

Reminders



HOMEWORK 7 DUE DATE IS
DUE BY MIDNIGHT ON 3/25.
HW8 WILL BE DUE 4/1.



QUIZ ON CHAPTERS 12-14
WILL BE RELEASED SOON, DUE
ON MONDAY



SIGN-UP SHEET FOR PROJECT
OPTION OR HOMEWORK
OPTION DUE TONIGHT

Review: Dependency Parsing

JURAFSKY AND MARTIN CHAPTER 15

Review: Dependency Formalism

The dependency structures are directed graphs.

$$G = (V, A)$$

where **V** is a **set of vertices** and **A** is a set of **ordered pairs of vertices (or directed arcs)**. Each arc points from the **head** to a **dependent**.



Directed arcs can also be **labeled** with the **grammatical relation** that holds between the head and a dependent.

Review: Dependency Trees

A dependency tree is a digraph where:

1. There is a **single designated root node** that has no incoming arcs
2. Each vertex has **exactly one incoming arc** (except the root node)
3. There is a **unique path** from the root node to each vertex in V

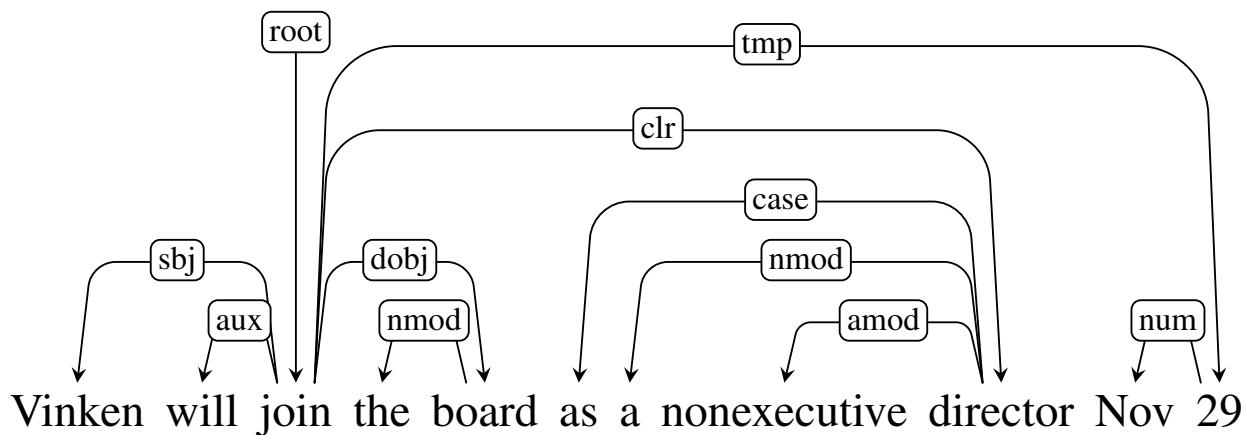
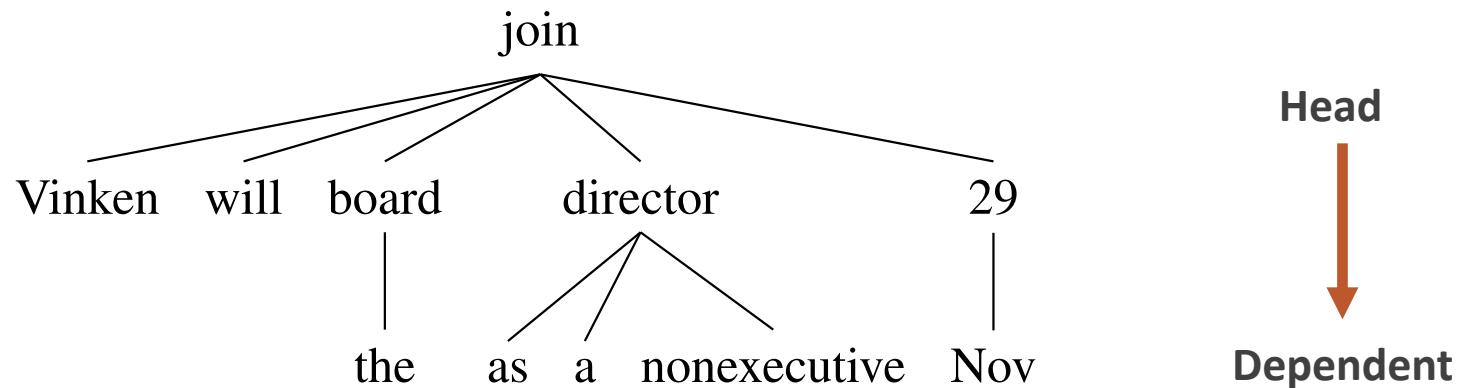
This means that each word in the sentence has exactly one head.

Labeled dependency trees add labels to arcs to specify the grammatical relationship between the head and the dependent.

Head → Dependent

Relations can be things like **nsubj** and **dobj** identify the subject and direct object

2 styles of drawing dep. trees



relation
Head → Dependent

Review: Transition-based Parsing

Transition-based parsing systems employ a **greedy stack-based** algorithm to create dependency structures.

Parsing consists of a **sequence of “shift-reduce” transitions**.

Once all the words have been moved off the stack, they have each and been assigned a head (and an appropriate relation).

The resulting configuration is a **dependency tree**.

Review: Operators

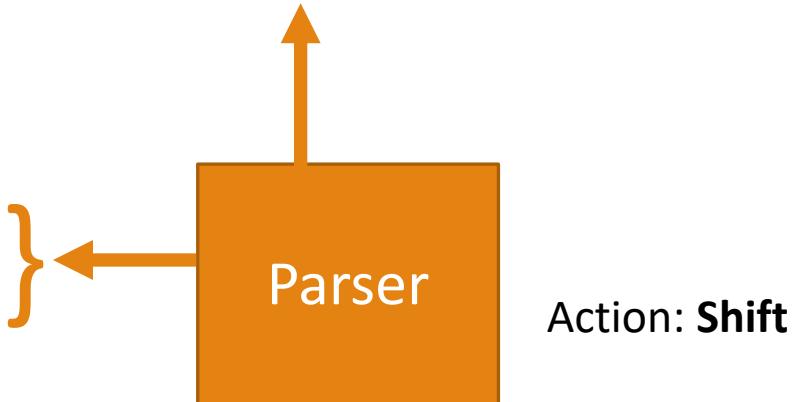
There are three **transition operators** that will operate on the top two elements of the stack:

1. **LEFTARC**: Assert a head-dependent relation between the word at the top of the stack and the word directly beneath it; *remove the lower word from the stack*.
2. **RIGHTARC**: Assert a head-dependent relation between the second word on the stack and the word at the top; *remove the word at the top of the stack*;
3. **SHIFT**: *Remove the word from the front of the input buffer and push it onto the stack*.

input buffer:

