More λ-Calculus
Lexical Semantics

LING 571 — Deep Processing Techniques for NLP

October 30, 2019

Shane Steinert-Threlkeld

Announcements

- HW5:
  - readme: be detailed!
    - What problems you ran into (e.g. agreement), how you solved them, and exhibit of the solution
  - HSPG-style subcategorization extra credit: yes, but with max 100 points

- We will drop your lowest score when calculating final grades. But you still need to do all the assignments!

Ambiguity of the Week

- Derivative of an alleged Groucho Marx-ism:
- In the US, a woman gives birth every fifteen minutes.
  - We must find her and put a stop to it.

- Thank you scope ambiguity! (Not the same as attachment ambiguity.)

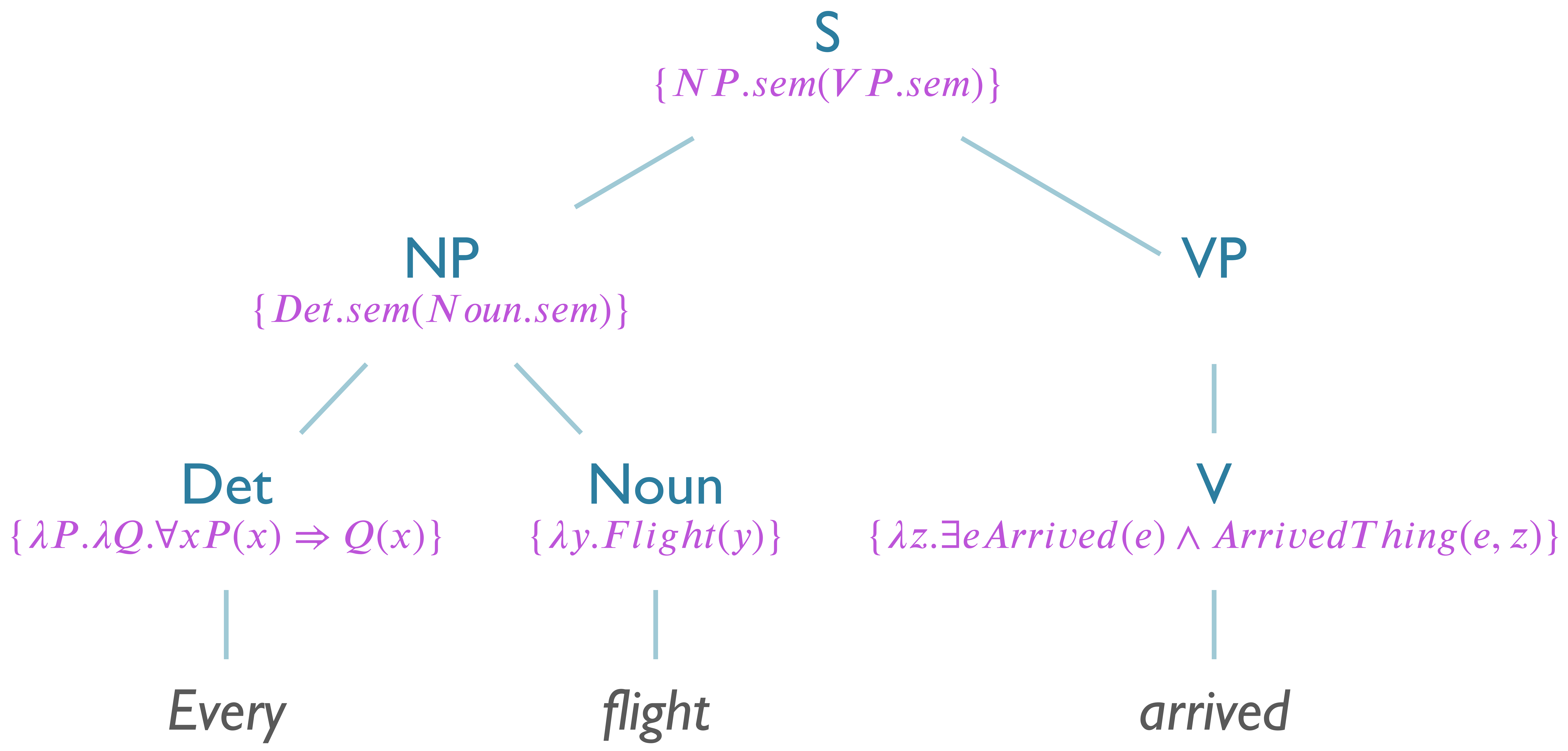
Roadmap

- More Lambdas
- Learning Semantic Parsers
- Lexical Semantics
  - Motivation & Definitions
  - Word Senses
  - Tasks:
    - Word sense disambiguation
    - Word sense similarity
  - **Distributional Similarity**

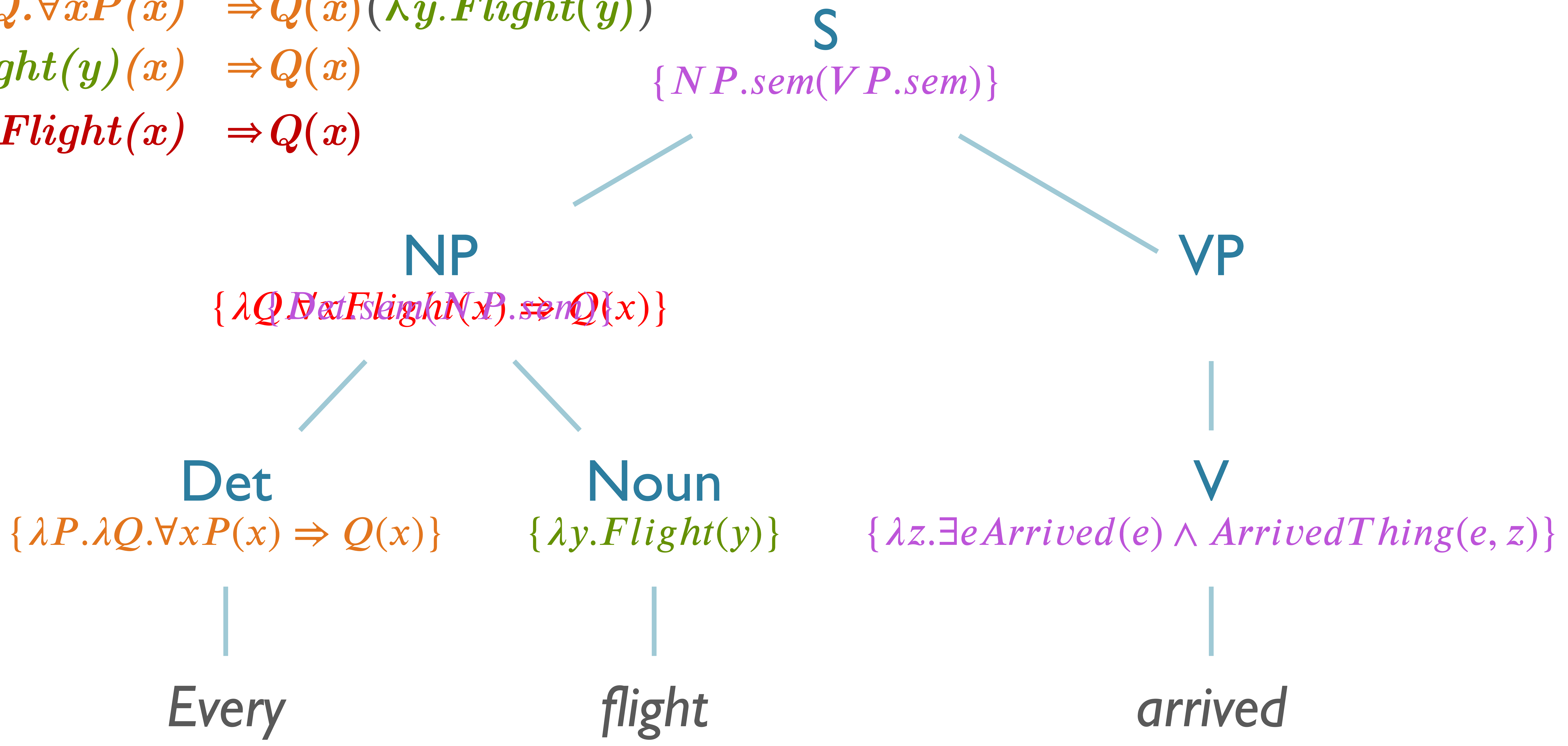
Creating Attachments

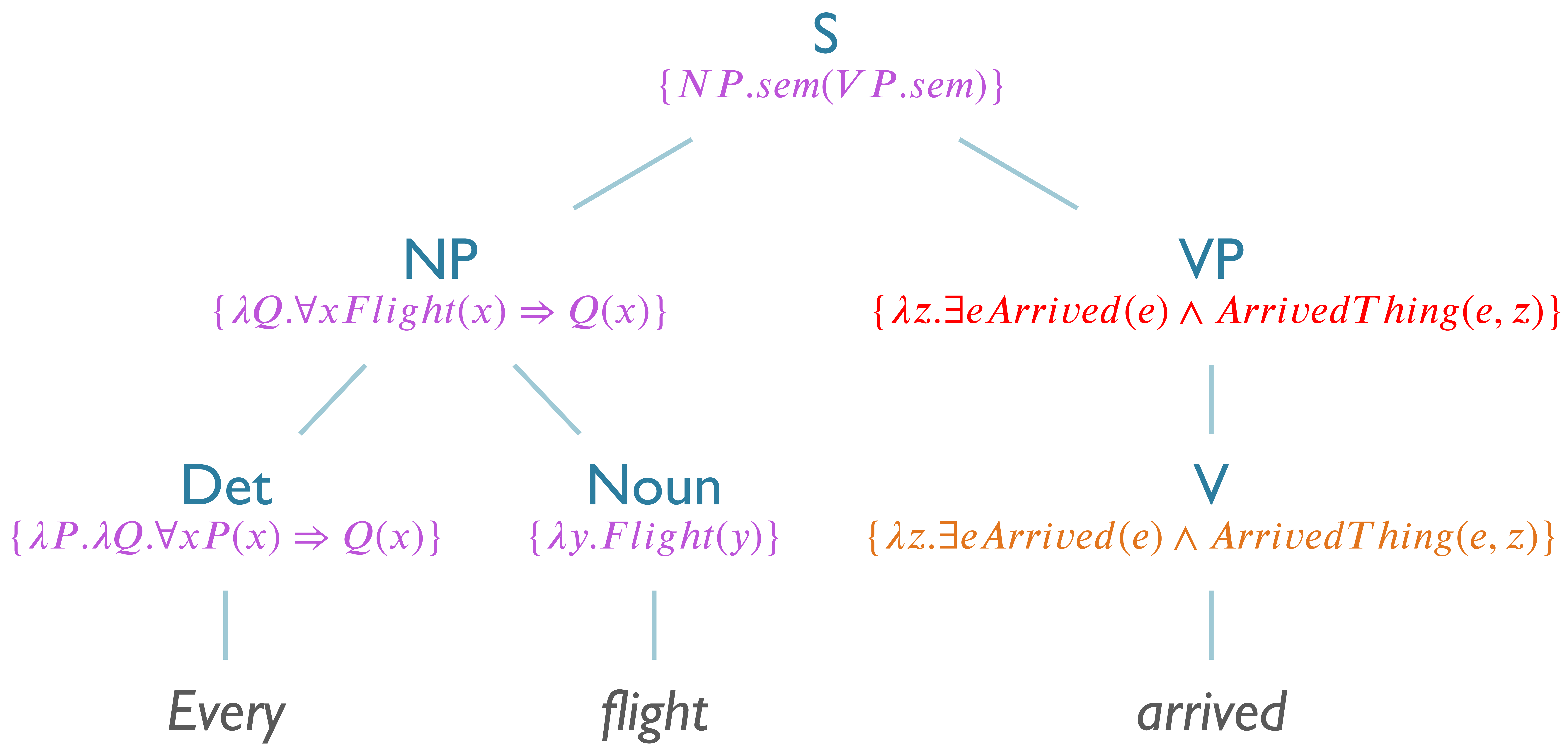
"Every flight arrived"

