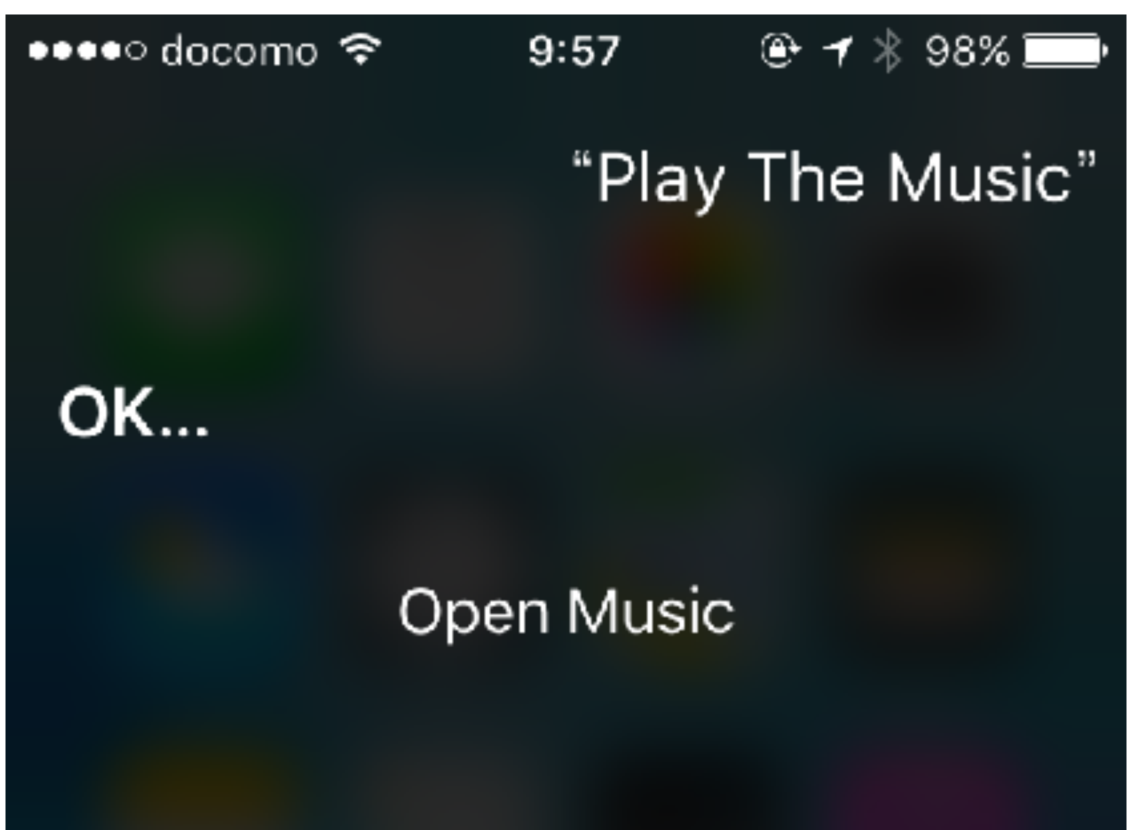
Syntactic and semantic parsing for natural language understanding

1. Introduction and Combinatory Categorical Grammars

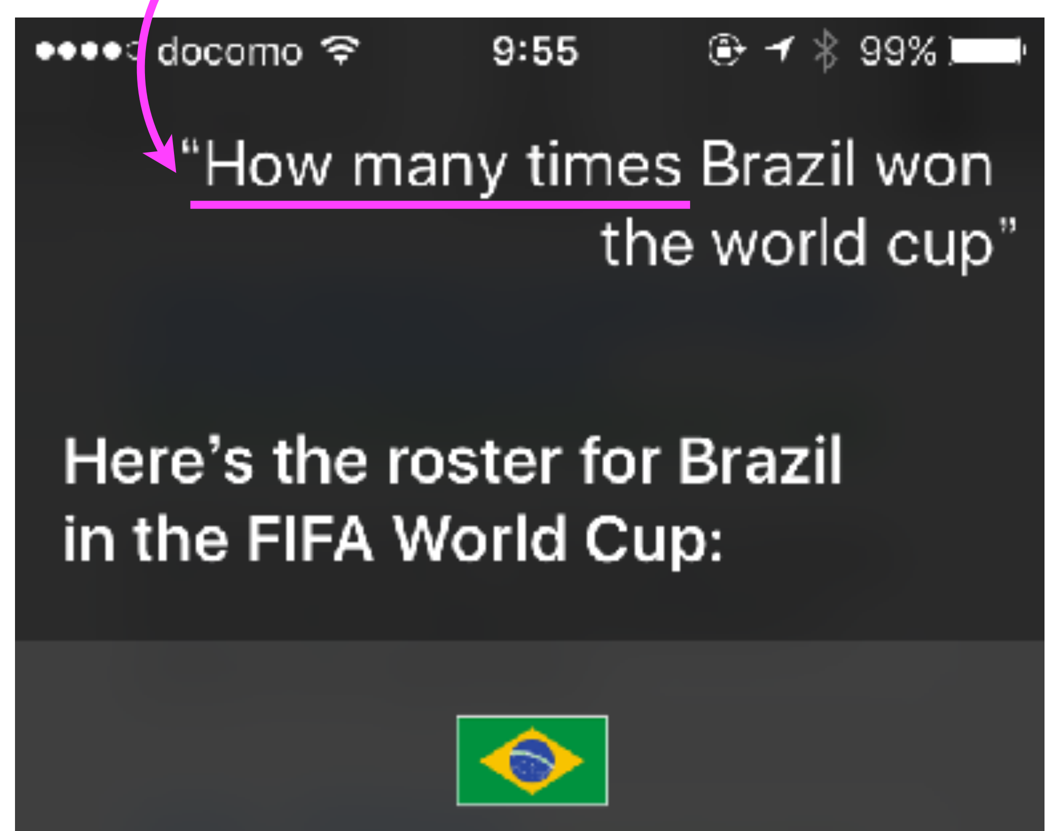
Hiroshi Noji

Natural language understanding?