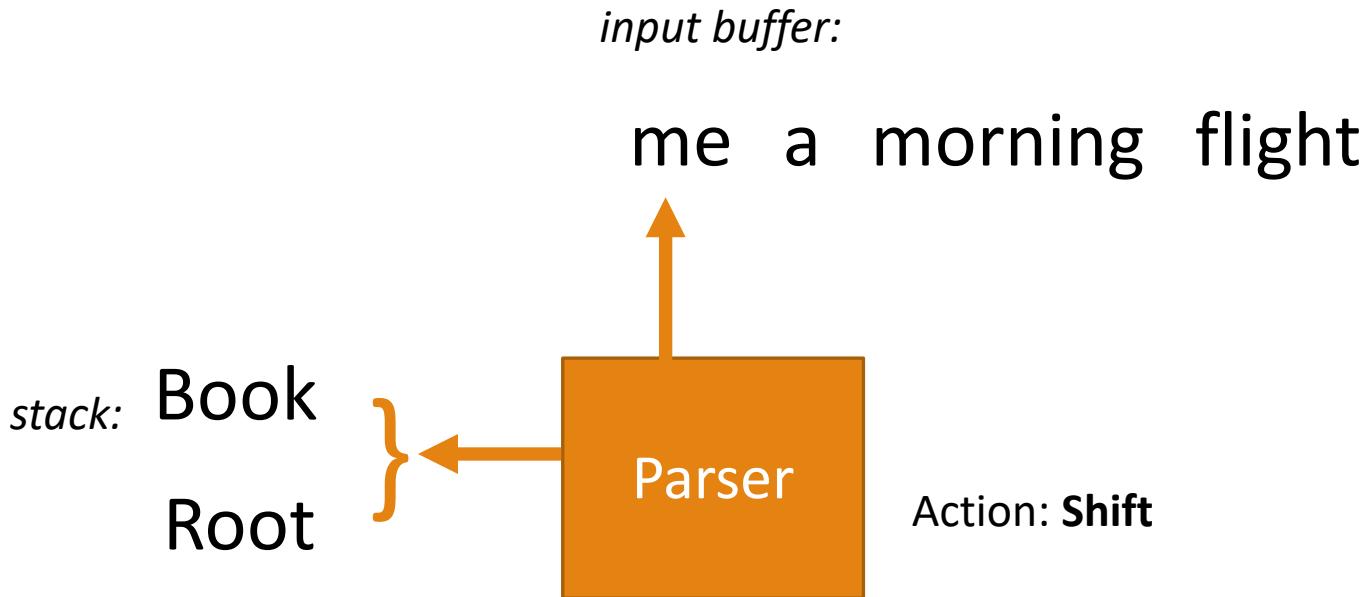
Book me a morning flight

stack: Root



Action: **Shift**

Root Book me a morning flight

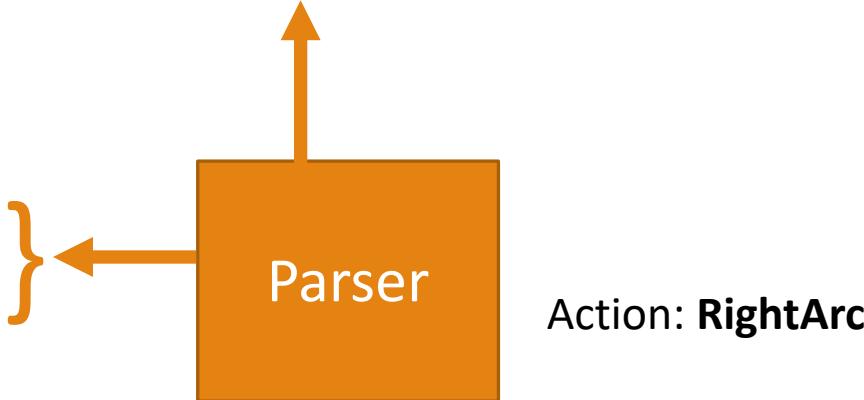


Root Book me a morning flight

input buffer:

a morning flight

stack:
me
Book
Root

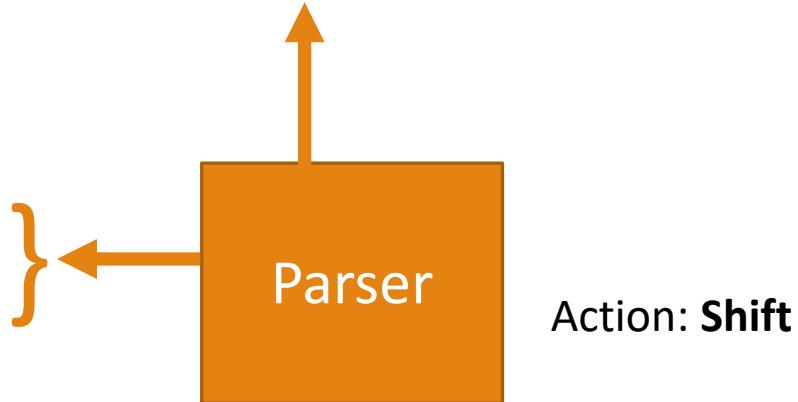


iobj
Root Book me a morning flight

input buffer:

a morning flight

stack:
Book
Root



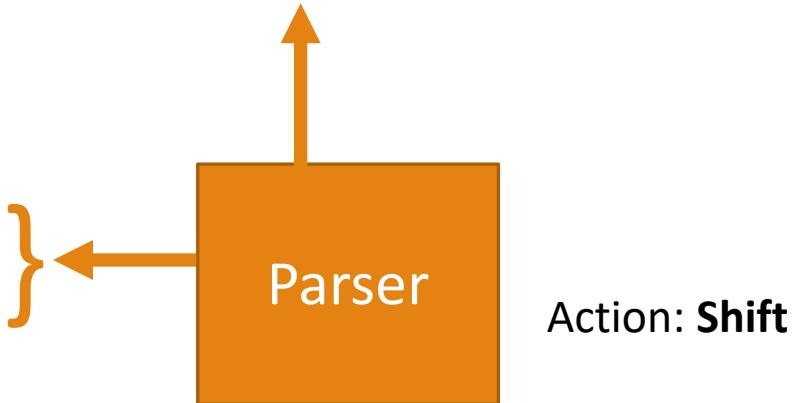
Action: **Shift**

iobj
Root Book me a morning flight

input buffer:

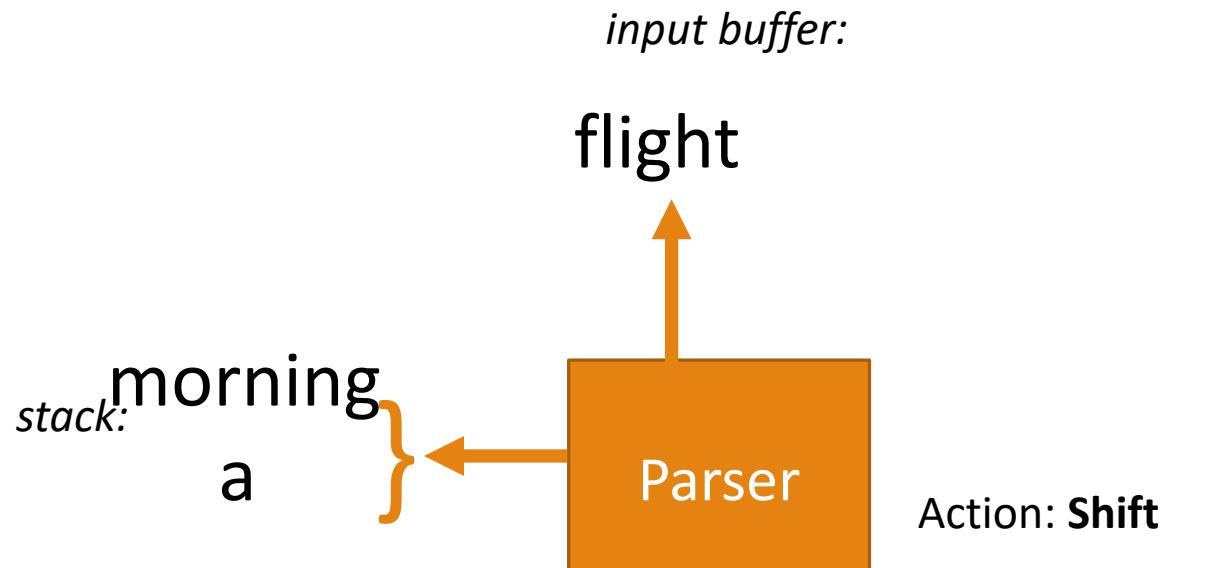
morning flight

stack:
a
Book
Root



Action: Shift

iobj
Root Book me a morning flight

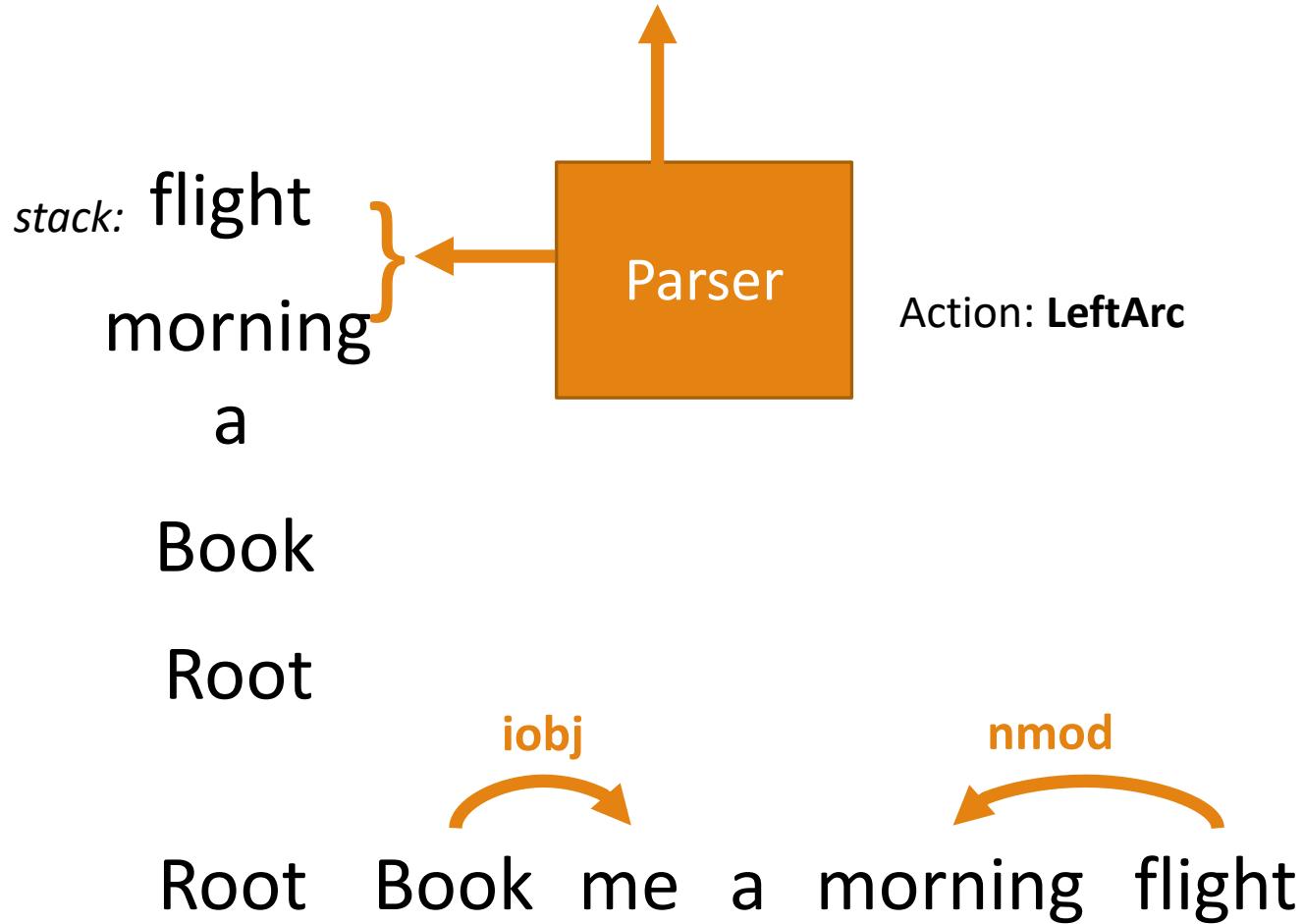


iobj

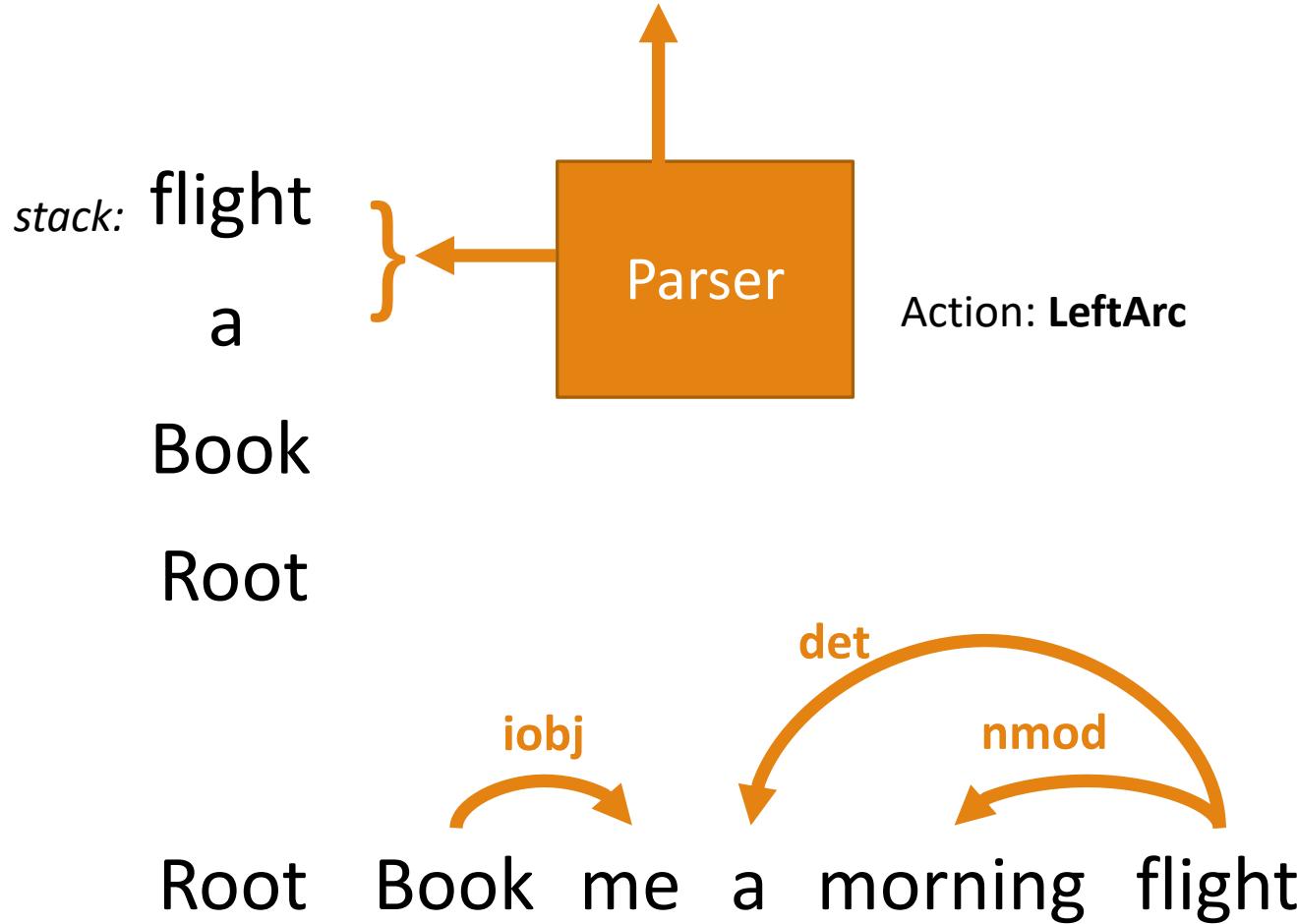
Root Book me a morning flight

