CS447: Natural Language Processing

http://courses.engr.illinois.edu/cs447

Lecture 22: Compositional Semantics

Julia Hockenmaier

juliahmr@illinois.edu

3324 Siebel Center

Natural language conveys information about the world

We can compare statements about the world with the actual state of the world:

Champaign is in California. (false)

We can learn new facts about the world from natural language statements:

The earth turns around the sun.

We can answer questions about the world:

Where can I eat Korean food on campus?

We draw inferences from natural language statements

Some inferences are purely linguistic:

All blips are foos.

Blop is a blip.

Blop is a foo (whatever that is).

Some inferences require world knowledge.

Mozart was born in Salzburg.

Mozart was born in Vienna.

No, that can't be - these are different cities.

What does it mean to “understand” language?

The ability to identify the intended literal meaning is a prerequisite for any deeper understanding

“eat sushi with chopsticks” does not mean that chopsticks were eaten

True understanding also requires the ability to draw appropriate inferences that go beyond literal meaning:

— **Lexical** inferences (depend on the meaning of words)

You are running —> you are moving.

— **Logical** inferences (e.g. syllogisms)

All men are mortal. Socrates is a man —> Socrates is mortal.

— **Common sense** inferences (require world knowledge):

It’s raining —> You get wet if you’re outside.

— **Pragmatic** inferences (speaker’s intent, speaker’s assumptions about the state of the world, social relations)

What does it mean to “understand” language?

Linguists have studied (and distinguish between) semantics and pragmatics

— Semantics is concerned with literal meaning (e.g. truth conditions: when is a statement true), lexical knowledge (running is a kind of movement).
— Pragmatics is (mostly) concerned with speaker intent and assumptions, social relations, etc.

NB: Linguistics has little to say about extralinguistic (commonsense) inferences that are based on world knowledge, although some of this is captured by lexical knowledge.

How do we get computers to “understand” language?

Not all aspects of understanding are equally important for all NLP applications

Historically, even just identifying the correct literal meaning has been difficult.

In recent years, more efforts on task such as entailment recognition that aim to evaluate the ability to draw inferences.

Semantics: getting at literal meaning

In order to understand language, we need to be able to identify its (literal) meaning.

- How do we represent the meaning of a word? (**Lexical semantics**)
- How do we represent the meaning of a sentence? **(Compositional semantics)**
- How do we represent the meaning of a text? **(Discourse semantics)**

NB: Although we clearly need to handle all levels of semantics, historically these have often been studied in (relative) isolation, so these subareas each have their own theories and models.

Today's lecture

Our initial question:

What is the meaning of (declarative) sentences?

Declarative sentences: *"John likes coffee"*.

(We won't deal with questions (*"Who likes coffee?"*) and imperative sentences (commands: *"Drink up!"*))

Follow-on question 1:

How can we represent the meaning of sentences?

Follow-on question 2:

How can we map a sentence to its meaning representation?

What do nouns and verbs mean?

In the simplest case, an NP is just a name: *John*
Names refer to entities in the world.

Verbs define n-ary predicates: depending on the arguments they take (and the state of the world), the result can be true or false.

What do sentences mean?

Declarative sentences (statements) can be true or false, depending on the state of the world:

John sleeps.

In the simplest case, the consist of a verb and one or more noun phrase arguments.

Principle of compositionality (Frege):
The meaning of an expression depends on the meaning of its parts and how they are put together.

First-order predicate logic (FOL) as a meaning representation language

Predicate logic expressions

Terms: refer to entities

- Variables: x, y, z
- Constants: John', Urbana'
- Functions applied to terms (fatherOf(John'))'

Predicates: refer to properties of, or relations between, entities

- tall'(x), eat'(x,y), …

Formulas: can be true or false

- Atomic formulas: predicates, applied to terms: tall'(John')
- Complex formulas: constructed recursively via logical connectives and quantifiers

Formulas

Atomic formulas are predicates, applied to terms:

book(x), eat(x,y)

Complex formulas are constructed recursively by

...**negation** (¬): $\neg book(John')$

...**connectives** (∧,∨,→): $book(y) \wedge read(x,y)$

conjunction (and): $\varphi \wedge \psi$ disjunction (or): $\varphi \vee \psi$ implication (if): $\varphi \rightarrow \psi$

...**quantifiers** ($\forall x$, $\exists x$)

universal (typically with implication) $\forall x[\varphi(x) \rightarrow \psi(x)]$

existential (typically with conjunction) $\exists x[\varphi(x)]$, $\exists x[\varphi(x) \wedge \psi(x)]$

Interpretation: formulas are either **true or false**.

