2. Combinatory Categorial Grammar for NLP

Mark Steedman, University of Edinburgh

Mar 14th 2017

School of informatics

Prologue

- One reaction to the theoretical catastrophe of the '70s was to try to define alternatives to transformational grammar
- Alternative theories came in two flavors:
  - Formalisms that were equally expressive in theory, but whose correctness was easier to check automatically in practice: (ATN, FUG, LFG, HPSG);
  - Formalisms that were much less expressive in the first place: (GPSG, TAG, CCG).

CCG

- Combinatory Categorial Grammar (CCG) is a Radically Lexicalized grammar formalism, in which:
  - All information specific to a given language is specified in its lexicon, including valency, word-order, and semantics;
  - Syntactic projection from the lexicon to the sentences of the language is by strictly adjacent combinatory merger, without movement or "action at a distance" of any kind;
  - Syntactic and semantic composition are entirely homomorphic, and related "rule-to-rule".
- CCG is currently the only linguistically adequate grammar formalism that is widely used in applied computational linguistics.

It is currently less well understood in Linguistics departments.

Outline

- I: CCG
- II: Descriptive Adequacy of CCG
- III: Explanatory Adequacy of CCG
- IV: The Shape of Things to Come
- V: Conclusion

I: Combinatory Categorial Grammar

- CCG eschews language-specific syntactic rules like (3) for English.

(1) $S \rightarrow NP \ VP$
$VP \rightarrow TV \ NP$
$TV \rightarrow \{proved, found, sees, \ldots\}$

- Instead, all language-specific syntactic information is lexicalized, via lexical entries like (4) for the English transitive verb:

(2) $\text{sees} := (S\backslash NP)/NP$

- This syntactic "category" identifies the transitive verb as a function, and specifies the type and directionality of its arguments and the type of its result.

Combinatory Categorial Grammar

- CCG eschews language-specific syntactic rules like (3) for English.

(3) $S \rightarrow NP \ VP$
$VP \rightarrow TV \ NP$
$TV \rightarrow \{proved, found, sees, \ldots\}$

- Instead, all language-specific syntactic information is lexicalized, via lexical entries like (4) for the English transitive verb:

(4) $\text{sees} := (S\backslash NP)/NP : \lambda x \lambda y.sees\,x\,y$

- This syntactic "category" identifies the transitive verb as a function, and specifies the type and directionality of its arguments and the type of its result.

Anatomy of a Category

- (5) a. sees $:= (S\backslash NP_{3s})/NP : \lambda x \lambda y.sees' x y$

b. $\overbrace{\text{sees}}^{\text{phonological form}} := \overbrace{\overbrace{(S\ \underbrace{fin}_{\text{feature}}\ \backslash NP_{3s})/NP}^{\text{syntactic type}} : \overbrace{\underbrace{\lambda x \lambda y.}_{\lambda\text{-binders}}\ \underbrace{sees' x y}_{\text{predicate-argument structure}}}^{\text{logical form}}}^{\text{category}}$

Lexicalizing "A-movement"

- To take a slightly more complex lexical entry, the following is the category for a subject control verb for a sentence such as *He promises her to leave*:

(6) a. promises $:= ((S\backslash NP_{3s})/VP_{to})/NP : \lambda x \lambda p \lambda y.promise'\,(p\,y)\,x\,y$

b. $\overbrace{\text{promises}}^{\text{phonological form}} := \overbrace{\overbrace{((S\ \underbrace{fin}_{\text{feature}}\ \backslash NP_{3s})/VP_{to})/NP}^{\text{syntactic type}} : \overbrace{\underbrace{\lambda x \lambda p \lambda y.}_{\lambda\text{-binders}}\ \underbrace{promise'\,(p\,y)\,x\,y}_{\text{predicate-argument structure}}}^{\text{logical form}}}^{\text{category}}$

Pure Categorial Grammar

- (7) a. Forward Application:

$$X/Y : f \quad Y : a \quad \Rightarrow \quad X : f\,a \qquad (>)$$

b. Backward Application:

$$Y : a \quad X\backslash Y : f \quad \Rightarrow \quad X : f\,a \qquad (<)$$

- (8)

$$\dfrac{\dfrac{\text{Harry}}{NP_{3s}}{>}\mathbf{T} \quad \dfrac{\text{sees}}{(S\backslash NP_{3s})/NP} \quad \dfrac{\text{Sally}}{NP}{<}\mathbf{T}}{\dfrac{\dfrac{}{S\backslash NP_{3s}}{>}}{S}{<}}$$