- An ultimate goal of natural language processing (NLP)
- Example: dialogue system
  - current systems can understand some basic commands
  - but they cannot understand complex utterances people naturally use



need to count the events

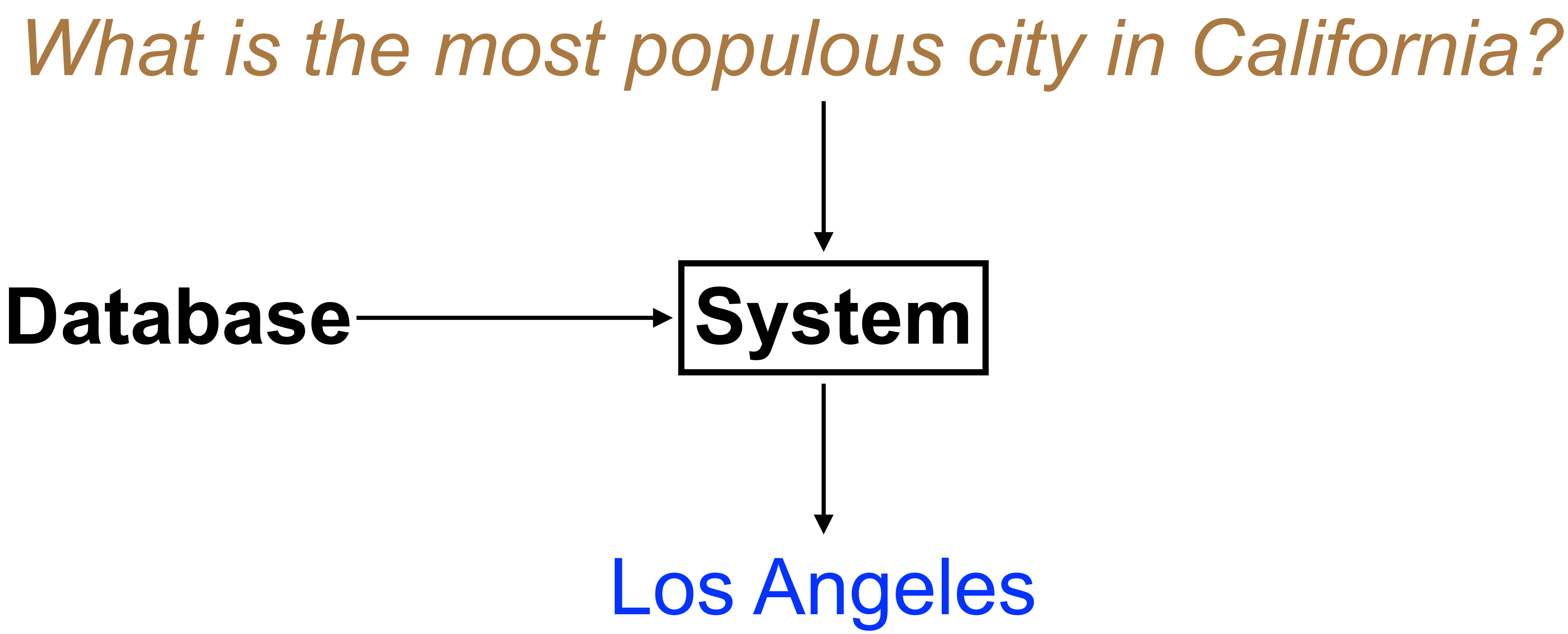


Goal of this course

- Getting insights on the current **state-of-the-art** techniques around natural language **understanding**
- In particular, we focus on two NLP tasks related to language understanding
  - Question answering
  - Recognizing textual entailment (RTE)
  - These tasks require **deep understanding of the text**

Question answering

- **Main focus:** question answering with external **database**
- System may convert an input question into a query to the database, and obtain the answer



Recognizing textual entailment (RTE)

- **Task:** judge whether the input sentences (premise; 前提) semantically **entail** another sentence (hypothesis; 仮定)
  - Given **P**, **H** is true or false?

P Smoking in restaurants is prohibited by low in most cities in Japan

---

H Some cities does not allow smoking in public spaces

⇒ **true (entail)**, because **most cities does not mean all cities**

  - *The best way of testing an NLP system's semantic capacity* (Cooper, et al., 1994)
  - Applications: Summarization, QA on articles, etc.

Approach

- In this course, we focus on a specific approach called **semantic parsing** on these tasks
- Semantic parsing:
  - Converting a natural language utterance into the corresponding **meaning representation** (typically some **logical forms**)
- Example in question answering:

What is the most populous city in California?

