CS6741: Structured Prediction for NLP
Fall 2015

Semantic Parsing: Lambda Calculus and CCG

Instructor: Yoav Artzi

Parsing | Learning | Modeling

- Lambda calculus
- Parsing with Combinatory Categorial Grammars
- Linear CCGs
- Factored lexicons

Lambda Calculus

- Formal system to express computation
- Allows high-order functions

$$\lambda a.move(a) \wedge dir(a, LEFT) \wedge to(a, \iota y.chair(y)) \wedge$$
$$pass(a, \mathcal{A}y.sofa(y) \wedge intersect(\mathcal{A}z.intersection(z), y))$$

[Church 1932]

Lambda Calculus
Base Cases

- Logical constant
- Variable
- Literal
- Lambda term

Lambda Calculus
Logical Constants

- Represent objects in the world

$$NYC, CA, RAINIER, LEFT, \ldots$$
$$located\_in, depart\_date, \ldots$$

Lambda Calculus
Variables

- Abstract over objects in the world
- Exact value not pre-determined

$$x, y, z, \ldots$$

Lambda Calculus
Literals

- Represent function application

$$city(AUSTIN)$$

$$located\_in(AUSTIN, TEXAS)$$

Lambda Calculus
Literals

- Represent function application

$$city(AUSTIN)$$

$$located\_in(AUSTIN, TEXAS)$$

Predicate

Arguments

Logical expression

List of logical expressions

Lambda Calculus
Lambda Terms

- Bind/scope a variable
- Repeat to bind multiple variables

$$\lambda x.city(x)$$

$$\lambda x.\lambda y.located\_in(x, y)$$

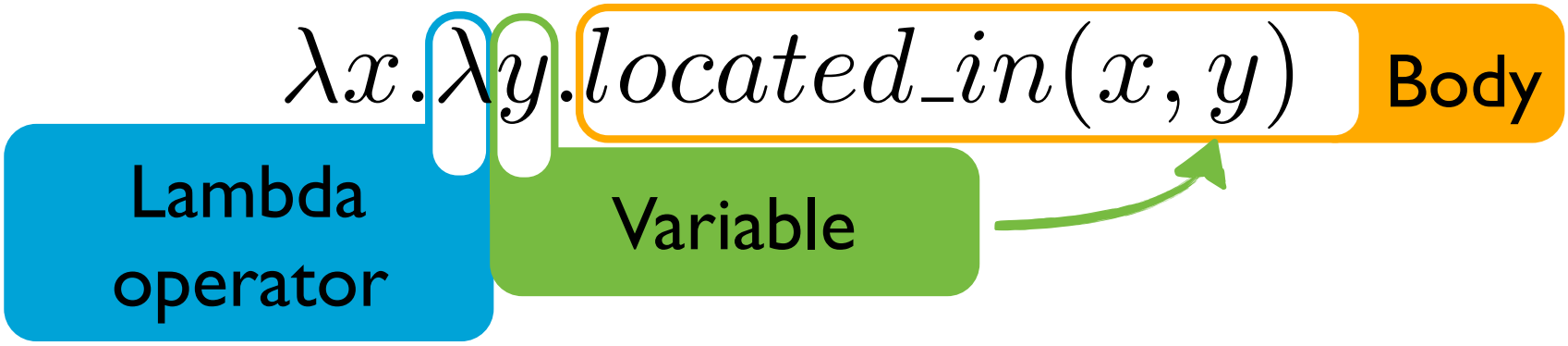
Lambda Calculus

Lambda Terms

- Bind/scope a variable
- Repeat to bind multiple variables

$$\lambda x.city(x)$$

$$\lambda x.\lambda y.located\_in(x, y)$$



Lambda Calculus
Quantifiers?

- Higher order constants
- No need for any special mechanics
- Can represent all of first order logic

$$\forall(\lambda x.big(x) \wedge apple(x))$$
$$\neg(\exists(\lambda x.lovely(x))$$
$$\iota(\lambda x.beautiful(x) \wedge grammar(x))$$

Lambda Calculus

Syntactic Sugar

$$\wedge\left(A, \wedge(B, C)\right) \Leftrightarrow A \wedge B \wedge C$$

$$\vee\left(A, \vee(B, C)\right) \Leftrightarrow A \vee B \vee C$$

$$\neg(A) \Leftrightarrow \neg A$$

$$\mathcal{Q}(\lambda x.f(x)) \Leftrightarrow \mathcal{Q}x.f(x)$$

$$\text{for } \mathcal{Q} \in \{\iota, \mathcal{A}, \exists, \forall\}$$

Lambda Calculus

$$\lambda a.pre(a, \iota x.chair(x)) \wedge move(a) \wedge len(a, 3) \wedge$$
$$dir(a, forward) \wedge past(a, \iota y.sofa(y))$$

$$\lambda x.flight(x) \wedge to(x, move)$$
$$\lambda x.flight(x) \wedge to(x, NYC)$$
$$\lambda x.NYC(x) \wedge x(to, move)$$

✗ $\lambda x.flight(x) \wedge to(x, move)$

✓ $\lambda x.flight(x) \wedge to(x, NYC)$

✗ $\lambda x.NYC(x) \wedge x(to, move)$

Simply-typed Lambda Calculus

- Like lambda calculus
- But, typed

✗ $\lambda x.flight(x) \wedge to(x, move)$

✓ $\lambda x.flight(x) \wedge to(x, NYC)$

✗ $\lambda x.NYC(x) \wedge x(to, move)$

[Church 1940]