Pure Categorial Grammar

- (9) a. Forward Application:

$$X/Y : f \quad Y : a \quad \Rightarrow \quad X : f\,a \qquad (>)$$

b. Backward Application:

$$Y : a \quad X\backslash Y : f \quad \Rightarrow \quad X : f\,a \qquad (<)$$

- (10)

$$\dfrac{\dfrac{\text{Harry}}{\underset{\text{: } harry'}{NP_{3s}}}>\mathbf{T} \quad \dfrac{\text{sees}}{\underset{\text{: } \lambda x\lambda y.sees'\,x\,y}{(S\backslash NP_{3s})/NP}} \quad \dfrac{\text{Sally}}{\underset{\text{: } sally'}{NP}}<\mathbf{T}}{\underset{\text{: } \lambda y.sees'\,sally'\,y}{S\backslash NP_{3s}}}> \Bigg/ \underset{\text{: } sees'\,sally'\,harry'}{S}<$$

The Combinatory Projection Principle

- (11) *The Combinatory Projection Principle (CPP)*
  Syntactic combinatory rules are binary linearly ordered type-dependent rules, applying to string-adjacent categories, consistent with their directional types and linear order, and must project unchanged onto the result the type and directionality of any argument of the inputs that also appears there.
- This principle is defined more formally in *SP* in terms of three more fundamental principles of Adjacency, Directional Inheritance, and Directional Consistency, which collectively forbid rules like (a), (b), and (c):

(12)
$$a.\quad Y:a \qquad X/Y:f \;\not\Rightarrow\; X:fa$$
$$b.\quad (X/Y)/Z:f \qquad Y:a \;\not\Rightarrow\; X/Z:fa$$
$$c.\quad (X/Y)/Z:f \qquad Z:a \;\not\Rightarrow\; X\backslash Y:fa$$

Type Raising as Case

- Type-raising, in the form of case, is a universal primitive of grammar.

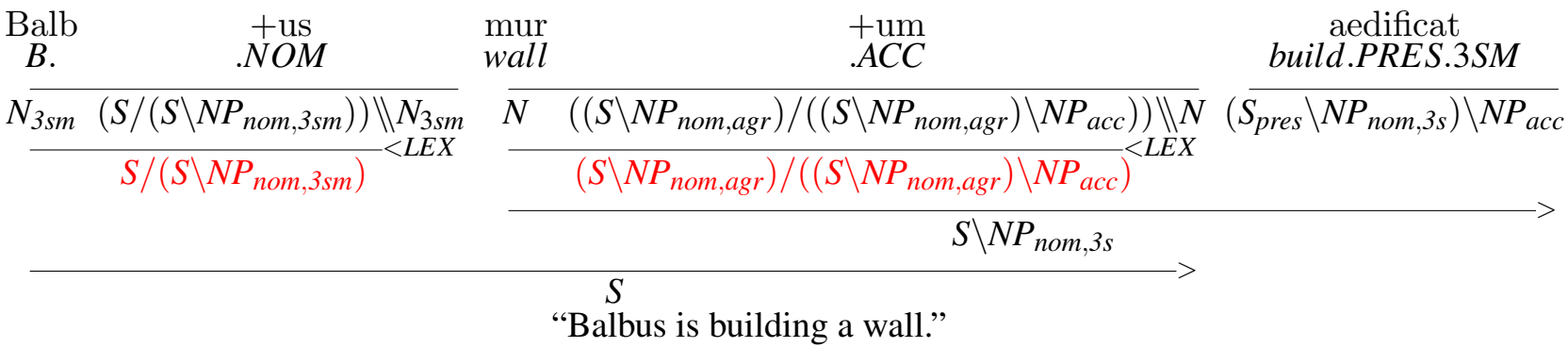
All noun-phrases (NP) like "Harry" are (polymorphically) type-raised in the morpholexicon.

- In Japanese and Latin this is the job of case morphemes like nominative *-ga* and *-us*.