The diagram shows the input buffer at the bottom, consisting of the words "Root", "Book", "me", "a", "morning", and "flight". An orange curved arrow labeled "iobj" points from the "Root" word to the "a" word, indicating it is the current object being processed by the parser.

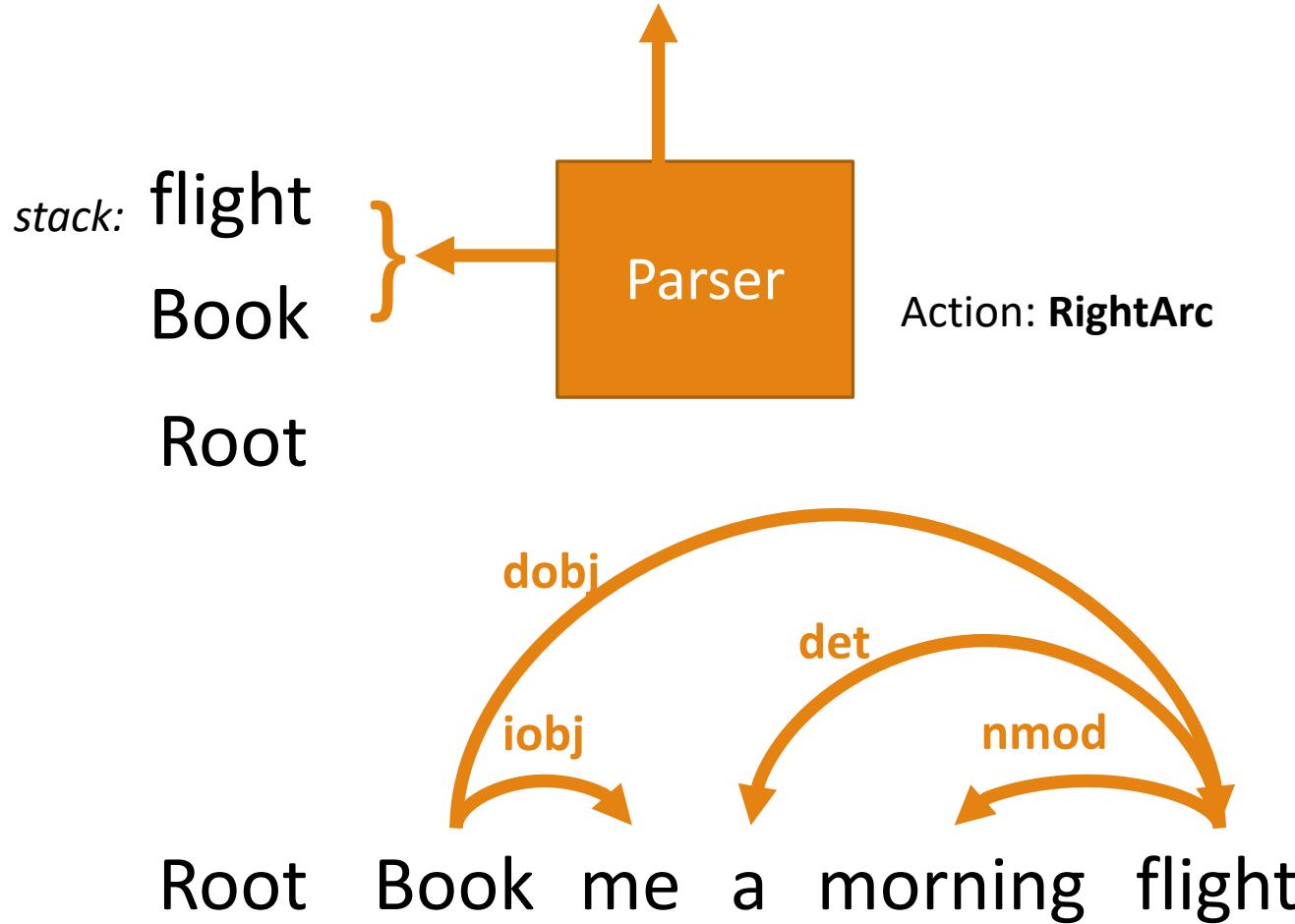
input buffer:



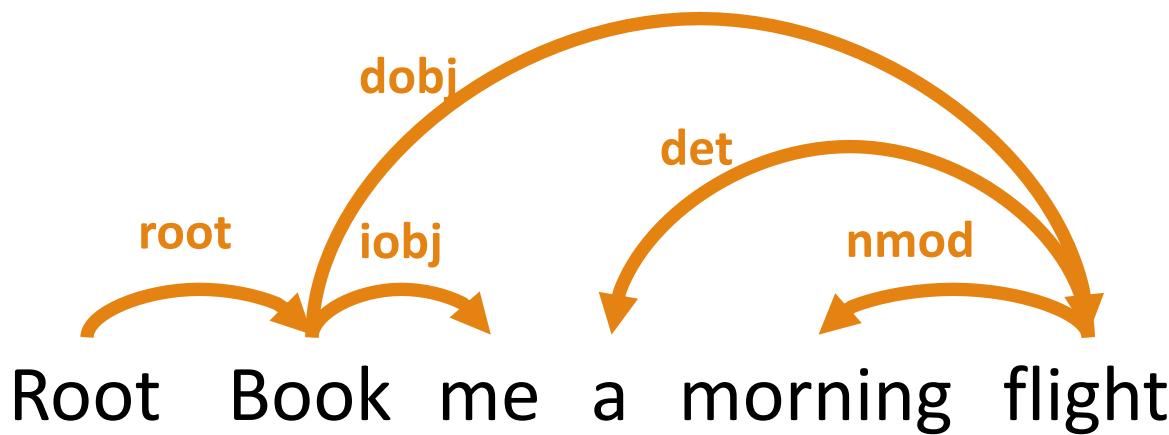
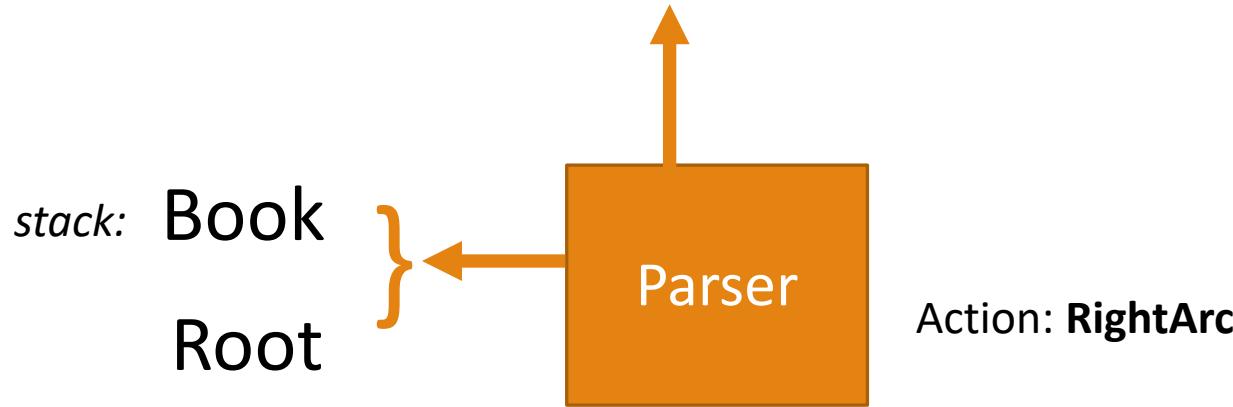
input buffer:



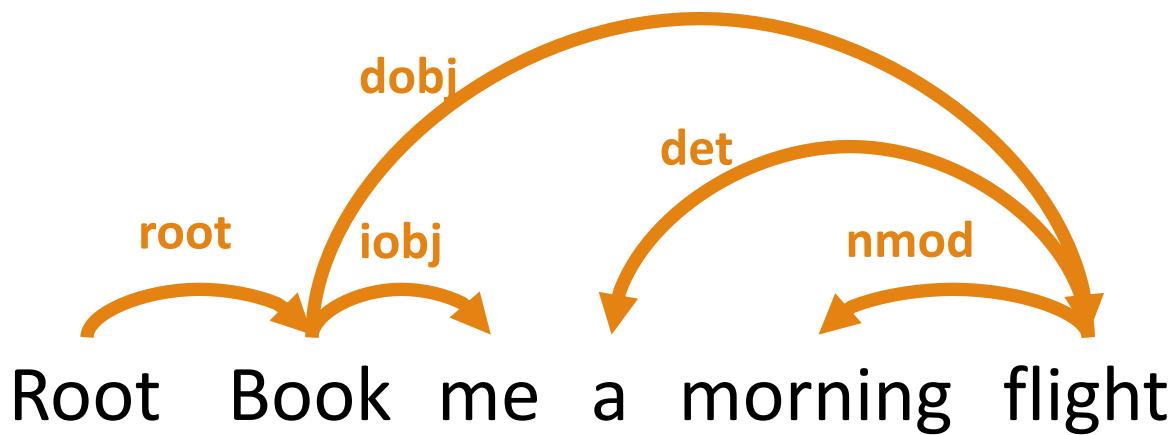
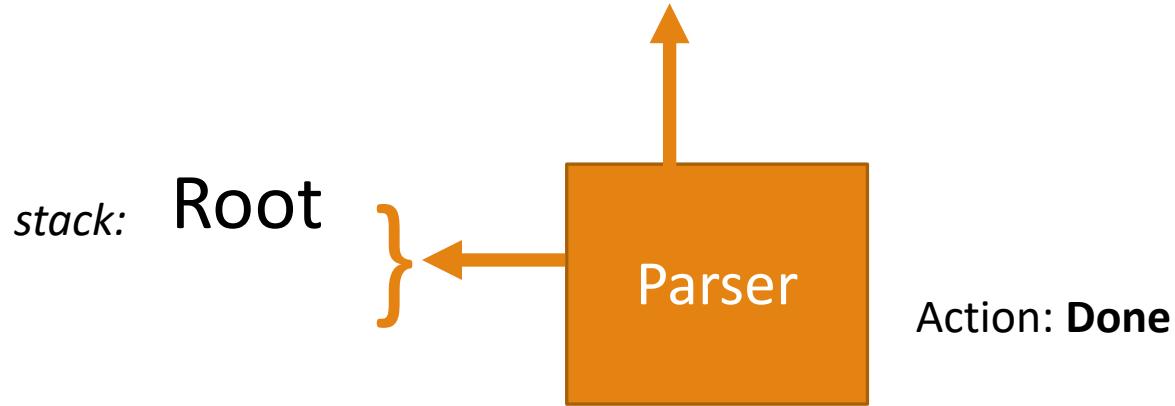
input buffer:



input buffer:



input buffer:



Running time

The transition-based parser takes a single pass through the input sentence. It places each word on the stack exactly once, and pops each word exactly once.

The running time is therefore $O(N)$, for sentence length of N .

The algorithm is **greedy** and makes its best prediction for which **operation** should be take at each time step.

How does it decide between the 3 options? The Oracle

Creating the Oracle

SOTA transition-based systems use supervised machine learning methods to train classifiers that play the role of the **oracle**, which takes in as input a configuration and returns as output a transition operator.

Problem: What about the training data? To train the oracle, we need configurations paired with transition operators, which aren't provided by the Treebanks...

Solution: simulate the operation of the parser by running the algorithm and relying on a new training oracle to give correct transition operators for each successive operation.