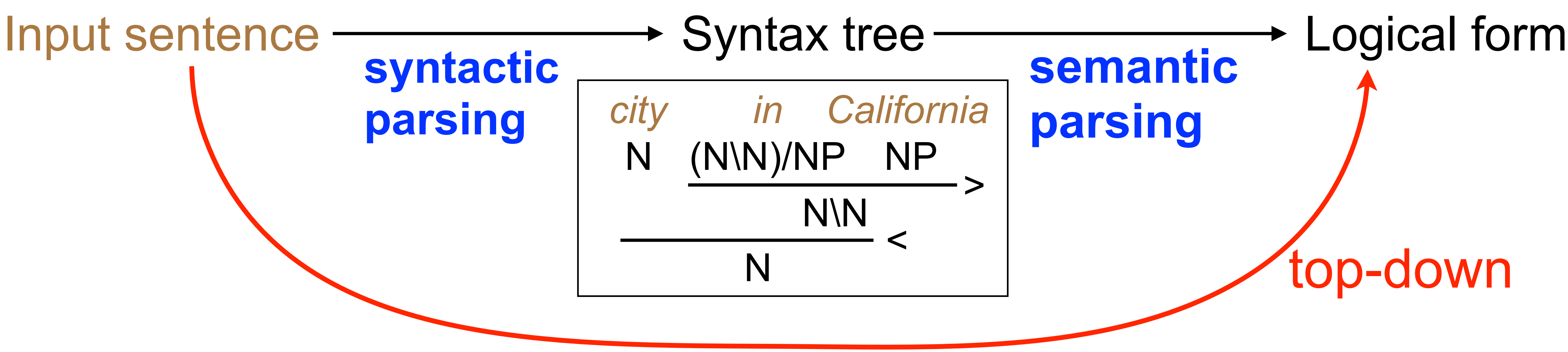
$$\big\downarrow \textbf{ semantic parsing}$$

$$\textbf{argmax}(\lambda x.\textbf{city}(x) \wedge \textbf{loc}(x, \textbf{CA}), \lambda x.\textbf{populous}(x))$$

- We may convert the output into other form (e.g., SQL) by some rules

Two approaches for semantic parsing

- There are two different approaches broadly
- **Bottom-up** approach (also called **non-grounding** approach)
  - Given a sentence, first find its syntactic structure, and then convert it to a logical form



- **Top-down** approach (called **grounding** approach)
  - We do not use an external syntactic parser (direct conversion)

Both have pros and cons

Plan of the course

- Schedule and grading:
  - 4 lectures: **12/14** (today); **12/19** (Tue); **12/21** (Thu); **1/4** (Thu)
  - 100% grading by an assignment (report) given in the last lecture
- Materials:
  - Combinatory categorial grammar (CCG)
    - Linguistically-motivated grammar suitable for converting natural language into logical forms
  - Techniques for efficient CCG parsing
  - Bottom-up and top-down semantic parsing with CCG
  - Semantic parsing without CCG (lightly-supervised approach), and challenges for the future research

today

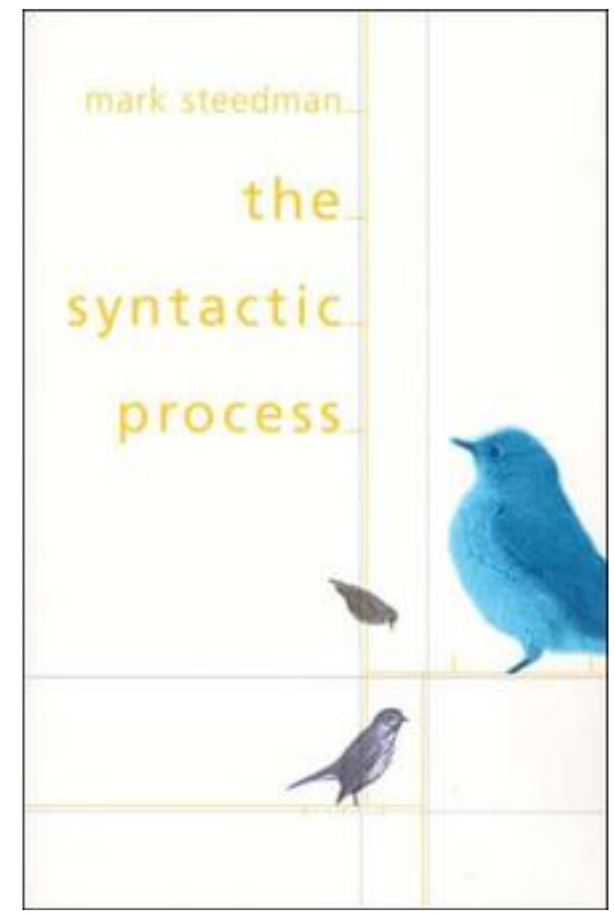
12/14

12/19,
1/4

What is CCG?

- For NLP, Combinatory Categorical Grammar (CCG) is a grammar that makes computation of logical forms easier

- For linguistics, CCG is known as a lexicalized grammar, which recognizes **mildly context-sensitive languages** (slightly beyond **context-free languages**, on Chomsky hierarchy)
  - Formal linguists (syntacticians) are interested in the grammar that best fits the generative capacity of the human languages
  - CCG is one of such attempts
  - Other related grammars: HPSG, TAG, LFG



(MIT Press, 2000)

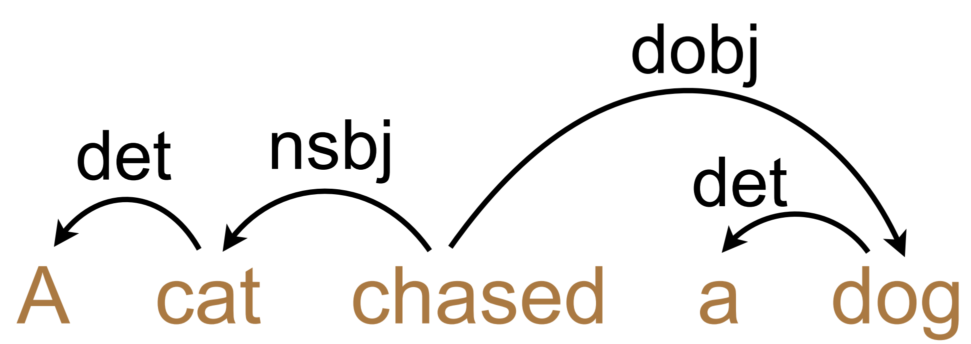
Outline

- Brief introduction to (syntactic) parsing
  - Phrase structure and dependency grammars
- Combinatory Categorical Grammar (CCG)
  - First introduce Categorical Grammar (CG)
  - CCG can be obtained by extending CG
- Properties of CCG
  - Transparency to the logical form
  - Ability to handle long-distance dependencies

What is (syntactic) parsing?