The syntax of FOL expressions

Term ⇒ Constant |
Variable |
Function(Term,...,Term)

Formula ⇒ Predicate(Term, ...Term) |
¬ Formula |
∀ Variable Formula |
∃ Variable Formula |
Formula ∧ Formula |
Formula ∨ Formula |
Formula → Formula

Some examples

John is a student:
$\text{student}(\text{john})$

All students take at least one class:
$\forall x\ \text{student}(x) \rightarrow \exists y(\text{class}(y) \wedge \text{takes}(x,y))$

There is a class that all students take:
$\exists y(\text{class}(y) \wedge \forall x\ (\text{student}(x) \rightarrow \text{takes}(x,y))$

FOL is sufficient for many Natural Language inferences

All blips are foos.
Blop is a blip.
Blop is a foo

$\forall x\ blip(x) \rightarrow foo(x)$
$blip(blop)$
$foo(blop)$

Some inferences require world knowledge.

Mozart was born in Salzburg.
Mozart was born in Vienna.

No, that can't be-
these are different cities

$bornIn(Mozart, Salzburg)$
$bornIn(Mozart, Vienna)$

$bornIn(Mozart, Salzburg)$
$\wedge \neg bornIn(Mozart, Salzburg)$

Not all of natural language can be expressed in FOL:

Tense:

It was hot yesterday.

I will go to Chicago tomorrow.

Modals:

You can go to Chicago from here.

Other kinds of quantifiers:

Most students hate 8:00am lectures.

λ-Expressions

We often use **λ-expressions**
to construct complex logical formulas:

- $\lambda x.\varphi(..x...)$ is a **function** where x is a variable, and φ some FOL expression.

- **β-reduction** (called λ-reduction in textbook):
  Apply $\lambda x.\varphi(..x...)$ to some argument a:
  $(\lambda x.\varphi(..\mathrm{x}...)\ a) \Rightarrow \varphi(..\mathrm{a}...)$
  Replace all occurrences of x in $\varphi(..\mathrm{x}...)$ with a

- **n-ary functions** contain embedded λ-expressions:
  $\lambda x.\lambda y.\lambda z.give(\mathrm{x},\mathrm{y},\mathrm{z})$

(Combinatory) Categorial Grammar

CCG: the machinery

Categories:

specify subcat lists of words/constituents.

Combinatory rules:

specify how constituents can combine.

The lexicon:

specifies which categories a word can have.

Derivations:

spell out process of combining constituents.

CCG categories

Simple (atomic) categories: NP, S, PP

Complex categories (functions):

Return a **result** when combined with an **argument**

VP, intransitive verb	**S\NP**
Transitive verb	**(S\NP)/NP**
Adverb	**(S\NP)\(S\NP)**
Prepositions	**((S\NP)\(S\NP))/NP** **(NP\NP)/NP** **PP/NP**

