CS447: Natural Language Processing

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

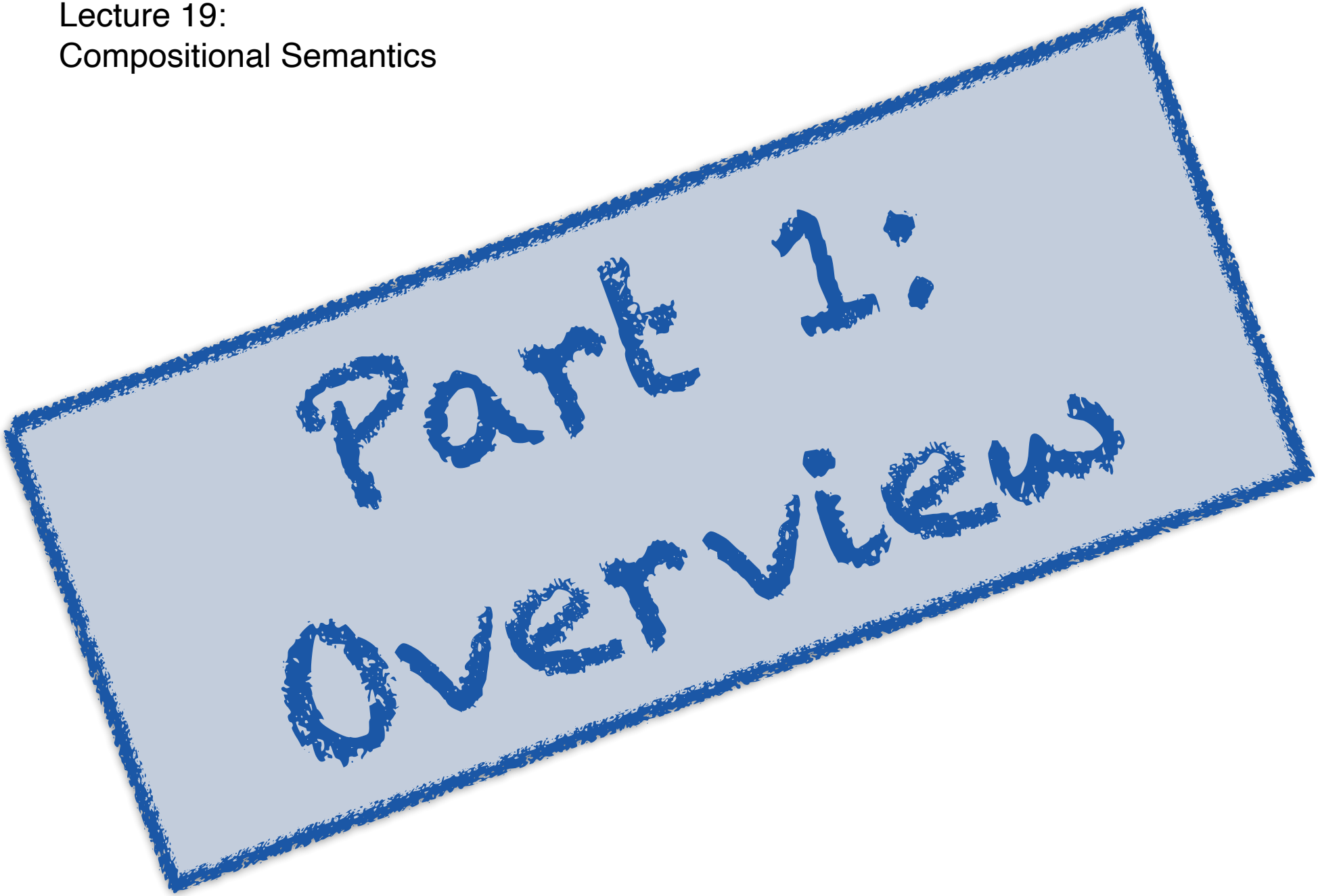
Lecture 19: Compositional Semantics

Julia Hockenmaier

juliahmr@illinois.edu

3324 Siebel Center

Lecture 19:
Compositional Semantics



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)

Boss says "It's cold here" —> Assistant gets up to close the window.

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, Urbana, USA, Thanksgiving,

Names refer to (real or abstract) entities in the world.

Verbs define n-ary predicates:

stand, run, eat, win,

Depending on the arguments they take (and the state of the world), the proposition that is obtained when we apply these predicates to the arguments can be true or false in a given situation.

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, they 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.

Today's lecture

Part 1: Overview, Principle of Compositionality

Part 2: First-order predicate logic
as a meaning representation language

Part 3: Using CCG to map sentences
to predicate logic

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?