- In English NPs are underspecified as to case, and must be disambiguated by the parsing model.

Latin Morphological Case

- (13)



"Balbus is building a wall."

English "Structural" Case

- English case is ambiguous and polymorphic

(14)
$$\dfrac{\dfrac{\text{Harry}}{S/(S\backslash NP)}\,{>}\mathbf{T} \quad \dfrac{\dfrac{\text{sees}}{(S\backslash NP)/NP} \quad \dfrac{\text{Sally}}{(S\backslash NP)\backslash((S\backslash NP)/NP)}\,{<}\mathbf{T}}{S\backslash NP}\,{<}}{S}\,{>}$$

- (15)
$$\dfrac{\dfrac{\text{missed}}{S/NP} \quad \dfrac{\text{the Saturday dance}}{S\backslash(S/NP)}}{S:}\,{<}$$

English "Structural" Case

- English case is ambiguous and polymorphic

(16)
$$\dfrac{\dfrac{\text{Harry}}{S/(S\backslash NP) : \lambda p.p\ harry'}\,{>}\mathbf{T} \quad \dfrac{\text{sees}}{(S\backslash NP)/NP : sees'} \quad \dfrac{\text{Sally}}{(S\backslash NP)\backslash((S\backslash NP)/NP) : \lambda p.p\ sally'}\,{<}\mathbf{T}}{\dfrac{S\backslash NP : sees'\,sally'}{S : sees'\,sally'\,harry'}\,{<}}\,{>}$$

- (17)
$$\dfrac{\dfrac{\text{missed}}{S/NP \quad \lambda x.missed'\,x\,one'_{1s}} \quad \dfrac{\text{the Saturday dance}}{S\backslash(S/NP) \quad \lambda p.p\,saturdaydance}}{S : missed'\,saturdaydance\,one'_{1s}}\,{<}$$

Relativization

- Relativization and all unbounded dependencies crucially involve syntactic combinatory rules of function composition, subject to the CPP.

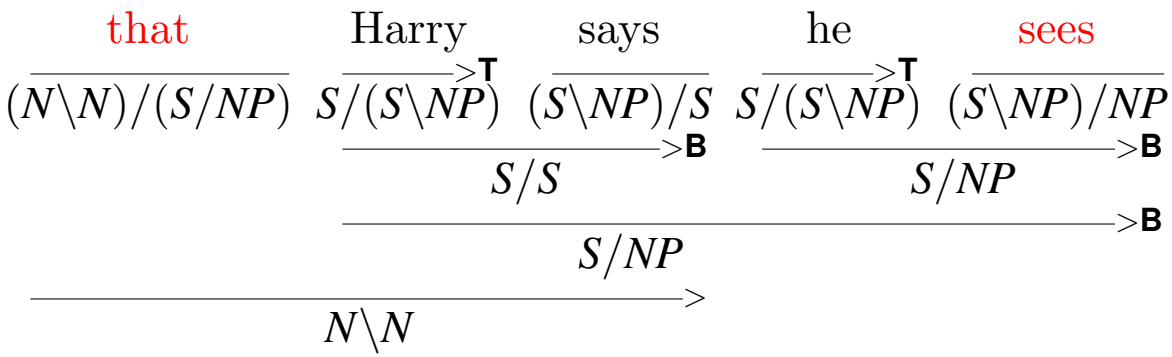
- (18) that $:= (N\backslash N)/(S/NP)$

(19) (The woman)

