THE OHIO STATE UNIVERSITY

CSE 5525: Foundations of Speech and Language Processing

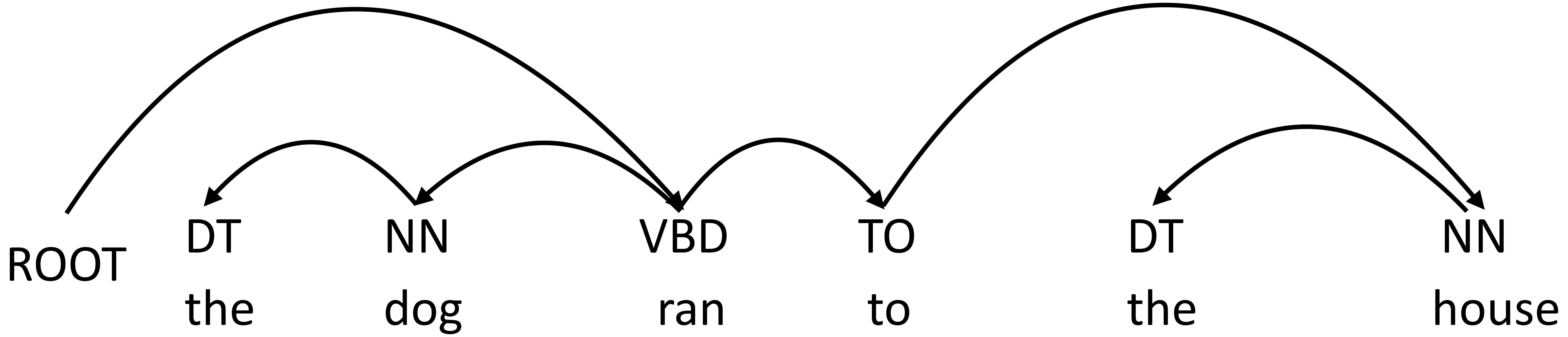
Semantics

Huan Sun (CSE@OSU)

Many thanks to Prof. Greg Durrett @ UT Austin for sharing his slides.
Some images/examples were from the two textbooks by (1) Jurafsky and Martin and (2) Eisenstein.

Recall: Dependencies

- Dependency syntax: syntactic structure is defined by dependencies
  - Head (parent, governor) connected to dependent (child, modifier)
  - Each word has exactly one parent except for the ROOT symbol
  - Dependencies must form a directed acyclic graph



Recall: arc standard approach; Shift-reduce parsing

- 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.
- 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;
- SHIFT: Remove the word from the front of the input buffer and push it onto the stack.

Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book $\rightarrow$ me)
3	[root, book]	[the, morning, flight]	SHIFT	
4	[root, book, the]	[morning, flight]	SHIFT	
5	[root, book, the, morning]	[flight]	SHIFT	
6	[root, book, the, morning, flight]	[]	LEFTARC	(morning $\leftarrow$ flight)
7	[root, book, the, flight]	[]	LEFTARC	(the $\leftarrow$ flight)
8	[root, book, flight]	[]	RIGHTARC	(book $\rightarrow$ flight)
9	[root, book]	[]	RIGHTARC	(root $\rightarrow$ book)
10	[root]	[]	Done	

Figure 15.7 Trace of a transition-based parse.

Where are we now?

- Early in the class: bags of word (classifiers) => sequences of words (sequence modeling)

- Now we can understand sentences in terms of tree structures as well

Where are we now?

- Early in the class: bags of word (classifiers) => sequences of words (sequence modeling)
- Now we can understand sentences in terms of tree structures as well
- Why is this useful? What does this allow us to do?
- We're going to see how syntactic parsing can be a stepping stone towards more formal representations of language meaning

Today

- Montague semantics*:
    - Model theoretic semantics
    - Compositional semantics with first-order logic
- CCG parsing for database queries
- Lambda-DCS for question answering

*The approach of algorithmically building up meaning representations from a series of operations on the syntactic structure of a sentence is generally attributed to the philosopher Richard Montague, who published a series of influential papers on the topic in the early 1970s (Chapter 12.3.1, Eisenstein)

Today

- Model theoretic semantics

- Compositional semantics with first-order logic

- CCG parsing for database queries

- Lambda-DCS for question answering

Model-Theoretic Semantics

Meaning Representation

- To be useful, a meaning representation must meet several criteria:

- **c1**: it should be unambiguous: unlike natural language, there should be exactly one meaning per statement;
- **c2**: it should provide a way to link language to external knowledge, observations, and actions;
- **c3**: it should support computational **inference**, so that meanings can be combined to derive additional knowledge;
- **c4**: it should be expressive enough to cover the full range of things that people talk about in natural language.

Meaning Representation

- To be useful, a meaning representation must meet several criteria:

- **c1**: it should be unambiguous: unlike natural language, there should be exactly one meaning per statement;
- **c2**: it should provide a way to link language to external knowledge, observations, and actions;
- **c3**: it should support computational **inference**, so that meanings can be combined to derive additional knowledge;
- **c4**: it should be expressive enough to cover the full range of things that people talk about in natural language.

Meaning Representation

- What does it mean for a statement to be unambiguous?

  Programming languages:

$$[\![5+3]\!] = [\![(4*4)-(3*3)+1]\!] = [\![((8))]\!] = 8.$$

- The output of a program (or, denotation) determined by constants (e.g., 4, 3) and relations (e.g., +, -)

Meaning Representation

- What does it mean for a statement to be unambiguous?

Programming languages:

$$[\![5+3]\!] = [\![(4*4)-(3*3)+1]\!] = [\![((8))]\!] = 8.$$

- The output of a program (or, denotation) determined by constants (e.g., 4, 3) and relations (e.g., +, -)

How about double(4)? Is it unambiguous?

Meaning Representation

- What does it mean for a statement to be unambiguous?

Programming languages:

$$[\![5+3]\!] = [\![(4*4)-(3*3)+1]\!] = [\![((8))]\!] = 8.$$

- The output of a program (or, denotation) determined by constants (e.g., 4, 3) and relations (e.g., +, -)

How about double(4)? Is it unambiguous?

depends on the meaning of double, which is defined in a *world model* *M*.

- As long as the denotation of a statement in model M can be computed unambiguously, it can be said to be unambiguous.

Model-Theoretic Semantics Approach

- Addresses c1 (no ambiguity) and c2 (connecting language with external knowledge/observations/actions)
  - e.g. connect the meaning of “the capital of Georgia” with a world model that includes a knowledge base

Model-Theoretic Semantics

- Addresses c1 (no ambiguity) and c2 (connecting language with external knowledge/observations/actions)

- Criterion c3: meaning representation supports inference (A=>B)
- Criterion c4: meaning representation be sufficiently expressive

- The most mature is the language of *first-order logic*

First-order Logic

- Powerful logic formalism including things like constants/entities, relations/predicates, and quantifiers

Lady Gaga sings

- sings is a *predicate* (with one argument); function f: entity → true/false

- sings(Lady Gaga) = true or false, which we can find out by executing this against some database (*world*)

- [[sings]] = *denotation*, set of entities which sing (found by executing this predicate on the *world* — we'll come back to this)

Quantifiers

- **Universal quantifier**: "forall" operator
  - $\forall x \; \text{sings}(x) \vee \text{dances}(x) \rightarrow \text{performs}(x)$

    "Everyone who sings or dances performs"

Quantifiers

- Universal quantifiers: "forall" operator
  - $\forall x\ \text{sings}(x) \lor \text{dances}(x) \rightarrow \text{performs}(x)$

    "Everyone who sings or dances performs"
- **Existential quantifiers**: "there exists" operator
  - $\exists x\ \text{sings}(x)$ *"Someone sings"*

Quantifiers

- Universal quantifiers: “forall” operator
  - $\forall x\ \text{sings}(x) \vee \text{dances}(x) \rightarrow \text{performs}(x)$

    “Everyone who sings or dances performs”
- **Existential quantifiers**: “there exists” operator
  - $\exists x\ \text{sings}(x)$ *“Someone sings”*
- Source of ambiguity! *“Everyone is friends with someone”*
  - $\forall x\ \exists y\ \text{friend}(x,y)$
  - $\exists y\ \forall x\ \text{friend}(x,y)$

Logic in NLP Tasks

- Question answering (or, semantic parsing):

  Who are all the American singers named Amy?

