Motivation for Categorial Grammar

Why investigate categorial grammar (CG)?

- CG is a minimal extension to CFGs.
- CG has a close relation to compositional semantics & is well-understood from a logical standpoint.
- Cross-linguistic generalizations can be made easily since the same set of rules always apply.
- Flexible constituency allows for coordination of unlikes and treatment of “unbounded” constructions.
  - This also supports psychologically plausible analyses, since processing can proceed in a left-to-right fashion.

Categorial Grammar: Categories

Rules of grammar are entirely conditioned on lexical categories

- i.e., many categories & a small set of applicable rules
- Categories (or types) come in two varieties:
  - **Primitive categories:** N, NP, PP, S, etc.
    - (1) Marcel := NP
    - (2) man := N
  - can further be distinguished by features
- **Functions:** a combination of primitive categories, i.e., a function from one category (primitive or function) to another:
  - S/NP, (S/NP)/(S/NP), etc.

Functional application (FA)

- Syntactically potent elements such as verbs are associated with a syntactic category that identifies them as functions
  - Functions specify the type and directionality of their arguments and the type of their result
  - S\(\text{NP}\) is an intransitive verb because it is looking for an NP (to the left) in order to form an S
  - A “result leftmost” notation is used here:
    - \(\alpha/\beta\) is a rightward-combining functor over a domain \(\beta\) into a range \(\alpha\)
    - \(\alpha\backslash\beta\) is the corresponding leftward-combining functor.
    - \(\alpha\) and \(\beta\) may themselves be functional categories.

Example lexical entries

- Intransitive, transitive, & ditransitive verbs:
  - (3) a. ran := (\(S\backslash\text{NP}\))
     - b. proved := (\(S\backslash\text{NP}\))/\(\text{NP}\)
     - c. gave := ((\(S\backslash\text{NP}\))/\(\text{NP}\))/\(\text{NP}\)
  - Sentence and verb modifying adverbs
    - (4) a. yesterday := S/S
     - b. always := (\(S\backslash\text{NP}\))/(\(S\backslash\text{NP}\))
Rules and derivations

- Functor categories can combine with their arguments by the following rules:
  
  (5) Forward application (>)
  \[ X/Y \Rightarrow X \]  
  (6) Backward application (<)
  \[ Y \setminus X \Rightarrow X \] 

- Derivations are written as shown below. Note the direct correspondence to an upside-down constituency tree.

```
  Marcel
  ┌─────────────┐
  │ NP         │
  │ proved     │
  │ (S\NP)/NP  │
  │ completeness │
  │ \(S\NP)    │
  ├─────────────┤
  │ S           │
```

- Exercise: What is the derivation of Marcel proved completeness yesterday?

Adjuncts

The method for combining a head with adjuncts is the same as for the combination with complements: functional application.

- Note how in the previous proof tree, always was added with the same forward application as we used to add chocolate.

- Adjuncts are generally defined as being of type \(X/X\) (or \(X\backslash X\)).

```
  Marcel NP:marcel' proved (S\NP)/NP completeness'
  ┌─────────────┐
  │ (S\NP)/NP   │
  │ λy.λx.prove'(x, y) │
  │ NP:completeness' │
  │ \(S\NP\:\lambda x.\text{prove}'(x, completeness')\) │
  ├─────────────┤
  │ S:prove'(marcel', completeness') │
```

- Exercise: What is the derivation for Marcel always gave John chocolate?

Semantics

- Lexical categories are augmented with an explicit identification of their semantic interpretation
- Rules of functional application are accordingly expanded with an explicit semantics

(7) \(\text{proved} := (S\backslash NP)\backslash NP : \lambda y.\lambda x.\text{prove}'(x, y)\)

(8) a. Forward application (>)
\[ X/Y : f \quad Y : a \Rightarrow X : fa \]

b. Backward application (<)
\[ Y : a \quad X \setminus Y : f \Rightarrow X : fa \]

- Note that the syntactic head is the semantic functor.

Example derivation with semantics

```
  Marcel NP:marcel' proved (S\NP)/NP completeness'
  ┌─────────────┐
  │ (S\NP)/NP   │
  │ λy.λx.prove'(x, y) │
  │ NP:completeness' │
  │ \(S\NP\:\lambda x.\text{prove}'(x, completeness')\) │
  ├─────────────┤
  │ S:prove'(marcel', completeness') │
```

- Exercise: What is the derivation for Marcel always gave John chocolate?

Principle of Type Transparency

- The semantic interpretation of all combinatory rules is fully determined by the Principle of Type Transparency:
  
  - **Categories**: All syntactic categories reflect the semantic type of the associated logical form.
  - **Rules**: All syntactic combinatory rules are type-transparent versions of one of a small number of semantic operations over functions including application, composition, & type-raising.

In other words: deriving a sentence in different ways results in the same semantics.
Raising and Control

- Raising verb

  \( (9) \) a. John seemed to be angry.
  \[ \text{b. } \text{seem} := (S\lambda y. \text{seem}'(y)) \]

- (Subject-)Control verb

  \( (10) \) a. John promised to take a bath.
  \[ \text{b. } \text{promise} := (S\lambda y. \text{promise}'(y, p(y))) \]

Syntactic functions are identical, but the semantics of control include \( y \) as the interpretation of the subject, which also serves as the subject of the embedded infinitive (the predicate \( p \)).