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)

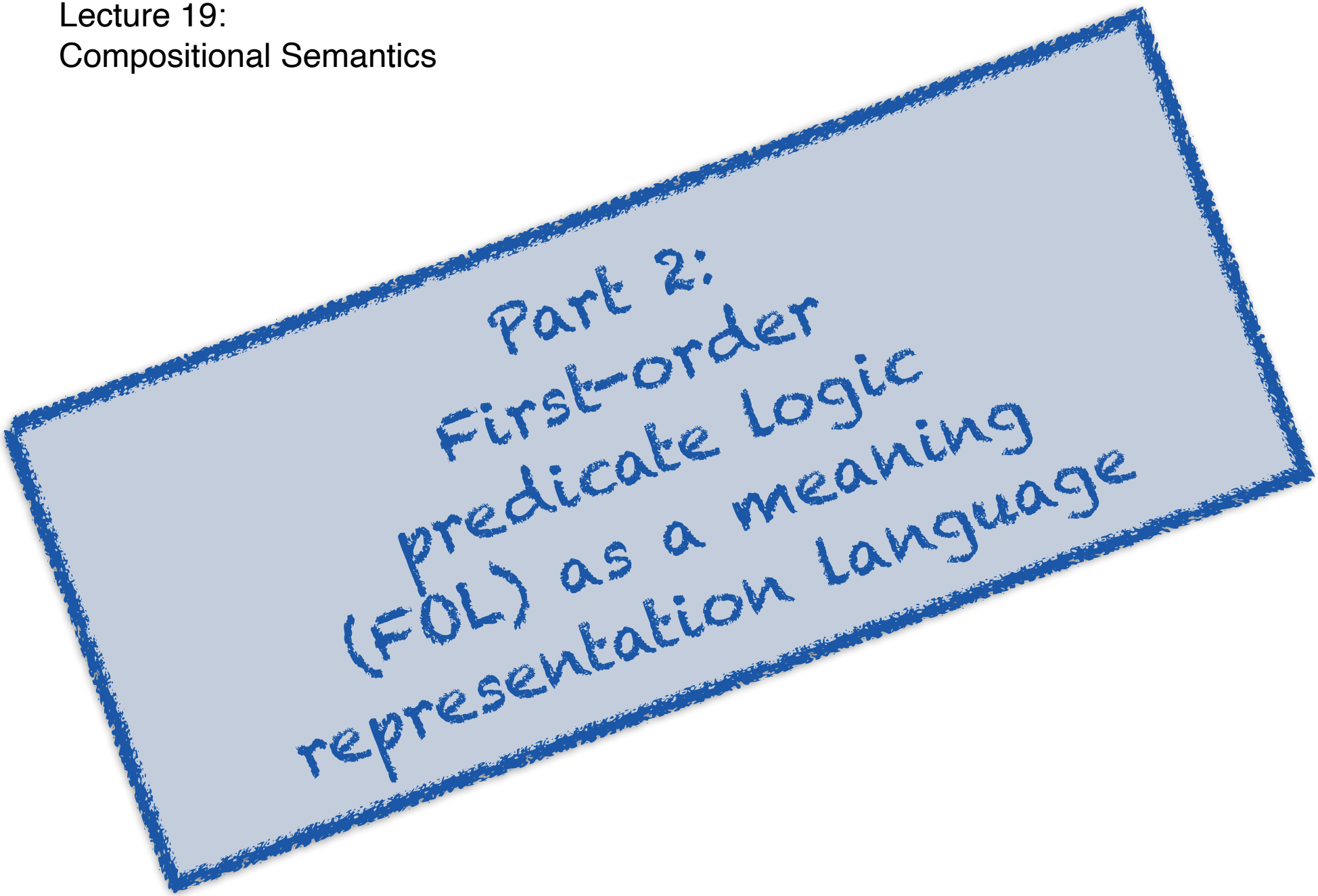
… how do we put together meaning representations of multiple sentences?

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?

Lecture 19:
Compositional Semantics



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:

$book(x)$, $eat(x,y)$, $tall(John')$

Complex formulas are constructed recursively by

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

...**connectives** ($\wedge, \vee, \rightarrow$): $book(y) \wedge read(x,y)$

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 $\Rightarrow$ Constant |
Variable |
Function(Term,...,Term)

Formula $\Rightarrow$ Predicate(Term, ...Term) |
$\neg$ Formula |
$\forall$ Variable Formula |
$\exists$ Variable Formula |
Formula $\wedge$ Formula |
Formula $\vee$ Formula |
Formula $\rightarrow$ Formula

Some examples

John is a student:
$student(john')$

All students take at least one class:
$\forall x\ student(x) \longrightarrow \exists y(class(y) \wedge take(x,y))$

There is a class that all students take:
$\exists y(class(y) \wedge \forall x\ (student(x) \longrightarrow take(x,y))$

FOL is sufficient for some natural language inferences

All blips are foos.

Blop is a blip.