  $\lambda x.\ \mathrm{nationality}(x, \mathrm{USA}) \land \mathrm{sings}(x) \land \mathrm{firstName}(x, \mathrm{Amy})$

  - Lambda calculus: (logical form for the question)

Logic in NLP

- Question answering:

  Who are all the American singers named Amy?

  $\lambda x.\ \text{nationality}(x,\text{USA}) \wedge \text{sings}(x) \wedge \text{firstName}(x,\text{Amy})$

  - Function that maps from x to true/false, like `filter`. Execute this on the *world* to answer the question (x's that meet these constraints are answers)
  - Lambda calculus: powerful system for expressing these functions

Logic in NLP

- Question answering:

  Who are all the American singers named Amy?

  $\lambda x.\ \text{nationality}(x,\text{USA}) \wedge \text{sings}(x) \wedge \text{firstName}(x,\text{Amy})$

  - Function that maps from x to true/false, like `filter`. Execute this on the *world* to answer the question
  - Lambda calculus: powerful system for expressing these functions

- Information extraction: *Lady Gaga and Eminem are both musicians*

  $\text{musician}(\text{Lady Gaga}) \wedge \text{musician}(\text{Eminem})$

  - Can now do reasoning. If knowing: $\forall x\ \text{musician}(x) \Rightarrow \text{performer}(x)$

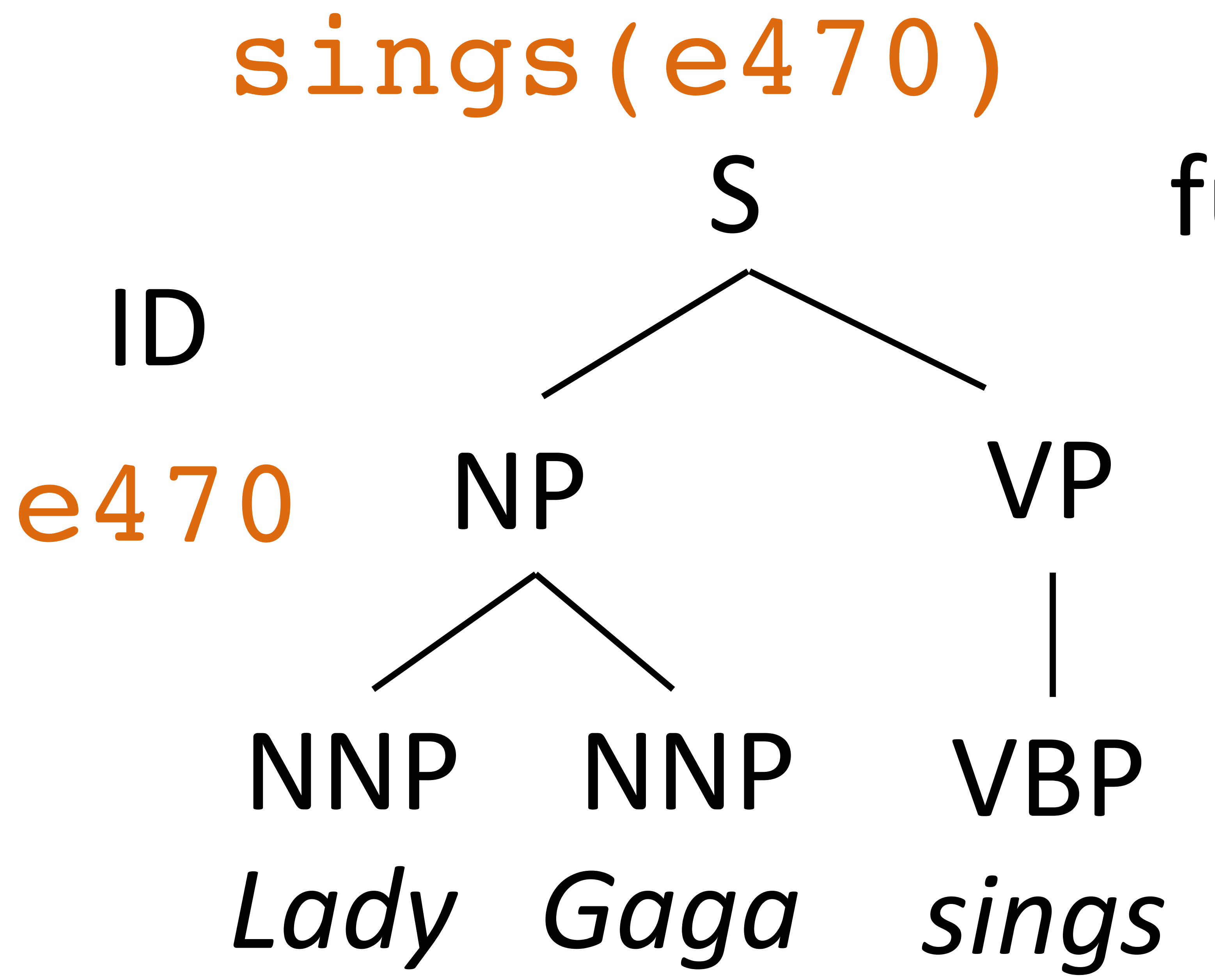
  Then: $\text{performer}(\text{Lady Gaga}) \wedge \text{performer}(\text{Eminem})$

Today

- Montague semantics*:
  - Model theoretic semantics
  - **Compositional semantics with first-order logic (For Offline Reading)**
- CCG parsing for database queries
- Lambda-DCS for question answering

*The approach of algorithmically building up meaning representations from a series of operations on the syntactic structure of a sentence is generally attributed to the philosopher Richard Montague, who published a series of influential papers on the topic in the early 1970s (Chapter 12.3.1, Eisenstein)