Constituent Coordination

- The coordination and can be given the following (schematic) lexical entry:

  \( (11) \) and := (X\lambda X/X)

illustrated by the following constituent coordination example:

\[
\begin{array}{cccc}
\text{Marcel} & \text{conjectured} & \text{and} & \text{proved} \\
\text{NP} & \text{NP} & \text{NP} & \text{NP} \\
(S\lambda y. \text{NP}) & (X\lambda y. S\lambda y. \text{NP}) & (S\lambda y. \text{NP}) & (S\lambda y. \text{NP}) \\
S & S & S & S \\
\end{array}
\]

Exercise: Can we handle Right Node Raising, e.g., John conjectured and Harry proved completeness?

Combinatory Categorial Grammar

So far: categorial grammar (CG)

Combinatory Categorial Grammar (CCG) increases the power of CG by adding combinators, e.g., these two combinatory rules:

- Functional composition
- Type-raising

Composition example

\[
\begin{array}{cccc}
\text{Marcel} & \text{conjectured} & \text{and} & \text{proved} \\
\text{NP} & \text{NP} & \text{NP} & \text{NP} \\
(S\lambda y. \text{NP}) & (X\lambda y. S\lambda y. \text{NP}) & (S\lambda y. \text{NP}) & (S\lambda y. \text{NP}) \\
S & S & S & S \\
\end{array}
\]

NB: to save space in the future, we can abbreviate \( S\lambda y. \text{NP} \) as \( \text{VP} \), thus making \( \text{might} \)'s lexical category more intuitive: \( \text{VP} / \text{VP} \)

Exercise: Without FC, what would the lexical entry for \( \text{might} \) need to be?

Without function composition

If we had not used functional composition, we would have had to posit \( \text{might} \)'s lexical entry as:

\( (13) \) \text{might} := ((S\lambda y. \text{NP})/\text{NP})/(S\lambda y. \text{NP})/\text{NP}

- Select a transitive verb, \( (S\lambda y. \text{NP})/\text{NP} \)
- Then select the object of that transitive verb, \( \text{NP} \)

Consequence: separate rules for every different kind of complement verb (intransitive, ditransitive, sentential complement, etc.)
More on function composition

- **Exercise**: Using function composition
  - What are the analyses for *Marcel might prove completeness*?
  - Do they correspond to different semantic representations?
- **Q**: Is function composition too powerful?
  - A1: Dowty (1988) shows that, in terms of word order, no new orders are allowed by function composition:
    
    Application:
    
    \[
    \frac{\text{NP: marcel}^\prime}{\text{S: run marcel}^\prime} \quad \frac{\text{run}}{\text{S: NP: run}} \quad < \quad \frac{\text{NP: marcel}^\prime}{\text{S: NP: marcel}^\prime} \quad \frac{\text{run}}{\text{S: NP: run}}
    \]
    
    Other word orderings simply won’t work for either case.
  - A2: Modalities can be used to limit the power of function composition where needed

Type-raising example

With type-raising, we have more than one option for derivation, where the semantics work out to be the same:

\[
\frac{\text{NP: marcel}^\prime}{\text{S: NP: marcel}^\prime} \quad \frac{\text{run}}{\text{S: NP: run}} < \quad \frac{\text{NP: marcel}^\prime}{\text{S: NP: marcel}^\prime} \quad \frac{\text{run}}{\text{S: NP: run}}
\]

- Allowing any category to type-raise allows for an infinite number of derivations
  - One has to be sure to restrict it while parsing.
  - Type-raising creates sometimes spurious structural ambiguity
    - But it is extremely useful . . .

Non-standard surface structures

- Complement-taking verbs like *think*, VP/S, can compose with fragments like *Marcel proved*, S/NP, which accounts for right-node raising (17), and also provides the basis for an analysis of unbounded dependencies (18).

(17) [John disproved](S/NP) and [you think that Marcel proved](S/NP) completeness.

(18) the result that [you think that Marcel proved](S/NP)

- **Key**: Strings such as you *think that Marcel proved* are taken to be surface constituents of type S/NP.
  - i.e., constituents can be missing material

Right Node Raising

By using type-raising and functional composition, we can account for such phenomena as right node raising:

\[
\begin{array}{c}
\frac{\text{Marcel}}{\text{NP: marcel}^\prime} \quad \frac{\text{proved}}{\text{S: NP: proved}} \quad \frac{\text{S/NP}}{\text{S: NP: S/NP}} \quad \frac{\text{NP}}{\text{NP}} \\
\frac{\text{NP}}{\text{S: NP}} \quad \frac{\text{NP}}{\text{S: NP}} \quad \frac{\text{S/NP}}{\text{S: NP}} \quad \frac{\text{NP}}{\text{NP}}
\end{array}
\]