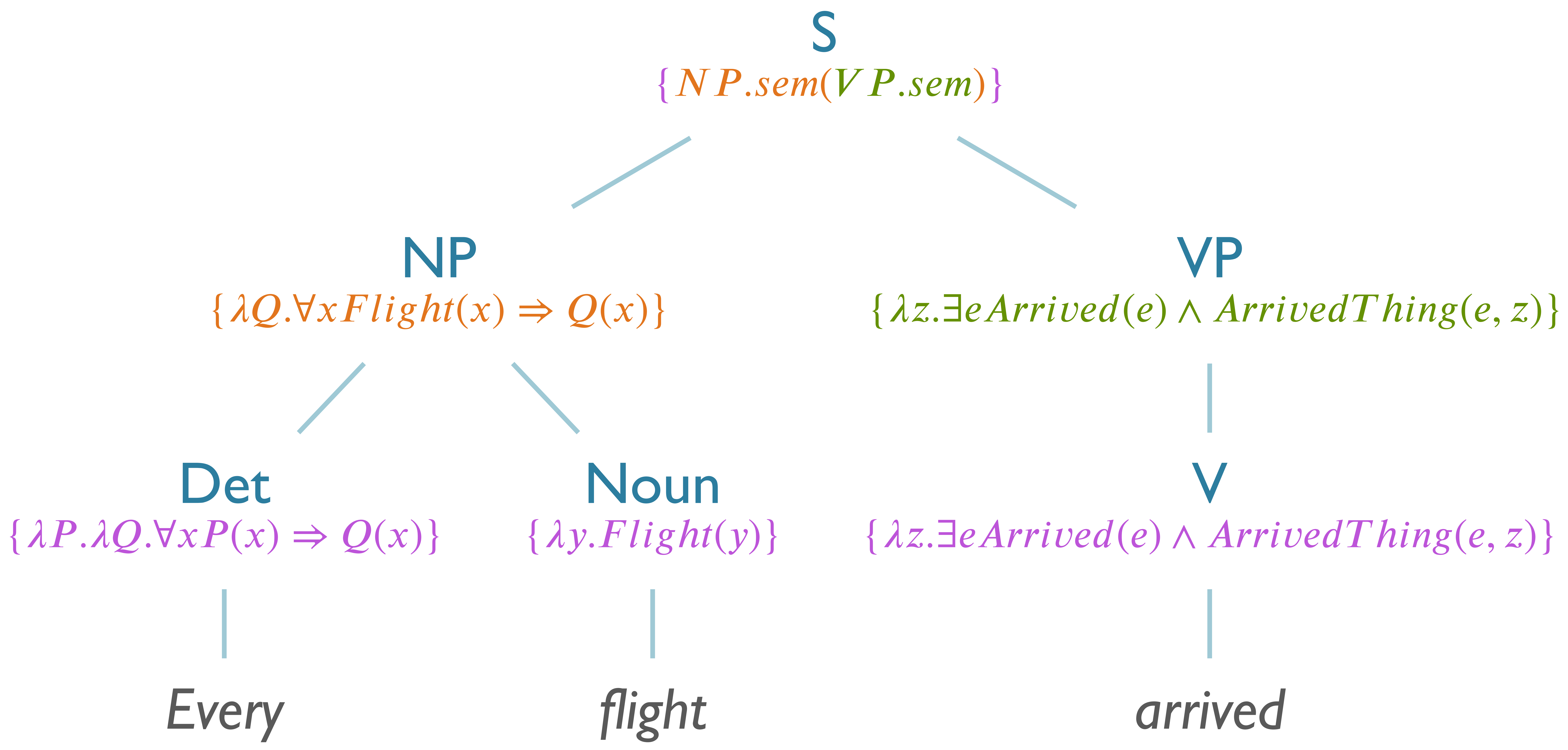
$Det \rightarrow$ 'Every'		$\{\ \lambda P.\lambda Q.\forall \boldsymbol{x}\ P(\boldsymbol{x}) \Rightarrow Q(\boldsymbol{x})\ \}$
$Noun \rightarrow$ 'flight'		$\{\ \lambda \boldsymbol{x}.Flight(\boldsymbol{x})\ \}$
$Verb \rightarrow$ 'arrived'		$\{\lambda \boldsymbol{y}.\exists \boldsymbol{e} Arrived(\boldsymbol{e}) \wedge ArrivedThing(\boldsymbol{e}, \boldsymbol{y})\}$
$VP \rightarrow Verb$		$\{\ Verb.sem\ \}$
$Nom \rightarrow Noun$		$\{\ Noun.sem\ \}$
$S \rightarrow NP\ VP$		$\{\ NP.sem(VP.sem)\ \}$
$NP \rightarrow Det\ Nom$		$\{\ Det.sem(Nom.sem)\ \}$


S
$\{NP.sem(VP.sem)\}$
NP
$\{Det.sem(Noun.sem)\}$
VP
Det
$\{\lambda P.\lambda Q.\forall x P(x) \Rightarrow Q(x)\}$
Noun
$\{\lambda y.Flight(y)\}$
V
$\{\lambda z.\exists e Arrived(e) \wedge ArrivedThing(e,z)\}$
Every
flight
arrived

$$
\begin{aligned}
NP &\rightarrow Det.sem(NP.sem) \\
\lambda P.\lambda Q.\forall x P(x) &\Rightarrow Q(x)(\lambda y.Flight(y)) \\
\lambda Q.\forall x \lambda y.Flight(y)(x) &\Rightarrow Q(x) \\
\lambda Q.\forall x Flight(x) &\Rightarrow Q(x)
\end{aligned}
$$


S
$\{NP.sem(VP.sem)\}$
NP
$\{Det.sem(NP.sem)\}$
$\{\lambda Q.\forall x Flight(x) \Rightarrow Q(x)\}$
VP
Det
$\{\lambda P.\lambda Q.\forall x P(x) \Rightarrow Q(x)\}$
Noun
$\{\lambda y.Flight(y)\}$
V
$\{\lambda z.\exists e Arrived(e) \wedge ArrivedThing(e,z)\}$
Every
flight
arrived


S
$\{NP.sem(VP.sem)\}$
NP
$\{\lambda Q.\forall x Flight(x) \Rightarrow Q(x)\}$
VP
$\{\lambda z.\exists e Arrived(e) \wedge ArrivedThing(e,z)\}$
Det
$\{\lambda P.\lambda Q.\forall x P(x) \Rightarrow Q(x)\}$
Noun
$\{\lambda y.Flight(y)\}$
V
$\{\lambda z.\exists e Arrived(e) \wedge ArrivedThing(e,z)\}$
Every
flight
arrived

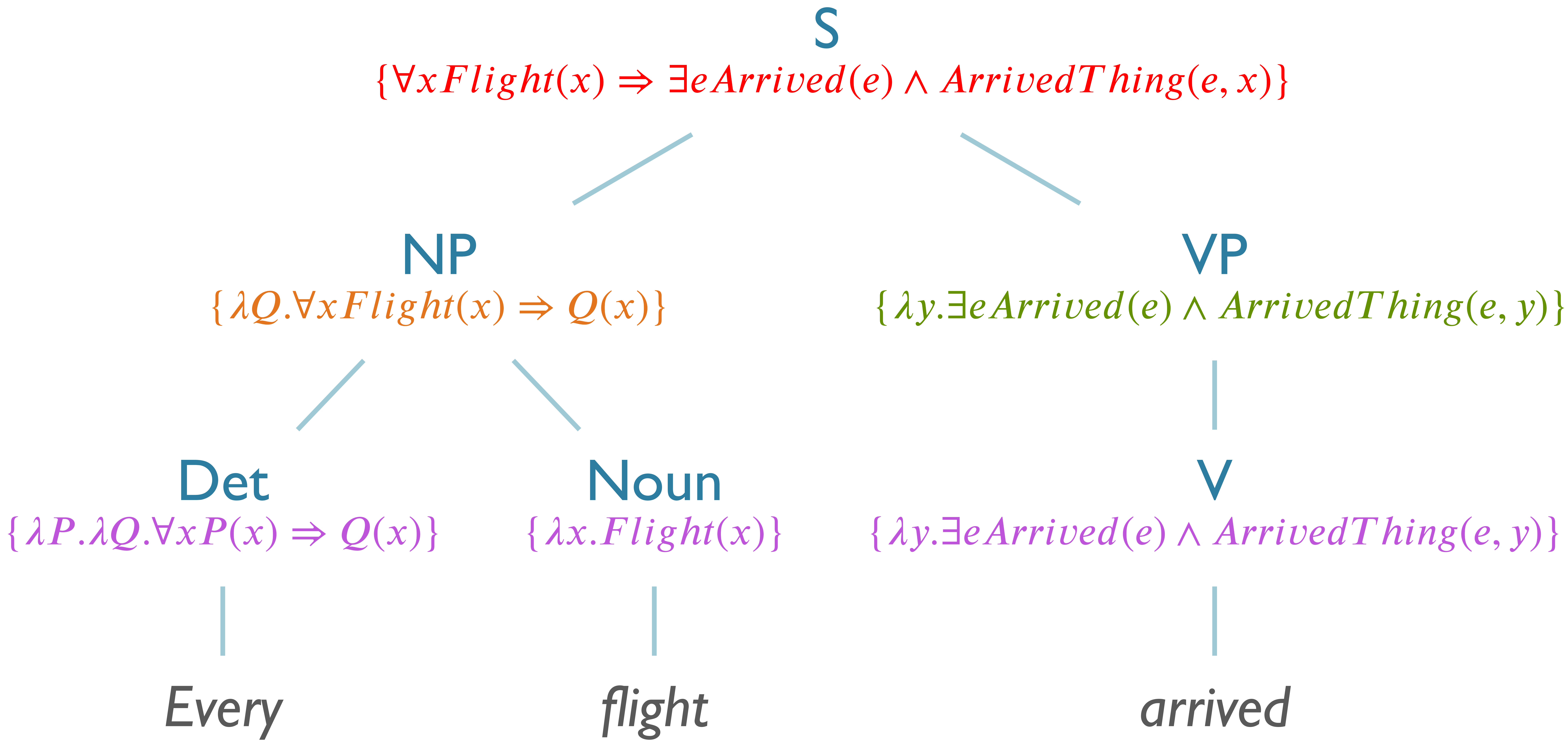

S
$\{NP.sem(VP.sem)\}$
NP
$\{\lambda Q.\forall x Flight(x) \Rightarrow Q(x)\}$
VP
$\{\lambda z.\exists e Arrived(e) \wedge ArrivedThing(e,z)\}$
Det
$\{\lambda P.\lambda Q.\forall x P(x) \Rightarrow Q(x)\}$
Noun
$\{\lambda y.Flight(y)\}$
V
$\{\lambda z.\exists e Arrived(e) \wedge ArrivedThing(e,z)\}$
Every
flight
arrived