Montague Semantics



function application: apply this to e470

`λy. sings(y)`

`λy. sings(y)`

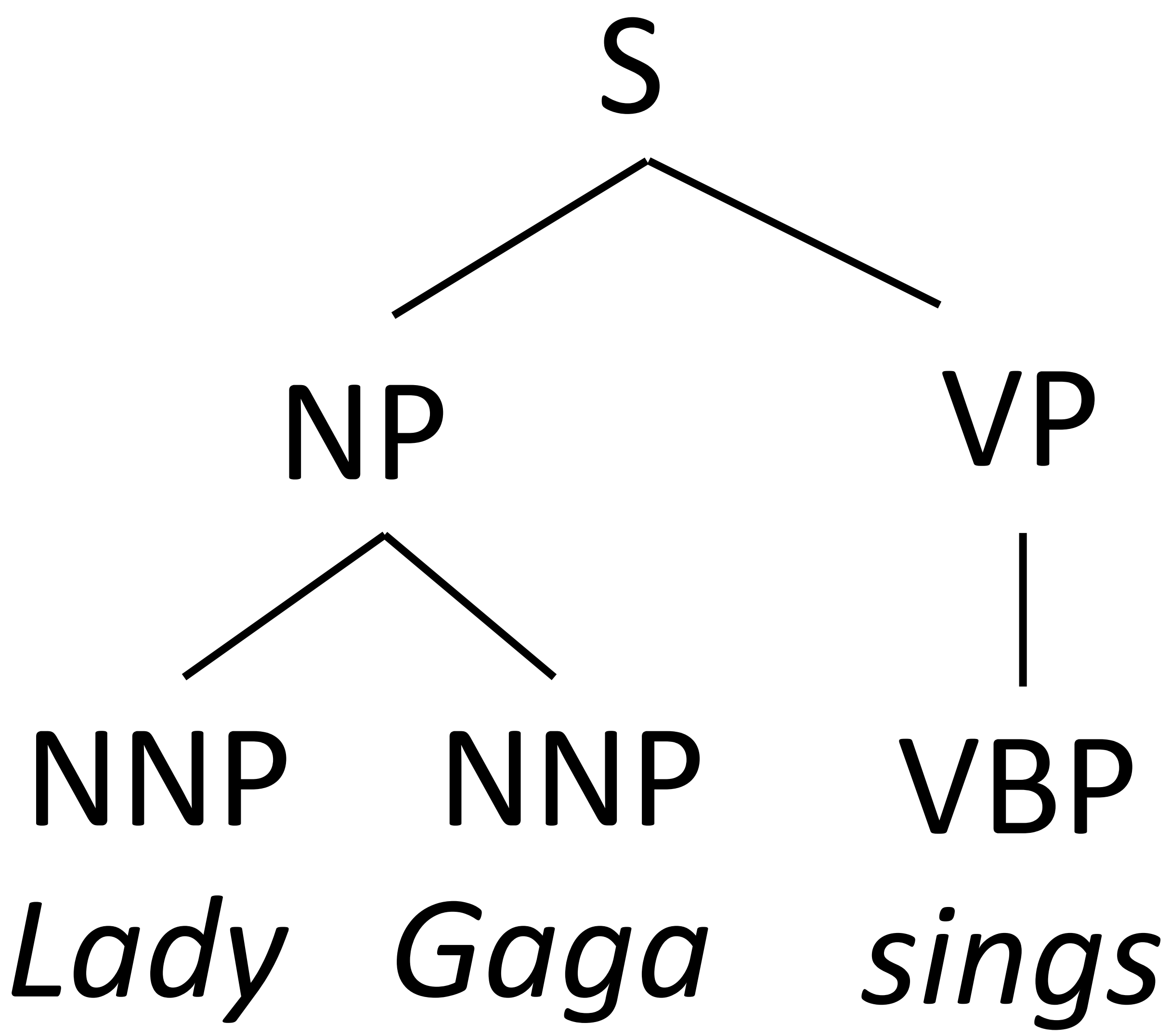
takes one argument (y, the entity) and returns a logical form `sings(y)`

- We can use the syntactic parse as a bridge to the lambda-calculus representation, build up a logical form (our model) *compositionally*

- **(For Offline Reading)**

Background

(For Offline Reading)



Id	Name	Alias	Birthdate	Sings?
e470	Stefani Germanotta	Lady Gaga	3/28/1986	T
e728	Marshall Mathers	Eminem	10/17/1972	T

- Database containing entities, predicates, etc.

- Sentence expresses something about the world which is either true or false

- Denotation: evaluation of some expression against this database

- `[[`*Lady Gaga*`]] = e470`

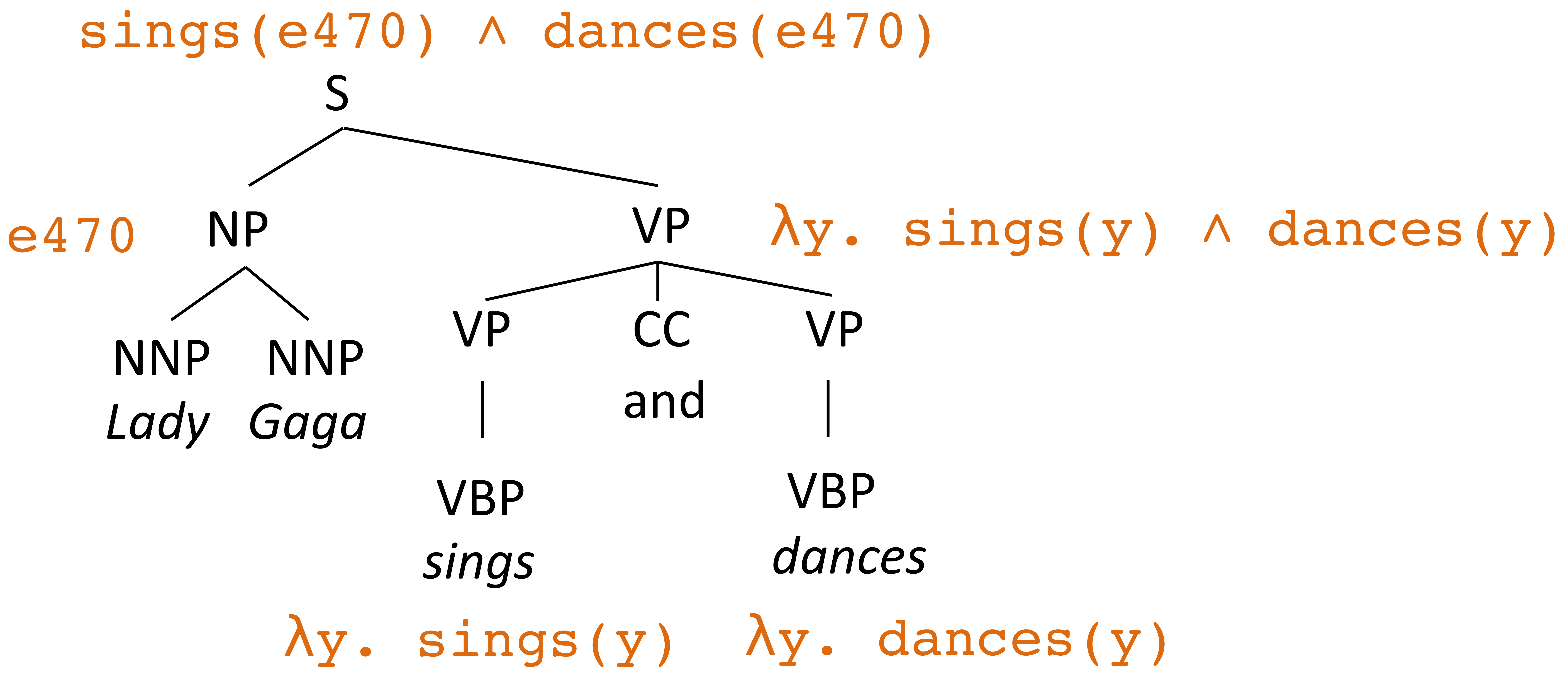
  denotation of this string is an entity

- `[[sings(e470)]] = True`

  denotation of this expression is T/F

Parses to Logical Forms

(For Offline Reading)