Lambda Calculus

Typing

t Truth-value

e Entity

- Simple types
- Complex types

$$< e, t >$$

$$<< e, t >, e >$$

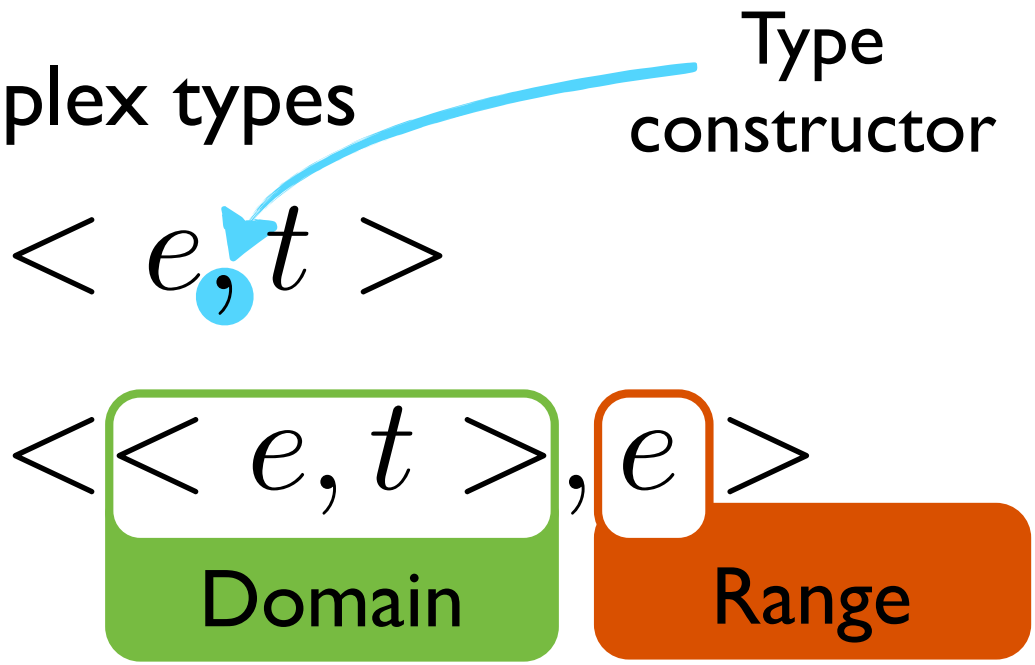
Lambda Calculus

Typing

t Truth-value

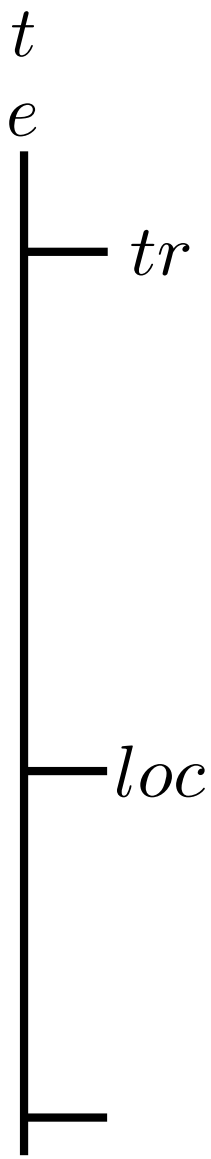
e Entity

- Simple types
- Complex types



Lambda Calculus

Typing



- Simple types
- Complex types

Type constructor

$$< e, t >$$

$$< < e, t >, e >$$

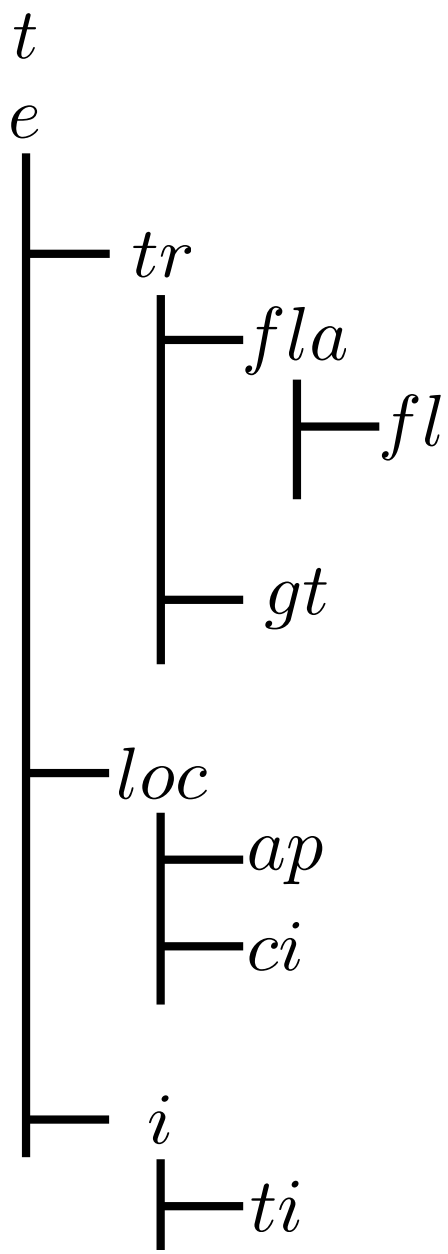
Domain

Range

- Hierarchical typing system

Lambda Calculus

Typing



- Simple types
- Complex types

Type constructor

$$< e, t >$$

$$< < e, t >, e >$$

Domain

Range

- Hierarchical typing system

Simply-typed Lambda Calculus

Base Cases

- Logical constant
- Variable
- Literal
- Lambda term

Lambda Calculus
Logical Constants

- Represent objects in the world

$$\mathrm{NYC}_{ci}, \mathrm{CA}_{st}, \mathrm{RAINIER}_{mt}, \mathrm{LEFT}_{dir}, \ldots$$
$$\mathrm{located\_in}_{\langle e,\langle loc,t\rangle\rangle}, \mathrm{depart\_date}_{\langle tr,\langle dt,t\rangle\rangle}, \ldots$$

Lambda Calculus
Variables

- Abstract over objects in the world
- Exact value not pre-determined

$$x_{ci}, y_{dir}, z_{\langle e,\langle loc,t\rangle\rangle}, \cdots$$

Lambda Calculus
Literals

- Represent function application

$$t: \quad \text{city}_{\langle loc,t \rangle}(\text{AUSTIN}_{ci})$$

$$t: \quad \text{located\_in}_{\langle e,\langle loc,t \rangle \rangle}(\text{AUSTIN}_{ci}, \text{TEXAS}_{st})$$

Lambda Calculus
Lambda Terms

- Bind/scope a variable
- Repeat to bind multiple variables

$$\langle loc, t\rangle : \quad \lambda x_{loc}.\mathrm{city}_{\langle loc,t\rangle}(x)$$

$$\langle e, \langle loc, t\rangle\rangle : \quad \lambda x_{ci}.\lambda y_{st}.\mathrm{located\_in}_{\langle e,\langle loc,t\rangle\rangle}(x_{ci}, y_{st})$$

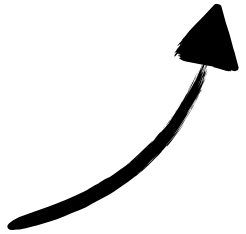
Function Application in Lambda Calculus

$(P_{\langle \alpha,\beta \rangle})(A_{\alpha'})$

s.t. α' inherits α

If $P_{\langle \alpha,\beta \rangle} = \lambda x_\alpha . B_\beta$, can reduce:

$(P_{\langle \alpha,\beta \rangle})(A_{\alpha'}) = (\lambda x_\alpha . B_\beta)(A_{\alpha'}) = B_\beta[x_\alpha \mapsto A_{\alpha'}]$



Replace all occurrences

Function Application in Lambda Calculus

$$((\lambda x_{ci}.\lambda y_{st}.\text{located\_in}_{\langle e,\langle loc,t\rangle\rangle}(x_{ci}, y_{st}))(\text{BOS}_{ci}))(\text{NY}_{st})$$

Function Application in Lambda Calculus

$$((\lambda x_{ci}.\lambda y_{st}.\text{located\_in}_{\langle e,\langle loc,t\rangle\rangle}(x_{ci}, y_{st}))(\text{BOS}_{ci}))(\text{NY}_{st})$$
$$\mapsto (\lambda y_{st}.\text{located\_in}_{\langle e,\langle loc,t\rangle\rangle}(\text{BOS}_{ci}, y_{st}))(\text{NY}_{st})$$
$$\mapsto \text{located\_in}_{\langle e,\langle loc,t\rangle\rangle}(\text{BOS}_{ci}, \text{NY}_{st})$$

Function Application in Lambda Calculus

$$(\lambda f_{\langle\langle e,t\rangle,\langle e,t\rangle\rangle}.f(\lambda x_e.\mathrm{book}_{\langle e,t\rangle}(x)))(\lambda g_{\langle e,t\rangle}.\lambda y_e.g(y) \wedge \mathrm{red}_{\langle e,t\rangle}(y))$$

Function Application in Lambda Calculus

$$(\lambda f_{\langle\langle e,t\rangle,\langle e,t\rangle\rangle}.f(\lambda x_e.\mathrm{book}_{\langle e,t\rangle}(x)))(\lambda g_{\langle e,t\rangle}.\lambda y_e.g(y) \wedge \mathrm{red}_{\langle e,t\rangle}(y))$$
$$\mapsto (\lambda g_{\langle e,t\rangle}.\lambda y_e.g(y) \wedge \mathrm{red}_{\langle e,t\rangle}(y))(\lambda x_e.\mathrm{book}_{\langle e,t\rangle}(x))$$
$$\mapsto \lambda y_e.(\lambda x_e.\mathrm{book}_{\langle e,t\rangle}(x))(y) \wedge \mathrm{red}_{\langle e,t\rangle}(y)$$
$$\mapsto \lambda y_e.\mathrm{book}_{\langle e,t\rangle}(y) \wedge \mathrm{red}_{\langle e,t\rangle}(y)$$

Function Composition

$$g_{\langle\langle e,t\rangle,\langle e,t\rangle\rangle} = \lambda g_{\langle e,t\rangle}.\lambda x_e.g(x) \wedge \text{yellow}(x)$$
$$f_{\langle\langle e,t\rangle,\langle e,t\rangle\rangle} = \lambda k_{\langle e,t\rangle}.\lambda y_e.k(y) \wedge \text{square}(y)$$

$$= (f_{\langle\langle e,t\rangle,\langle e,t\rangle\rangle} \cdot g_{\langle\langle e,t\rangle,\langle e,t\rangle\rangle})_{\langle\langle e,t\rangle,\langle e,t\rangle\rangle}$$