Training

While we can compute the score of tree as a sum of the scores of the edges that comprise it, each edge score can also be reduced to a **weighted sum of features extracted from it.**

$$\text{score}(S, e) = \sum_{i=1}^N w_i f_i(S, e) = w \cdot f$$

Com-

- Wordforms, lemmas and POS of the headword and dependent
- Corresponding features of contexts before, after and between words
- Word embeddings
- Dependency relation type
- Direction of the relation (to the right or to the left)
- Distance from the head to the dependent

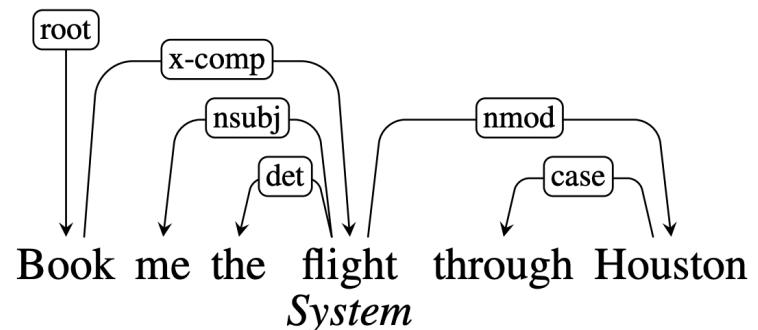
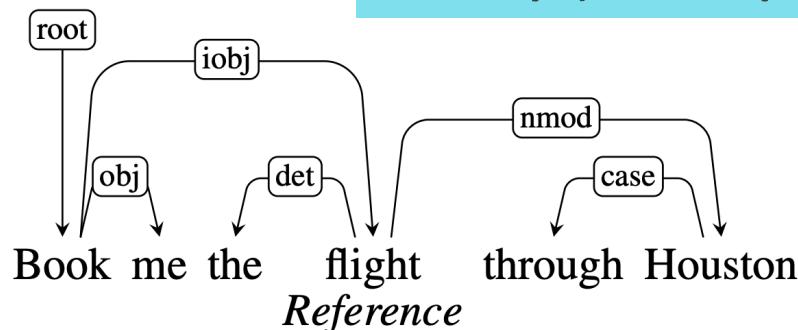
Evaluation

The common method for evaluating dependency parsers are **labeled attachment accuracy (LAS)** and **unlabeled attachment accuracy**.

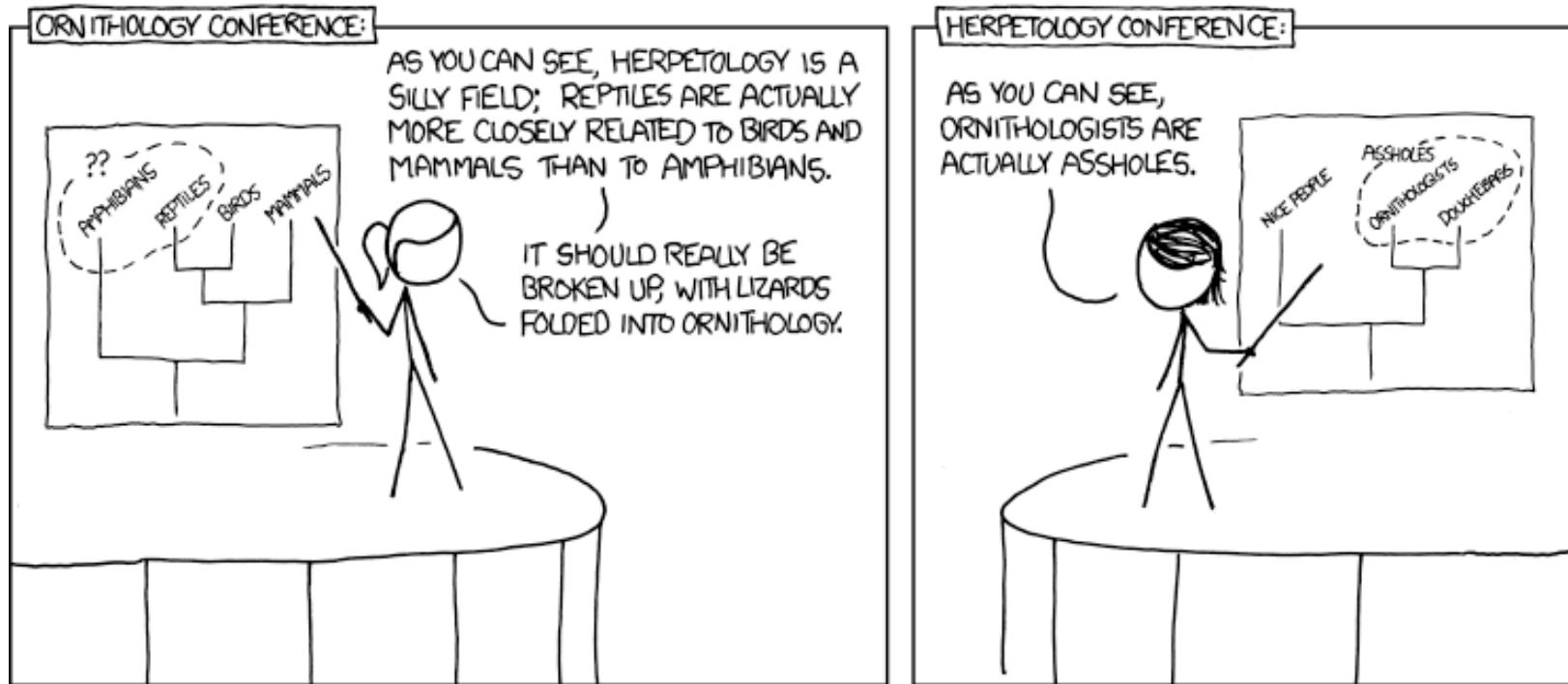
Labeled attachment refers to the proper assignment of a word to its head with the correct dependency relation.

Unlabeled attachment refers to the proper assignment of a word to its head ONLY (ignores dependency relation)

LAS = 4/6, UAS = 5/6



HW8: Learning Hypernyms



<http://computational-linguistics-class.org/homework/hypernyms/hypernyms.html>

Logical Representation of Sentence Meaning

JURAFSKY AND MARTIN CHAPTER 16

Computational Semantics

So far, we have discussed one kind of **meaning representation** in the form of word vectors or word embeddings.