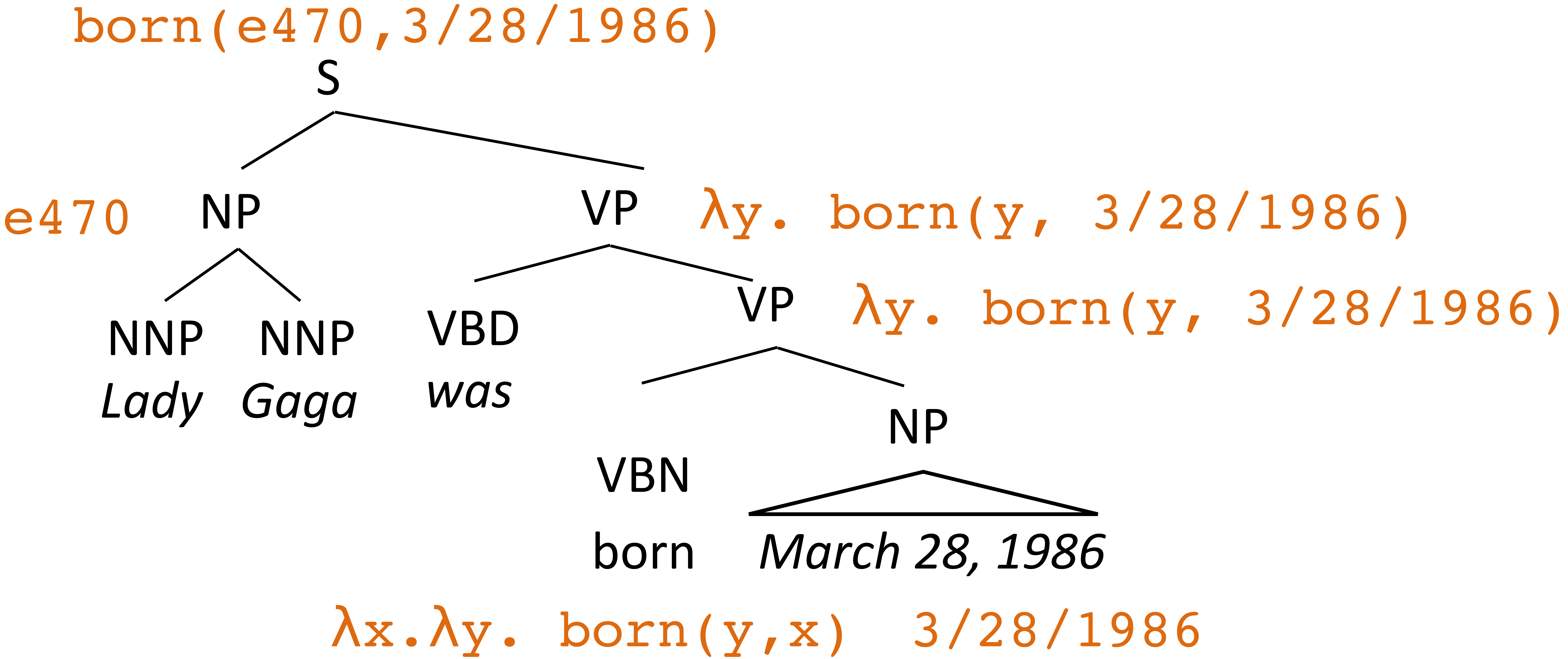
- General rules:

  VP: $\lambda y.\ a(y) \wedge b(y)$ -> VP: $\lambda y.\ a(y)$ CC VP: $\lambda y.\ b(y)$

  S: $f(x)$ -> NP: x VP: f

Parses to Logical Forms

(For Offline Reading)



- Function takes two arguments: first x (date), then y (entity)
- How to handle tense: should we indicate that this happened in the past?

Tricky things

(For Offline Reading)

- Adverbs/temporality: *Lady Gaga sang well yesterday*

  `sings(Lady Gaga, time=yesterday, manner=well)`

  - "Neo-Davidsonian" view of events: things with many properties:

  `∃e. type(e,sing) ∧ agent(e,e470) ∧ manner(e,well) ∧ time(e,…)`

- Quantification: *Everyone is friends with someone*

  `∃y ∀x friend(x,y)` (one friend)

  `∀x ∃y friend(x,y)` (different friends)

  - Same syntactic parse for both! So syntax doesn't resolve all ambiguities

- Indefinite: *Amy ate a waffle* `∃w. waffle(w) ∧ ate(Amy,w)`
- Generic: *Cats eat mice* (all cats eat mice? most cats? some cats?)

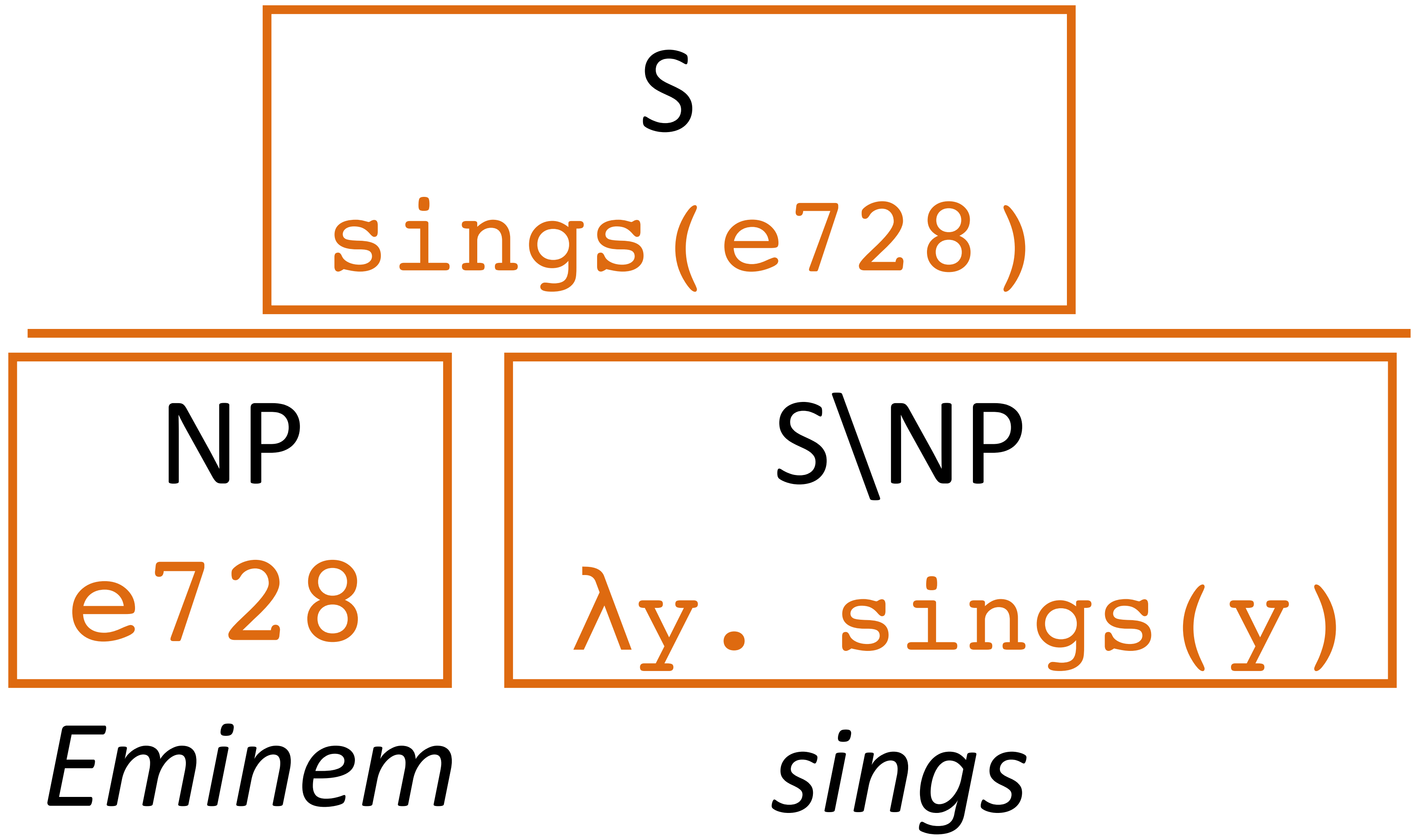
Semantic Parsing

- For question answering, syntactic parsing doesn't tell you everything you want to know, but indicates the right structure
- Solution: *semantic parsing* (many forms of this task depending on semantic formalisms)
- Two today: CCG (looks like what we've been doing) and lambda-DCS
- Applications: database querying/question answer: produce lambda-calculus expressions that can be executed in these contexts