Function Composition

$g_{\langle\langle e,t\rangle,\langle e,t\rangle\rangle} = \lambda g_{\langle e,t\rangle}.\lambda x_e.g(x) \wedge \text{yellow}(x)$

$f_{\langle\langle e,t\rangle,\langle e,t\rangle\rangle} = \lambda k_{\langle e,t\rangle}.\lambda y_e.k(y) \wedge \text{square}(y)$

Let $A_{\langle e,t\rangle}$ be a $\langle e,t\rangle$-typed logical expression

$g_{\langle\langle e,t\rangle,\langle e,t\rangle\rangle}(A_{\langle e,t\rangle}) = (\lambda g_{\langle e,t\rangle}.\lambda x_e.g(x) \wedge \text{yellow}(x))(A_{\langle e,t\rangle}) =$

$(\lambda x_e.g(x) \wedge \text{yellow}(x))[g \mapsto A_{\langle e,t\rangle}] =$

$\lambda x_e.A_{\langle e,t\rangle}(x) \wedge \text{yellow}(x)$

$f_{\langle\langle e,t\rangle,\langle e,t\rangle\rangle}(g_{\langle\langle e,t\rangle,\langle e,t\rangle\rangle}(A_{\langle e,t\rangle})) =$

$(\lambda k_{\langle e,t\rangle}.\lambda y_e.k(y) \wedge \text{square}(y))(\lambda x_e.A_{\langle e,t\rangle}(x) \wedge \text{yellow}(x)) =$

$(\lambda y_e.k(y) \wedge \text{square}(y))[k \mapsto \lambda x_e.A_{\langle e,t\rangle}(x) \wedge \text{yellow}(x)] =$

$\lambda y_e.(\lambda x_e.A_{\langle e,t\rangle}(x) \wedge \text{yellow}(x))(y) \wedge \text{square}(y)) =$

$\lambda y_e.(A_{\langle e,t\rangle}(x) \wedge \text{yellow}(x))[x \mapsto y] \wedge \text{square}(y)) =$

$\lambda y_e.A_{\langle e,t\rangle}(y) \wedge \text{yellow}(y) \wedge \text{square}(y))$

Add a new variable and mapping: $\lambda z_{\langle e,t\rangle}.f_{\langle\langle e,t\rangle,\langle e,t\rangle\rangle}(g_{\langle\langle e,t\rangle,\langle e,t\rangle\rangle}(A_{\langle e,t\rangle}))[A \mapsto z] =$

$(\lambda z_{\langle e,t\rangle}.\lambda y_e.A_{\langle e,t\rangle}(y) \wedge \text{yellow}(y) \wedge \text{square}(y)))[A \mapsto z] =$

$\lambda z_{\langle e,t\rangle}.\lambda y_e.z(y) \wedge \text{yellow}(y) \wedge \text{square}(y)) = (f_{\langle\langle e,t\rangle,\langle e,t\rangle\rangle} \cdot g_{\langle\langle e,t\rangle,\langle e,t\rangle\rangle})_{\langle\langle e,t\rangle,\langle e,t\rangle\rangle}$

Simply Typed Lambda Calculus

$$\lambda a.move(a) \wedge dir(a, LEFT) \wedge to(a, \iota y.chair(y)) \wedge$$
$$pass(a, \mathcal{A}y.sofa(y) \wedge intersect(\mathcal{A}z.intersection(z), y))$$

Type information usually omitted

Capturing Meaning with Lambda Calculus

$$\lambda a.pre(a, \iota x.chair(x)) \wedge move(a) \wedge len(a, 3) \wedge$$
$$dir(a, forward) \wedge past(a, \iota y.sofa(y))$$

- Flexible representation
- Can capture full complexity of natural language
- Done?

Capturing Meaning with Lambda Calculus

$$\lambda a.pre(a, \iota x.chair(x)) \wedge move(a) \wedge len(a, 3) \wedge$$
$$dir(a, forward) \wedge past(a, \iota y.sofa(y))$$

- Flexible representation
- Can capture full complexity of natural language
- Done? What about the symbols?

Constructing Lambda Calculus Expressions

at the chair, move forward three steps past the sofa

$$\lambda a.pre(a, \iota x.chair(x)) \wedge move(a) \wedge len(a, 3) \wedge$$
$$dir(a, forward) \wedge past(a, \iota y.sofa(y))$$

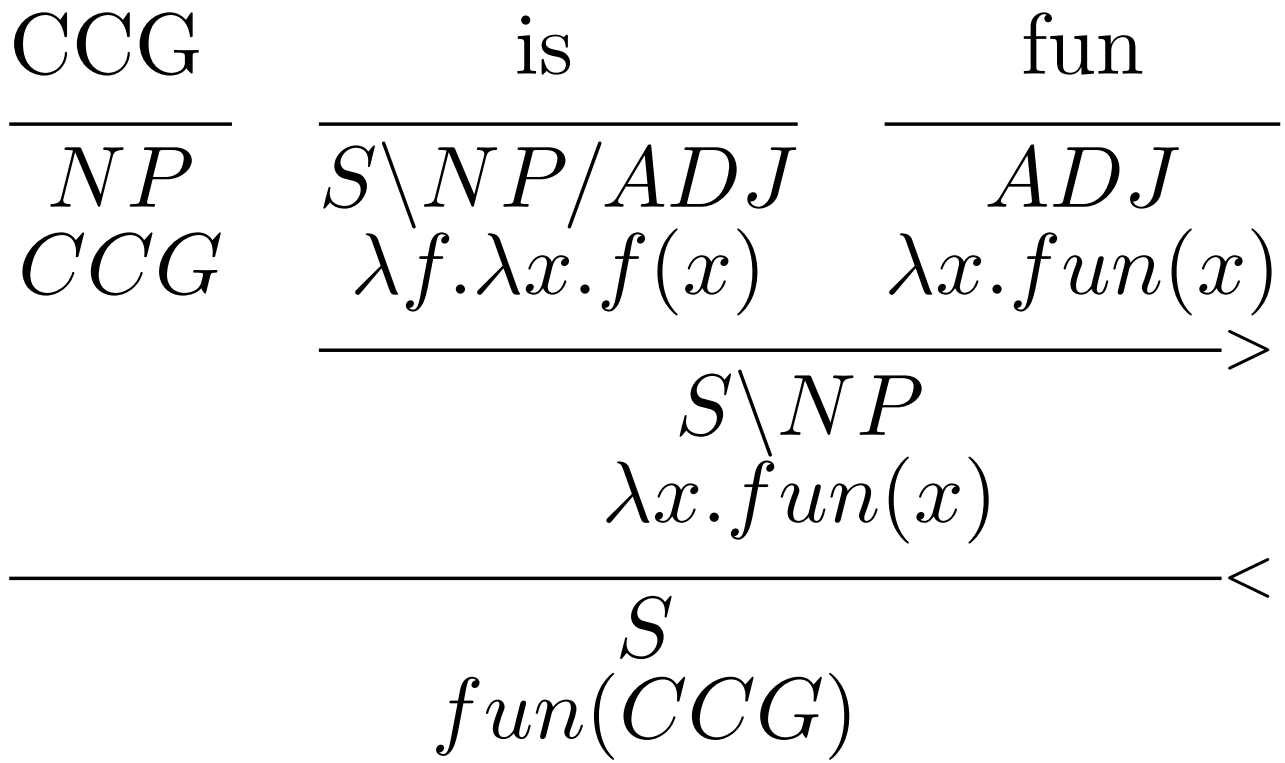
Constructing Lambda Calculus Expressions

move forward

$$\lambda a.move(a) \wedge dir(a, forward)$$

- What operations can we use?
- Build logical forms?

Combinatory Categorial Grammars



[Steedman 1996, 2000]

Combinatory Categorial Grammars

- Appeared around 1980
- Categorial formalism
  - Compositional
  - Puts more information on the words
- Transparent interface between syntax and semantics
- Designed with computation in mind
- Part of a class of mildly context sensitive formalisms (e.g., TAG, HG, LIG) [Joshi et al. 1990]

CCG Categories

$$ADJ : \lambda x.fun(x)$$

- Basic building block
- Capture syntactic and semantic information jointly (what's the difference from CFG?)

