity: with our new rules, for each *derived structure* of a sentence, there can be *many* derivations leading to that structure.

+

Syntax-only CCG parsing has polynomial time CKY-style algorithms, but parsing with *semantics* requires entire categories as chart signatures (e.g. *fun* := **ADJ λx.fun(x)**).

+

Inherent *lexical* and *grammatical* ambiguities of language

=

CCG parsing is a tough task!

Parsing with CCGs: (some) solutions

Many approaches have been tried so far:

- **Generative models** over *normal-form derivations* (Hockenmaier, 2001; Hockenmaier and Steedman, 2002);

Parsing with CCGs: (some) solutions

Many approaches have been tried so far:

- **Generative models** over *normal-form derivations* (Hockenmaier, 2001; Hockenmaier and Steedman, 2002);
- **Conditional models** over *dependency structures* (Clark et al., 2002), that is *CCG categories* + *word-word dependencies*;

Parsing with CCGs: (some) solutions

Many approaches have been tried so far:

- **Generative models** over *normal-form derivations* (Hockenmaier, 2001; Hockenmaier and Steedman, 2002);
- **Conditional models** over *dependency structures* (Clark et al., 2002), that is *CCG categories* + *word-to-word dependencies*;
- **Log-linear models** (Clark and Curran, 2003) capturing information from both dependencies and derivations:

$$P(\pi \mid S) = \sum_{d \in \Delta(\pi)} P(d, \pi \mid S)$$

conditional probability of the parse π given the sentence S

Δ(π): set of al possible CCG derivations of S

conditional probability of the parse π **and** the dependency structure *d*

Parsing with CCGs: log-linear model

The log-linear model (or **maximum entropy model**) looks like this:

$$P(\pi \mid S) = \frac{1}{Z_S} e^{\sum_i \lambda_i f_i(\pi)}$$

conditional probability
of the parse π
given the sentence S

Normalization factor

features of the parse: any
real-valued function over
the space of parses Π

- **Packed charts**: compact representation of a very large number of CCG derivations (retrieve the highest scoring parse or dependency structure without enumerating all derivations)

- The *derivation space* Δ(π) could be huge! CCG produces an extremely large number of parses: we need a way of *limiting* them.

Parsing with CCGs: supertagger

CCG parsing is best viewed as a *two-stage* process:

- first, assign *lexical categories* to the words in the sentence (**supertagging**)**;**
- then *combine the categories* together using the rules we already know (**parsing**)**.**

Parsing with CCGs: supertagger

CCG parsing is best viewed as a *two-stage* process:

- first, assign *lexical categories* to the words in the sentence (**supertagging**);
- then *combine the categories* together using the rules we already know (**parsing**).

The trivial (but stupid) way: simply assigning to each word all categories from the word's entry in the lexicon.

Parsing with CCGs: supertagger

CCG parsing is best viewed as a *two-stage* process:

- first, assign *lexical categories* to the words in the sentence (**supertagging**);
- then *combine the categories* together using the rules we already know (**parsing**).

The trivial (but stupid) way: simply assigning to each word all categories from the word's entry in the lexicon.

The complex (but smarter) way: try to guess the most likely category (or categories) given the word's context.

Log-linear supertagger (Clark and Curran, 2004) + parser is an order of magnitude faster than comparable systems!