Blop is a foo

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

$blip(blop')$

$foo(blop')$

Not all of natural language can be expressed in FOL:

Tense:

It was hot yesterday.

I will go to Chicago tomorrow.

Modals:

You can/must go to Chicago from here.

Other kinds of quantifiers:

Most students hate 8:00am lectures.

λ-Expressions

We can use **λ-expressions** and **β-reduction** to combine simpler logical formulas into complex logical formulas.

λ-expressions $\lambda x.\varphi(..x...)$ are (unary) **functions**
Here x is a variable, and φ is a FOL expression that we assume contains one or more free occurrences of x
(free = not bound by a quantifier, e.g. $\forall x$)

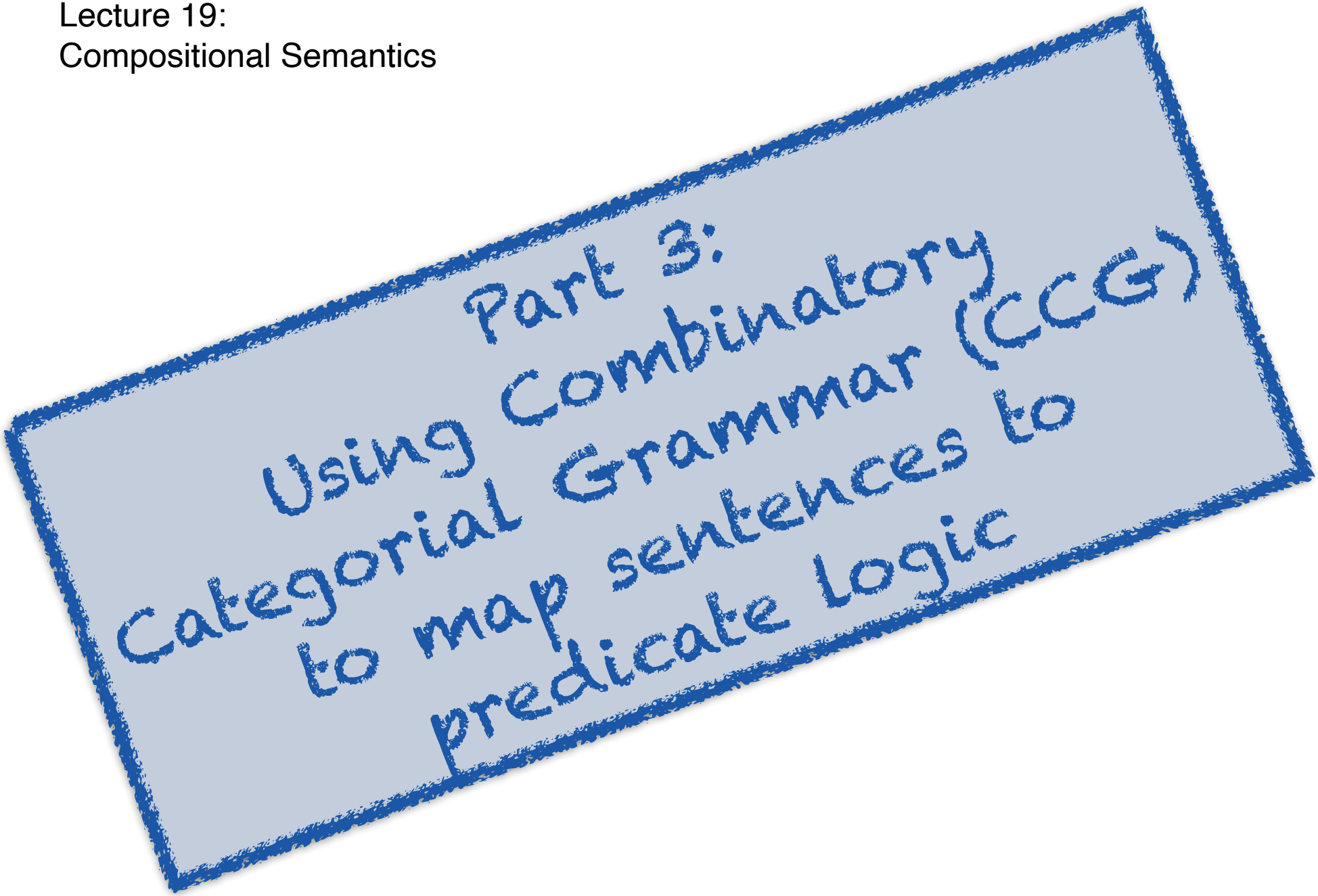
β-reduction (called λ-reduction in textbook):
Apply the function $\lambda x.\varphi(...x...)$ to some argument a:

$$(\lambda x.\varphi(..x...)\ a) \Rightarrow \varphi(...a...)$$

Replace all (free) occurrences of x in $\varphi(..x...)$ with a

n-ary functions contain embedded λ-expressions:

$$\lambda x.\lambda y.\lambda z.give(x,y,z)$$


Part 3:
Using Combinatory Categorial Grammar (CCG) to map sentences to predicate logic

Last lecture…

We've introduced CCG as a syntactic formalism.