CCG Categories

Syntax ADJ : $\lambda x.fun(x)$ Semantics

- Basic building block
- Capture syntactic and semantic information jointly

CCG Categories

Syntax $ADJ : \lambda x.fun(x)$

$$(S\backslash NP)/ADJ : \lambda f.\lambda x.f(x)$$

$$NP : CCG$$

- Primitive symbols: N, S, NP, ADJ and PP
- Syntactic combination operator (/,\)
- Slashes specify argument order and direction

CCG Categories

$$ADJ : \lambda x.fun(x)$$

Semantics

$$(S\backslash NP)/ADJ : \lambda f.\lambda x.f(x)$$

$$NP : CCG$$

- λ-calculus expression
- Syntactic type maps to semantic type

CCG Lexical Entries

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

- Pair words and phrases with meaning
- Meaning captured by a CCG category

CCG Lexical Entries

- Pair words and phrases with meaning
- Meaning captured by a CCG category

CCG Lexicons

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

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

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

- Pair words and phrases with meaning
- Meaning captured by a CCG category

Parsing with CCGs

Use lexicon to match words and phrases with their categories

Parsing with CCGs

CCG	is	fun
NP	$S\backslash NP/ADJ$	ADJ
CCG	$\lambda f.\lambda x.f(x)$	$\lambda x.fun(x)$

Use lexicon to match words and phrases with their categories

CCG Operations

- Small set of operators
  - Input: 1-2 CCG categories
  - Output: A single CCG category
- Operate on syntax semantics together
- Mirror natural logic operations

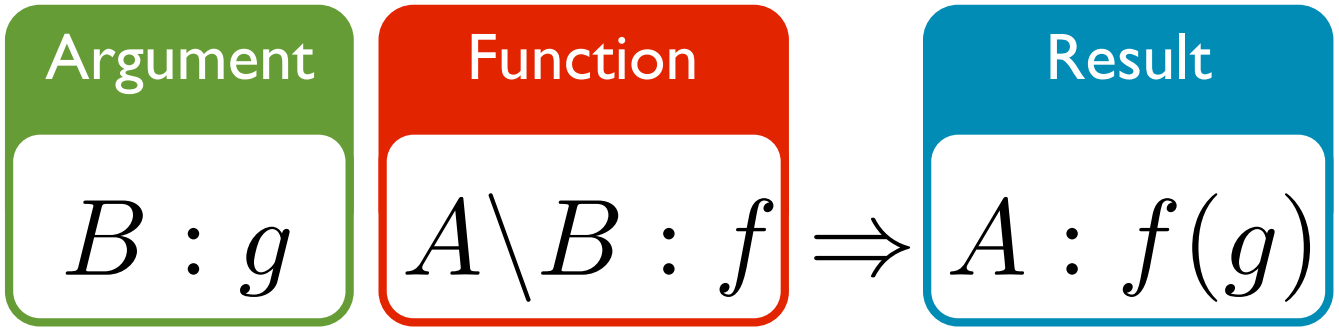
CCG Operations
Application

$$B : g \quad A\backslash B : f \Rightarrow A : f(g) \quad (<)$$

$$A/B : f \quad B : g \Rightarrow A : f(g) \quad (>)$$

- Equivalent to function application
- Two directions: forward and backward
  - Determined by slash direction

CCG Operations
Application



$$A/B : f \quad B : g \Rightarrow A : f(g) \quad (>)$$

- Equivalent to function application
- Two directions: forward and backward
  - Determined by slash direction

Parsing with CCGs

CCG	is	fun
NP	$S\backslash NP/ADJ$	ADJ
CCG	$\lambda f.\lambda x.f(x)$	$\lambda x.fun(x)$

Use lexicon to match words and phrases with their categories

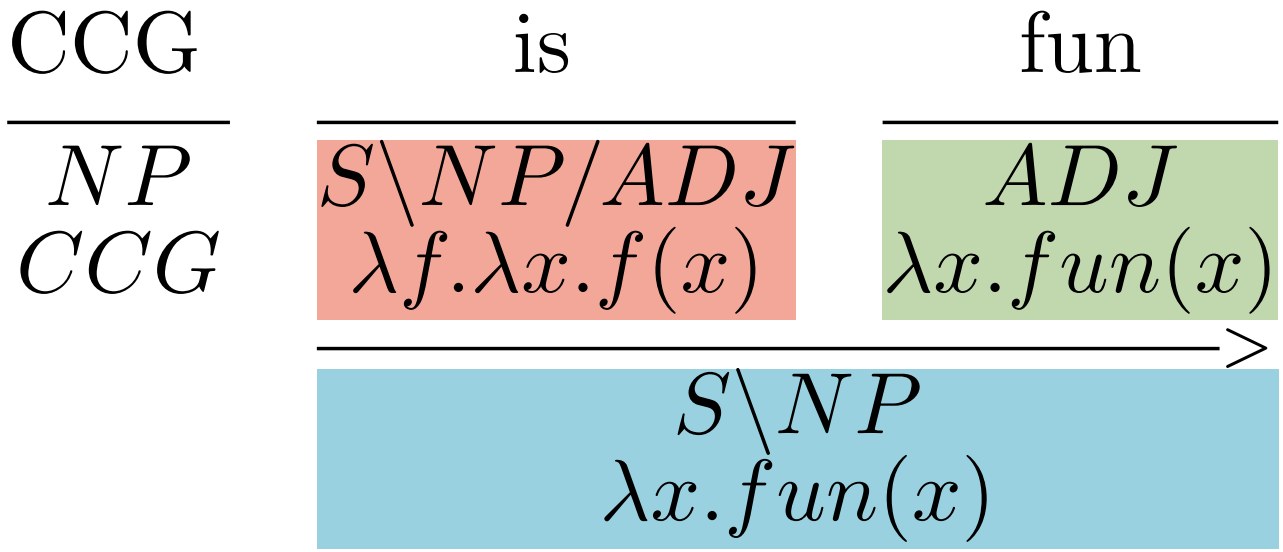
Parsing with CCGs

CCG	is	fun
NP	$S\backslash NP/ADJ$	ADJ
CCG	$\lambda f.\lambda x.f(x)$	$\lambda x.fun(x)$