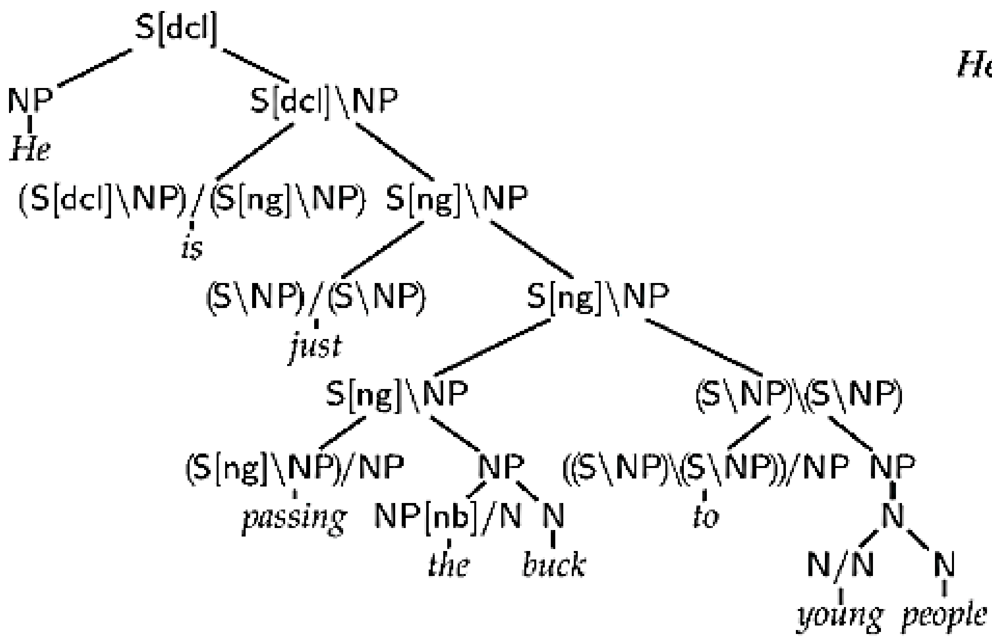
What about the training data?

CCGbank: a corpus of *CCG derivations* and *dependency structures* (Hockenmeier and Steedman, 2003) directly translated from Penn Treebank and suitable for training CCG-based systems.

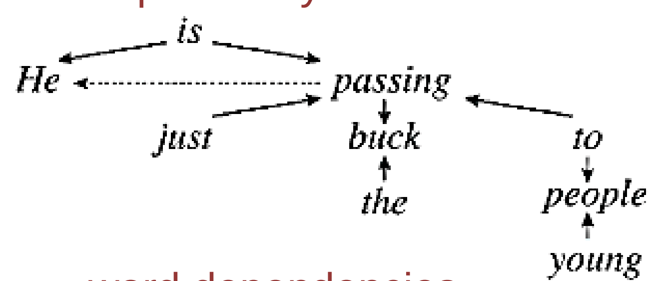
What about the training data?

CCGbank: a corpus of *CCG derivations* and *dependency structures* (Hockenmeier and Steedman, 2003) directly translated from Penn Treebank and suitable for training CCG-based systems.

CCG derivation tree



dependency tree



word dependencies

$\langle \textit{is}, (\mathrm{S[dcl]}\backslash\mathbf{NP_1})/(\mathrm{S[ng]}\backslash\mathrm{NP})_2, 1, \textit{He}\rangle,$

$\langle \textit{is}, (\mathrm{S[dcl]}\backslash\mathrm{NP_1})/(\mathbf{S[ng]}\backslash\mathbf{NP})_2, 2, \textit{passing}\rangle,$

$\langle \textit{just}, (\mathrm{S}\backslash\mathrm{NP_1})/(\mathbf{S}\backslash\mathbf{NP})_2, 2, \textit{passing}\rangle,$

$\langle \textit{passing}, (\mathrm{S[ng]}\backslash\mathbf{NP_1})/\mathrm{NP_2}, 1, \textit{He}\rangle,$

$\langle \textit{passing}, (\mathrm{S[ng]}\backslash\mathrm{NP_1})/\mathbf{NP_2}, 2, \textit{buck}\rangle,$

$\langle \textit{the}, \mathrm{NP[nb]}/\mathbf{N_1}, 1, \textit{buck}\rangle,$

$\langle \textit{to}, ((\mathrm{S}\backslash\mathrm{NP_1})/(\mathbf{S}\backslash\mathbf{NP})_2)/\mathrm{NP_3}, 2, \textit{passing}\rangle,$

$\langle \textit{to}, ((\mathrm{S}\backslash\mathrm{NP_1})/(\mathrm{S}\backslash\mathrm{NP})_2)/\mathbf{NP_3}, 3, \textit{people}\rangle,$