Syntactically, CCG's main advantages are:

CCG is more expressive than CFGs,
so it can handle non-projective dependencies.
(but it's still efficiently parseable)

Type-raising and composition give CCG
a "flexible constituent structure" that allows CCG
to capture non-local dependencies without traces
(e.g. by combing a subject and transitive verb into an S/NP constituent)

Today's lecture

Compositionality in CCG's syntax-semantics interface

- Every lexical entry can be paired with a semantic interpretation
- Every syntactic combinatory rule has a semantic counterpart

NB: We will use first-order predicate logic as one example of a meaning representation language, but these principles can be applied to any other kind of representation.

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**

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

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), same slashes

(S\NP)/PP PP/NP → (S\NP)/NP

S/(S\NP) (S\NP)/NP → S/NP

Crossing composition: X/Y Y\Z → X\Z

Composes two functions (complex categories), different slashes

(S\NP)/S S\NP → (S\NP)\NP

CCG semantics

Every syntactic constituent has a semantic interpretation:

Every **lexical entry** maps a word to a syntactic category
and a corresponding, appropriate semantic type, e.g.:

John=(**NP**, john') Mary= (**NP**, mary')
loves: ((**S\NP**)**/NP** λy.λx.loves(x,y))
[a transitive verb has two (paired) arguments in the syntax and the semantics]

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

Function application: **X/Y**:λy.f(y) **Y:**a → **X**:f(a)

Function composition: **X/Y**:λy.f(y) **Y/Z:**λz.g(z) → **X/Z**:λz.f(g(z))

Type raising: **X**:a → **T/(T\X)** λf.f(a)

A CCG derivation with semantics

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

Quantifier scope ambiguity

"Every chef cooks a meal"

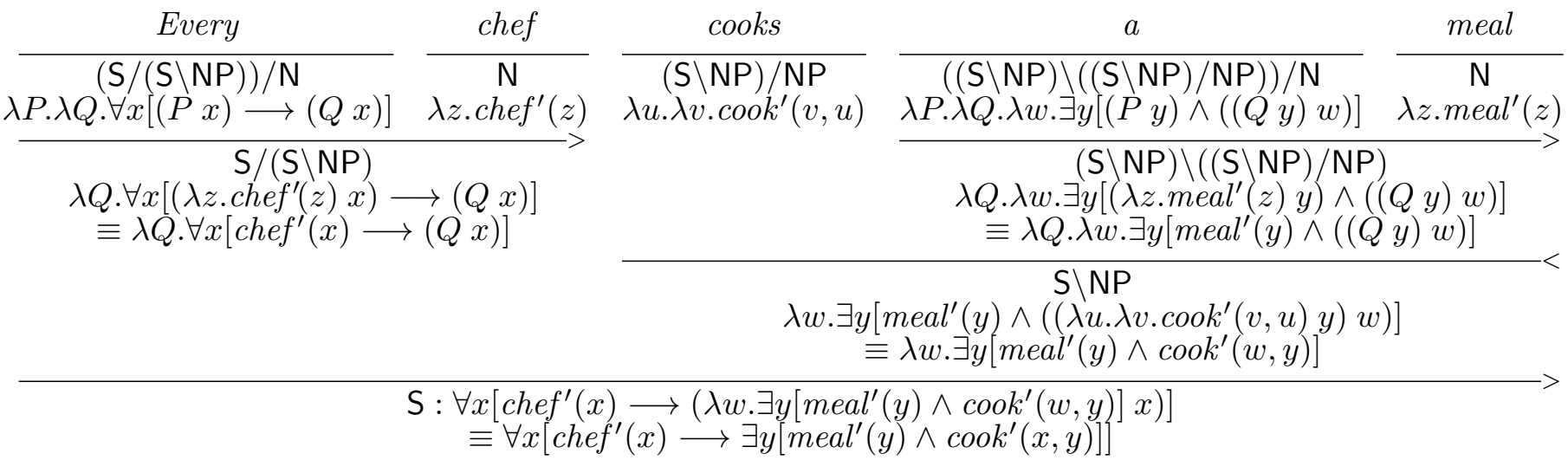
- **Interpretation A:**
  For every chef, there is a meal which he cooks.

$$\forall x \left[\text{chef}'(x) \longrightarrow \exists y \left[\text{meal}'(y) \wedge \text{cook}'(x, y) \right] \right]$$

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

$$\exists y \left[\text{meal}'(y) \wedge \forall x \left[\text{chef}'(x) \longrightarrow \text{cook}'(x, y) \right] \right]$$

Interpretation A



Interpretation B