Combine categories using operators

$$A/B : f \quad B : g \Rightarrow A : f(g) \quad (>)$$

Parsing with CCGs



Combine categories using operators

$$A/B : f \quad B : g \Rightarrow A : f(g) \quad (>)$$

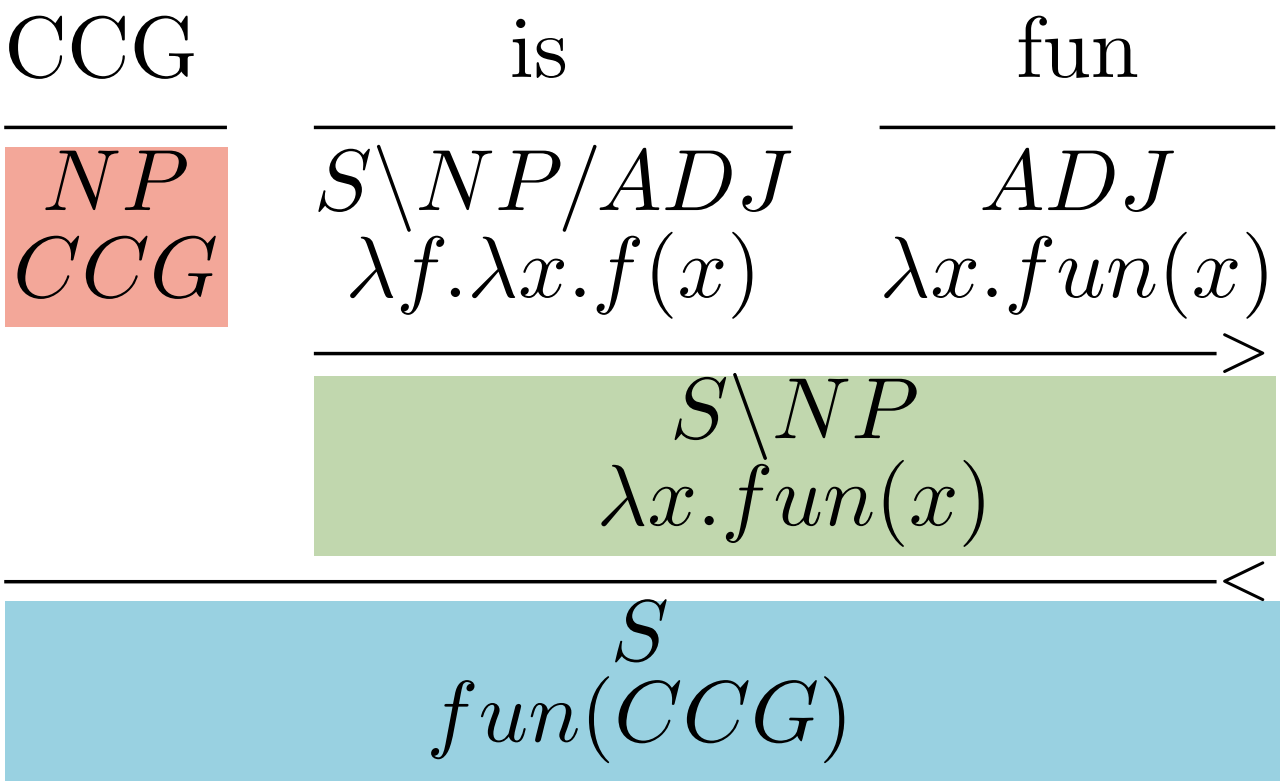
Parsing with CCGs

$$\dfrac{\dfrac{\dfrac{\text{CCG}}{\begin{array}{c}NP\\CCG\end{array}} \quad \dfrac{\text{is}}{\begin{array}{c}S\backslash NP/ADJ\\ \lambda f.\lambda x.f(x)\end{array}} \quad \dfrac{\text{fun}}{\begin{array}{c}ADJ\\ \lambda x.fun(x)\end{array}}}{S\backslash NP \quad \lambda x.fun(x)}\,>}{S\backslash NP \quad \lambda x.fun(x)}$$

Combine categories using operators

$$B : g \quad A\backslash B : f \Rightarrow A : f(g) \quad (<)$$

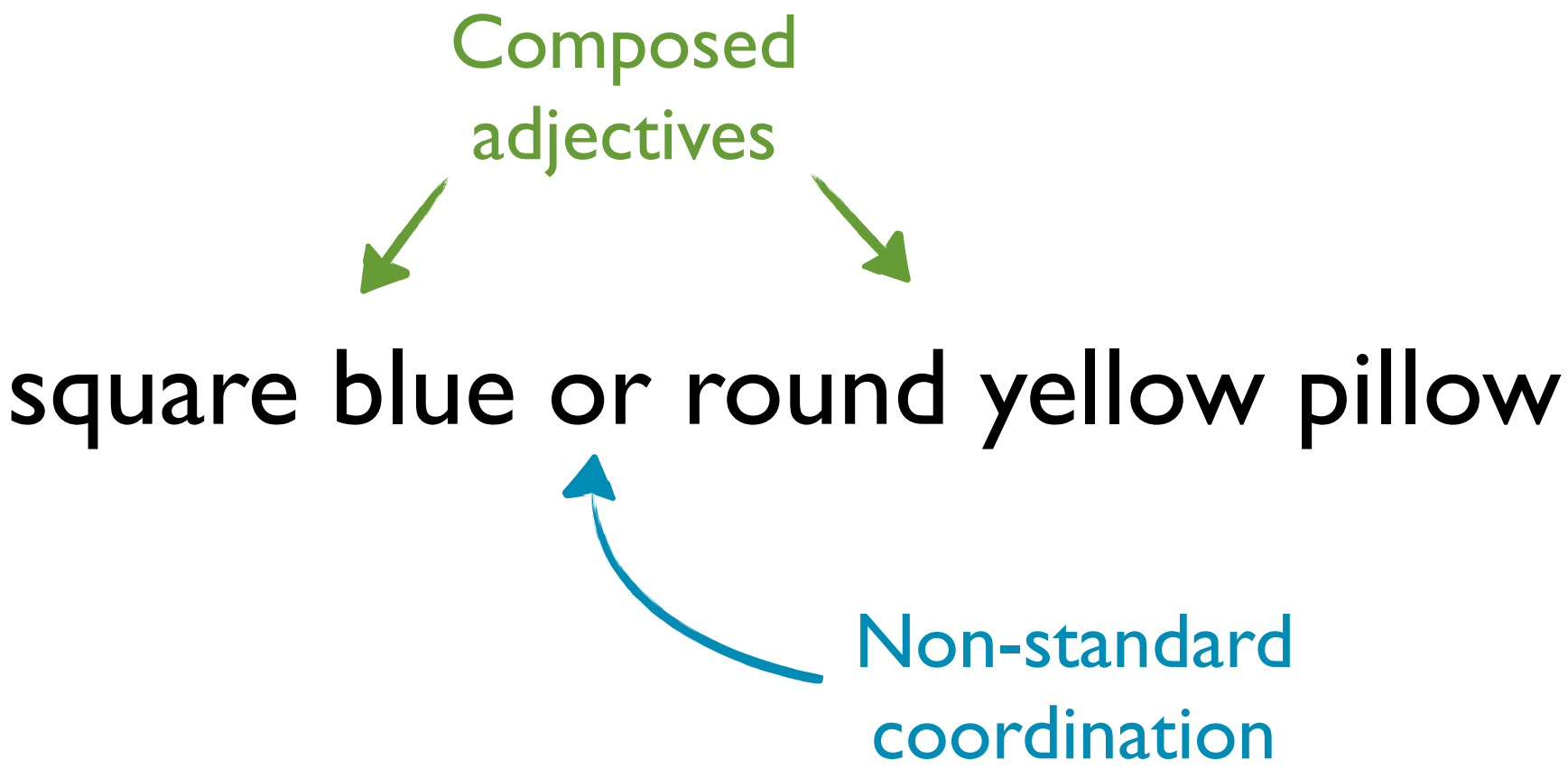
Parsing with CCGs



Combine categories using operators

$$B : g \quad A\backslash B : f \Rightarrow A : f(g) \quad (<)$$

Parsing with CCGs



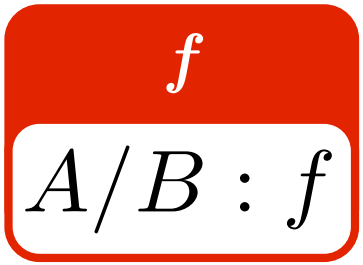
CCG Operations
Composition

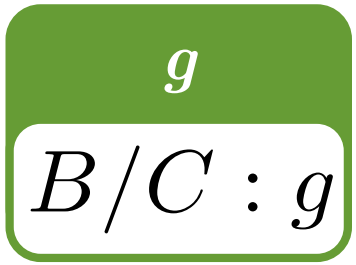
$$A/B : f \quad B/C : g \Rightarrow A/C : \lambda x.f(g(x)) \quad (> B)$$

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

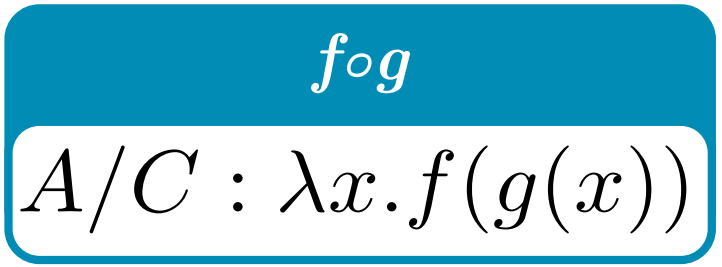
- Equivalent to function composition
- Two directions: forward and backward