CCG Parsing

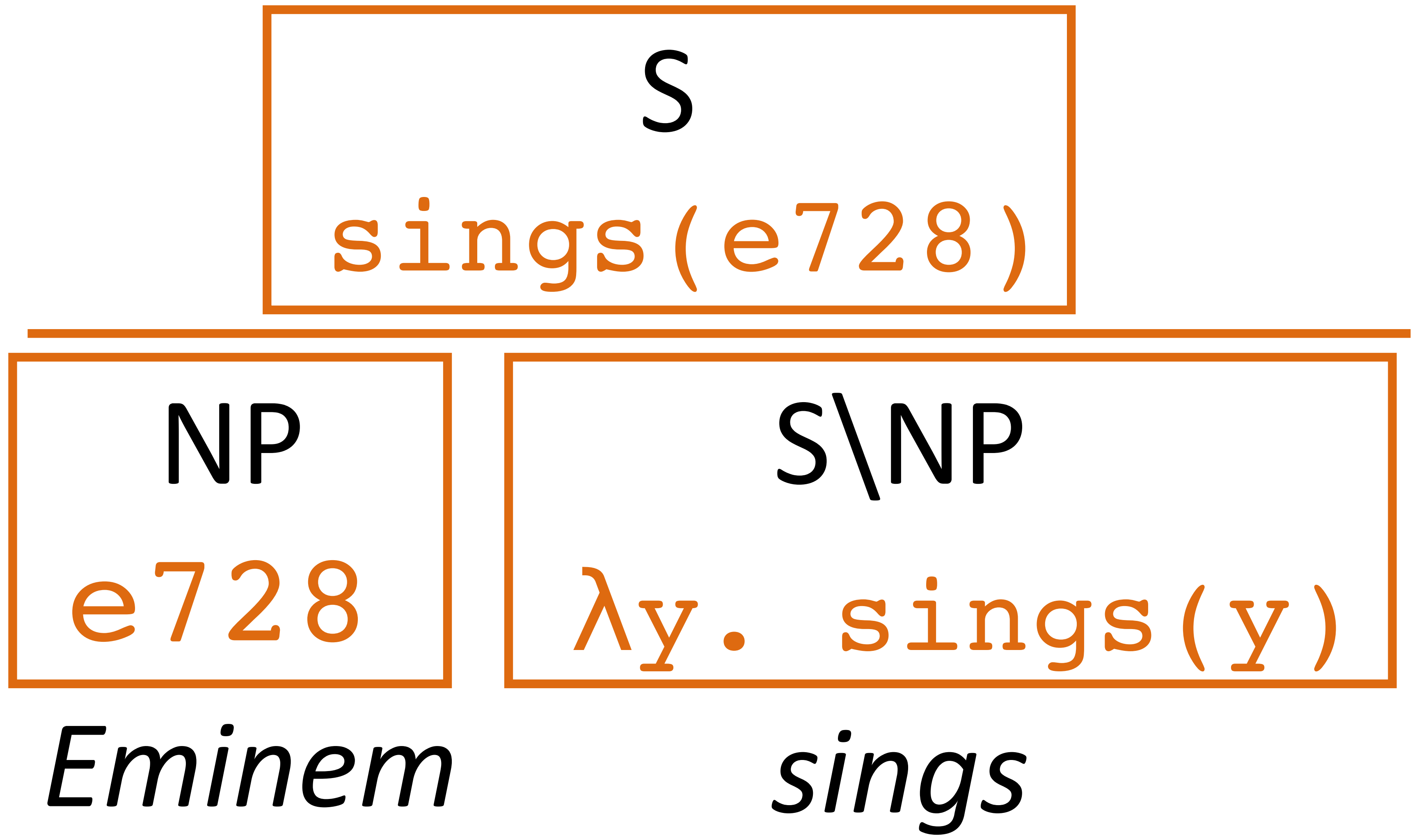
Combinatory Categorial Grammar

- Steedman+Szabolcsi (1980s): formalism bridging syntax and semantics
- Parallel derivations of syntactic parse and lambda calculus expression