$$\begin{tikzcd}[column sep=small]
 & \begin{array}{c}\text{S}\\ \{\forall x Flight(x) \Rightarrow \exists e Arrived(e) \wedge ArrivedThing(e,x)\}\\ \{NP.sem(VP.sem)\}\end{array} \arrow[dl, dash] \arrow[dr, dash] & \\
\begin{array}{c}\text{NP}\\ \{\lambda Q.\forall x Flight(x) \Rightarrow Q(x)\}\end{array} & & \begin{array}{c}\text{VP}\\ \{\lambda z.\exists e Arrived(e) \wedge ArrivedThing(e,z)\}\end{array}
\end{tikzcd}$$

$$\lambda Q.\forall x Flight(x) \Rightarrow Q(x)(\lambda z.\exists e Arrived(e) \wedge ArrivedThing(e, z))$$

$$\forall x Flight(x) \Rightarrow \lambda z.\exists e Arrived(e) \wedge ArrivedThing(e, z)(x)$$

$$\forall x Flight(x) \Rightarrow \exists e Arrived(e) \wedge ArrivedThing(e, x)$$


S
$\{\forall x Flight(x) \Rightarrow \exists e Arrived(e) \wedge ArrivedThing(e,x)\}$
NP
$\{\lambda Q.\forall x Flight(x) \Rightarrow Q(x)\}$
VP
$\{\lambda y.\exists e Arrived(e) \wedge ArrivedThing(e,y)\}$
Det
$\{\lambda P.\lambda Q.\forall x P(x) \Rightarrow Q(x)\}$
Noun
$\{\lambda x.Flight(x)\}$
V
$\{\lambda y.\exists e Arrived(e) \wedge ArrivedThing(e,y)\}$
Every
flight
arrived

More λ-Calculus

Target Representations

Common Nouns

- $\lambda x.Restaurant(x) \rightarrow$ '*restaurant*'
  - Somewhat similar to the NNP construction
    - λ *var.Predicate(var)*

- But common nouns represent ***relations***, rather than ***constants***
  - Meaning of the noun encoded in the predicate
  - Relate the concept of the noun to a particular instance of variable

Negation

- "No vegetarian restaurant serves meat."
  - $\neg(\exists x \; VegetarianRestaurant(x) \wedge Serves(x, Meat))$
- "All vegetarian restaurants do not serve meat."
  - $\forall x \; VegetarianRestaurant(x) \Rightarrow \neg Serves(x, Meat)$

- *These are semantically equivalent!*
  - ¬[IF P, THEN Q] ⇔ P AND NOT Q

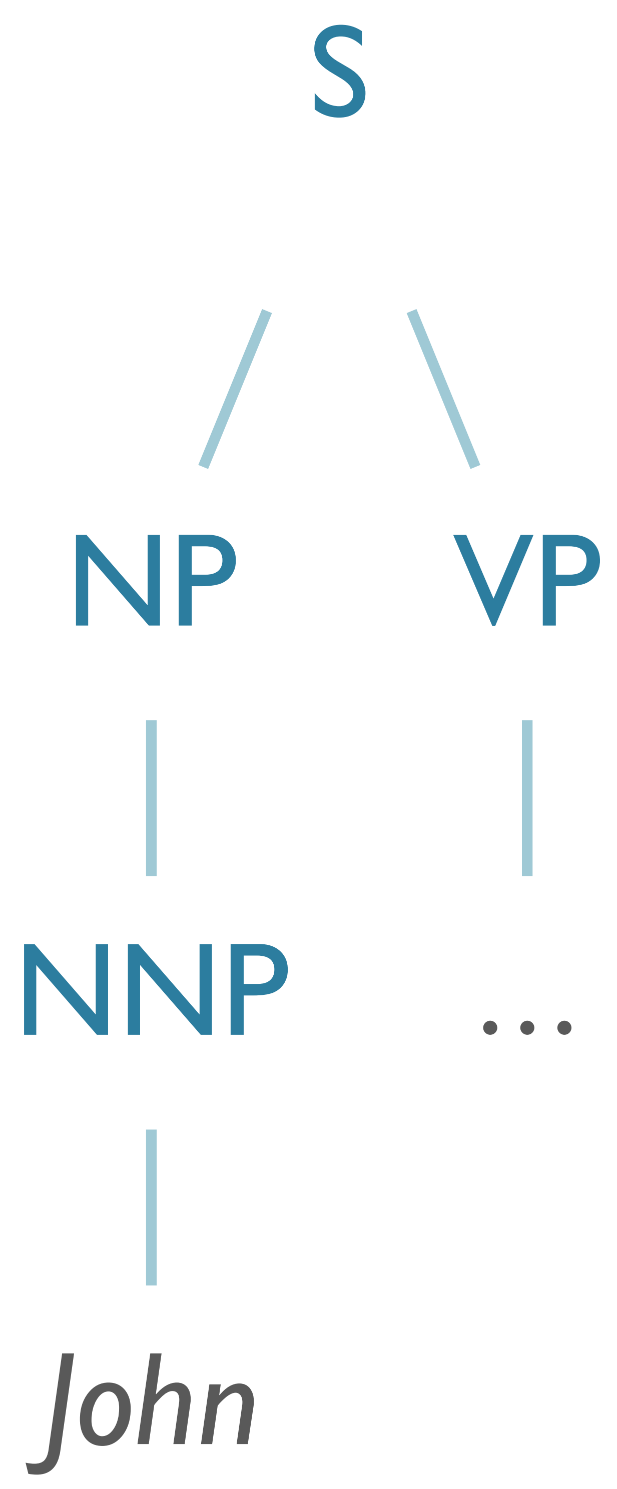
- For NLTK, use the hyphen/minus character: '–'

‘John booked a flight’

- Target representation:
    - $\exists x\ Flight(x) \wedge (\exists e Booked(\mathrm{e}) \wedge Booker(e,\ John) \wedge BookedThing(e,\ x))$

‘John booked a flight’

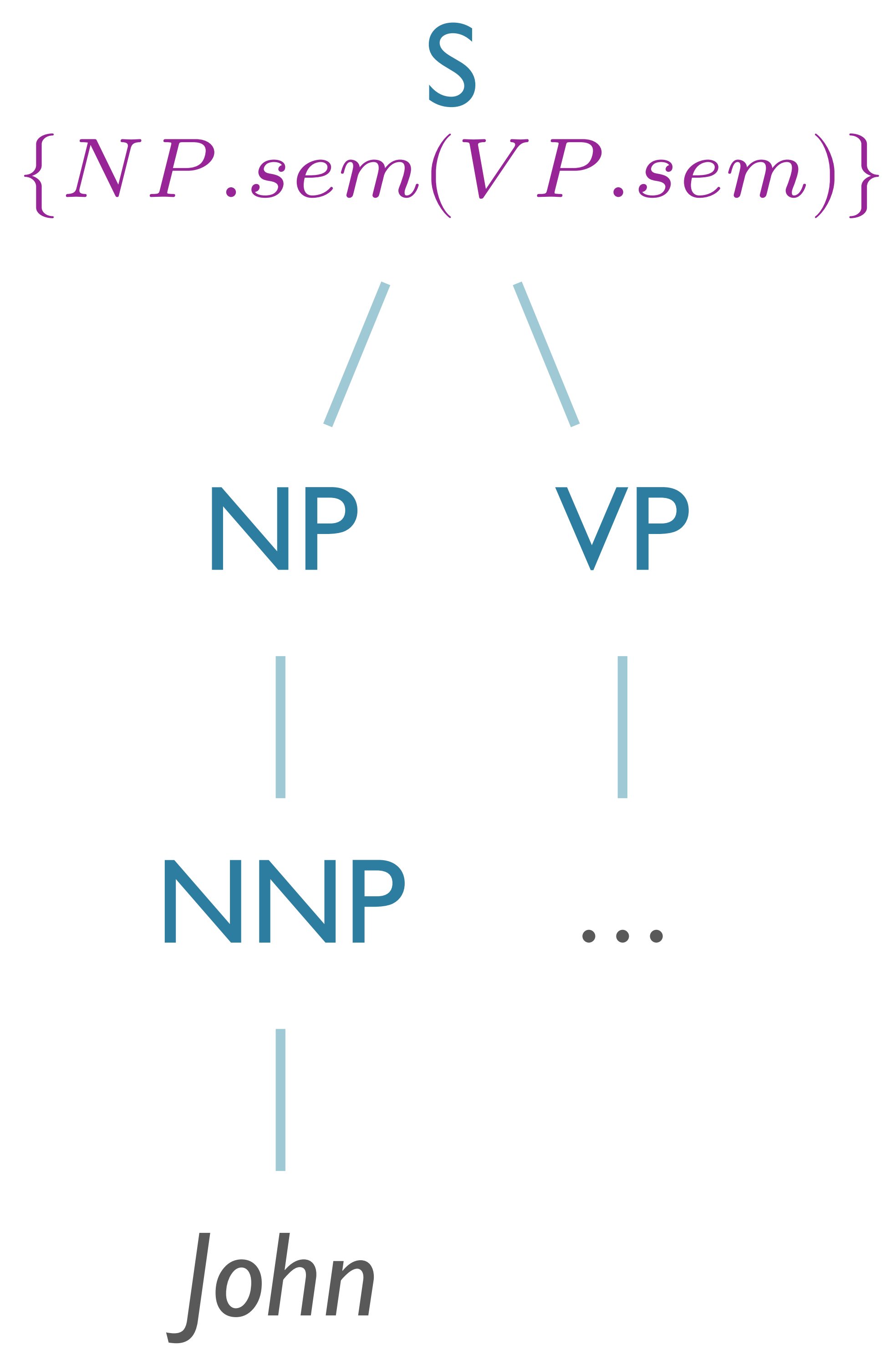
- $\exists x\ \mathit{Flight}(x) \land (\exists e \mathit{Booked}(\mathrm{e}) \land \mathit{Booker}(e,\ \mathit{John}) \land \mathit{BookedThing}(e,\ x))$



$S \rightarrow NP\ VP \qquad \{NP.sem(VP.sem)\}$

‘John booked a flight’