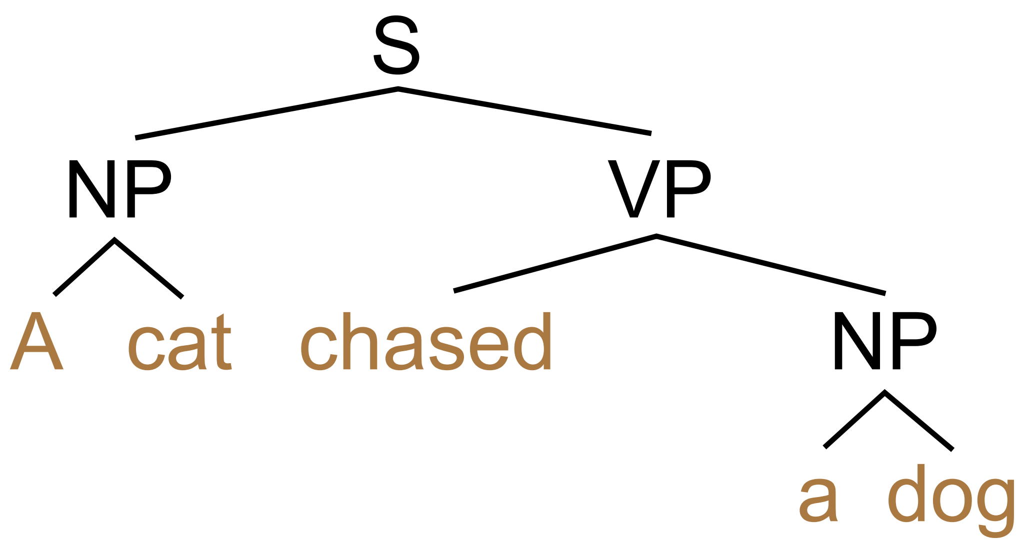
- Linguists like to use **trees** to describe the syntactic structure of the sentence
- Parsing is the problem to find the correct tree given an input sentence
- Different grammar describes a sentence in a different way

Dependency grammar



Focus: relationships between two words

Phrase-structure grammar

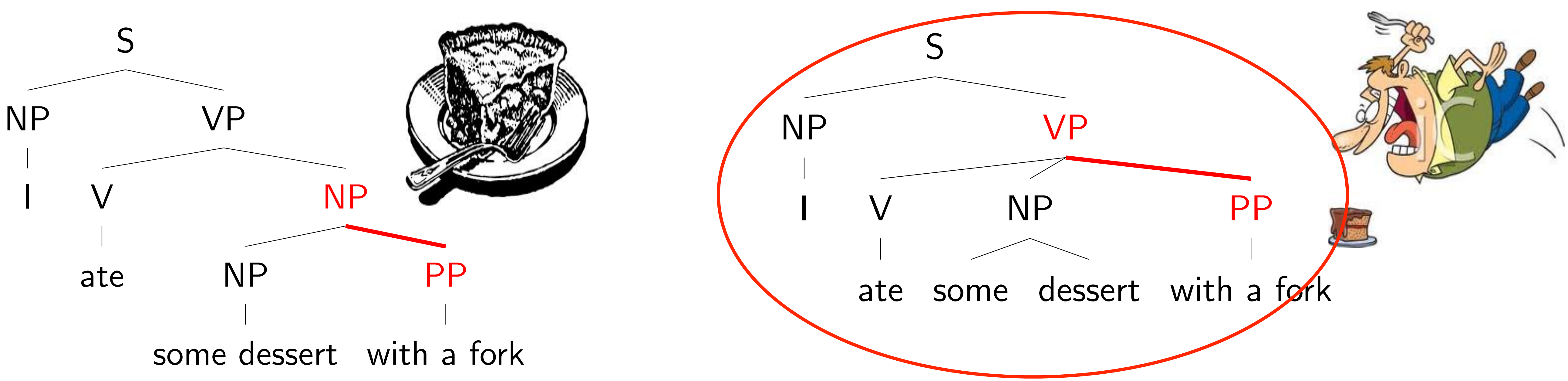


Focus: recursive structure

Goal of parsing: disambiguation

- What is the correct tree?
  - correct tree is the one **corresponding to human interpretation**
- Parsing aims at finding a tree that corresponds to human interpretation
- Example: PP-attachment ambiguity

I ate some dessert with a fork



Figures from Natural Language Understanding: Foundations and State-of-the-Art (Liang, 2015)

Statistical parsing since 2000's

- Current state-of-the-art parsers are mostly **statistical parsers** that learn the mapping from a sentence into a tree with **supervised machine learning (recently deep-learning)**
  - Typically we use 10,000~30,000 trees to train a parser
- Popular parsers:
  - English: Stanford parser https://stanfordnlp.github.io/CoreNLP/
  - Japanese: CaboCha https://taku910.github.io/cabocha/
- Today we do not touch on statistical disambiguation models
- **We focus on the roles of grammars (syntax theory)**