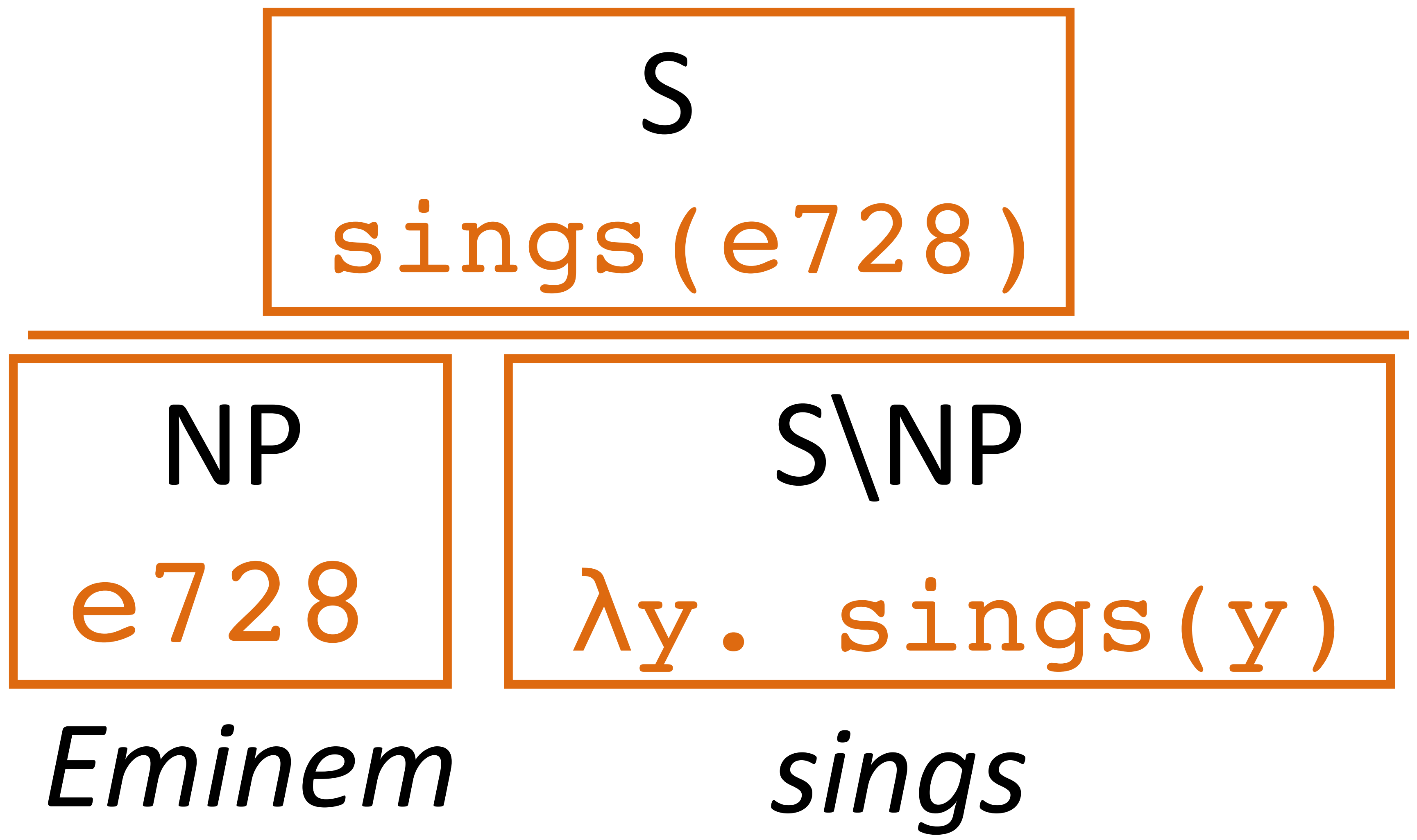
Combinatory Categorial Grammar

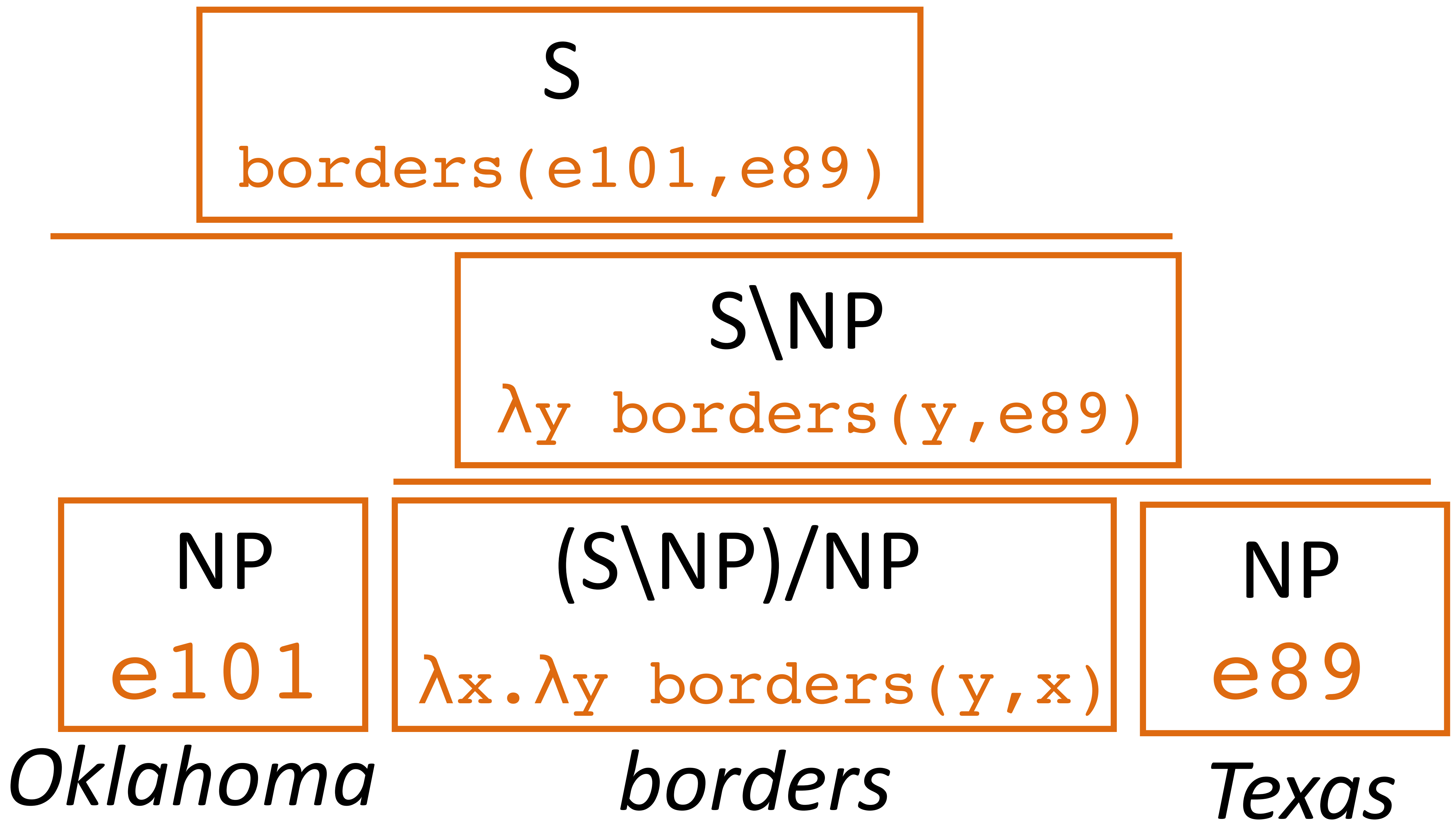
- Steedman+Szabolcsi (1980s): formalism bridging syntax and semantics
- Parallel derivations of syntactic parse and lambda calculus expression
- Syntactic categories (for this lecture): S, NP, "slash" categories
- S\NP: "if I combine with an NP on my left side, I form a sentence" — verb or vp
- When you apply this, there has to be a parallel instance of function application on the semantics side



Combinatory Categorial Grammar

- Steedman+Szabolcsi 1980s: formalism bridging syntax and semantics
- Syntactic categories (for this lecture): S, NP, “slash” categories
  - S\NP: “if I combine with an NP on my left side, I form a sentence” — verb
  - (S\NP)/NP: “I need an NP on my right and then on my left” — verb with a direct object





CCG Parsing

$$
\begin{array}{cccc}
\text{What} & \text{states} & \text{border} & \text{Texas} \\
\hline
(S/(S\backslash NP))/N & N & (S\backslash NP)/NP & NP \\
\lambda f.\lambda g.\lambda x.f(x) \wedge g(x) & \lambda x.state(x) & \lambda x.\lambda y.borders(y,x) & texas \\
 & & \multicolumn{2}{c}{\overline{\qquad\qquad\qquad\qquad}>} \\
 & & \multicolumn{2}{c}{(S\backslash NP)} \\
 & & \multicolumn{2}{c}{\lambda y.borders(y, texas)}
\end{array}
$$

- “What” is a ***very*** complex type: needs a noun and needs a S\NP to form a sentence. S\NP is basically a verb or verb phrase (*border Texas*)

Zettlemoyer and Collins (2005)

CCG Parsing

$$
\begin{array}{cccc}
\text{What} & \text{states} & \text{border} & \text{Texas} \\
\hline
(S/(S\backslash NP))/N & N & (S\backslash NP)/NP & NP \\
\lambda f.\lambda g.\lambda x.f(x) \wedge g(x) & \lambda x.state(x) & \lambda x.\lambda y.borders(y,x) & texas
\end{array}
$$

$$
\xrightarrow{\qquad\qquad\qquad} > \qquad\qquad \xrightarrow{\qquad\qquad\qquad} >
$$

$$
\begin{array}{cc}
S/(S\backslash NP) & (S\backslash NP) \\
\lambda g.\lambda x.state(x) \wedge g(x) & \lambda y.borders(y, texas)
\end{array}
$$

$$
\xrightarrow{\qquad\qquad\qquad\qquad\qquad\qquad} >
$$

$$
\begin{array}{c}
S \\
\lambda x.state(x) \wedge borders(x, texas)
\end{array}
$$

- “What” is a ***very*** complex type: needs a noun and needs a S\NP to form a sentence. S\NP is basically a verb phrase (*border Texas*)

Zettlemoyer and Collins (2005)