$\langle \textit{young}, \mathrm{N}/\mathbf{N_1}, 1, \textit{people}\rangle$

From Treebanks to CCGbanks

Available from the Linguistic Data Consortium (LDC):

```
{S[dcl] {S[dcl] {NP {NP {NP {NP {N {N/N  Pierre}
                                   {N  Vinken}}}
                            {,  ,}}
                        {NP\NP {S[adj]\NP {NP {N {N/N  61}
                                                 {N  years}}}
                                          {(S[adj]\NP)\NP  old}}}}
                   {,  ,}}
               {S[dcl]\NP {(S[dcl]\NP)/(S[b]\NP)  will}
                          {S[b]\NP {S[b]\NP {(S[b]\NP)/PP {((S[b]\NP)/PP)/NP  join}
                                                          {NP {NP[nb]/N  the}
                                                              {N  board}}}
                                            {PP {PP/NP  as}
                                                {NP {NP[nb]/N  a}
                                                    {N {N/N  nonexecutive}
                                                       {N  director}}}}}
                                   {(S\NP)\(S\NP) {((S\NP)\(S\NP))/N  Nov.}
                                                  {N  29}}}}}}
```

CCG derivation tree

Dependencies

Pierre	**(N/N)**	Vinken
61	**(N/N)**	years
old	**((S[adj]\NP)\NP)**	*Vinken* years
will	**((S[dcl]\NP)/(S[b]\NP))**	Vinken join
join	**(((S[b]\NP)/PP)/NP)**	*Vinken* as board
the	**(NP[nb]/N)**	board
as	**(PP/NP)**	director
a	**(NP[nb]/N)**	director
nonexecutive	**(N/N)**	director
Nov.	**(((S\NP)\(S\NP))/N)**	join 29

CCGbank — File wsj_0001.html

Further development: multi-modal CCG

CCG composition rules are all order-preserving: *cannot* derive sequences of expressions in which function categories are not *immediately* adjacent to their arguments.

Ed often sees his friend Ted.

- ✓ Fine: adverb is (S\NP)/(S\NP)

Ed saw today his friend Ted.

- × Heavy NP-shift: cannot derive this!

Further development: multi-modal CCG

CCG composition rules are all order-preserving: *cannot* derive sequences of expressions in which function categories are not *immediately* adjacent to their arguments.

Ed often sees his friend Ted.

✓ Fine: adverb is (S\NP)/(S\NP)

Ed saw today his friend Ted.

× Heavy NP-shift: cannot derive this!

Need some rule to allow permutations over arguments...
For instance:

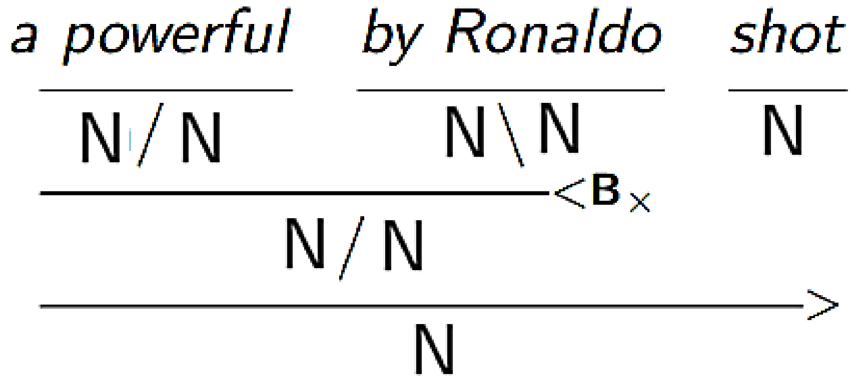
($>\mathbf{B}_x$)	$\mathbf{A/B \quad B\backslash C \Rightarrow A\backslash C}$	Forward crossed composition
($<\mathbf{B}_x$)	$\mathbf{B\backslash C \quad A/B \Rightarrow A/C}$	Backward crossed composition

Further development: multi-modal CCG

Problem: rules are now a bit *too loose*! We could derive something like:

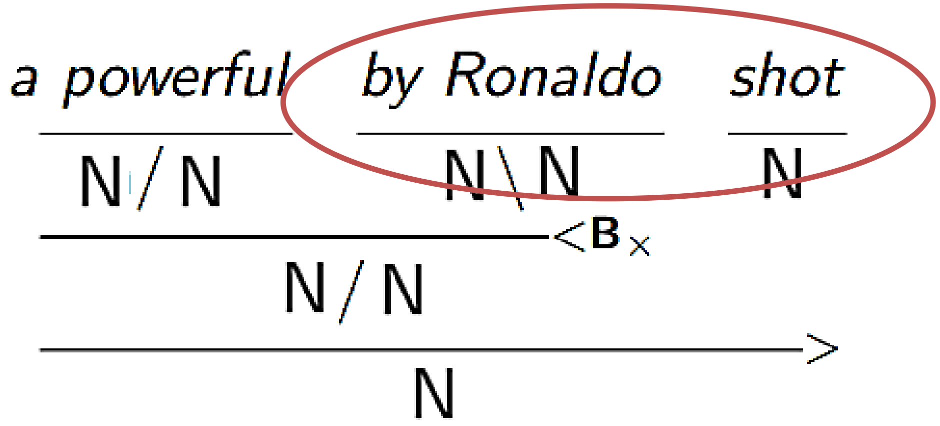
Further development: multi-modal CCG

Problem: rules are now a bit *too loose*! We could derive something like:



Further development: multi-modal CCG

Problem: rules are now a bit *too loose*! We could derive something like:



permutation not allowed!

Solution #1: Devise some language-specific rules for cases where such scrambling operations are allowed.

- Somehow unattractive: we would like to have a grammar where combinatory rules are *universal* (remember Steedman's words about CGs?)

Further development: multi-modal CCG

Solution #2: Create *modalized rules* (**multi-modal CCG**: Baldridge and Kruijff, 2003) by introducing typed slashes:

Further development: multi-modal CCG

Solution #2: Create *modalized rules* (**multi-modal CCG**: Baldridge and Kruijff, 2003) by introducing typed slashes:

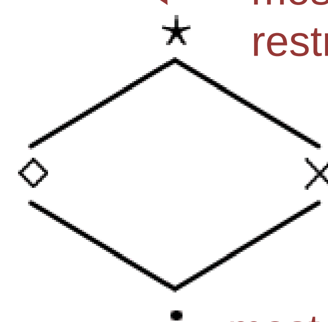
- $/_{\star}$ non-associative, non-commutative
  (‘old’ uni-modal slash)
- $/_{\diamond}$ associative, non-commutative
- $/_{\times}$ non-associative, commutative
- $/_{\cdot}$ associative, commutative

Further development: multi-modal CCG

Solution #2: Create *modalized rules* (**multi-modal CCG**: Baldridge and Kruijff, 2003) by introducing typed slashes:

- $/_\star$ non-associative, non-commutative (‘old’ uni-modal slash)
- $/_\diamond$ associative, non-commutative
- $/_\times$ non-associative, commutative
- $/_\cdot$ associative, commutative