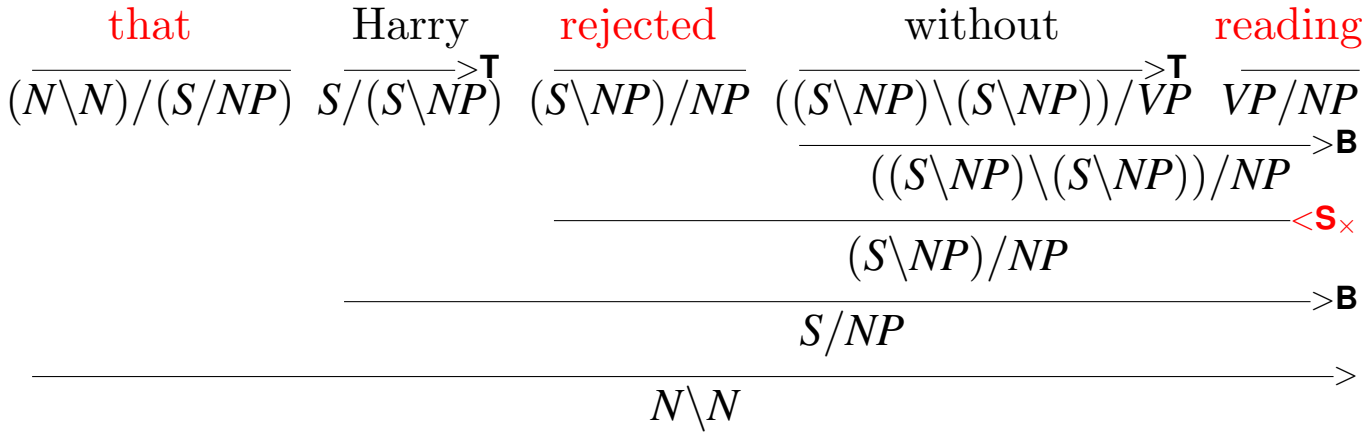
$$\dfrac{\dfrac{\text{that}}{(N\backslash N)/(S/NP)} \quad \dfrac{\dfrac{\text{Harry}}{S/(S\backslash NP)}\,{>}\mathbf{T} \quad \dfrac{\text{sees}}{(S\backslash NP)/NP}}{S/NP}\,{>}\mathbf{B}}{N\backslash N}\,{>}$$

Relativization

- (20) (The woman)



- (21) (The articles)



Coordination

- (22)

$$\dfrac{\dfrac{\dfrac{\dfrac{\text{[Harry sees]}}{S/NP}\,{>}\mathbf{B} \quad \dfrac{\text{and}}{(X\backslash X)/X} \quad \dfrac{\dfrac{\text{[Fred says he likes]}}{S/NP}\,{>}\mathbf{B} \quad \dfrac{\text{Sally}}{S\backslash(S/NP)}\,{<}\mathbf{T}}{(S/NP)\backslash(S/NP)}\,{>}}{(S/NP)}\,{<}}{S}\,{<}$$

- (23)

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{give}}{DTV} \quad \dfrac{\text{Harry}}{TV\backslash DTV}\,{<}\mathbf{T} \quad \dfrac{\text{a book}}{VP\backslash TV}\,{<}\mathbf{T}}{VP\backslash DTV}\,{<}\mathbf{B} \quad \dfrac{\text{and}}{(X\backslash X)/X} \quad \dfrac{\dfrac{\text{Sally}}{TV\backslash DTV}\,{<}\mathbf{T} \quad \dfrac{\text{a record}}{VP\backslash TV}\,{<}\mathbf{T}}{VP\backslash DTV}\,{<}\mathbf{B}}{(VP\backslash DTV)\backslash(VP\backslash DTV)}\,{>}}{VP\backslash DTV}\,{<}}{VP}\,{<}$$

II: Descriptive Adequacy of CCG

- The argument cluster coordination construction (23) is an example of a universal tendency for "deletion under coordination" to respect basic word order: in all languages, if arguments are on the left of the verb then argument clusters coordinate on the left, if arguments are to the right of the verb then argument clusters coordinate to the right of the verb (Ross 1970):

(24)

SVO:	*SO and SVO	SVO and SO
VSO:	*SO and VSO	VSO and SO
SOV:	SO and SOV	*SOV and SO

CCG reduces the linguists' MOVE and COPY/DELETE to adjacent MERGE

CCG is Trans- Context-Free

(25)

das	mer	em Hans	es huus	hälfed	aastriiche
that	we−NOM	Hans−DAT	the house−ACC	helped	paint
	$NP^{\uparrow}_{nom}$	$NP^{\uparrow}_{dat}$	$NP^{\uparrow}_{acc}$	$((S_{+SUB}\backslash NP_{nom})\backslash NP_{dat})/VP$	$VP\backslash NP_{acc}$
				$(((S_{+SUB}\backslash NP_{nom})\backslash NP_{dat})\backslash NP_{acc}$ (>$\mathbf{B}_{\times}$)	
			$(S_{+SUB}\backslash NP_{nom})\backslash NP_{dat}$ (>)		
		$S_{+SUB}\backslash NP_{nom}$ (>)			
	S_{+SUB} (>)				

"that we helped Hans paint the house"

- The following is correctly also allowed (Shieber, 1985):

  (26) Das mer em Hans hälfed es huus aastriiche.