Outline

- Brief introduction to (syntactic) parsing
  - Phrase structure and dependency grammars
- Combinatory Categorial Grammar (CCG)
  - First introduce Categorial Grammar (CG)
  - CCG can be obtained by extending CG
- Properties of CCG
  - Transparency to the logical form
  - Ability to handle long-distance dependencies

Categorical Grammar

- Fundamental units: Category
  - Atomic category: S, NP, N
  - Complex category:
    - consists of atomic categories and “/” “\”
    - X/Y means it accepts Y in the right and becomes X
    - X\Y means it accepts Y in the left and becomes X
- Example:
  - Intransitive verbs (e.g., walk): S\NP
  - Transitive verbs (e.g., eat): S\NP/NP

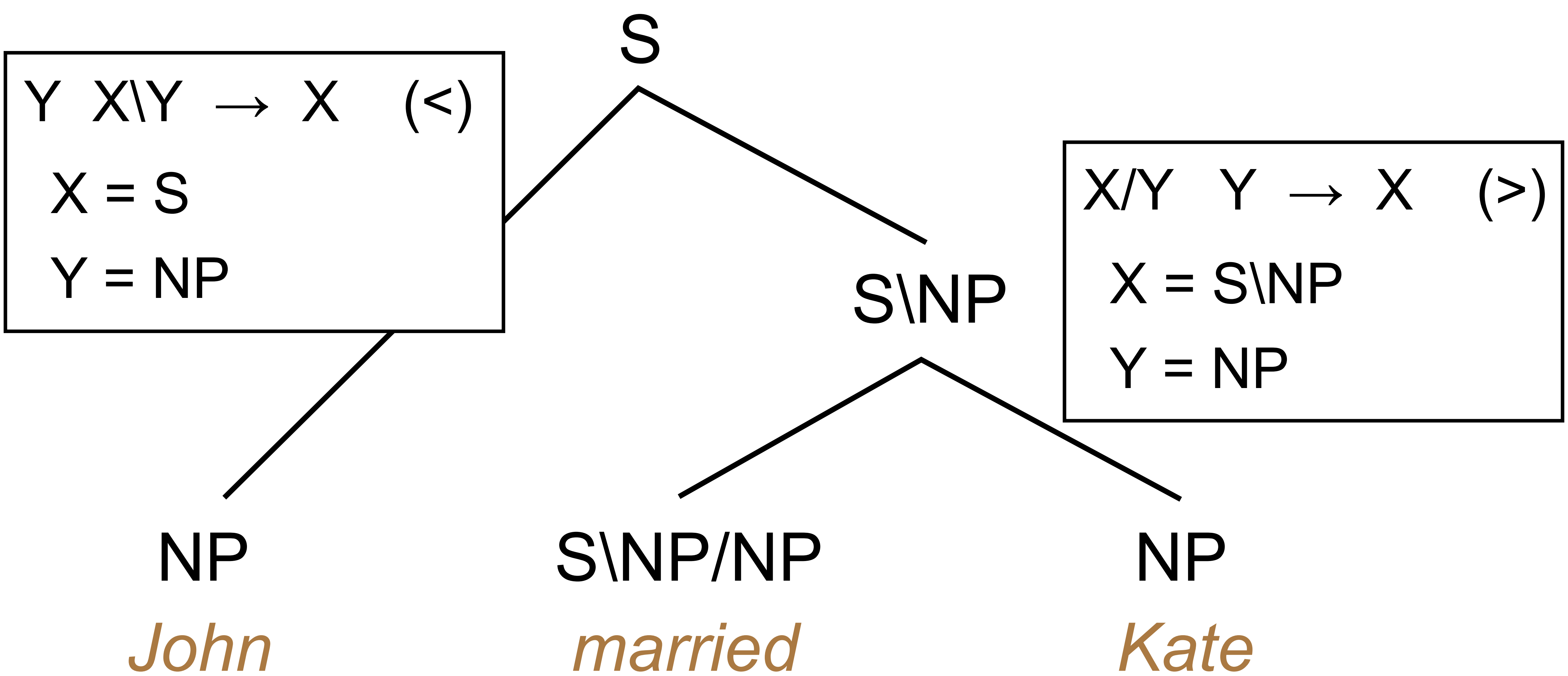
NP	S\NP	→	S
John	*walked*		

Rules

- There are only two rules in CG
- Function application rules:
  - X/Y   Y   →   X   (>)
  - Y   X\Y   →   Y   (<)
- Example:
  - S/NP   NP   →   S
  - NP   S\NP   →   S