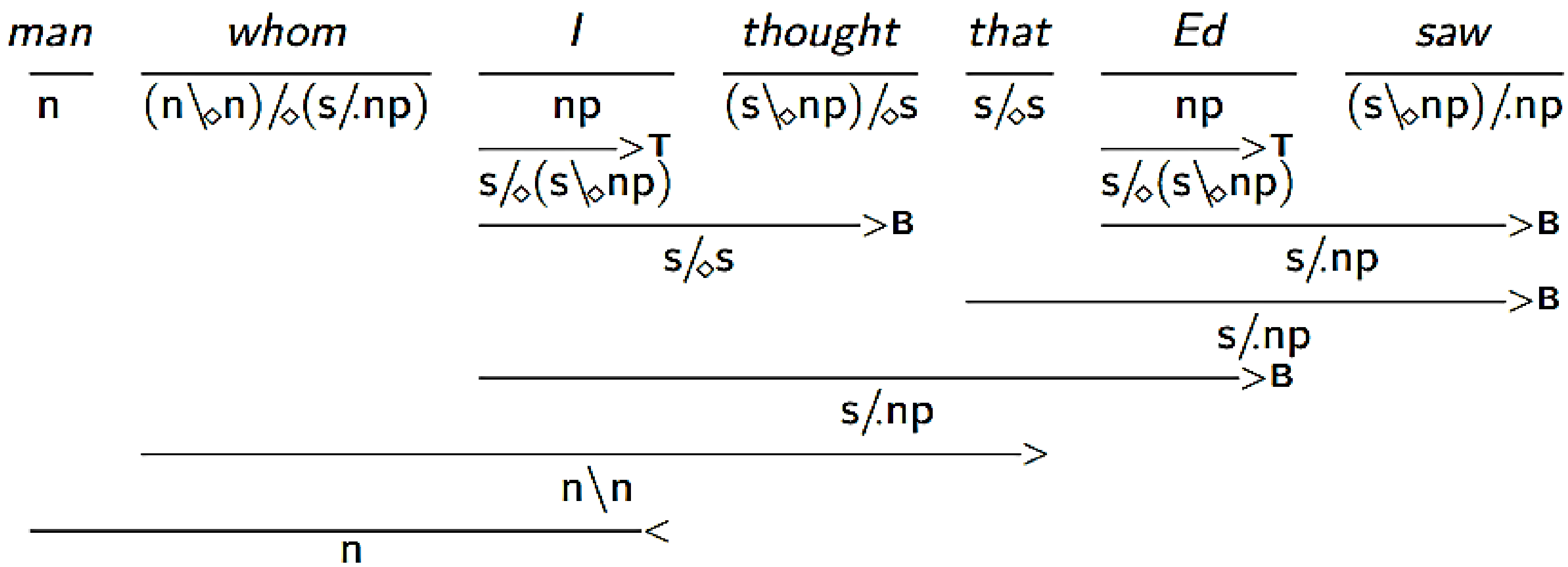
most restrictive



most permissive

Now restate *application rules* with $/_\star$, *composition rules* with $/_\diamond$ and *crossed composition rules* with $/_\times$ and everything will *magically* fix!

Further development: multi-modal CCG



Even further extension: modalities (like $\Box_i, i \in \{1, \ldots, n\}$), dependency grammars and other fancy things borrowed from CTL (Categorical Type Logic).

What about semantics then?

Note: from the point of view of semantics, the use of λ-terms is simply a convenient device to bind arguments when presenting derivations. What actually matters are dependencies!

What about semantics then?

Note: from the point of view of semantics, the use of λ-terms is simply a convenient device to bind arguments when presenting derivations. What actually matters are dependencies!

- **Hybrid Logic Dependency Semantics** (Baldridge and Kruijff, 2003): directly adapt the CTL framework to encode the semantics of CCG derivations

Hybrid modal logic (Blackburn, 2000) allows explicit references to states in the object language

What about semantics then?

Note: from the point of view of semantics, the use of λ-terms is simply a convenient device to bind arguments when presenting derivations. What actually matters are dependencies!

- **Hybrid Logic Dependency Semantics** (Baldridge and Kruijff, 2003): directly adapt the CTL framework to encode the semantics of CCG derivations

Hybrid modal logic (Blackburn, 2000): allows explicit references to states in the object language

For instance:

$$Ed \vdash \mathsf{n} : @_{d_1}\mathbf{Ed}$$

n: syntactic category

d_1: discourse referent (unique for Ed!)

What about semantics then?

Note: from the point of view of semantics, the use of λ-terms is simply a convenient device to bind arguments when presenting derivations. What actually matters are dependencies!