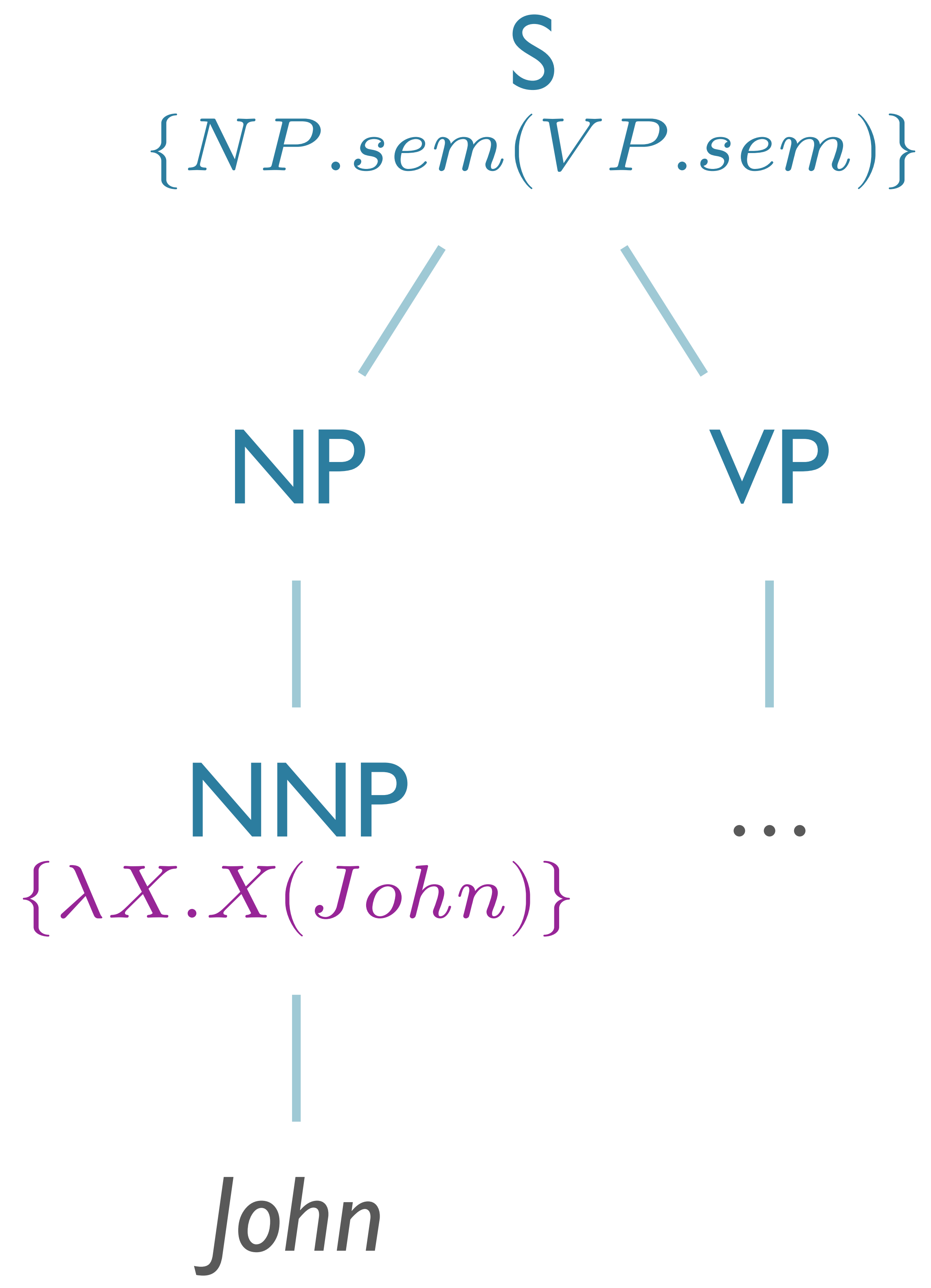
- $\exists x\ Flight(x) \wedge (\exists e Booked(\mathrm{e}) \wedge Booker(e,\ John) \wedge BookedThing(e,\ x))$



$S \rightarrow NP\ VP \qquad \{NP.sem(VP.sem)\}$

‘John booked a flight’

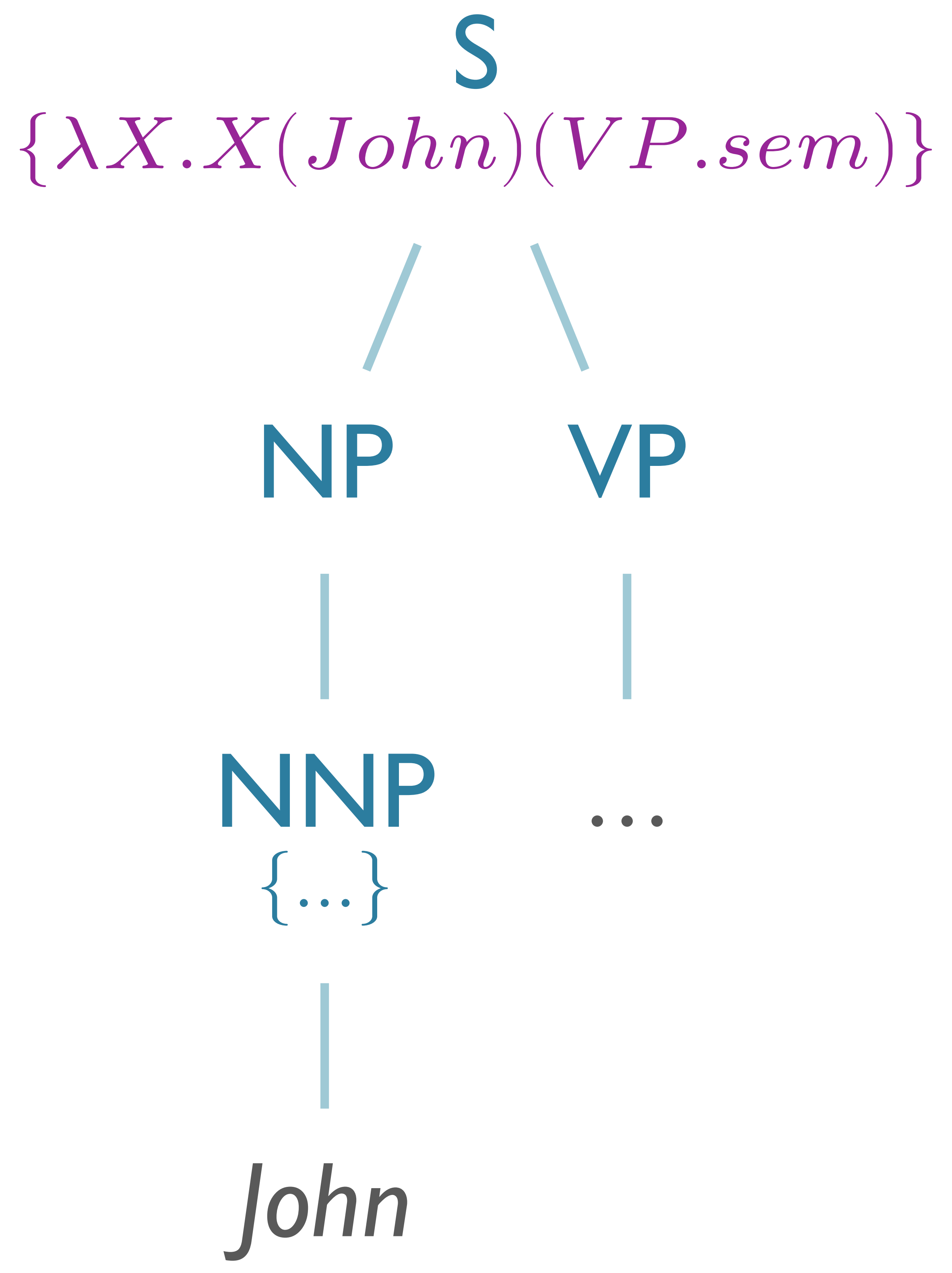
- $\exists x\ Flight(x) \wedge (\exists e Booked(\mathrm{e}) \wedge Booker(e,\ John) \wedge BookedThing(e,\ x))$



$S \rightarrow NP\ VP$	$\{NP.sem(VP.sem)\}$
$NNP \rightarrow$ ‘$John$’	$\{\lambda X.X(John)\}$
$NP \rightarrow NNP$	$\{NNP.sem\}$
$VP \rightarrow Verb\ NP$	$\{Verb.sem(NP.sem)\}$

‘John booked a flight’

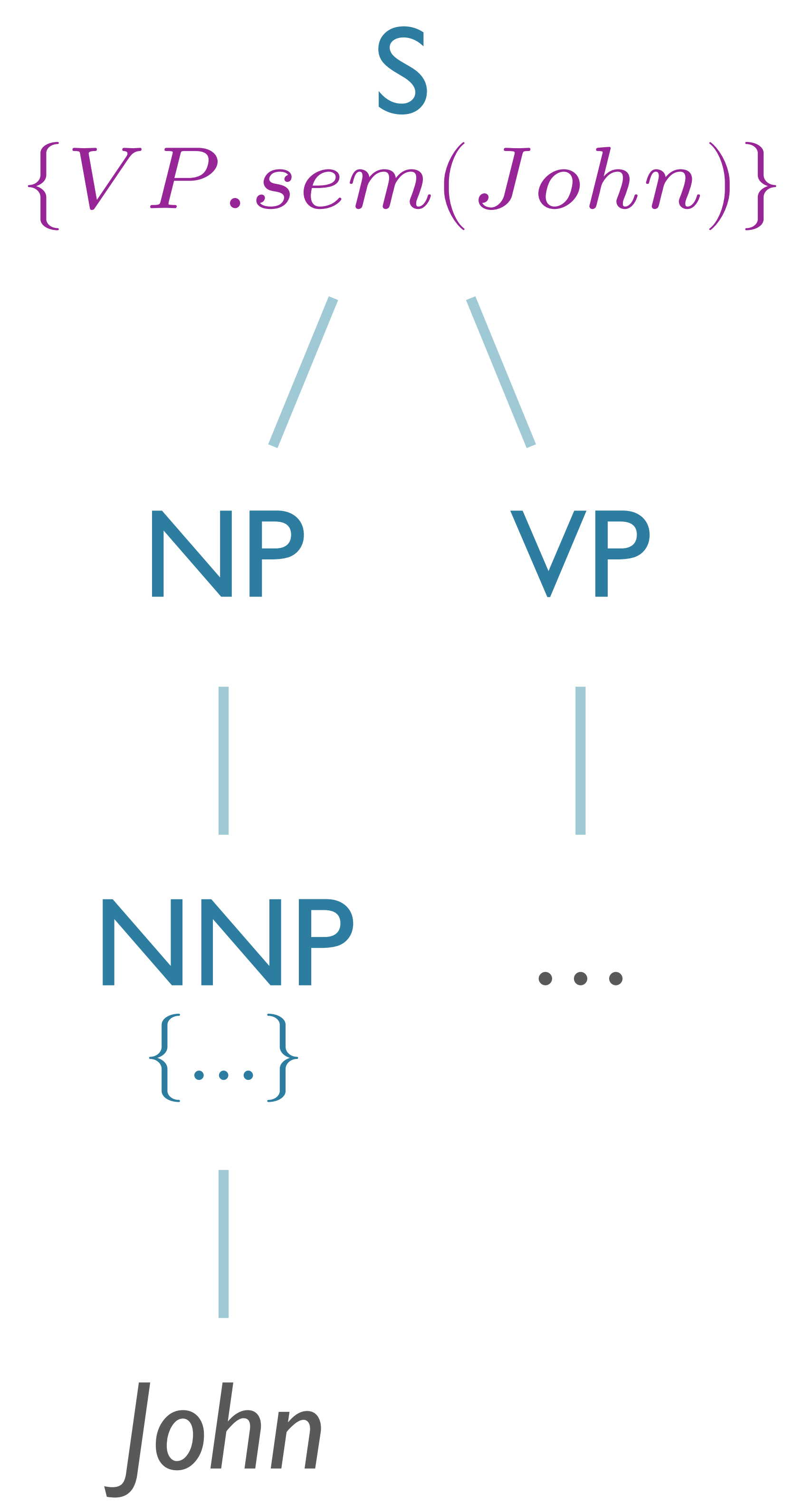
- $\exists x\ Flight(x) \wedge (\exists e Booked(\mathrm{e}) \wedge Booker(e, John) \wedge BookedThing(e, x))$