CCG categories are functions

CCG has **a few atomic categories, e.g**

S, NP, PP

All other CCG categories are **functions:**

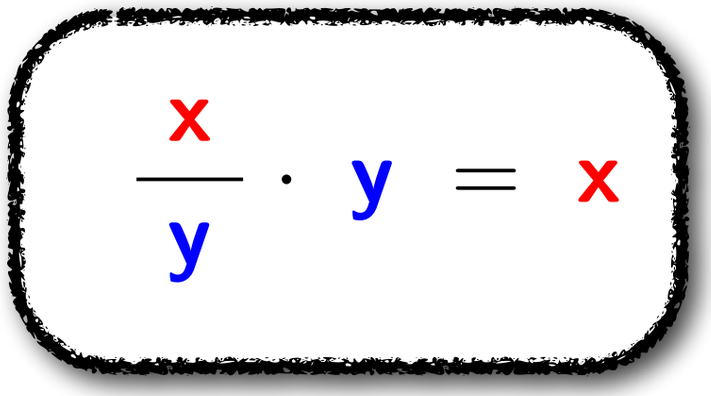
S / NP

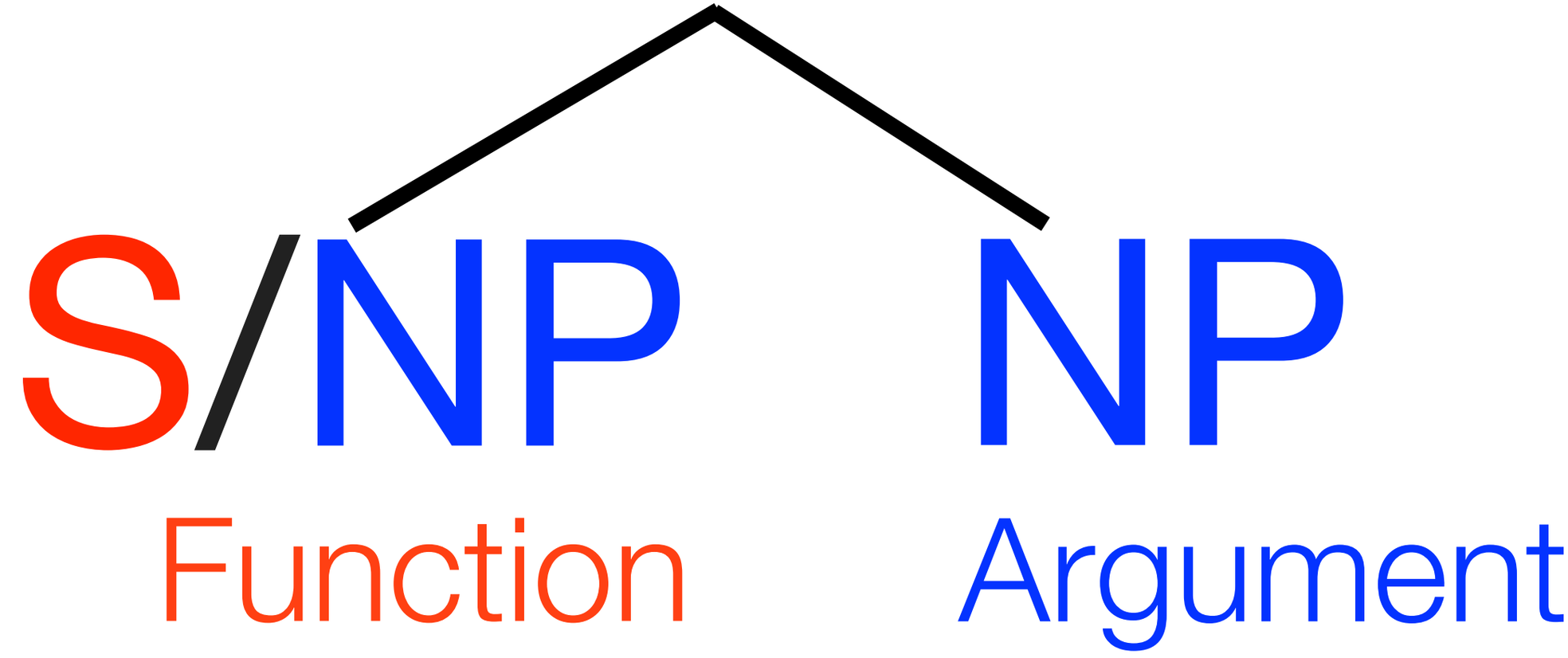
Result Dir. Argument

Rules: Function application

Result

S





Rules: Function application

Result

S

$$y \cdot \frac{x}{y} = x$$

NP

Argument

S\NP

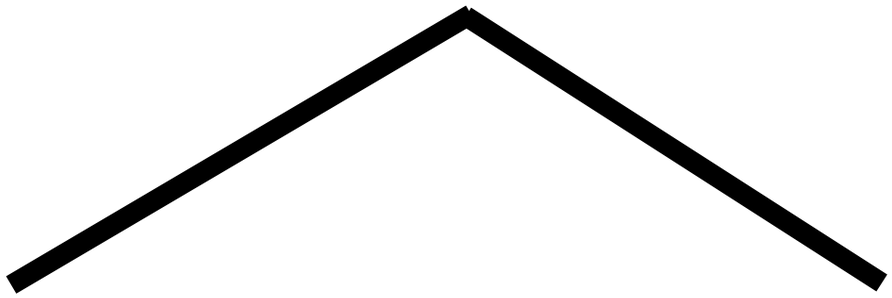
Function

Rules: Function application

Result

S\NP

$$\frac{x}{y} \cdot y = x$$



(S\NP)/NP NP

Function Argument

Function application

Forward application (>):