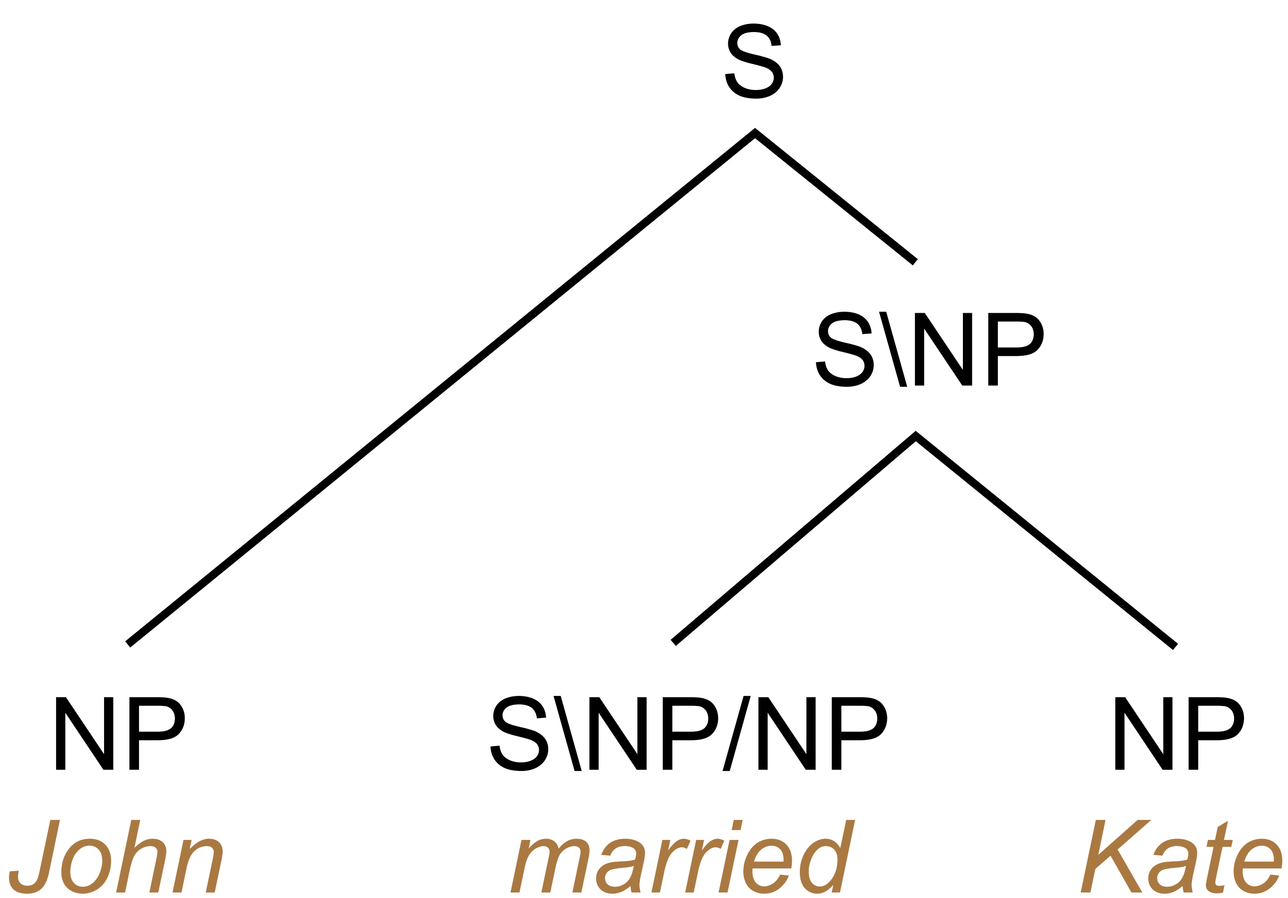
How parsing proceeds

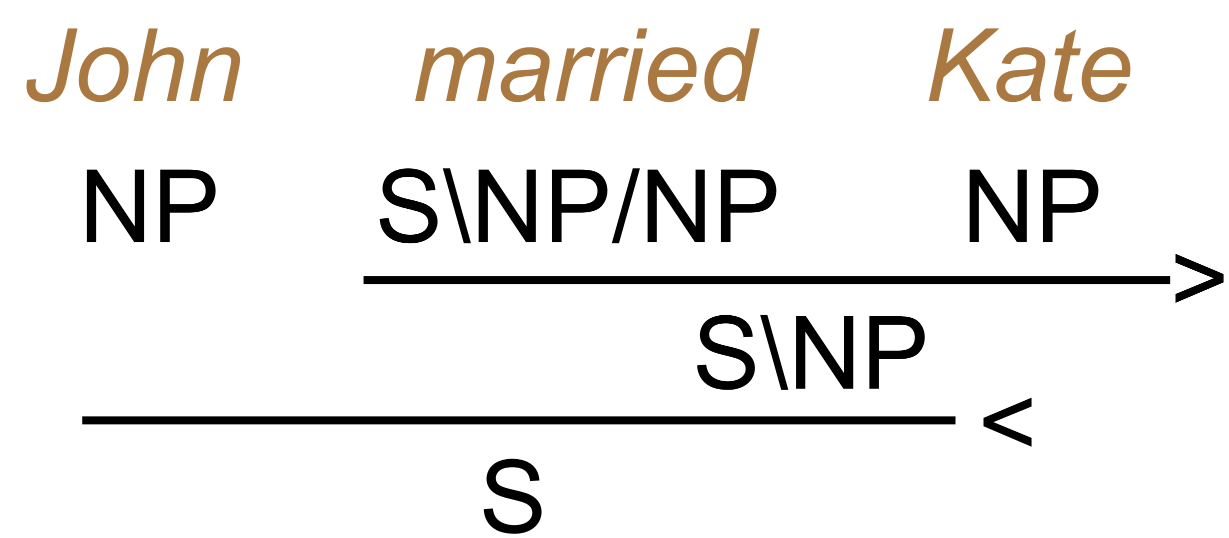
- Assign lexical categories to words
- Apply a rule to categories to build tree



Parsing as derivation

- In CG and CCG, we usually write down a parsing process as a logical derivation of a proof





Typical categories

- Modifier have the category X/X or X\X
  - Adjective (*a **tall** man, a **great** book*): N/N
  - Adverb (*He runs **quickly***): (S\NP)\(S\NP)
- Many other categories can be understood intuitively
  - We can understand **S\NP** as a category, who wants to become **S** but lacks **NP** in the left (intransitive verbs)
  - ***in*** in "*He walks **in** the park*"?
    - (S\NP)\(S\NP)/NP
    - *"the park"* is NP
    - *"**in** the park"* becomes modifier modifying *walks* (S\NP)