$S \rightarrow NP\ VP$	$\{NP.sem(VP.sem)\}$
$NNP \rightarrow$ ‘*John*’	$\{\lambda X.X(John)\}$
$NP \rightarrow NNP$	$\{NNP.sem\}$
$VP \rightarrow Verb\ NP$	$\{Verb.sem(NP.sem)\}$

‘John booked a flight’

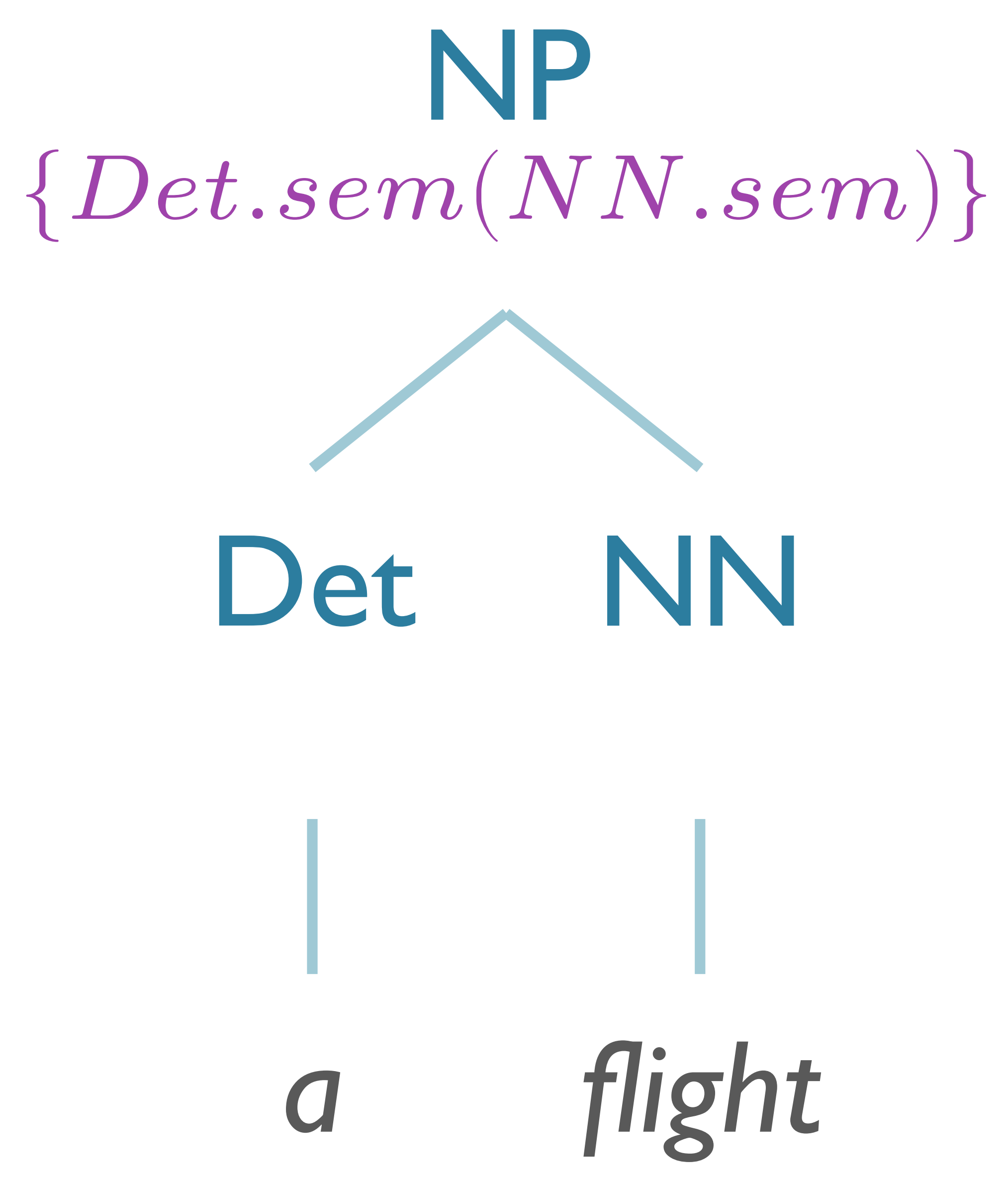
- $\exists x\ Flight(x) \wedge (\exists e Booked(\mathrm{e}) \wedge Booker(e, John) \wedge BookedThing(e, x))$



$S \rightarrow NP\ VP$	$\{NP.sem(VP.sem)\}$
$NNP \rightarrow$ ‘John’	$\{\lambda X.X(John)\}$
$NP \rightarrow NNP$	$\{NNP.sem\}$
$VP \rightarrow Verb\ NP$	$\{Verb.sem(NP.sem)\}$

‘John booked a flight’

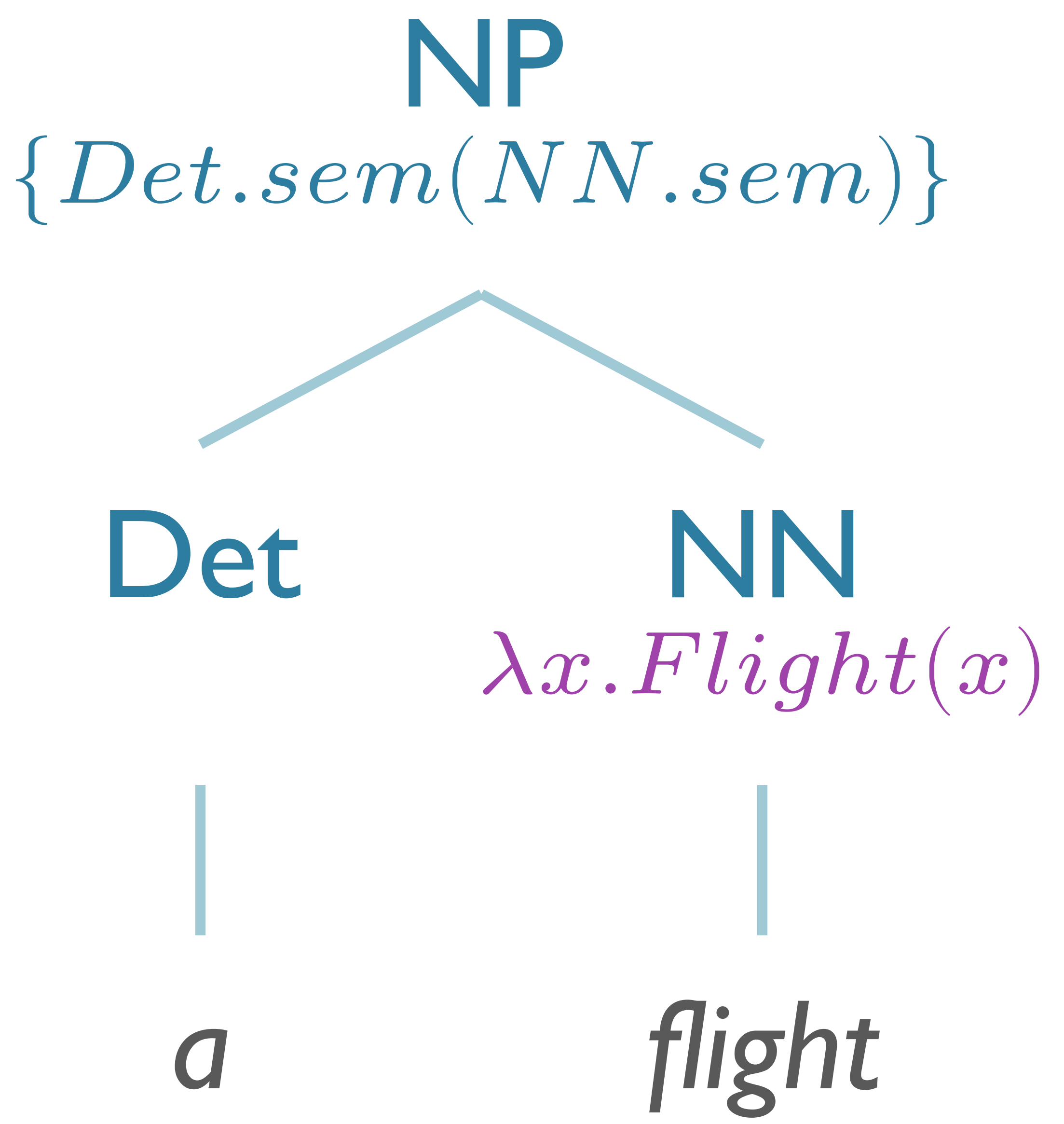
- $\exists x\ Flight(x) \wedge (\exists e Booked(\mathrm{e}) \wedge Booker(e, John) \wedge BookedThing(e, x))$



$NP \rightarrow Det\ NN \qquad \{Det.sem(NN.sem)\}$

‘John booked a flight’