CCG is Trans- Context-Free

(27)

das	mer	d′chind	em Hans	es huus	lönd	hälfe	aastriiche
that	we−NOM	the children−ACC	Hans−DAT	the house−ACC	let	help	paint
	$NP^{\uparrow}_{nom}$	$NP^{\uparrow}_{acc}$	$NP^{\uparrow}_{dat}$	$NP^{\uparrow}_{acc}$	$((S_{+SUB}\backslash NP_{nom})\backslash NP_{acc})/VP$	$(VP\backslash NP_{dat})/VP$	$VP\backslash NP_{acc}$
					$(((S_{+SUB}\backslash NP_{nom})\backslash NP_{acc})\backslash NP_{dat})/VP$ (>$\mathbf{B}^2_\times$)		
					$(((S_{+SUB}\backslash NP_{nom})\backslash NP_{acc})\backslash NP_{dat})\backslash NP_{acc}$ (>$\mathbf{B}_\times$)		
				$((S_{+SUB}\backslash NP_{nom})\backslash NP_{acc})\backslash NP_{dat}$ (>)			
			$(S_{+SUB}\backslash NP_{nom})\backslash NP_{acc}$ (>)				
		$S_{+SUB}\backslash NP_{nom}$ (>)					
	S_{+SUB} (>)						

"that we let the children help Hans paint the house"

- Again, other word orders are correctly allowed.

III: Explanatory Adequacy of CCG

- CCG's combination of type-raising (**T**), composition (**B**), and substitution (**S**), subject to CPP yields a permuting and rebracketing calculus closely tuned to the needs of natural grammar.

- CCG and TAG are provably weakly equivalent to Linear Indexed Grammar (LIG) Vijay-Shanker and Weir (1994).

- Hence they are not merely "Mildly Context Sensitive" (Joshi 1988), but rather "Nearly Context Free," (NCF) in the Extended Chomsky Hierarchy:

Language Type	Automaton	Rule-types	Exemplar
Type 0: RE	Universal Turing Machine	$\alpha \rightarrow \beta$	PA-valid
Type 1: CS	Linear Bound Automaton (LBA)	$\phi A \psi \rightarrow \phi \alpha \psi$	$a^{n!}$
MCF (LCFRS)	ith-order EPDA	$A_{[[(i),\ldots]\ldots]} \rightarrow \phi B_{[[(i),\ldots]\ldots]} \psi$	$\mathcal{P}(a^n b^n c^n)$
IG	Nested Stack Automaton(NSA)	$A_{[(i),\ldots]} \rightarrow \phi B_{[(i),\ldots]} \psi C_{[(i),\ldots]} \xi$	a^{2^n}
LIG/CCG/TAG	Embedded PDA (EPDA)	$A_{[(i),\ldots]} \rightarrow \phi B_{[(i),\ldots]} \psi$	$a^n b^n c^n$
Type 2: CF	Push-Down Automaton (PDA)	$A \rightarrow \alpha$	$a^n b^n$
Type 3: FS	Finite-state Automaton (FSA)	$A \rightarrow \begin{cases} \text{a B} \\ \text{a} \end{cases}$	a^n

All higher language classes properly contain all lower except MCF and I, which properly intersect.

Why is it Good to be Near-Context-Free?

- To achieve explanatory adequacy by limiting degrees of freedom in the theory.
- How many degrees of freedom are there in CCG?
- A lot less than in LCFRS/MCFG, and therefore a lot less than under the Minimalist Program
- A lot less than the set of all possible categories would allow, defined by the rule "if α and β are types, then α/β and $\alpha\backslash\beta$ are types"

The Type-System of Predicates

- (28) 1. t and e are elementary types.
  2. predicate types have bounded valency ≤ 4.
  3. $e \rightarrow t$ and $t \rightarrow t$ are predicate types.
  4. If α is an elementary type and $e \rightarrow \beta$ is a predicate type then $\alpha \rightarrow (e \rightarrow \beta)$ is a predicate type.
  5. Nothing else is a type.