CCG Parsing

$$
\begin{array}{cccc}
\text{What} & \text{states} & \text{border} & \text{Texas} \\
\hline
(S/(S\backslash NP))/N & N & (S\backslash NP)/NP & NP \\
\lambda f.\lambda g.\lambda x.f(x) \wedge g(x) & \lambda x.state(x) & \lambda x.\lambda y.borders(y,x) & texas \\
\hline
\multicolumn{2}{c}{S/(S\backslash NP) \quad >} & \multicolumn{2}{c}{(S\backslash NP) \quad >} \\
\multicolumn{2}{c}{\lambda g.\lambda x.state(x) \wedge g(x)} & \multicolumn{2}{c}{\lambda y.borders(y, texas)} \\
\hline
\multicolumn{4}{c}{S \quad >} \\
\multicolumn{4}{c}{\lambda x.state(x) \wedge borders(x, texas)}
\end{array}
$$

- "What" is a ***very*** complex type: needs a noun and needs a S\NP to form a sentence. S\NP is basically a verb phrase (*border Texas*)
- Lexicon is highly ambiguous — all the challenge of CCG parsing is in picking the right lexicon entries

Zettlemoyer and Collins (2005)

CCG Parsing

- Many ways to build these parsers

- One approach: run a "supertagger" (tagging the sentence with complex labels), then run the parser

What	states	border	Texas
$(S/(S\backslash NP))/N$	N	$(S\backslash NP)/NP$	NP
$\lambda f.\lambda g.\lambda x.f(x) \wedge g(x)$	$\lambda x.state(x)$	$\lambda x.\lambda y.borders(y,x)$	$texas$

  - Parsing is easy once you have the tags, so we've reduced it to a (hard) tagging problem

Zettlemoyer and Collins (2005)

Building CCG Parsers

- Model: log-linear model over derivations, with features on rules:

$$P(d|x) \propto \exp w^\top \left(\sum_{r \in d} f(r, x) \right)$$

f(S `sings(e728)` over NP `e728` and S\NP `λy. sings(y)`) = Indicator(S -> NP S\NP)

f(NP `e728`) *Eminem*

f(S\NP `λy. sings(y)`) = Indicator(S\NP -> *sings*)

sings

Zettlemoyer and Collins (2005)

Building CCG Parsers

- Training data looks like pairs of sentences and logical forms

What states border Texas $\lambda x.\ \texttt{state(x)} \wedge \texttt{borders(x, e89)}$

Zettlemoyer and Collins (2005)

Building CCG Parsers

- Training data looks like pairs of sentences and logical forms

What states border Texas `λx. state(x) ∧ borders(x, e89)`

- Problem: we don't know the derivation
  - _Texas_ corresponds to NP | `e89` in the logical form (easy to figure out)
  - _What_ corresponds to (S/(S\NP))/N | `λf.λg.λx. f(x) ∧ g(x)`
  - How do we infer that without being told it?

Zettlemoyer and Collins (2005)

Lexicon

- GENLEX: takes sentence S and logical form L. Break up logical form into chunks C(L), assume any substring of S might map to any chunk

What states border Texas $\lambda x.\ \mathtt{state}(x) \wedge \mathtt{borders}(x, \mathtt{e89})$

Zettlemoyer and Collins (2005)

Lexicon

- GENLEX: takes sentence S and logical form L. Break up logical form into chunks C(L), assume any substring of S might map to any chunk

What states border Texas $\lambda x.\ \mathtt{state}(x) \land \mathtt{borders}(x,\ \mathtt{e89})$

- Chunks inferred from the logic form based on rules:
  - NP: $\mathtt{e89}$
  - (S\NP)/NP: $\lambda x.\ \lambda y.\ \mathtt{borders}(x,y)$

Zettlemoyer and Collins (2005)

Lexicon

- GENLEX: takes sentence S and logical form L. Break up logical form into chunks C(L), assume any substring of S might map to any chunk

What states border Texas $\lambda x.\ \mathtt{state}(x) \wedge \mathtt{borders}(x,\ \mathtt{e89})$

- Chunks inferred from the logic form based on rules:
  - NP: `e89`
  - (S\NP)/NP: $\lambda x.\ \lambda y.\ \mathtt{borders}(x,y)$
- Any substring can parse to any of these in the lexicon
  - *Texas* -> NP: e89
  - *border Texas* -> NP: e89
  - *What states border Texas* -> NP: e89

  …

Zettlemoyer and Collins (2005)

Lexicon

- GENLEX: takes sentence S and logical form L. Break up logical form into chunks C(L), assume any substring of S might map to any chunk

What states border Texas $\lambda x.\ state(x) \wedge borders(x, e89)$

- Chunks inferred from the logic form based on rules:
  - NP: $e89$
  - (S\NP)/NP: $\lambda x.\ \lambda y.\ borders(x,y)$
- Any substring can parse to any of these in the lexicon
  - *Texas* -> NP: e89 is correct
  - *border Texas* -> NP: e89
  - *What states border Texas* -> NP: e89

  …

learning from data

Zettlemoyer and Collins (2005)

Applications

- GeoQuery: answering questions about states (~80% accuracy)
- Jobs: answering questions about job postings (~80% accuracy)
- ATIS: flight search
- Can do well on all of these tasks if you handcraft systems and use plenty of training data: these domains aren't that rich
- What about broader QA? A simpler form?

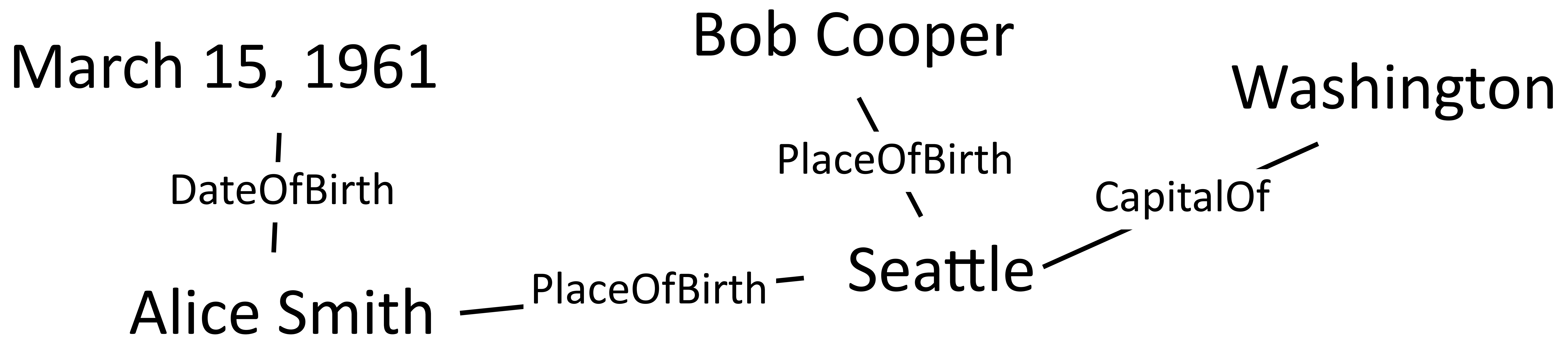
Lambda-DCS

Lambda-DCS

- Dependency-based compositional semantics — original version was less powerful than lambda calculus, lambda-DCS is as powerful

- Designed in the context of building a QA system from Freebase

- Freebase: set of entities and relations



- [[PlaceOfBirth]] = set of pairs of (person, location)

Liang et al. (2011), Liang (2013)

Lambda-DCS

Lambda-DCS	Lambda calculus
`Seattle`	`λx. x = Seattle`
`PlaceOfBirth`	`λx.λy. PlaceOfBirth(x,y)`
`PlaceOfBirth.Seattle`	`λx. PlaceOfBirth(x,Seattle)`