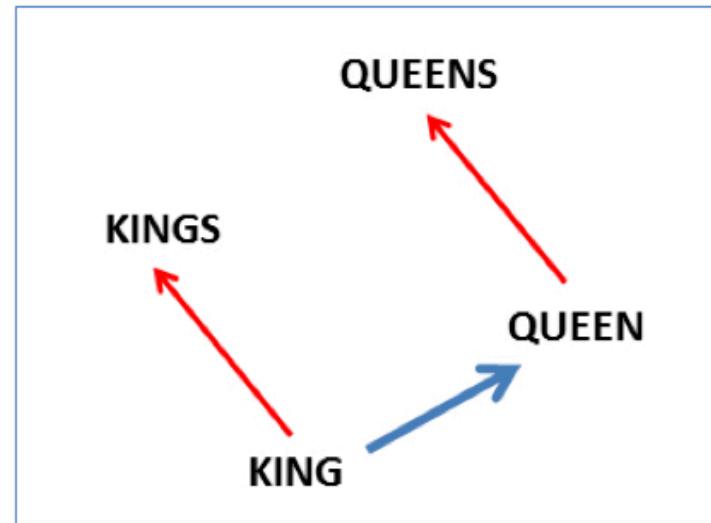
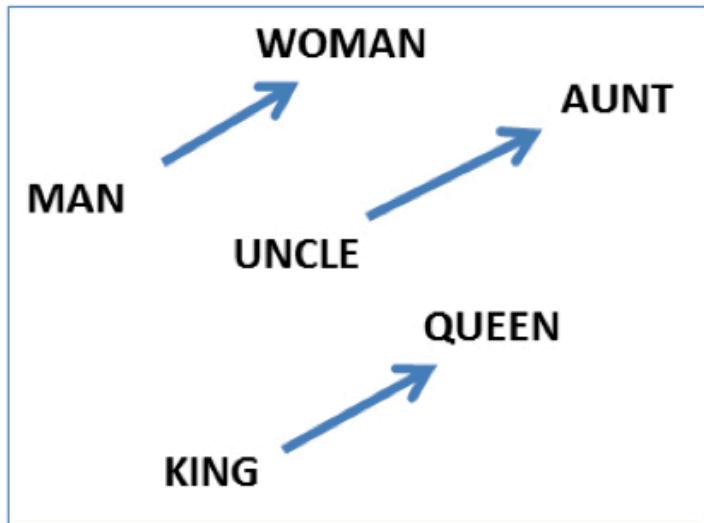
What kind of **reasoning** does that representation allow us to draw?

1. Words are similar to each other
2. Analogical reasoning

Analogy: Embeddings capture relational meaning

$\text{vector('king')} - \text{vector('man')} + \text{vector('woman')} \approx \text{vector('queen')}$

$\text{vector('Paris')} - \text{vector('France')} + \text{vector('Italy')} \approx \text{vector('Rome')}$



Computational Semantics

An important kind of reasoning that word embeddings don't support is **logical inference** like this:

All animals have an ulnar artery.

Dogs are a kind of animal.

⇒

All dogs have an ulnar artery

Today we will discuss **meaning representations** that allow us to link linguistics structures (words and sentences) onto a representation of the state of the world, and perform inferences.

Semantic Parsing

The process whereby **meaning representations** are created and assigned to linguistic inputs is called **semantic parsing** or **semantic analysis**.

Linguistic Expression: *I have a car*

Meaning representation:

$\exists e, y \text{Having}(e) \wedge \text{Haver}(e, \text{Speaker}) \wedge \text{HadThing}(e, y) \wedge \text{Car}(y)$

Meaning Representation

A meaning representation consists of structures composed from a set of **symbols**, or **representational vocabulary**, which correspond to

1. Objects (speaker, car)
2. Properties of objects (red (car))
3. Relation among objects (owns (speaker, car))

These describe some state of the world, which we are trying to represent and reason about.

Meaning Representation

Meaning representations can be viewed **both** as

1. Representations of the **meaning of a linguistic input**, and
2. As representations of **the state of affairs in the world**.

This is going to allow us to link linguistic inputs to the world and use our knowledge of the world to reason about whether statements are true, or to answer questions by returning objects that match variables in the question.

Desirable Properties for Meaning Representations

1. Verifiability
2. Unambiguous Representations
3. Canonical Forms
4. Make Inferences
5. Match variables

Verifiability

One application that we would like to use meaning representations for is to support **question answering systems** against a **knowledge base**.

Does Zahav serve vegetarian food?

We want a representation like `serves(Zahav, vegetarian)`
representation that could be queried against Yelp's KB.

Zahav  Claimed



\$\$\$ · Middle Eastern [Edit](#)

[Write a Review](#) [Add Photo](#) [Share](#) [Save](#)

Popular Dishes



[View Full Menu](#)

Amenities

 **Health Score 91 out of 100**

Powered by HDScores

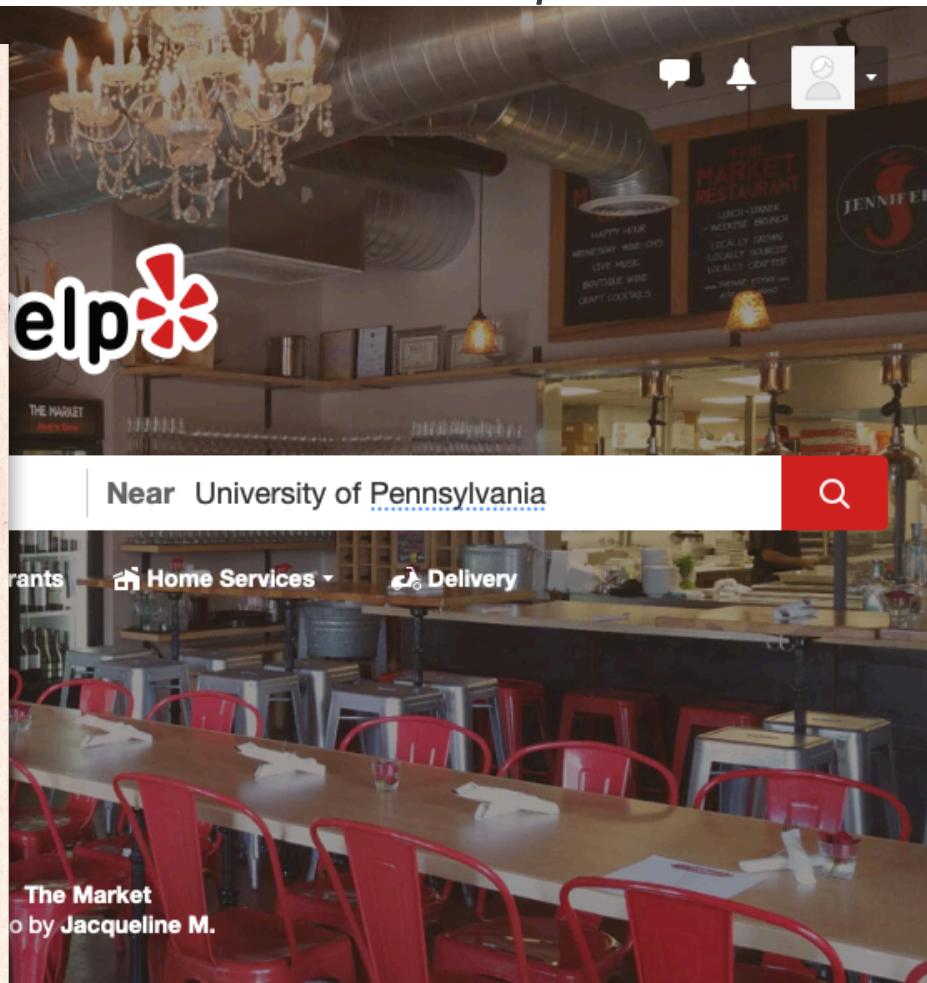


Vegetarian Options Yes

[21 More Attributes](#)

Unambiguous representation

I want to eat someplace that's near Penn's campus.