A bit complex case

A	*young*	*boy*	*was*	*walking*	*in*	*the*	*park*
NP/N	N/N	N	(S\NP)/(S\NP)	S\NP	(S\NP)\(S\NP)/NP	NP/N	N

N/N N → N (>)

NP/N N → NP (>)

NP/N N → NP (>)

(S\NP)\(S\NP)/NP NP → (S\NP)\(S\NP) (>)

S\NP (S\NP)\(S\NP) → S\NP (<)

(S\NP)/(S\NP) S\NP → S\NP (>)

NP S\NP → S (<)

CG to CCG

- Pure CG has only function application rules (<, >)
  - This is not sufficient to cover all linguistic phenomena
  - Formally, its generative capacity is equivalent to CFG
- CCG is a CG with some extended rules
  - Extended rules elegantly solve many problems of natural language syntax
    - e.g., coordination and long-distance dependencies

CG	CCG
categories function application	+ extended rules

Simple coordination

John *met* *and* *married* *Kate*

NP S\NP/NP CONJ S\NP/NP NP

met and married: S\NP/NP (Φ)

met and married Kate: S\NP (>)

John met and married Kate: S (<)

Coordination rule:

X CONJ X → X (Φ)

Another coordination

John	*met*	*and*	*might*	*married*	*Kate*	
NP	S\NP/NP	CONJ	(S\NP)/(S\NP)	S\NP/NP	NP	
				S\NP		>
			S\NP			<

Different categories
⇒ cannot apply coordination rule

Function composition

John	*met*	*and*	*might*	*married*	*Kate*
NP	S\NP/NP	CONJ	(S\NP)/(S\NP)	S\NP/NP	NP

>B: (S\NP)/NP

Φ: S\NP/NP

>: S\NP

<: S

- **Idea:** make the category of *might married* S\NP/NP, the same as *met* to apply the coordination rule

Function composition rule

$$X/Y \quad Y/Z \quad \rightarrow \quad X/Z \qquad (>B)$$

$$Y\backslash Z \quad X\backslash Y \quad \rightarrow \quad X\backslash Z \qquad (<B)$$

What does composition?

John	*met*	*and*	*might*	*married*	*Kate*
NP	S\NP/NP	CONJ	(S\NP)/(S\NP)	S\NP/NP	NP

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

- It changes the order of function applications
- Originally, *married* has to take right NP (*Kate*), and then combines with *might*
- By composition, we can first combine *might* and *married*

Another rule: type raising

Type raising rule

$$X \rightarrow T/(T\backslash X) \qquad (>T)$$

$$X \rightarrow T\backslash(T/X) \qquad (<T)$$

- T can be any category
- Example: $NP \rightarrow S/(S\backslash NP) \quad (>T)$
  - X = NP, T = S
  - S/(S\NP) takes right S\NP (verb phrase) to become S
  - We can change the subject NP into a **function** receiving a verb phrase

Type raising for relative clauses

The man | *that* | *I* | *saw*

NP | (NP\NP)/(S/NP) | NP | (S\NP)/NP

S/(S\NP) >T

S/NP >B

NP\NP >

NP >

- Type-raising and then forward composition enables the analysis of relative clauses (that, who, which)
- Again, it changes the order of filling arguments

Outline

- Brief introduction to (syntactic) parsing
  - Phrase structure and dependency grammars
- Combinatory Categorial Grammar (CCG)
  - First introduce Categorial Grammar (CG)
  - CCG can be obtained by extending CG
- Properties of CCG
  - Transparency to the logical form
  - Ability to handle long-distance dependencies

What we want is logical forms

- Example:
  - *Some woman ordered tea*
  - Logical form: $\exists x.(woman(x) \wedge \exists y.(tea(y) \wedge order(x, y)))$
- Different logical form for a sentence with different semantics
  - *All women ordered coffee or tea*
  - Logical form: $\forall x.(women(x) \rightarrow \exists y.((tea(y) \vee cofffee(y)) \wedge order(x, y)))$
- We can use these logical forms to calculate whether one sentence entails the other (next week)
- For QA, logical forms can be a query to DB

$$\textbf{argmax}(\lambda x.\textbf{city}(x) \wedge \textbf{loc}(x,\textbf{CA}),\ \lambda x.\textbf{populous}(x))$$

CCG and logical forms

- One attractive property of CCG is its transparency between syntax and semantics (among syntactic theories)
- By assigning a logical form to each word, the sentence logical form can be obtained automatically
  - some: *λFλG.∃x.(Fx∧Gx)*
  - woman: *λx.woman(x)*

$$
\dfrac{
  \dfrac{
    \begin{array}{c}\text{Some}\\ NP/N\\ \lambda F\lambda G.\exists x.(Fx \wedge Gx)\end{array}
    \quad
    \begin{array}{c}\text{woman}\\ N\\ \lambda x.\mathsf{woman}(x)\end{array}
  }{
    \begin{array}{c}NP\\ \lambda G.\exists x(\mathsf{woman}(x) \wedge G(x))\end{array}
  }>
  \quad
  \dfrac{
    \begin{array}{c}\text{ordered}\\ (S\backslash NP)/NP\\ \lambda Q_1\lambda Q_2.Q_2(\lambda x.Q_1(\lambda y.\mathsf{order}(x,y)))\end{array}
    \quad
    \dfrac{
      \begin{array}{c}\text{tea}\\ N\\ \lambda y.\mathsf{tea}(y)\end{array}
    }{
      \begin{array}{c}NP\\ \lambda F.\exists y.(\mathsf{tea}(y) \wedge F(y))\end{array}
    }\text{lex}
  }{
    \begin{array}{c}S\backslash NP\\ \lambda Q_2.Q_2(\lambda x.\exists y.(\mathsf{tea}(y) \wedge \mathsf{order}(x,y)))\end{array}
  }>
}{
  \begin{array}{c}S\\ \exists x.(\mathsf{woman}(x) \wedge \exists y.(\mathsf{tea}(y) \wedge \mathsf{order}(x,y)))\end{array}
}<
$$

Figure from Martínez-Gómez et al., 2016. ccg2lambda: A Compositional Semantics System.