- CCG includes type-raising rules, which turn arguments into functions over functions-over-such-arguments.
  - (14) Forward type-raising (> T)
    \[
    X : a \Rightarrow T / (T \backslash X) : \lambda f . f a
    \]
  - (15) Backward type-raising (< T)
    \[
    X : a \Rightarrow T / (T \backslash X) : \lambda f . f a
    \]
  - \( X \) ranges over argument categories (e.g., NP and PP).
  - The rules are order-preserving, e.g., (14) can turn an NP into a rightward-looking function over leftward functions, preserving the linear order of subjects and predicates.
Argument Cluster Coordination

Likewise, argument cluster coordination is licensed:

\[
\begin{align*}
\lambda &\rightarrow TV/DTV \\
\lambda x &\rightarrow TV/DTV \\
\lambda y &\rightarrow TV/DTV
\end{align*}
\]

where:
- \( VP = S/NP \)
- \( TV (\text{transitive verb}) = VP/NP \)
- \( DTV (\text{ditransitive verb}) = (VP/NP)/NP \)

Non-standard surface structures are licensed throughout

- The non-traditional constituents motivated for right-node raising and similar coordinations are also possible constituents of non-coordinate sentences like Marcel proved completeness.

\[
\begin{align*}
\text{Marcel} &\rightarrow NP:marcel \\
S_{1}/(S_{2} (NP):MNP) &\rightarrow (S_{2} (NP):NP:prove') \\
\lambda &\rightarrow NP:completeness' \\
S_{1}/NP &\rightarrow S_{2}/NP:prove' \rightarrow marcel' \\
\lambda &\rightarrow S_{2}/NP:completeness' \\
S_{1}/NP &\rightarrow S_{2}/NP:prove' \rightarrow completeness' marcel' \\
\lambda &\rightarrow S_{2}/NP:completeness' \\
S_{1}/NP &\rightarrow S_{2}/NP:prove' \rightarrow completeness' marcel'
\end{align*}
\]

The Principle of Type Transparency guarantees that all such non-standard derivations yield identical interpretations.

Extraction

Try to work out derivations for the following object extractions:

19. what John knows (NP)
20. What does John know?

Remember:
- Categories can select for complex categories
- Wh-pronouns (e.g., what) can select for gapped constructions

Non-standard surface structures: incremental processing

One potential advantage of CCG is that you can derive sentences in a completely incremental, left-to-right fashion (some type-raising omitted here) (Dowty 1988):

\[
\begin{align*}
1 &\rightarrow S/VP \\
S_{1}/VP &\rightarrow S_{2}/VP \\
S_{1}/NP &\rightarrow S_{2}/VP \\
S_{1}/NP &\rightarrow S_{2}/VP \\
S_{1}/NP &\rightarrow S_{2}/NP \\
S_{1}/NP &\rightarrow S_{2}/NP
\end{align*}
\]

Constituent Condition on Rules

Traditionally, there has been the notion of the Constituent Condition on Rules:

The inputs and outputs of all rules of syntax should be constituents

Thus, CCG has to claim one of the following:
1. This claim is not true.
2. Items like Marcel proved can be constituents.

Non-standard surface structures: information structure

According to Steedman, the non-standard surface structures are not spurious ambiguities but relevant since they subsume the intonation structures needed to explain the possible intonation contours for sentences of English.

- Intonational boundaries contribute to determining which of the possible combinatory derivations is intended.
- The interpretations of the constituents that arise from these derivations are related to semantic distinctions of information structure and discourse focus.
- Steedman’s claims:
  - Where intonational boundaries are present, they contribute to disambiguation.
  - Conversely, any such boundaries must be consistent with some syntactic derivation, or ill-formedness will result.
Examples for impossible intonation boundaries

    b. * (Seymour prefers the nuts) (and bolts approach).
    c. * (They only asked whether John knew the woman who chaired) (the zoning board).

Syntactic structure and intonation

Steedman’s claims:
- Surface structure and information structure coincide, the latter simply consisting in the interpretation associated with a constituent analysis of the sentence.
- Intonation coincides with surface structure, and hence information structure, in the sense that all intonational boundaries coincide with syntactic boundaries (but not all syntactic boundaries are intonationally marked).

Syntactic structure and intonation (cont.)

As a result, fragments such as Marcel proved in (22c), are not only prosodic constituents but surface syntactic constituents, complete with interpretations.

(22) a. Marcel proved completeness.
    b. [S Marcel [VP proved completeness ] ]
    c. [S [IP Marcel proved ] completeness ]

Parsing with CCG

To deal with the exponential potential of type-raising, CCG parsers can do different things (see Hockenmaier & Steedman 2002)
- Logical form check: Put logical/semantic forms into a chart & only add a new constituent when form isn’t already in there
- Last resort: Only use the rules of type-raising and functional composition when no other parse works

NB: The Penn Treebank has been put into CCG format and is available as the CCGBank.

Summary

- CG has complex lexical entries with syntactic and semantic information and a few rules to apply to them.
- CCG adds combinators, which allows for flexible constituency and all sorts of coordination phenomena.
- The “non-constituent” clustering of information is claimed to be consistent with intonational/information structure.