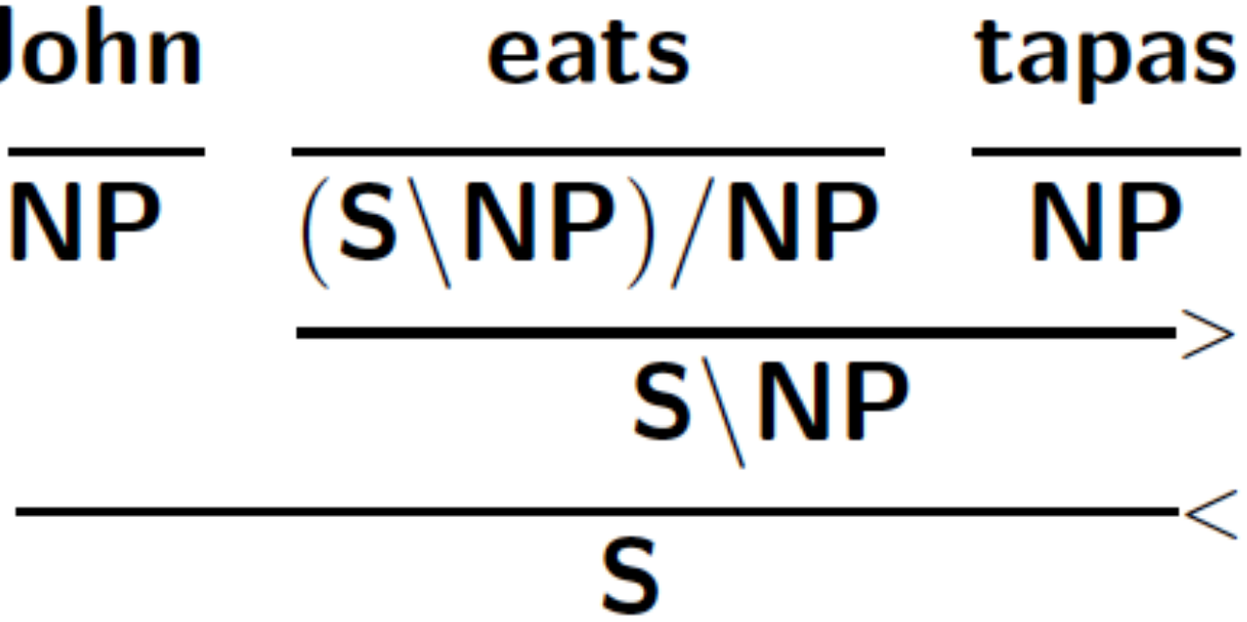
(S\NP)/NP	**NP**	$\Rightarrow_{>}$	**S\NP**
eats	tapas		eats tapas

Backward application (<):