- For example, we have $seem'_{t \rightarrow t}$,$want'_{t \rightarrow (e \rightarrow t)}$,$promise'_{t \rightarrow (e \rightarrow (e \rightarrow t))}$

We do not assume that the predicate we write $promise'$ is actually atomic.

The Type System of Logical Form

- We have seen that the lexical logical forms constituted via the lambda-binders in(5b) are more diverse. They allow *properties* of type $e \rightarrow t$ as arguments of raising and control verbs, as well as e and t. They also allow those arguments to combine in any order, regardless of lf-command relations.

  (29) If a predicate p' as defined in (100) has an argument of type t, then it can be realized in the predicate-argument structure of a corresponding lexical VP stem as *either* (i) an free-variable argument, also of type t, *or*; (ii) as the application of an argument free variable of type $e \rightarrow t$ to another free-variable argument of type e. The latter e-type argument can either be unique to this predication (”raising”), or it can be one of the original arguments of p' (”control”, ”ECM”).

- For example, we can now have $seem'(p\,e)$ as well as $seem'\,t$.

Defining the Lexical Stem Types

- To specify the stem types of the lexicon, we also need a language-independent mapping between the elementary types and syntactic argument types (generalizing over minor features such as S', VP_{to}, etc.):

(30)
$$\begin{array}{ll} S & \mapsto t \\ NP, PP & \mapsto e \\ N, VP & \mapsto (e \rightarrow t) \end{array}$$

- Languages are then free to associate directional syntactic categories specifying the linear position of constituents corresponding to arguments of those types in all possible ways, with linking λ-binders to pass their values into universal predicate-argument structure (UPAS).
- The present language-specific logical forms are a proxy for UPAS

Functional Projections

- It is important at this point to note that $S\backslash NP$ and S/NP, the types of the tensed intransitive, are not yet syntactically typable, All verbs so far are functions into $e \rightarrow t$, and all subcategorized predicates are VP.
- The stem types are then mapped onto further functional projections, in English either morphologically as with tense or by auxiliary verbs.
- Either tense-morphology or a lexical rule maps infinitival verb stems $VP\$$ (where \$ schematizes over subcategorizations) onto corresponding tensed forms of the same semantic type requiring a subject—for English, $S\backslash NP_{agr}\$$; for Welsh, $S\$/NP_{agr}$.
- English passive morphology and Dyirbal antipassive morphology similarly map accusative and absolutive transitive VPs onto intransitive VP. (Thus, morphological operators may either increase or decrease valency.)

Case

- Similarly, case morphology (or structurally disambiguated underspecification) maps argument categories into functors over all and only the lexical (tensed, etc.) functor types that take them as arguments.

- The lexicon also includes non-order preserving higher types, such as wh-words and *tough*-predicates, which change the type of their result. In the case of topicalizers, the changed category is a root construction.

- These various levels of lexicalization are reminiscent of Emonds's 1976 structure-preserving, local, and root transformations, and of Williams, 2003 "levels" of movement phenomena.

Combinatory Syntactic Projection

- All and only the Combinatory Rules allowed under the CPP are necessary to project the lexical categories onto sentence derivations.

- The language-specific lexicon can specify restrictions on which rules can apply to which categories, via slash-types not discussed here.

- Constraints on dependency projection, such as the **that-t* condition and the across-the-board condition arise from the combinatorics of CCG rather than stipulated constraints (See *SP, passim*).

On Having Fewer Degrees of Freedom

- What we want from a theory is that all and only the degrees of freedom it allows are observed in the data it seeks to explain.
- For example, to allow (32) we need it to allow the following categories for raising verbs:

(31) a. seems := $(S\backslash NP)/VP_{to} : \lambda p \lambda y.seems'\,(p\,y)$
b. seems := $(S\backslash NP_{xpl})/S : \lambda p \lambda s.seems'\,s$

(32) a. John seems to be certain to leave.
b. It seems (that) John is certain to leave.

- Such examples have appeared to require a Minimal Link Condition to prevent "superraising" (Chomsky, 1995) to more distant targets,

On Having Fewer Degrees of Freedom

- However, in order allow the "superraising" example (33a) we would either need a CCP-violating rule applying non-adjacent merger, or a category that is not a possible CCG lexical item, and would allow (33b) if it were:

(33) a. *John seems (that) it is certain to leave.
b. *John seems (that) is certain to leave.