- **Hybrid Logic Dependency Semantics** (Baldridge and Kruijff, 2003): directly adapt the CTL framework to encode the semantics of CCG derivations

Hybrid modal logic (Blackburn, 2000): allows explicit references to states in the object language

For instance:

$$Ed \vdash \mathsf{n} : @_{d_1}\mathbf{Ed}$$

n: syntactic category

d_1: discourse referent (unique for Ed!)

$$sleeps \vdash \underbrace{\mathsf{s} : @_{h_2}(\mathbf{sleep} \wedge \langle \textsc{Act} \rangle (i \wedge p))}_{\text{resulting sentence with attached semantics}} \backslash \underbrace{\mathsf{n} : @_i p}_{\text{argument: subject}}$$

CCG applications

- Semantics and Knowledge Representation
  (Bos et al. 2004, Bos 2005, Harrington et al. 2007);
- Discourse Theory, Dialogue Systems
  (Steedman 2003, Curran et al. 2007);
- Object Extraction and Question Parsing
  (Clark et al. 2004);
- Natural Language Generation
  (White et al. 2003-2007);
- Semantic Parsing and Semantic Role Labeling
  (Gildea et al. 2003, Zettlemoyer et al. 2007);
- Statistical Machine Translation
  (Birch et al. 2007, Mehay et al. 2012);
- ...

CCG applications

- Semantics and Knowledge Representation
  (Bos et al. 2004, Bos 2005, Harrington et al. 2007);
- Discourse Theory, Dialogue Systems
  (Steedman 2003, Curran et al. 2007);

C&C tools
`http://svn.ask.it.usyd.edu.au/trac/candc`

- **Wide-coverage CCG parser and supertagger:** provides categories, derivation trees and dependencies
- **Additional tools**: POS tagger, Lemmatizer, NER, …
- **Boxer**: takes a CCG derivation and generates a semantic representation (*Discourse Representation Structures*)

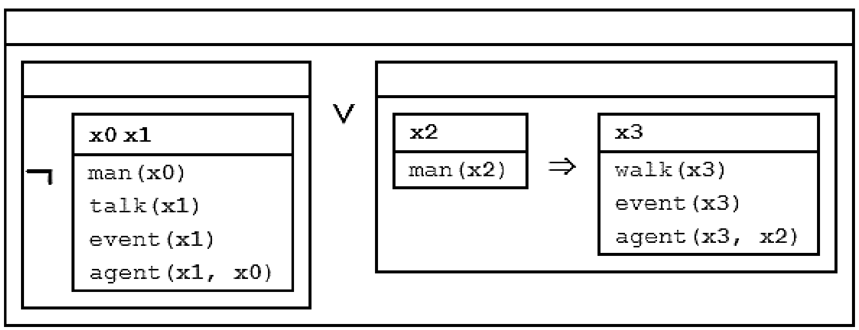
C&C tools: a demo

A man does not talk or every man walks.

```
(det man_1 A_0)
(ncmod _ does_2 not_3)
(aux talk_4 does_2)
(ncsubj talk_4 man_1 _)
(det man_7 every_6)
(ncsubj walks_8 man_7 _)
(conj or_5 walks_8)
(conj or_5 does_2)
<c> A|a|DT|I-NP|O|NP[nb]/N man|man|NN|I-NP|O|N does|do|VBZ|I-VP|O|
(S[dcl]\NP)/(S[b]\NP) not|not|RB|I-VP|O|(S\NP)\(S\NP) talk|talk|VB|I-VP|O|S[b]\NP
or|or|CC|O|O|conj every|every|DT|I-NP|O|NP[nb]/N man|man|NN|I-NP|O|N
walks|walk|VBZ|I-VP|O|S[dcl]\NP .|.|.|O|O|.
```

dependencies
(same formalism of the Stanford Parser)

supertags



Boxer output

Online demo available at:
http://svn.ask.it.usyd.edu.au/trac/candc/wiki/Demo

CCG applications

OpenCCG
`http://openccg.sourceforge.net`

- Part of *OpenNLP* (an open source library written in Java) focused on *parsing* and *realization*
- Makes use of *multi-modal* extensions to CCG and the *hybrid logic* semantics just described
- Current development efforts towards the use of the realizer in a *dialogue* system

- Natural Language Generation
  (White et al. 2003-2007);

References

M. Steedman, *Categorial Grammar, A short encyclopedia entry for MIT Encyclopedia of Cognitive Sciences*. In R. Wilson and F. Keil (eds.), 1999.