Lexicon = category + logical form

- So far, each word (lexicon) is only associated with a category
  - e.g., likes: S\NP/NP
- In general, each word is associated with a category and its meaning representation (typically lambda expression)
  - likes: S\NP/NP: $\lambda y.\lambda x.like(x,y)$
- Each rule also defines how to calculate logical forms

Function application rules:

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

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

Beta-reduction

- When a rule is applied, the logical form is updated with beta-reduction

$$\dfrac{\begin{array}{ccc} \ldots & \text{likes} & \text{Bob} \\ & S\backslash NP/NP\text{: } \lambda y.\lambda x.like(x,y) & NP\text{: } bob \end{array}}{S\backslash NP\text{: } \lambda x.like(x,bob)} >$$

- $\lambda y.\lambda x.like(x,y)$ is a function that next takes y
- Forward application results in $\lambda y.\lambda x.like(x,y)\ (bob)$
- Substituting *bob* to *y, we get* $\lambda x.like(x,bob)$

Advantage of CCG

- Transparency between syntax and semantics (logical forms)
- Every rule defines the way to combine logical forms
- **Point:** we can define the way to combine logical forms because CCG only has a small number of rules
  - This is impossible for other grammars, e.g., ordinary phrase-structure grammars, where the number of rules is huge (> 100)

$S \rightarrow NP\ VP$	$VP \rightarrow VP\ PP$
$NP \rightarrow DT\ NN$	$VP \rightarrow V\ NN$
$NP \rightarrow NP\ PP$	$VP \rightarrow V$
$NP \rightarrow DT$	…

Other rules with semantics

Function composition rule

X/Y: f $\quad Y/Z$: g $\quad \rightarrow \quad X/Z$: $\lambda x.f(g\ x)$ $\quad$ (>B)

$Y\backslash Z$: g $\quad X\backslash Y$: f $\quad \rightarrow \quad X\backslash Z$: $\lambda x.f(g\ x)$ $\quad$ (<B)

Type raising rule

X: a $\quad \rightarrow \quad T/(T\backslash X)$: $\lambda a.f\ a$ $\quad$ (>T)

X: a $\quad \rightarrow \quad T\backslash(T/X)$: $\lambda a.f\ a$ $\quad$ (<T)

Spurious ambiguity

- **Important property of CCG**: different syntax derivations lead to the same semantics

$$
\dfrac{\mathit{John} \atop \text{NP}: \mathit{john} \quad \dfrac{\mathit{married} \atop \text{S}\backslash\text{NP/NP}: \lambda y.\lambda x.\mathit{married}(x,y) \quad \mathit{Kate} \atop \text{NP}: \mathit{kate}}{\text{S}\backslash\text{NP}: \lambda x.\mathit{married}(x,\mathit{kate})}>}{\text{S}: \mathit{married}(\mathit{john},\mathit{kate})}>
$$

$$
\dfrac{\dfrac{\dfrac{\mathit{John} \atop \text{NP}: \mathit{john}}{\text{S/(S}\backslash\text{NP)}: \lambda f.f(\mathit{john})}{>}\text{T} \quad \mathit{married} \atop \text{S}\backslash\text{NP/NP}: \lambda y.\lambda x.\mathit{married}(x,y)}{\text{S/NP}: \lambda x.\mathit{married}(\mathit{john},x)}{>}\text{B} \quad \mathit{Kate} \atop \text{NP}: \mathit{kate}}{\text{S}: \mathit{married}(\mathit{john},\mathit{kate})}>
$$

Beta-reduction in composition

$$\frac{S/(S\backslash NP): \lambda f.f(john) \qquad S\backslash NP/NP: \lambda y.\lambda x.married(x,y)}{S/NP: \lambda x.married(john,x)} >B$$

Function composition rule

$$X/Y: f \quad Y/Z: g \quad \rightarrow \quad X/Z: \lambda x.f(g\ x) \quad (>B)$$

$$Y\backslash Z: g \quad X\backslash Y: f \quad \rightarrow \quad X\backslash Z: \lambda x.f(g\ x) \quad (<B)$$

x is already used so we avoid

$\lambda x'.(\lambda f.f(john)\ (\lambda y.\lambda x.married(y,x)\ (x')))$

$\Rightarrow \lambda x'.(\lambda f.f(john)\ (\lambda x.married(x',x)))$

$\Rightarrow \lambda x'.\ ((\lambda x.married(x',x))\ (john))$

$\Rightarrow \lambda x'.(married(x',john)) \Rightarrow \lambda x.married(x,john)$

Messages

- In CCG, syntactic derivation itself is not meaningful
  - Rules are designed so that different derivations can lead to the same semantics
  - The semantics is irrelevant to the order of applications
  - Tree = “syntactic process” to obtain the semantics
- Important properties:
  - Transparency between syntax and semantics
  - Syntax is governed by a few rules, and is language independent
  - Much information is encoded into the lexicon, which is language dependent

some: NP/N: $\lambda F \lambda G.\exists x.(Fx \wedge Gx)$

every: NP/N: $\lambda F \lambda G.\forall x.(Gx \rightarrow Fx)$