CCG Operations
Composition





$\Rightarrow$



$(> B)$

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

- Equivalent to function composition
- Two directions: forward and backward

CCG Operations
Type Shifting

$$ADJ : \lambda x.g(x) \Rightarrow N/N : \lambda f.\lambda x.f(x) \wedge g(x)$$

$$PP : \lambda x.g(x) \Rightarrow N\backslash N : \lambda f.\lambda x.f(x) \wedge g(x)$$

$$AP : \lambda e.g(e) \Rightarrow S\backslash S : \lambda f.\lambda e.f(e) \wedge g(e)$$

$$AP : \lambda e.g(e) \Rightarrow S/S : \lambda f.\lambda e.f(e) \wedge g(e)$$

- Category-specific unary operations
- Modify category type to take an argument
- Helps in keeping a compact lexicon

CCG Operations
Type Shifting

Input		Output
$ADJ : \lambda x.g(x)$	$\Rightarrow$	$N/N : \lambda f.\lambda x.f(x) \wedge g(x)$

$$PP : \lambda x.g(x) \Rightarrow N\backslash N : \lambda f.\lambda x.f(x) \wedge g(x)$$

$$AP : \lambda e.g(e) \Rightarrow S\backslash S : \lambda f.\lambda e.f(e) \wedge g(e)$$

$$AP : \lambda e.g(e) \Rightarrow S/S : \lambda f.\lambda e.f(e) \wedge g(e)$$

- Category-specific unary operations
- Modify category type to take an argument
- Helps in keeping a compact lexicon

CCG Operations
Type Shifting

Input		Output
$ADJ : \lambda x.g(x)$	$\Rightarrow$	$N/N : \lambda f.\lambda x.f(x) \wedge g(x)$
$PP : \lambda x.g(x)$	$\Rightarrow$	$N\backslash N : \lambda f.\lambda x.f(x) \wedge g(x)$
$AP : \lambda e.g(e)$	$\Rightarrow$	$S\backslash S : \lambda f.\lambda e.f(e) \wedge g(e)$
Topicalization $AP : \lambda e.g(e)$	$\Rightarrow$	$S/S : \lambda f.\lambda e.f(e) \wedge g(e)$

- Category-specific unary operations
- Modify category type to take an argument
- Helps in keeping a compact lexicon

CCG Operations
Coordination

$$\text{and} \vdash C : conj$$

$$\text{or} \vdash C : disj$$

- Coordination is special cased
  - Specific rules perform coordination
  - Coordinating operators are marked with special lexical entries

Parsing with CCGs

square blue or round yellow pillow

Parsing with CCGs

square	blue	or	round	yellow	pillow

Use lexicon to match words and phrases with their categories

Parsing with CCGs

square	blue	or	round	yellow	pillow
ADJ	ADJ	C	ADJ	ADJ	N
$\lambda x.square(x)$	$\lambda x.blue(x)$	$disj$	$\lambda x.round(x)$	$\lambda x.yellow(x)$	$\lambda x.pillow(x)$

Use lexicon to match words and phrases with their categories

Parsing with CCGs

square	blue	or	round	yellow	pillow
ADJ	ADJ	C	ADJ	ADJ	N
$\lambda x.square(x)$	$\lambda x.blue(x)$	$disj$	$\lambda x.round(x)$	$\lambda x.yellow(x)$	$\lambda x.pillow(x)$

Shift adjectives to combine

$$ADJ : \lambda x.g(x) \Rightarrow N/N : \lambda f.\lambda x.f(x) \wedge g(x)$$

Parsing with CCGs

square	blue	or	round	yellow	pillow
ADJ $\lambda x.square(x)$	ADJ $\lambda x.blue(x)$	C $disj$	ADJ $\lambda x.round(x)$	ADJ $\lambda x.yellow(x)$	N $\lambda x.pillow(x)$
N/N $\lambda f.\lambda x.f(x) \wedge square(x)$					

Shift adjectives to combine

$$ADJ : \lambda x.g(x) \Rightarrow N/N : \lambda f.\lambda x.f(x) \wedge g(x)$$

Parsing with CCGs

square	blue	or	round	yellow	pillow
ADJ	ADJ	C	ADJ	ADJ	N
$\lambda x.square(x)$	$\lambda x.blue(x)$	$disj$	$\lambda x.round(x)$	$\lambda x.yellow(x)$	$\lambda x.pillow(x)$
N/N	N/N		N/N	N/N	
$\lambda f.\lambda x.f(x) \wedge square(x)$	$\lambda f.\lambda x.f(x) \wedge blue(x)$		$\lambda f.\lambda x.f(x) \wedge round(x)$	$\lambda f.\lambda x.f(x) \wedge yellow(x)$	

Shift adjectives to combine

$$ADJ : \lambda x.g(x) \Rightarrow N/N : \lambda f.\lambda x.f(x) \wedge g(x)$$

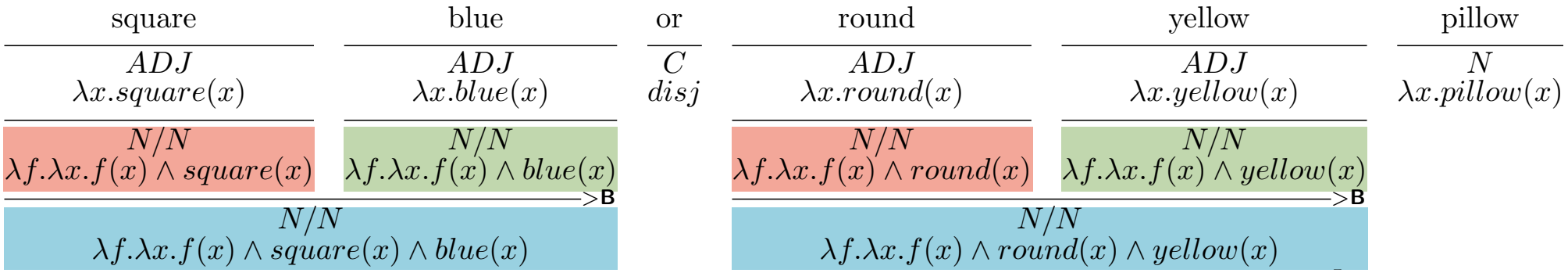
Parsing with CCGs

square	blue	or	round	yellow	pillow
ADJ	ADJ	C	ADJ	ADJ	N
$\lambda x.square(x)$	$\lambda x.blue(x)$	$disj$	$\lambda x.round(x)$	$\lambda x.yellow(x)$	$\lambda x.pillow(x)$
N/N	N/N		N/N	N/N	
$\lambda f.\lambda x.f(x) \wedge square(x)$	$\lambda f.\lambda x.f(x) \wedge blue(x)$		$\lambda f.\lambda x.f(x) \wedge round(x)$	$\lambda f.\lambda x.f(x) \wedge yellow(x)$	