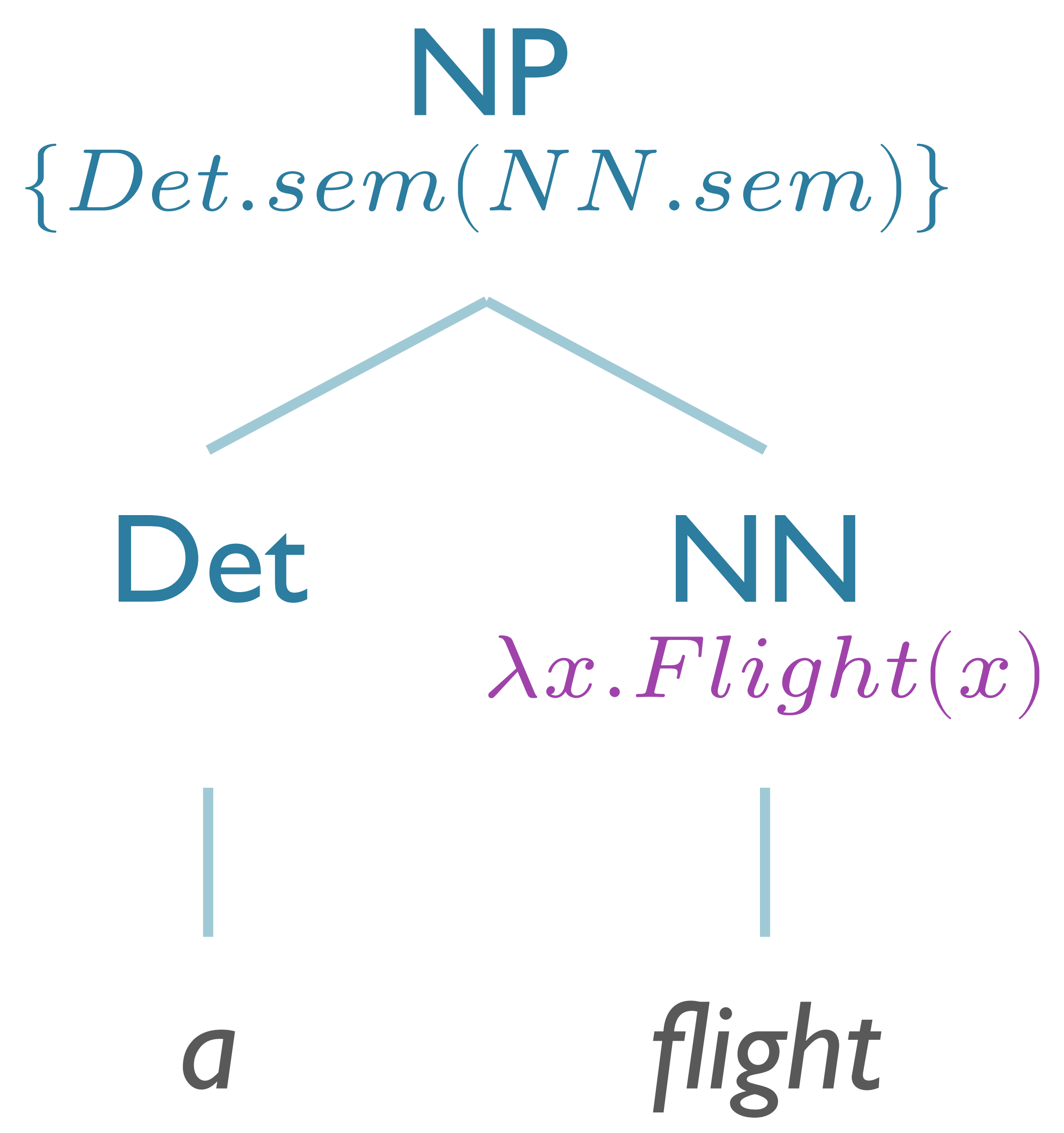
- $\exists x\ Flight(x) \wedge (\exists e Booked(\mathrm{e}) \wedge Booker(e, John) \wedge BookedThing(e, x))$



$NP \rightarrow Det\ NN$	$\{Det.sem(NN.sem)\}$
$NN \rightarrow$ ‘*flight*’	$\{\lambda x.Flight(x)\}$

‘John booked a flight’

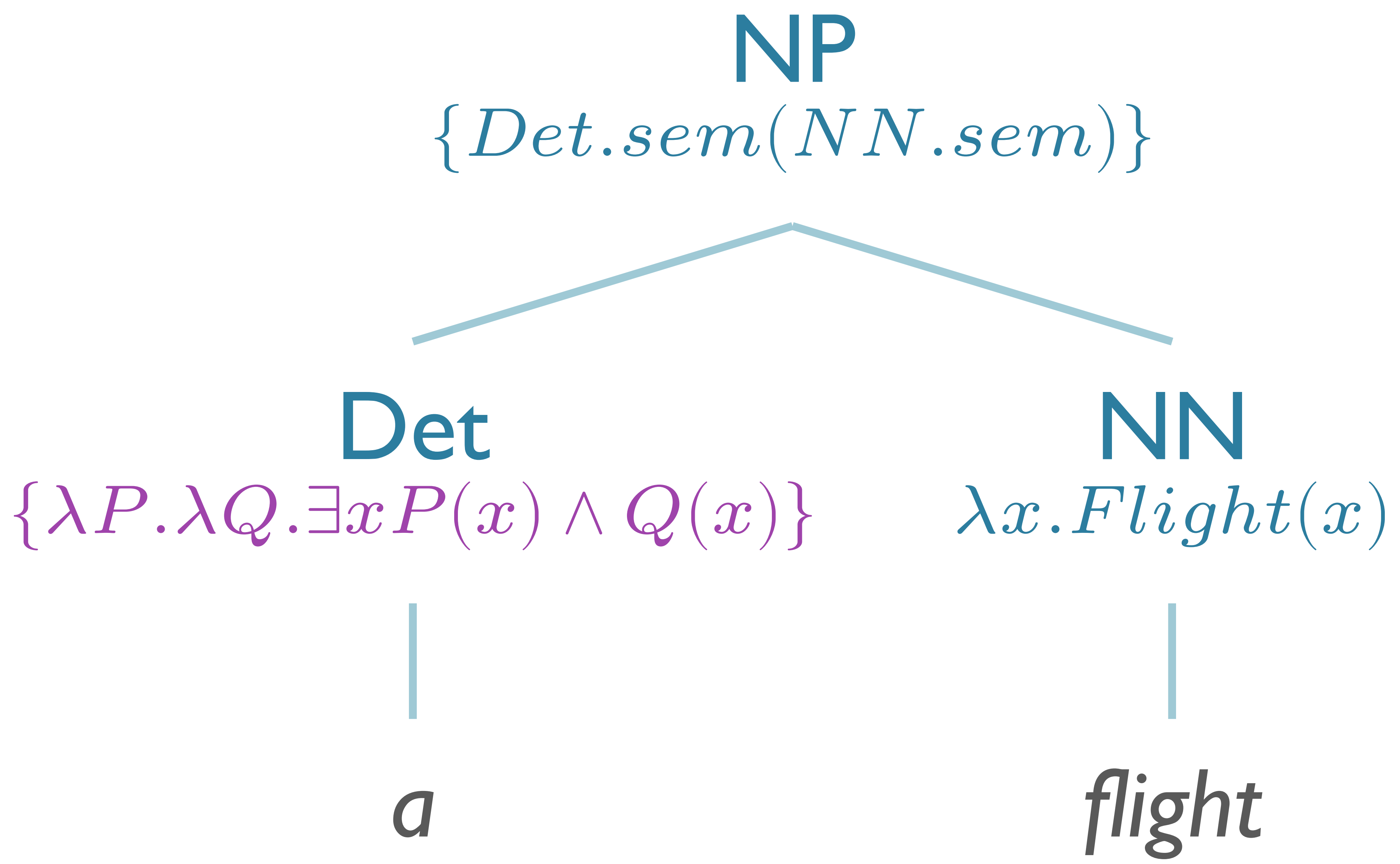
- $\exists x\ Flight(x) \wedge (\exists e Booked(\mathrm{e}) \wedge Booker(e, John) \wedge BookedThing(e, x))$



$NP \rightarrow Det\ NN$	$\{Det.sem(NN.sem)\}$
$NN \rightarrow$ ‘flight’	$\{\lambda x.Flight(x)\}$
$Det \rightarrow$ ‘a’	$\{\ \lambda P.\lambda Q.\exists x\ P(x) \wedge Q(x)\ \}$

‘John booked a flight’

- $\exists x\ Flight(x) \wedge (\exists e Booked(\mathrm{e}) \wedge Booker(e,\ John) \wedge BookedThing(e,\ x))$



$NP \rightarrow Det\ NN$	$\{Det.sem(NN.sem)\}$
$NN \rightarrow$ ‘*flight*’	$\{\lambda x.Flight(x)\}$
$Det \rightarrow$ ‘*a*’	$\{\ \lambda P.\lambda Q.\exists x\ P(x) \wedge Q(x)\ \}$

‘John booked a flight’

- $\exists x\ Flight(x) \wedge (\exists e Booked(\mathrm{e}) \wedge Booker(e, John) \wedge BookedThing(e, x))$

NP
$\{Det.sem(NN.sem)\}$

$NP \rightarrow Det\ NN$	$\{Det.sem(NN.sem)\}$
$NN \rightarrow$ ‘flight’	$\{\lambda x.Flight(x)\}$
$Det \rightarrow$ ‘a’	$\{\ \lambda P.\lambda Q.\exists x\ P(x) \wedge Q(x)\ \}$

‘John booked a flight’

- $\exists x\ Flight(x) \wedge (\exists e Booked(\mathrm{e}) \wedge Booker(e, John) \wedge BookedThing(e, x))$

NP

$\{Det.sem(NN.sem)\}$

$\{\lambda P.\lambda Q.\exists x P(x) \wedge Q(x)(\lambda x.Flight(x))\}$

$NP \rightarrow Det\ NN$		$\{Det.sem(NN.sem)\}$
$NN \rightarrow$ ‘*flight*’		$\{\lambda x.Flight(x)\}$
$Det \rightarrow$ ‘*a*’		$\{\ \lambda P.\lambda Q.\exists x\ P(x) \wedge Q(x)\ \}$

‘John booked a flight’

- $\exists x\ Flight(x) \wedge (\exists e Booked(\mathrm{e}) \wedge Booker(e, John) \wedge BookedThing(e, x))$

NP

$\{Det.sem(NN.sem)\}$

$\{\lambda P.\lambda Q.\exists x P(x) \wedge Q(x)(\lambda x.Flight(x))\}$

$\{\lambda Q.\exists x(\lambda x.Flight(x))(x) \wedge Q(x)\}$

$NP \rightarrow Det\ NN$	$\{Det.sem(NN.sem)\}$
$NN \rightarrow$ ‘*flight*’	$\{\lambda x.Flight(x)\}$
$Det \rightarrow$ ‘*a*’	$\{\ \lambda P.\lambda Q.\exists x\ P(x) \wedge Q(x)\ \}$

'John booked a flight'

- $\exists x\ Flight(x) \wedge (\exists e Booked(e) \wedge Booker(e, John) \wedge BookedThing(e, x))$

NP

$\{Det.sem(NN.sem)\}$

$\{\lambda P.\lambda Q.\exists x P(x) \wedge Q(x)(\lambda x.Flight(x))\}$

$\{\lambda Q.\exists x(\lambda x.Flight(x))(x) \wedge Q(x)\}$

$\{\lambda Q.\exists x Flight(x) \wedge Q(x)\}$

$NP \rightarrow Det\ NN$	$\{Det.sem(NN.sem)\}$
$NN \rightarrow$ 'flight'	$\{\lambda x.Flight(x)\}$
$Det \rightarrow$ 'a'	$\{\ \lambda P.\lambda Q.\exists x\ P(x) \wedge Q(x)\ \}$

‘John booked a flight’

- $\exists x\ Flight(x) \wedge (\exists e Booked(\mathrm{e}) \wedge Booker(e, John) \wedge BookedThing(e, x))$