NP	**S\NP**	$\Rightarrow_{<}$	**S**
John	eats tapas		John eats tapas

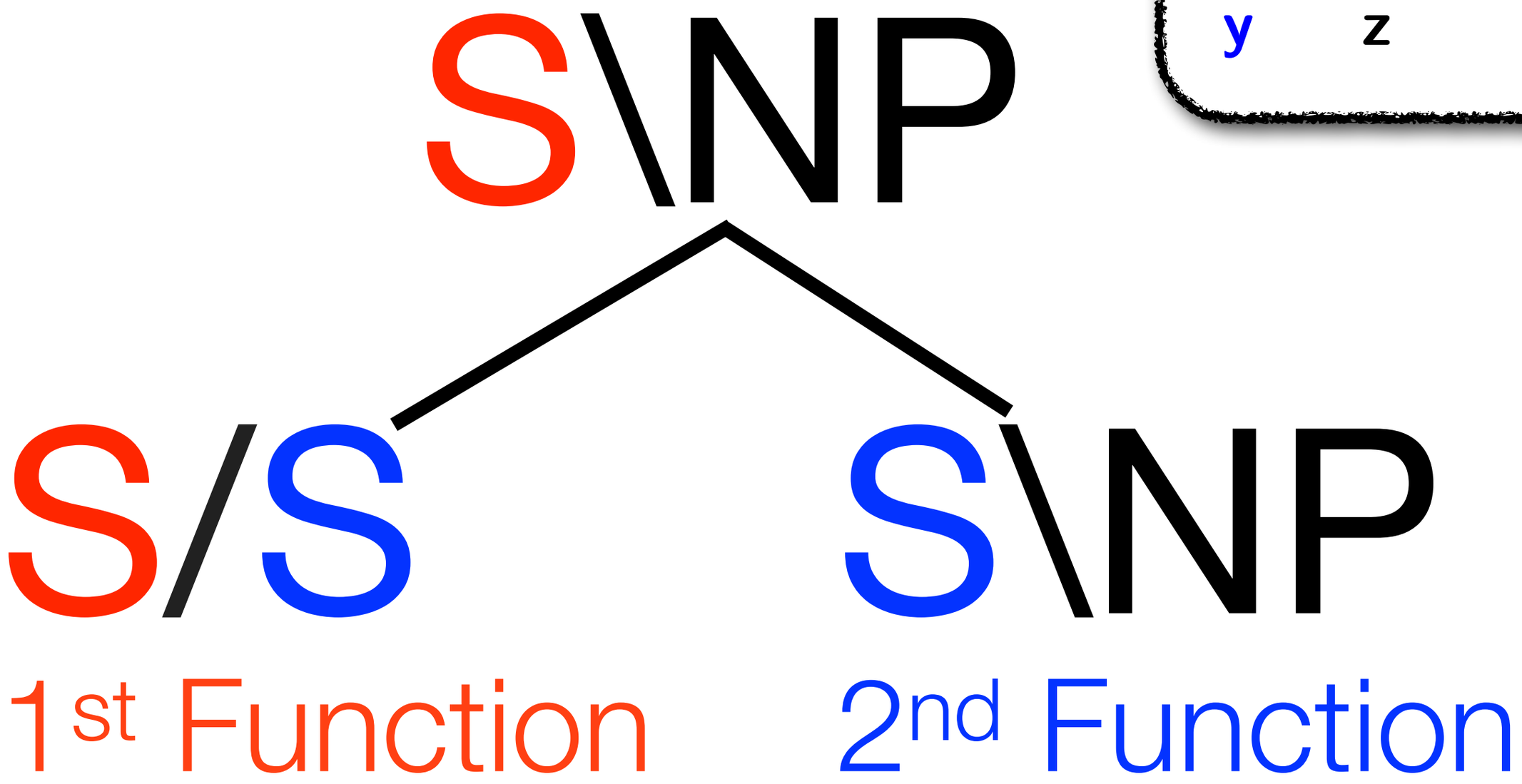
Combines function X/Y or X\Y with argument Y to yield result X
Used in all variants of categorial grammar

A (C)CG derivation



Rules: Function Composition

$$\frac{x}{y} \cdot \frac{y}{z} = \frac{x}{z}$$



Rules: Type-Raising

$$S/(S\backslash NP)$$

$$NP$$

$$y = \frac{x}{x} \cdot y = \frac{x}{\left(\frac{x}{y}\right)}$$

Type-raising and composition

Type-raising: X → T/(T\X)

Turns an argument into a function.

NP → S/(S\NP) (subject)

NP → (S\NP)\((S\NP)/NP) (object)

Harmonic composition: X/Y Y/Z → X/Z

Composes two functions (complex categories)