Compose pairs of adjectives

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

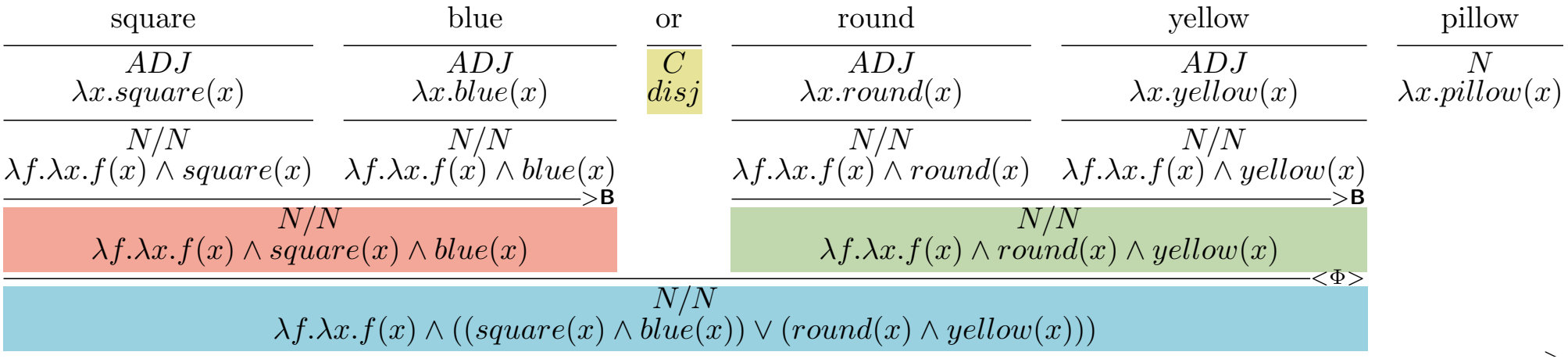
Parsing with CCGs



Compose pairs of adjectives

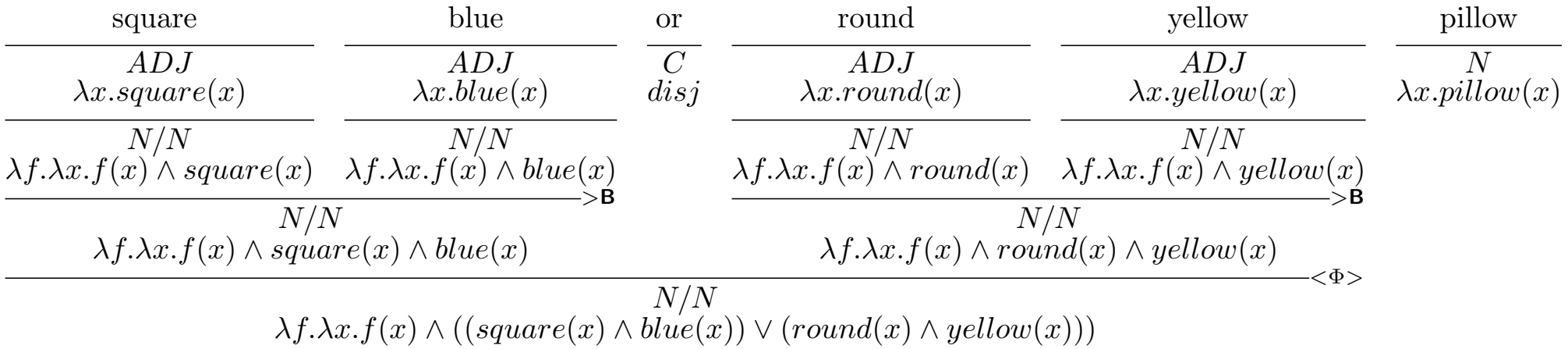
$$A/B : f \quad B/C : g \Rightarrow A/C : \lambda x.f(g(x)) \quad (> B)$$

Parsing with CCGs



Coordinate composed adjectives

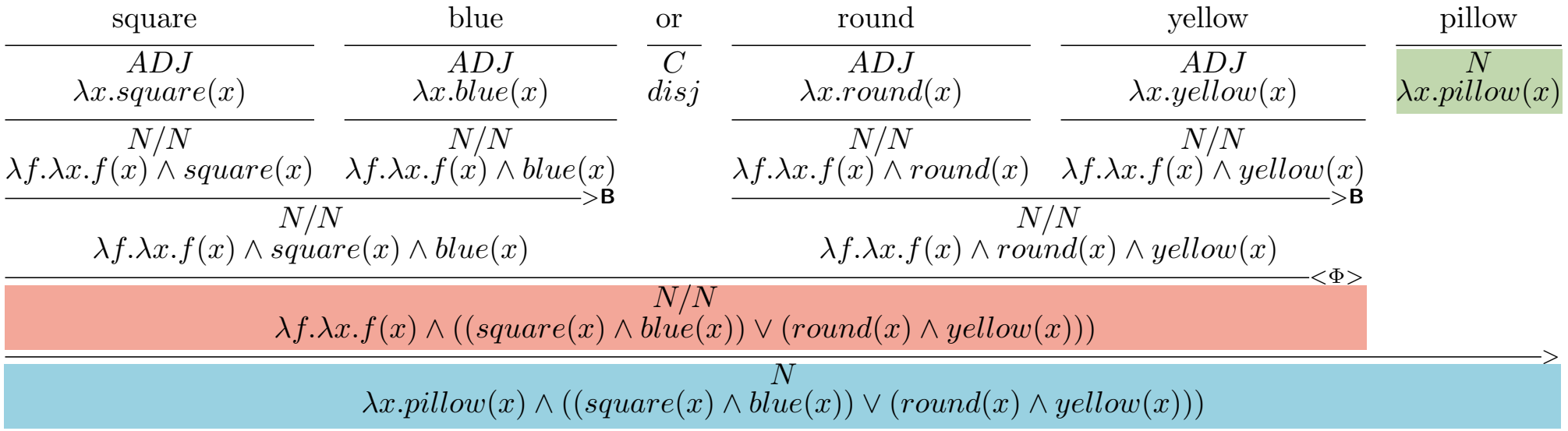
Parsing with CCGs



Apply coordinated adjectives to noun

$$A/B : f \quad B : g \Rightarrow A : f(g) \quad (>)$$

Parsing with CCGs



Apply coordinated adjectives to noun

$$A/B : f \quad B : g \Rightarrow A : f(g) \quad (>)$$

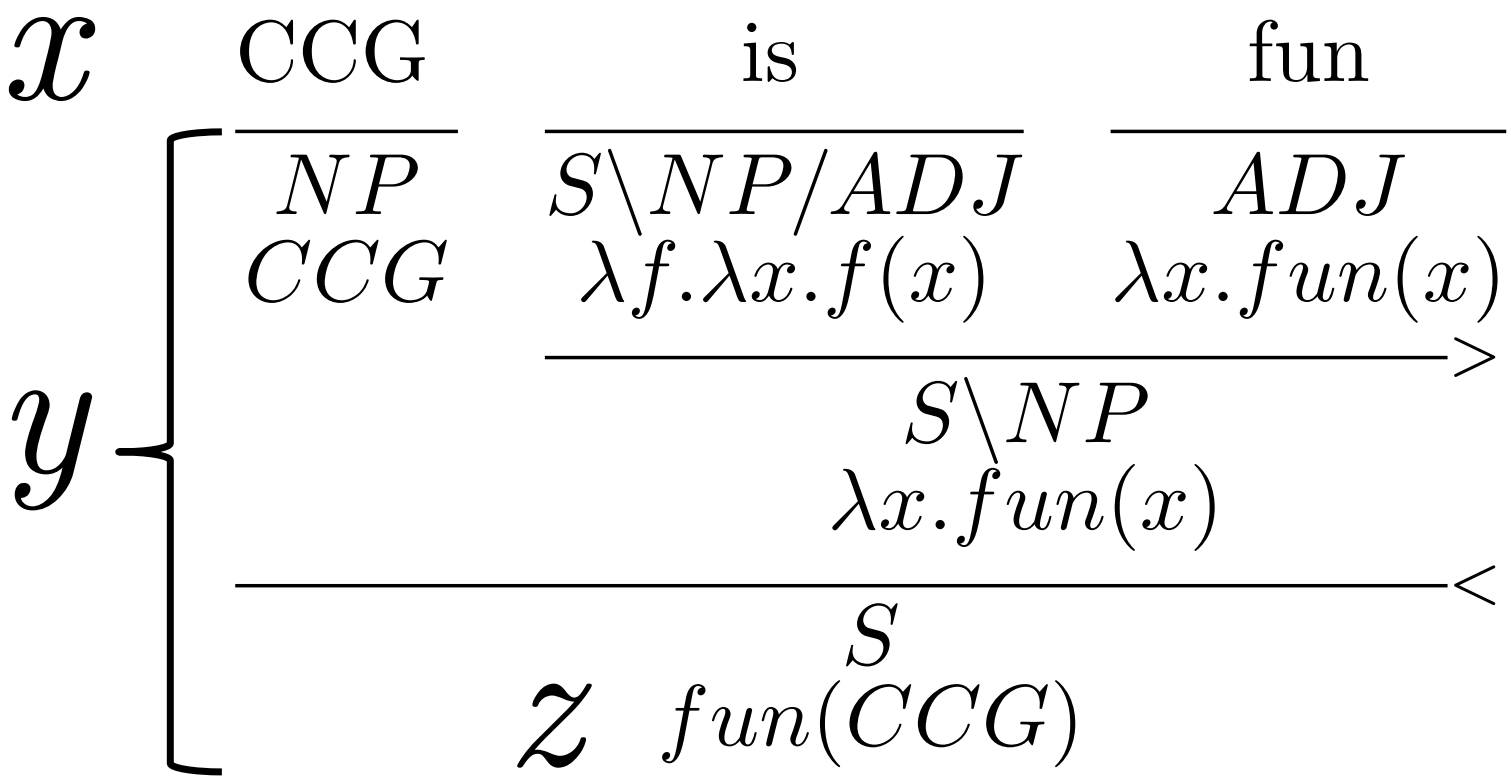
Between CCGs and CFGs

	CFGs	CCGs
Combination operations		
Parse tree nodes		
Syntactic symbols		
Paired with words		