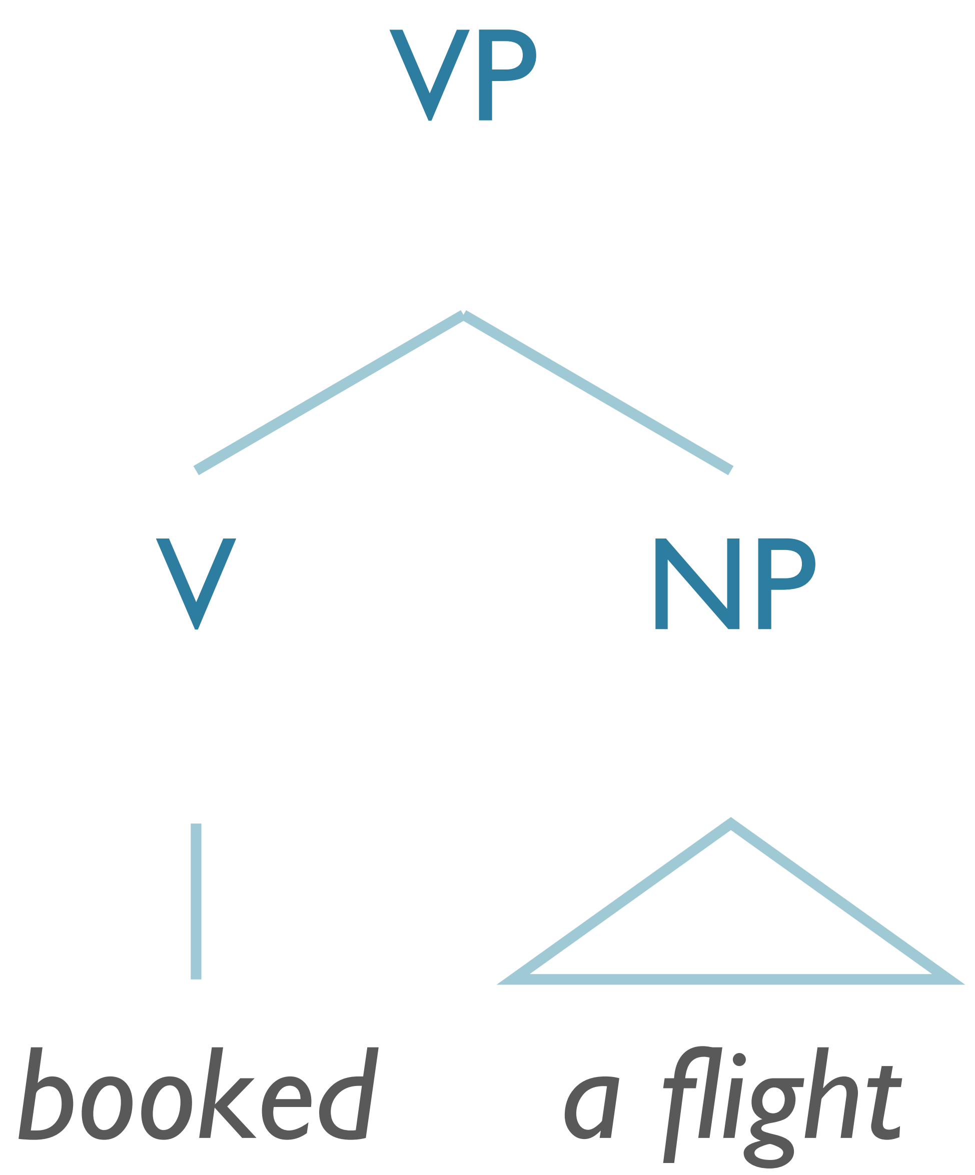
NP

$$\{Det.sem(NN.sem)\}$$
$$\{\lambda P.\lambda Q.\exists x P(x) \wedge Q(x)(\lambda x.Flight(x))\}$$
$$\{\lambda Q.\exists x(\lambda x.Flight(x))(x) \wedge Q(x)\}$$
$$\{\lambda Q.\exists x Flight(x) \wedge Q(x)\}$$

$NP \rightarrow Det\ NN$	$\{Det.sem(NN.sem)\}$
$NN \rightarrow$ *‘flight’*	$\{\lambda x.Flight(x)\}$
$Det \rightarrow$ *‘a’*	$\{\ \lambda P.\lambda Q.\exists x\ P(x) \wedge Q(x)\ \}$
‘a flight’	$\{\ \lambda Q.\exists x\ Flight(x) \wedge Q(x)\ \}$

‘John booked a flight’

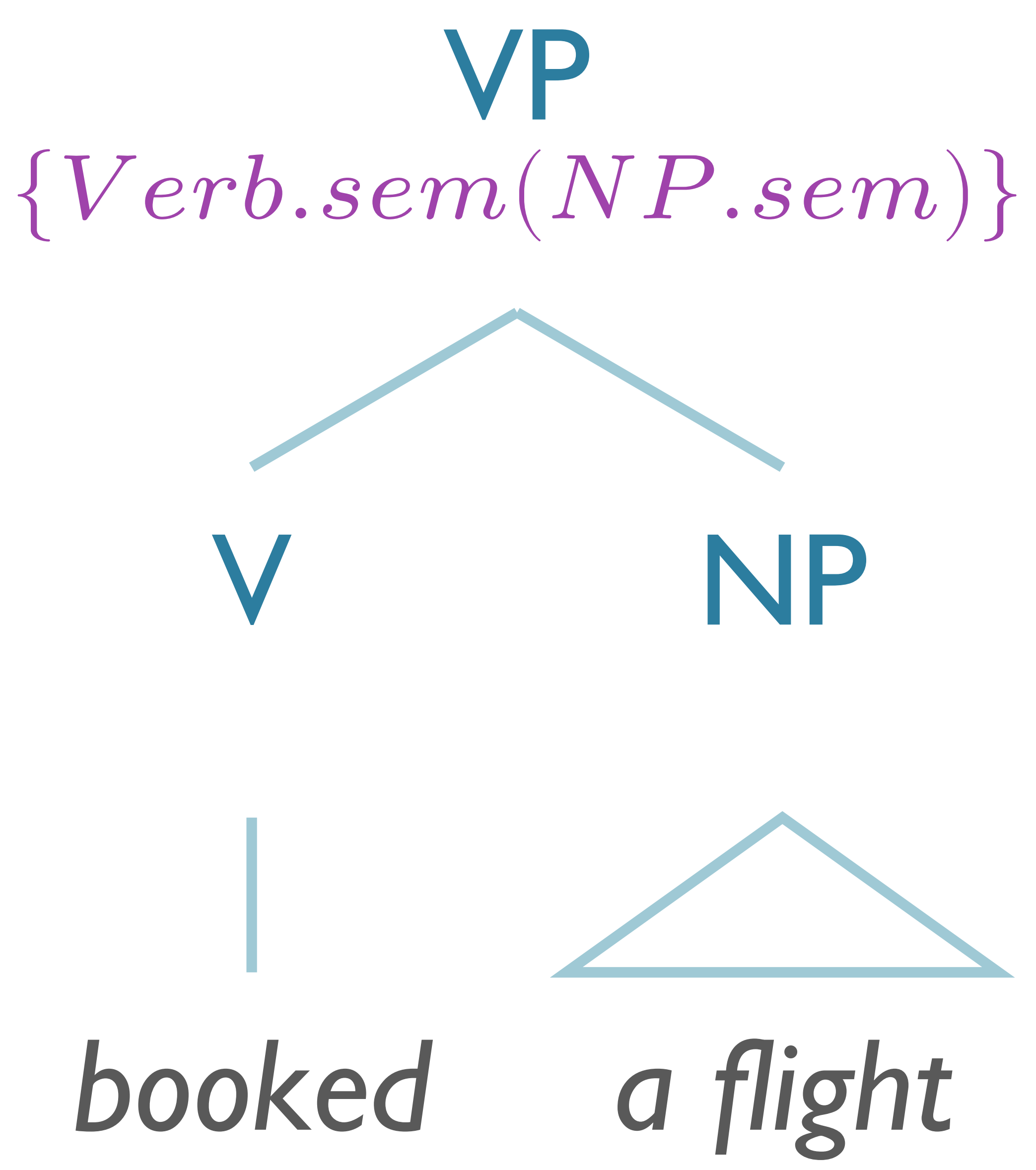
- $\exists x\ Flight(x) \wedge (\exists e Booked(\text{e}) \wedge Booker(e,\ John) \wedge BookedThing(e,\ x))$



$VP \rightarrow Verb\ NP$	$\{Verb.sem(NP.sem)\}$
‘a flight’	$\{\ \lambda Q.\exists x\ Flight(x) \wedge Q(x)\ \}$

‘John booked a flight’

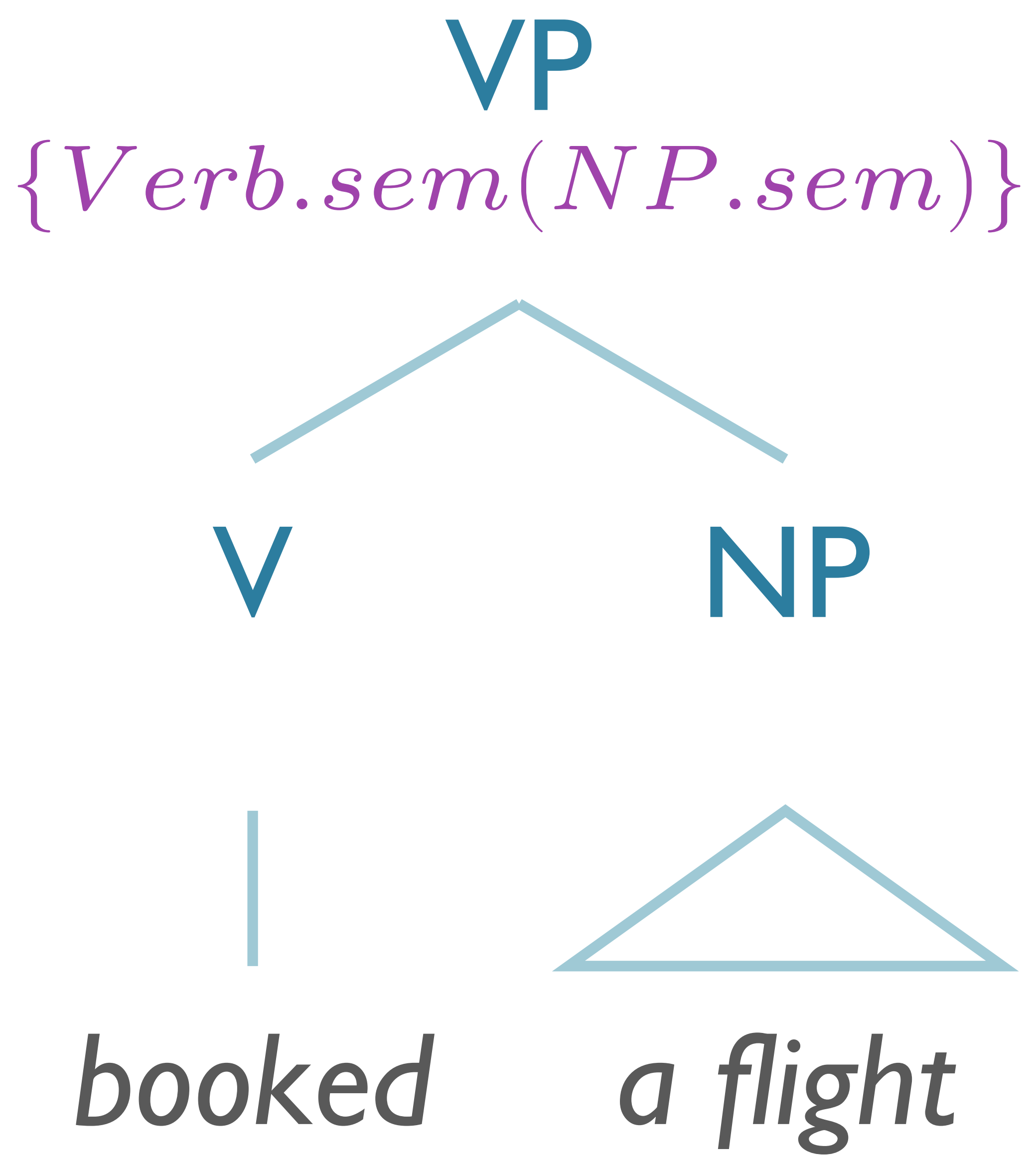
- $\exists x\ Flight(x) \wedge (\exists e Booked(e) \wedge Booker(e, John) \wedge BookedThing(e, x))$