(S\NP)/PP PP/NP → (S\NP)/NP

S/(S\NP) (S\NP)/NP → S/NP

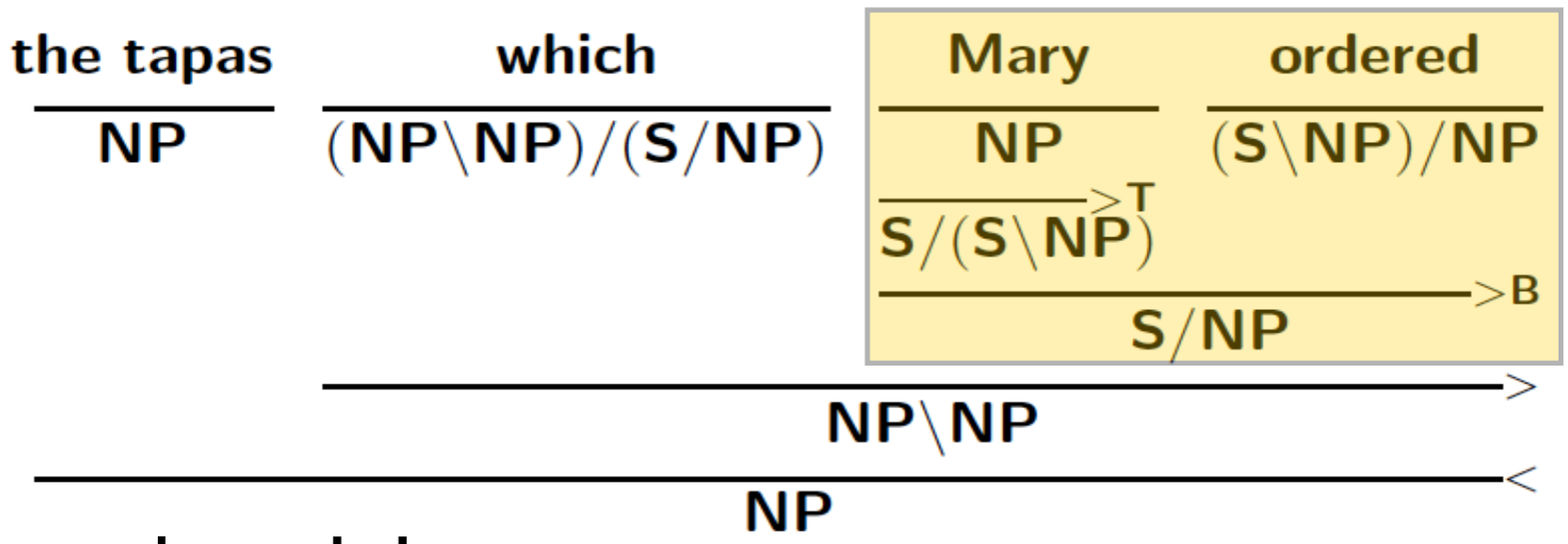
Crossing function composition: X/Y Y\Z → X\Z

Composes two functions (complex categories)

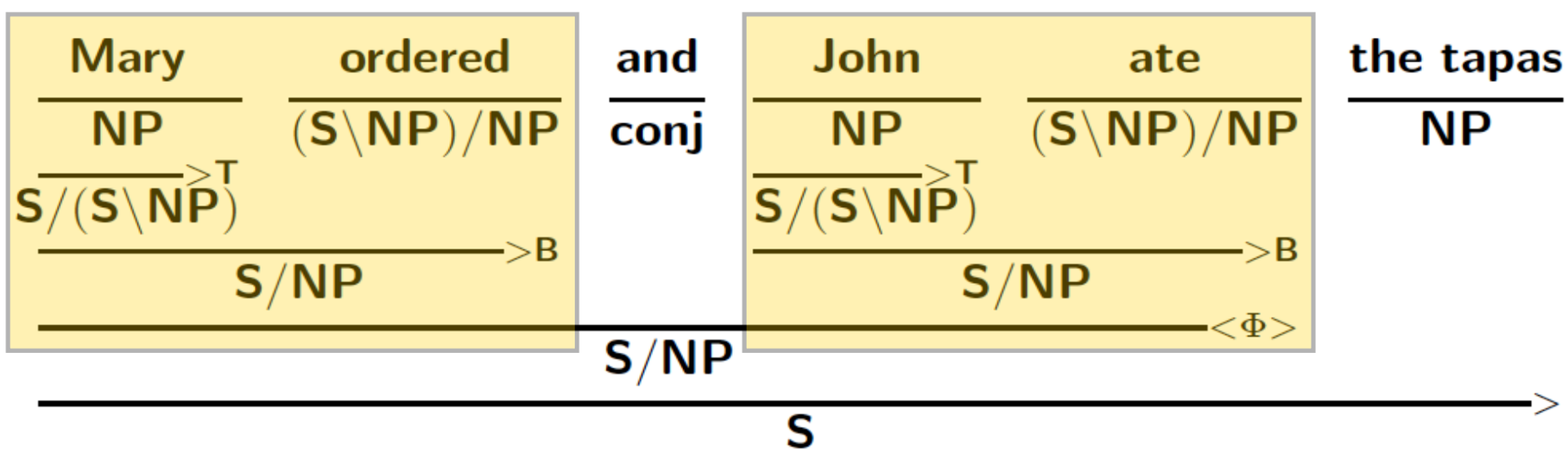
(S\NP)/S S\NP → (S\NP)\NP

Type-raising and composition

Wh-movement (relative clause):



Right-node raising:



Using Combinatory Categorial Grammar (CCG) to map sentences to predicate logic

λ-Expressions

λ-expressions can be used to construct complex logical formulas:

- $\lambda x.\varphi(..x...)$ is a **function** where x is a variable, and φ some FOL expression.