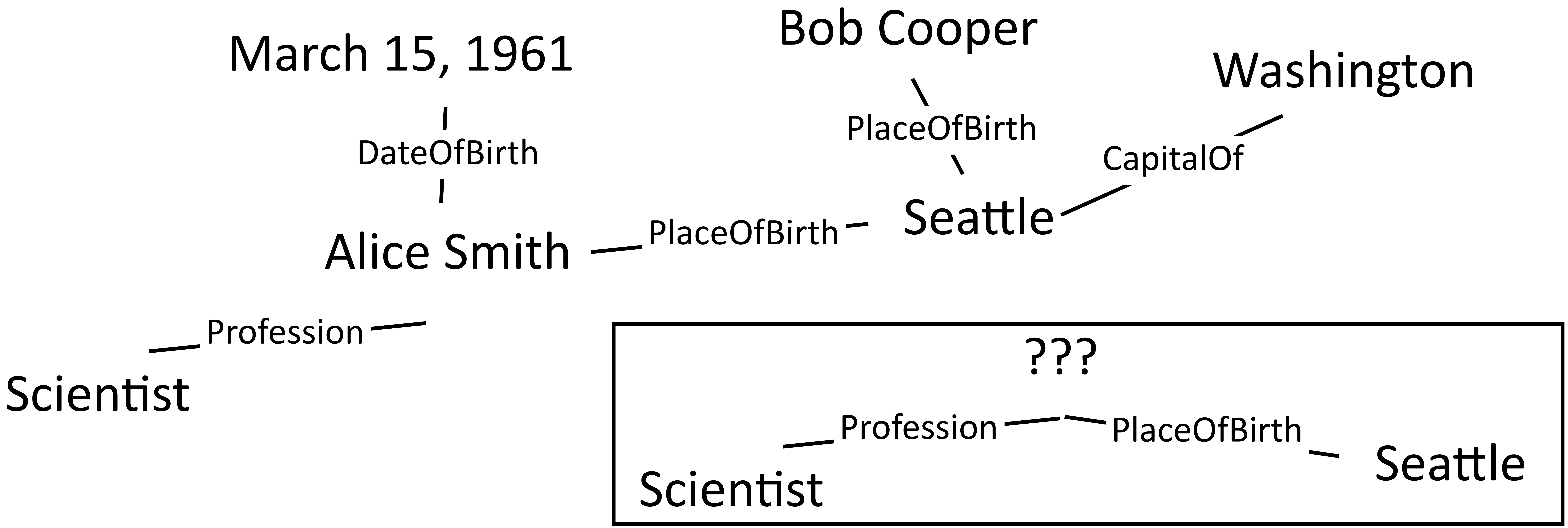
- Looks like a tree fragment over Freebase

??? —PlaceOfBirth— Seattle

`Profession.Scientist ∧` `PlaceOfBirth.Seattle`	`λx. Profession(x,Scientist)` `∧ PlaceOfBirth(x,Seattle)`

Liang et al. (2011), Liang (2013)

Lambda-DCS



"list of scientists born in Seattle"

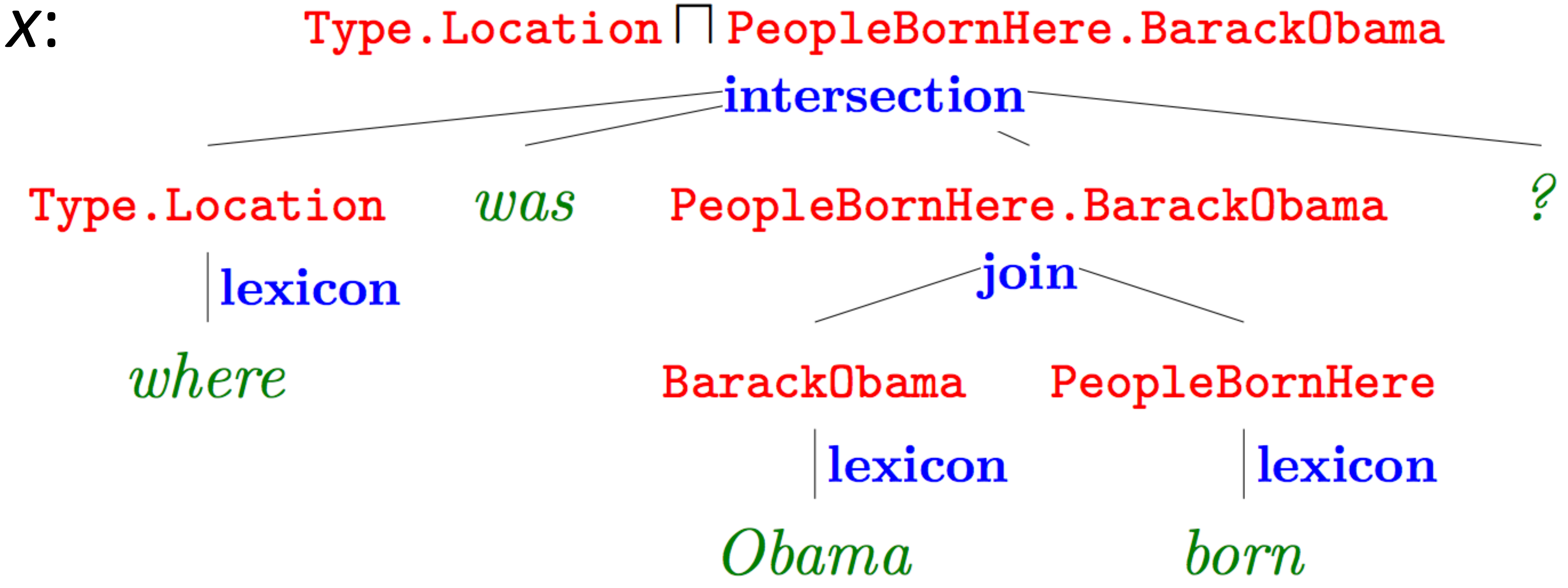
```
Profession.Scientist ∧
PlaceOfBirth.Seattle
```

- Execute this fragment against Freebase, returns Alice Smith (and others)

Liang et al. (2011), Liang (2013)

Parsing into Lambda-DCS

- Derivation *d* on sentence *x*:



- No more explicit syntax in these derivations like we had in CCG

- Building the lexicon: more sophisticated process than GENLEX, but can handle thousands of predicates

- Log-linear model with features on rules: $P(d|x) \propto \exp w^\top \left(\sum_{r \in d} f(r, x) \right)$

- Similar to CRF parsers

Berant et al. (2013)

Parsing with Lambda-DCS

- Learn just from question-answer pairs: maximize the likelihood of the right denotation *y* with the derivation *d* marginalized out

$$\mathcal{O}(\theta) = \sum_{i=1}^{n} \log \sum_{d \in D(x) : [\![d.z]\!]_{\mathcal{K}} = y_i} p_\theta(d \mid x_i).$$

sum over derivations *d* such that the denotation of *d* on knowledge base *K* is y_i

For each example:

Run beam search to get a set of derivations

Let d = highest-scoring derivation in the beam

Let d* = highest-scoring derivation in the beam *with correct denotation*

Do a structured perceptron update towards d* away from d

Berant et al. (2013)

Takeaways

- Can represent meaning with first order logic and lambda calculus
- Can bridge syntax and semantics and create semantic parsers that can interpret language into lambda-calculus expressions
- Useful for querying databases, question answering, etc.
- Recall: Previous encoder-decoder methods for doing this that rely less on having explicit grammars (HW3)