Unambiguous representation

I want to eat someplace that's near Penn's campus.

Our **meaning representations** itself cannot be ambiguous, so that the system can reason over a representation that means either one thing or the other in order to decide how to answer.

Note: Vagueness is OK!

Vagueness leaves some parts of the meaning underspecified but doesn't give rise to multiple representations,

Canonical form

Distinct inputs that mean the same thing should have the same meaning representation.

1. Does Zahav have vegetarian dishes?
2. Do they have vegetarian food at Zahav?
3. Are vegetarian dishes served at Zahav?
4. Does Zahav serve vegetarian fare?

This is related to the task of *paraphrase identification*. Variations can be syntactic as well as lexical.

Inference

A system needs to draw conclusions based on the meaning representation of inputs *and* its background knowledge in order to perform **inference**.

1. Does Zahav have vegetarian dishes?
2. **Can vegetarians eat at Zahav?**

These are different questions, and we need to use our commonsense reasoning to be able to answer them.

A system will need to use serves (Zahav, vegetarian) and other background knowledge to make the inference.

Variables

Finally, meaning representations must support variables that are not connected to a particular object.

1. I would like to find a restaurant where I can get vegetarian food.
2. Which restaurants serve vegetarian food?

We will need to have a meaning representation like
serves (**x**, vegetarian)

Where **x** can be replaced by some object in the KB that matches the whole proposition.

Model-Theoretic Semantics

A **model** allows us to bridge the gap between a formal representation and the world. The model stands in for a particular state of affairs in the world.

The **domain** of a model is the set of objects that are being represented. Each distinct thing (*person, restaurant, cuisine*) corresponds to a unique element in the domain

Properties of objects (like whether a restaurant is *expensive*) in a model correspond to sets of objects.

Relations between object (like whether a restaurant *serves* a cuisine) are are sets of tuples.

Domain	$\mathcal{D} = \{a, b, c, d, e, f, g, h, i, j\}$
Matthew, Franco, Katie and Caroline	a, b, c, d
Frasca, Med, Rio	e, f, g
Italian, Mexican, Eclectic	h, i, j

Properties

Noisy $Noisy = \{e, f, g\}$

Frasca, Med, and Rio are noisy

Relations

Likes

Matthew likes the Med

Katie likes the Med and Rio

Franco likes Frasca

Caroline likes the Med and Rio

$Likes = \{\langle a, f \rangle, \langle c, f \rangle, \langle c, g \rangle,$
 $\langle b, e \rangle, \langle d, f \rangle, \langle d, g \rangle\}$

Serves

Med serves eclectic

Rio serves Mexican

Frasca serves Italian

$Serves = \{\langle f, j \rangle, \langle g, i \rangle, \langle e, h \rangle\}$

Domain

Matthew, Franco, Katie and Caroline
Frasca, Med, Rio
Italian, Mexican, Eclectic

Properties

Noisy

Frasca, Med, and Rio are noisy

Relations

Likes

Matthew likes the Med

Katie likes the Med and Rio

Franco likes Frasca

Caroline likes the Med and Rio

Serves

Med serves eclectic

Rio serves Mexican

Frasca serves Italian

$$\mathcal{D} = \{a, b, c, d, e, f, g, h, i, j\}$$

$$a, b, c, d$$

$$e, f, g$$

Katie likes Rio

Katie → c

Rio → g

likes → Likes

$$\text{Likes} = \{\langle a, f \rangle, \langle c, f \rangle, \langle c, g \rangle, \langle b, e \rangle, \langle d, f \rangle, \langle d, g \rangle\}$$

$\langle c, g \rangle \in \text{Likes}$

so *Katie likes Rio*

is True

$\} \}$

Denotation and Interpretation

A meaning representation has open-ended **vocabulary** of names for the objects, properties, and relations that make up the world we're trying to represent.

Each element of the vocabulary must have a **denotation** in the model, meaning that every element corresponds to a fixed, well-defined part of the model.

The function that maps from the **vocabulary** to the proper **denotations** in the model is an **interpretation**.

More complex sentences

1. Katie likes the Rio and Matthew likes the Med.
2. Katie and Caroline like the same restaurants.
3. Franco likes noisy, expensive restaurants.
4. Not everybody likes Frasca.

In order to verify whether the meaning representations corresponding to these sentences are true in our model, we need an additional set of **logical operators** like **and**, **or**, **not**, and **quantifiers**, and corresponding **truth tables**.

*Assessing the **truth conditions** of complex examples still just involves simple set operations.*

First-Order Logic

FOL is a meaning representation language that satisfies the desirable qualities that we outlined. It provides a computational basis for **verifiability** and **inference**.

It doesn't have many requirements other than the represented world consists of objects, properties of objects, and relations among objects.

Basics of FOL

A **term** in FOL can consist of a **constant**, a **function** or a **variable**.

Constants are the objects in the world model

Functions are mapping to unique objects and can be expressed like LocationOf (Zahav)

Variables let us make assertions and draw inferences about objects without referring to a named object.

Relations

Predicates are symbols that name the relations that hold among a fixed number of objects.

A FOL representation for *Zahav serves vegetarian food* might look like the following formula:

Serves (Zahav, Vegetarian)

Predicates can have different number of arguments so the formula for *Zahav is a restaurant*

Restaurant (Zahav)

Logical Connectives

We can conjoin formula with logical connectives like **and** (\wedge), **or** (\vee), **not** (\neg), and **implies** (\Rightarrow)

Each one has a **truth table**:

P	Q	$P \vee Q$
<i>False</i>	<i>False</i>	<i>False</i>
<i>False</i>	<i>True</i>	<i>True</i>
<i>True</i>	<i>False</i>	<i>True</i>
<i>True</i>	<i>True</i>	<i>True</i>

Quantifiers and Variables

There are two quantifiers in FOL:

1. \exists – There exists
2. \forall – For all

For an **indefinite noun phrase** like

a restaurant that serves Mexican food near Penn

We use the existential quantifier and a variable.

$\exists x \text{ Restaurant}(x) \wedge \text{Serves}(x, \text{MexicanFood}) \wedge \text{Near}((\text{LocationOf}(x), \text{LocationOf}(Penn)))$

Quantifiers and Variables

All restaurants in Philly are closed.

$$\begin{aligned} & \forall x \text{Restaurant}(x) \wedge \text{Is}((\text{LocationOf}(x), \\ & \quad \text{Philadelphia}) \\ & \Rightarrow \text{Closed}(x,) \end{aligned}$$