- **β-reduction** (called λ-reduction in textbook):
  Apply $\lambda x.\varphi(..x...)$ to some argument a:
  $(\lambda x.\varphi(..\mathrm{x}...)\ a) \Rightarrow \varphi(..\mathrm{a}...)$
  Replace all occurrences of x in $\varphi(..\mathrm{x}...)$ with a

- **n-ary functions** contain embedded λ-expressions:
  $\lambda x.\lambda y.\lambda z.give(\mathrm{x},\mathrm{y},\mathrm{z})$

CCG semantics

Every syntactic constituent has a semantic interpretation:

Every **lexical entry** maps a word to a syntactic category and a corresponding semantic type:

John=(**NP**, john') Mary= (**NP**, mary')

loves: (**(S\NP)/NP** λx.λy.loves(x,y))

Every **combinatory rule** has a syntactic and a semantic part:

Function application: $\mathbf{X/Y}$:λx.f(x) $\mathbf{Y}$:a → $\mathbf{X}$:f(a)

Function composition: $\mathbf{X/Y}$:λx.f(x) $\mathbf{Y/Z}$:λy.g(y) → $\mathbf{X/Z}$:λz.f(λy.g(y).z)

Type raising: $\mathbf{X}$:a → $\mathbf{T/(T\backslash X)}$ λf.f(a)

An example with semantics

$$\dfrac{\dfrac{\dfrac{\textit{John}}{\mathbf{NP} : \textit{John}} \quad \dfrac{\textit{sees}}{(\mathbf{S}\backslash\mathbf{NP})/\mathbf{NP} : \lambda x.\lambda y.sees(x,y)} \quad \dfrac{\textit{Mary}}{\mathbf{NP} : \textit{Mary}}}{\mathbf{S}\backslash\mathbf{NP} : \lambda y.sees(Mary,y)}>}{\mathbf{S} : sees(Mary,John)}<$$

Understanding sentences

"Every chef cooks a meal"

$$\forall x[chef(x) \rightarrow \exists y[meal(y) \wedge cooks(y,x)]]$$
$$\exists y[meal(y) \wedge \forall x[chef(x) \rightarrow cooks(y,x)]]$$

We translate sentences into (first-order) predicate logic.

Every (declarative) sentence corresponds to a proposition, which can be true or false.

But…

… what can we do with these representations?

Being able to translate a sentence into predicate logic is not enough, unless we also know what these predicates mean.

Semantics joke (B. Partee): The meaning of life is *life'*

Compositional formal semantics tells us how to fit together pieces of meaning, but doesn't have much to say about the meaning of the basic pieces (i.e. lexical semantics)

… how do we put together meaning representations of multiple sentences?

We need to consider discourse (there are approaches within formal semantics, e.g. Discourse Representation Theory)

… Do we really need a *complete* analysis of each sentence?

This is pretty brittle (it's easy to make a parsing mistake)

Can we get a more shallow analysis?

Supplementary material: quantifier scope ambiguities in CCG

Quantifier scope ambiguity

"Every chef cooks a meal"

- **Interpretation A:**
  For every chef, there is a meal which he cooks.