Between CCGs and CFGs

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

Parsing with CCGs



Lexical Ambiguity + Many parsing decisions → Many potential trees and LFs

Weighted Linear CCGs

- Given a weighted linear model:
  - CCG lexicon Λ
  - Feature function $f : X \times Y \rightarrow \mathbb{R}^m$
  - Weights $w \in \mathbb{R}^m$
- The best parse is:

$$y^* = \arg\max_y w \cdot f(x, y)$$

- We consider all possible parses y for sentence x given the lexicon Λ

Parsing Algorithms

- Syntax-only CCG parsing has polynomial time CKY-style algorithms
- Parsing with semantics requires entire category as chart signature
  - e.g., $ADJ : \lambda x.fun(x)$
- In practice, prune to top-N for each span
  - Approximate, but polynomial time

The Lexicon Problem

- Key component of CCG
- Same words often paired with many different categories
- Difficult to learn with limited data

Factored Lexicons

the house dog

the dog of the house

$$\iota x.dog(x) \wedge of(x, \iota y.house(y))$$

the garden dog

$$\iota x.dog(x) \wedge of(x, \iota y.garden(y))$$

- Lexical entries share information
- Decomposition of entries can lead to more compact lexicons

[Kwiatkowski et al. 2011]

Factored Lexicons

the house dog $\quad \text{house} \vdash ADJ : \lambda x.of(x, \iota y.house(y))$

the dog of the house $\quad \text{house} \vdash N : \lambda x.house(x)$

$$\iota x.dog(x) \wedge of(x, \iota y.house(y))$$

the garden dog $\quad \text{garden} \vdash ADJ : \lambda x.of(x, \iota y.garden(y))$

$$\iota x.dog(x) \wedge of(x, \iota y.garden(y))$$

- Lexical entries share information
- Decomposition of entries can lead to more compact lexicons

Factored Lexicons

the house dog $\quad \text{house} \vdash ADJ : \lambda x.of(x, \iota y.house(y))$

the dog of the house $\quad \text{house} \vdash N : \lambda x.house(x)$

$$\iota x.dog(x) \wedge of(x, \iota y.house(y))$$

the garden dog $\quad \text{garden} \vdash ADJ : \lambda x.of(x, \iota y.garden(y))$

$$\iota x.dog(x) \wedge of(x, \iota y.garden(y))$$

- Lexical entries share information
- Decomposition of entries can lead to more compact lexicons

Factored Lexicons

the house dog $\quad \text{house} \vdash ADJ : \lambda x.of(x, \iota y.house(y))$

the dog of the house $\quad \text{house} \vdash N : \lambda x.house(x)$

$$\iota x.dog(x) \wedge of(x, \iota y.house(y))$$

the garden dog $\quad \text{garden} \vdash ADJ : \lambda x.of(x, \iota y.garden(y))$

$$\iota x.dog(x) \wedge of(x, \iota y.garden(y))$$

- Lexical entries share information
- Decomposition of entries can lead to more compact lexicons

Factored Lexicons

house $\vdash ADJ : \lambda x.of(x, \iota y.house(y))$

house $\vdash N : \lambda x.house(x)$

garden $\vdash ADJ : \lambda x.of(x, \iota y.garden(y))$

Lexemes

$(\text{garden}, \{garden\})$

$(\text{house}, \{house\})$

Templates

$\lambda(\omega, \{v_i\}_1^n).$

$[\omega \vdash ADJ : \lambda x.of(x, \iota y.v_1(y))]$

$\lambda(\omega, \{v_i\}_1^n).$

$[\omega \vdash N : \lambda x.v_1(x)]$

Factored Lexicons

Templates

$\lambda(\omega, \{v_i\}_1^n).$
$[\omega \vdash ADJ : \lambda x.of(x, \iota y.v_1(y))]$

$\lambda(\omega, \{v_i\}_1^n).$
$[\omega \vdash N : \lambda x.v_1(x)]$

Lexemes

$(\text{garden}, \{garden\})$

$(\text{house}, \{house\})$

- Capture systematic variations in word usage
- Each variation can then be applied to compact units of lexical meaning

- Model word meaning
- Abstracts the compositional nature of the word

Factored Lexicons

$$(\text{garden}, \{garden\})$$

$$\lambda(\omega, \{v_i\}_1^n).$$

$$[\omega \vdash N : \lambda x.v_1(x)]$$

$$\omega \leftarrow \text{garden}$$

$$v_1 \leftarrow garden$$

$$\text{garden} \vdash N : \lambda x.garden(x)$$

Factored Lexicons

Original Lexicon

$$\text{flight} \vdash S|NP : \lambda x.flight(x)$$
$$\text{flight} \vdash S|NP/(S|NP) : \lambda f.\lambda x.flight(x) \wedge f(x)$$
$$\text{flight} \vdash S|NP\backslash(S|NP) : \lambda f.\lambda x.flight(x) \wedge f(x)$$
$$\text{ground transport} \vdash S|NP : \lambda x.trans(x)$$
$$\text{ground transport} \vdash S|NP/(S|NP) : \lambda f.\lambda x.trans(x) \wedge f(x)$$
$$\text{ground transport} \vdash S|NP\backslash(S|NP) : \lambda f.\lambda x.trans(x) \wedge f(x)$$

Factored Lexicons

Original Lexicon

$\text{flight} \vdash S|NP : \lambda x.flight(x)$

$\text{flight} \vdash S|NP/(S|NP) : \lambda f.\lambda x.flight(x) \wedge f(x)$

$\text{flight} \vdash S|NP\backslash(S|NP) : \lambda f.\lambda x.flight(x) \wedge f(x)$

$\text{ground transport} \vdash S|NP : \lambda x.trans(x)$

$\text{ground transport} \vdash S|NP/(S|NP) : \lambda f.\lambda x.trans(x) \wedge f(x)$

$\text{ground transport} \vdash S|NP\backslash(S|NP) : \lambda f.\lambda x.trans(x) \wedge f(x)$

Factored Lexicon

$(\text{flight}, \{flight\})$

$(\text{ground transport}, \{trans\})$

$\lambda(\omega, \{v_i\}_1^n).[\omega \vdash S|NP : \lambda x.v_1(x)]$

$\lambda(\omega, \{v_i\}_1^n).[\omega \vdash S|NP/(S|NP) : \lambda f.\lambda x.v_1(x) \wedge f(x)]$

$\lambda(\omega, \{v_i\}_1^n).[\omega \vdash S|NP\backslash(S|NP) : \lambda f.\lambda x.v_1(x) \wedge f(x)]$

Factoring a Lexical Entry

$$\text{house} \vdash ADJ : \lambda x.of(x, \iota y.house(y))$$

Three possible factorings

Factoring a Lexical Entry

$$\text{house} \vdash ADJ : \lambda x.of(x, \iota y.house(y))$$

Partial factoring

$$(\text{house}, \{house\})$$
$$\lambda(\omega, \{v_i\}_1^n).[\omega \vdash ADJ : \lambda x.of(x, \iota y.v_1(y))]$$

Partial factoring

$$(\text{house}, \{of\})$$
$$\lambda(\omega, \{v_i\}_1^n).[\omega \vdash ADJ : \lambda x.v_1(x, \iota y.house(y))]$$

Maximal factoring

$$(\text{house}, \{of, house\})$$
$$\lambda(\omega, \{v_i\}_1^n).[\omega \vdash ADJ : \lambda x.v_1(x, \iota y.v_2(y))]$$

Parsing | Learning | Modeling

- Lambda calculus
- Parsing with Combinatory Categorial Grammars
- Linear CCGs
- Factored lexicons