M. Steedman, J. Baldridge, *Combinatory Categorial Grammar.* In R. Borsley and K. Borjars (eds.), *Non-Transformational Syntax*, Blackwell, 2005.

J. Hockenmaier, M. Steedman, *CCGbank: A Corpus of CCG Derivations and Dependency Structures Extracted from the Penn Treebank.* In Journal of Computational Linguistics, vol. 33-3, pp. 355-396, 2007

J. R. Curran, S. Clark, J. Bos, *Linguistically Motivated Large-Scale NLP with C&C and Boxer*. In Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL-07), pp.29-32, 2007.

The CCG site

`http://groups.inf.ed.ac.uk/ccg`

Additional readings

S. Clark, J. R. Curran, *Log-Linear Models for Wide-Coverage CCG Parsing*. In Proceedings of the SIGDAT Conference on Empirical Methods in Natural Language Processing, 2003.

J. Baldridge, G.-J. Kruijff, *Coupling CCG with Hybrid Logic Dependency Semantics*. In Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL-02), 2002.

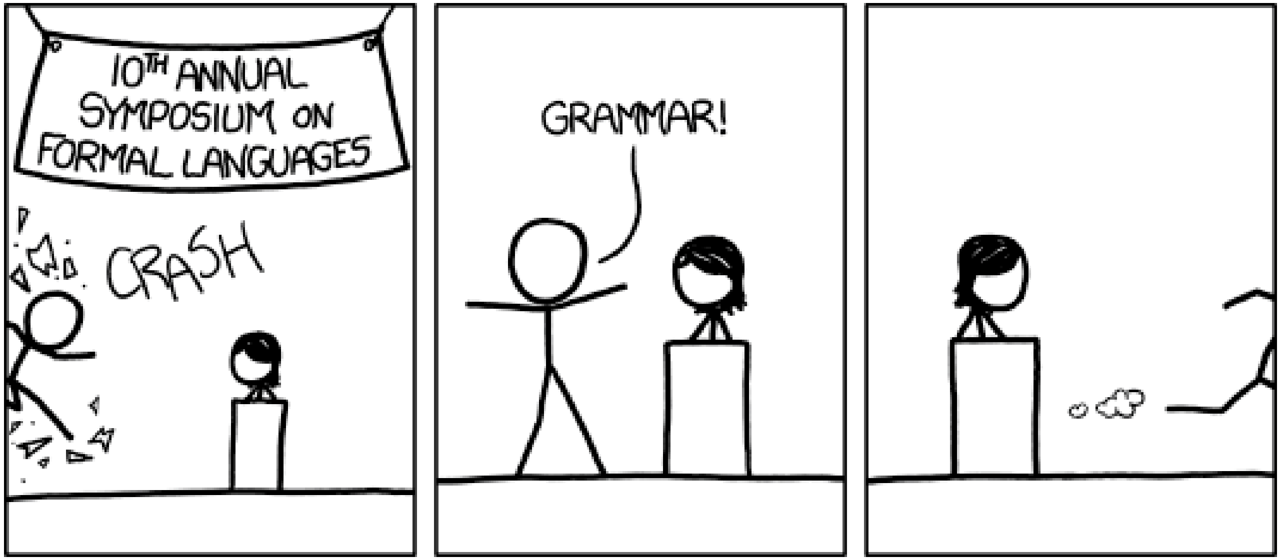
J. Baldridge, G.-J. Kruijff, *Multi-Modal Combinatory Categorial Grammar*. In Proceedings of the 11th Conference of the European Chapter for the Association of Computational Linguistics, 2003.

...and a thousand more at:

The CCG site

`http://groups.inf.ed.ac.uk/ccg`

Thanks for your attention!



[*audience looks around*] 'What just happened?' 'There must be some context we're missing.'

xkcd, *Formal languages*