$$\forall x[chef(x) \rightarrow \exists y[meal(y) \wedge cooks(y,x)]]$$

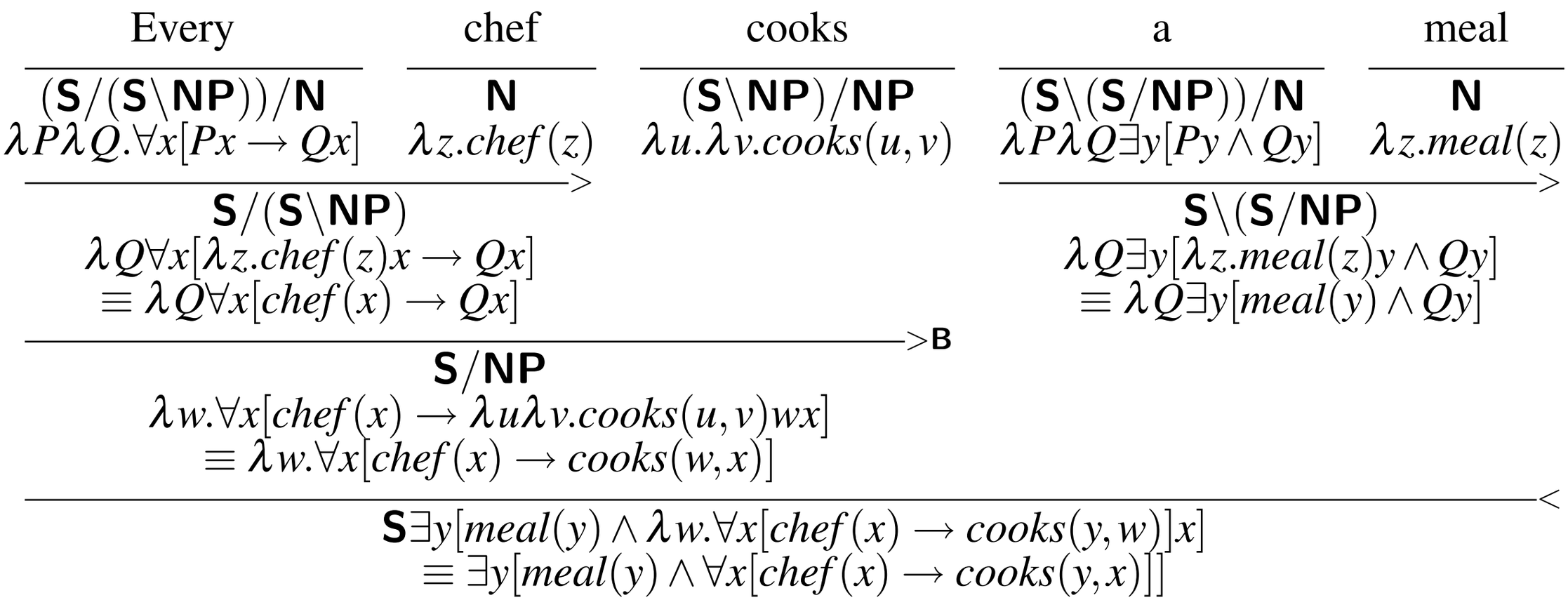
- **Interpretation B:**
  There is some meal which every chef cooks.

$$\exists y[meal(y) \wedge \forall x[chef(x) \rightarrow cooks(y,x)]]$$

Interpretation A

Every	chef	cooks	a	meal
$(\mathbf{S}/(\mathbf{S}\backslash\mathbf{NP}))/\mathbf{N}$	$\mathbf{N}$	$(\mathbf{S}\backslash\mathbf{NP})/\mathbf{NP}$	$((\mathbf{S}\backslash\mathbf{NP})\backslash((\mathbf{S}\backslash\mathbf{NP})/\mathbf{NP}))/\mathbf{N}$	$\mathbf{N}$
$\lambda P\lambda Q.\forall x[Px \rightarrow Qx]$	$\lambda z.chef(z)$	$\lambda u.\lambda v.cooks(u,v)$	$\lambda P\lambda Q\exists y[Py \wedge Qy]$	$\lambda z.meal(z)$
	$\mathbf{S}/(\mathbf{S}\backslash\mathbf{NP})$ $\lambda Q.\forall x[\lambda z.chef(z)x \rightarrow Qx]$ $\equiv \lambda Q.\forall x[chef(x) \rightarrow Qx]$		$(\mathbf{S}\backslash\mathbf{NP})\backslash((\mathbf{S}\backslash\mathbf{NP})/\mathbf{NP})$ $\lambda Q\exists y[\lambda z.meal(z)y \wedge Qy]$ $\equiv \lambda Q\lambda w.\exists y[meal(y) \wedge Qyw]$	
		$\mathbf{S}\backslash\mathbf{NP}$ $\lambda w.\exists y[meal(y) \wedge \lambda u\lambda v.cooks(u,v)yw]$ $\equiv \lambda w.\exists y[meal(y) \wedge cooks(y,w)]$		
$\mathbf{S}$: $\forall x[chef(x) \rightarrow \lambda w.\exists y[meal(y) \wedge cooks(y,w)]x]$ $\equiv \forall x[chef(x) \rightarrow \exists y[meal(y) \wedge cooks(y,x)]]$				

Interpretation B



Additional topics

Representing events and temporal relations:

- Add event variables *e* to represent the events described by verbs, and temporal variables *t* to represent the time at which an event happens.

Other quantifiers:

- What about "*most / at least two / … chefs*"?

Underspecified representations:

- Which interpretation of *"Every chef cooks a meal"* is correct? This might depend on context. Let the parser generate an underspecified representation from which both readings can be computed.

Going beyond single sentences:

- How do we combine the interpretations of single sentences?