- Similarly, we are allowed the following control verbs:

(34) a. persuades := $((S\backslash NP)/VP_{to})/NP : \lambda x \lambda p \lambda y.persuades'\,(p\,x)\,x\,y$
b. promises := $((S\backslash NP)/VP_{to})/NP : \lambda x \lambda p \lambda y.promises'\,(p\,y)\,x\,y$

- However, we do not need to invoke an exception to the MLC to explain why the "link" in (34b) is not minimal.

On Having Fewer Degrees of Freedom

- The CPP means that we cannot simulate Move-α.
- Of the $n!$ possible permutations we only allow the Large Schroeder Number $S(n)$ (cf. Williams (2003):25).
- For example, for a "cartographic" right-branching spine of 8 functional projections, more than 75% of the 8! permutations are excluded

The Spurious Problem of "Spurious Ambiguity"

- CCG allows (many) alternative derivations for the same logical form.
- As well as the earlier derivation, it allows the following:

(35)
$$\dfrac{\dfrac{\dfrac{\text{Harry}}{S/(S\backslash NP) : \lambda p.p\ harry'}\,{>}\mathbf{T} \quad \dfrac{\text{sees}}{(S\backslash NP)/NP : sees'} \quad \dfrac{\text{Sally}}{(S\backslash NP)\backslash((S\backslash NP)/NP) : \lambda p.p\ sally'}\,{<}\mathbf{T}}{S/NP : \lambda x.sees'\ x\ harry'}\,{>}\mathbf{B}}{S : sees'\ sally'\ harry'}\,{<}$$

For longer sentences there will up to Catalan numbers of alternatives.

Practical Applications of CCG

- It was widely assumed in the '80s that the degree of derivational ambiguity CCG allows would make it completely impractical for parsing.
- However, any grammar that covers these data has the same degree of nondeterminism.
- The catastrophic realization in the '90s of the need for statistical modeling in NLP due to the advent of the Penn WSJ Treebank was a great leveller.
- With such models, CCG can be parsed as fast and accurately as anything else. . .
- . . . with the advantage of a surface compositional semantics including discontinuity and "non-projectivity",

IV: The Shape of Thing to Come

- Thanks to the tools developed by colleagues, like Treebanks, Parsers, OpenCCG, Semantic Parser Learners etc., CCG is quite widely used in practical applications that require semantics, like semantic parsing, entailment, and generation.

- The advent of Deep Learning, LSTM, RNN, etc offers support for the original Ades and Steedman (1982) Shift-Reduce Parser architecture, using neural networks as supertaggers and transition models.

- Underspecification and disambiguation via superposition of contexts seems to be what sequential neural models and "embeddings" are good for, rather than representing semantics itself.,

Conclusion

- Natural Language Syntax = Case + Composition[1]

[1] We also need **S** combinatory rules, subject of course to CPP

References

Ades, Anthony and Steedman, Mark, 1982. "On the Order of Words." *Linguistics and Philosophy* 4:517–558.

Chomsky, Noam, 1995. *The Minimalist Program*. Cambridge, MA: MIT Press.

Emonds, Joseph, 1976. *A Transformational Approach to English Syntax*. New York: Academic Press.

Joshi, Aravind, 1988. "Tree-Adjoining Grammars." In David Dowty, Lauri Karttunen, and Arnold Zwicky (eds.), *Natural Language Parsing*, Cambridge: Cambridge University Press. 206–250.

Ross, John Robert, 1970. "Gapping and the Order of Constituents." In Manfred Bierwisch and Karl Heidolph (eds.), *Progress in Linguistics*, The Hague: Mouton. 249–259.

Shieber, Stuart, 1985. "Evidence against the Context-Freeness of Natural Language." *Linguistics and Philosophy* 8:333–343.

Steedman, Mark, 2000. *The Syntactic Process*. Cambridge, MA: MIT Press.

Vijay-Shanker, K. and Weir, David, 1994. "The Equivalence of Four Extensions of Context-Free Grammar." *Mathematical Systems Theory* 27:511–546.

Williams, Edwin, 2003. *Representation theory*. MIT Press.