$VP \rightarrow Verb\ NP$	$\{Verb.sem(NP.sem)\}$
‘a flight’	$\{\ \lambda Q.\exists x\ Flight(x) \wedge Q(x)\ \}$

‘John booked a flight’

- $\exists x\ Flight(x) \wedge (\exists e Booked(\mathrm{e}) \wedge Booker(e,\ John) \wedge BookedThing(e,\ x))$



$Verb \rightarrow$ ‘*booked*’

$\{\lambda W.\lambda z.W(\lambda y.\exists e Booked(\mathrm{e}) \wedge Booker(e,z) \wedge BookedThing(e,y))\}$

$VP \rightarrow Verb\ NP \qquad \{Verb.sem(NP.sem)\}$

‘*a flight*’ $\qquad \{\ \lambda Q.\exists x\ Flight(x) \wedge Q(x)\ \}$

‘John booked a flight’

- $\exists x\ Flight(x) \wedge (\exists e Booked(\mathrm{e}) \wedge Booker(e,\ John) \wedge BookedThing(e,\ x))$

$Verb.sem(NP.sem)$

$\lambda W.\lambda z.\ W(\lambda y.\exists e Booked(\mathrm{e}) \wedge Booker(e,z) \wedge BookedThing(e,y))(\lambda Q.\exists x\ Flight(x) \wedge Q(x))$

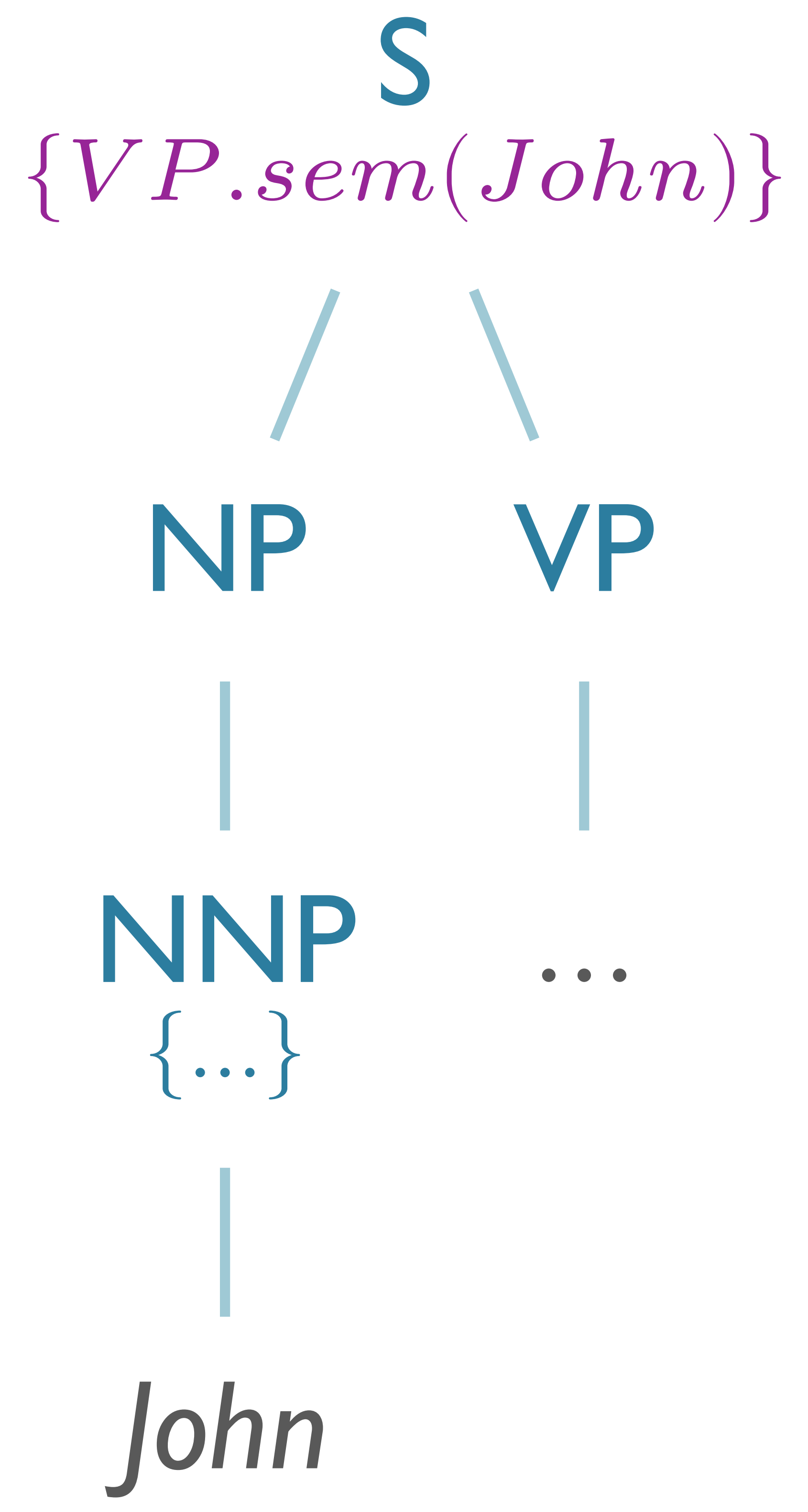
$\lambda z.(\lambda Q.\exists x\ Flight(x) \wedge Q(x))(\lambda y.\exists e Booked(\mathrm{e}) \wedge Booker(e,z) \wedge BookedThing(e,y))$

$\lambda z.\exists x\ Flight(x) \wedge (\lambda y.\exists e Booked(\mathrm{e}) \wedge Booker(e,z) \wedge BookedThing(e,y))(x)$

$\lambda z.\exists x\ Flight(x) \wedge (\exists e Booked(\mathrm{e}) \wedge Booker(e,z) \wedge BookedThing(e,\boldsymbol{x})$

‘John booked a flight’

- $\exists x\ Flight(x) \wedge (\exists e Booked(\mathrm{e}) \wedge Booker(e,\ John) \wedge BookedThing(e,\ x))$



S

$‘booked\ a\ flight'$

$VP.sem(John)$

$\lambda z.\exists x\ Flight(x) \wedge (\exists e Booked(\mathrm{e}) \wedge Booker(e,z)$
$\wedge\ BookedThing(e,\boldsymbol{x})$

‘John booked a flight’

- $\exists x\ Flight(x) \wedge (\exists e Booked(\mathrm{e}) \wedge Booker(e,\ John) \wedge BookedThing(e,\ x))$

S $\qquad\qquad\qquad\qquad\qquad\qquad$ $VP.sem(John)$

$‘booked\ a\ flight'$ $\qquad$ $\lambda z.\exists x\ Flight(x) \wedge (\exists e Booked(\mathrm{e}) \wedge Booker(e,z) \wedge BookedThing(e,\boldsymbol{x})$

$\boldsymbol{\lambda z}.\exists x\ Flight(x) \wedge (\exists e Booked(\mathrm{e}) \wedge Booker(e,z) \wedge BookedThing(e,x)(\boldsymbol{John})$

$\exists x\ Flight(x) \wedge (\exists e Booked(\mathrm{e}) \wedge Booker(e,\boldsymbol{John}) \wedge BookedThing(e,x)$

‘John booked a flight’

$Det \rightarrow$ ‘a’	$\{\ \lambda P.\lambda Q.\exists x\ P(x) \wedge Q(x)\ \}$
$Det \rightarrow$ ‘$every$’	$\{\ \lambda P.\lambda Q.\forall x\ P(x) \Rightarrow Q(x)\ \}$
$NN \rightarrow$ ‘$flight$’	$\{\lambda x.Flight(x)\}$
$Verb \rightarrow$ ‘$booked$’	$\{\lambda W.\lambda z.W(\lambda y.\exists e Booked(\mathrm{e}) \wedge Booker(e,z) \wedge BookedThing(e,y))\}$
$NNP \rightarrow$ ‘$John$’	$\{\lambda X.X(John)\}$
$NP \rightarrow NNP$	$\{NNP.sem\}$
$NP \rightarrow Det\ NN$	$\{Det.sem(NN.sem)\}$
$S \rightarrow NP\ VP$	$\{NP.sem(VP.sem)\}$
$VP \rightarrow Verb\ NP$	$\{Verb.sem(NP.sem)\}$

‘John booked no flight’

- $\neg(\exists x\ Flight(x) \land (\exists e Booked(\mathrm{e}) \land Booker(e,\ John) \land BookedThing(e,\ x)))$
- $\forall x Flight(x) \Rightarrow \neg(\exists e Booked(\mathrm{e}) \land Booker(e,\ John) \land BookedThing(e,\ x)$

‘John booked no flight’

$Det \rightarrow$ ‘no’	$\{ \lambda P.\lambda Q.\neg\exists x\ P(x) \wedge Q(x) \mid \lambda P.\lambda Q.\forall x\ P(x) \Rightarrow \neg Q(x)\}$
$Det \rightarrow$ ‘a’	$\{ \lambda P.\lambda Q.\exists x\ P(x) \wedge Q(x) \}$
$Det \rightarrow$ ‘every’	$\{ \lambda P.\lambda Q.\forall x\ P(x) \Rightarrow Q(x) \}$
$NN \rightarrow$ ‘flight’	$\{\lambda x.Flight(x)\}$
$Verb \rightarrow$ ‘booked’	$\{\lambda W.\lambda z.W(\lambda y.\exists e Booked(\mathrm{e}) \wedge Booker(e,z) \wedge BookedThing(e,y))\}$
$NNP \rightarrow$ ‘John’	$\{\lambda X.X(John)\}$
$NP \rightarrow NNP$	$\{NNP.sem\}$
$NP \rightarrow Det\ NN$	$\{Det.sem(NN.sem)\}$
$S \rightarrow NP\ VP$	$\{NP.sem(VP.sem)\}$
$VP \rightarrow Verb\ NP$	$\{Verb.sem(NP.sem)\}$

Other Lambda Calculus

Adjectives

- Similar to nouns, but with an extra conjunction and dummy predicate:
  - "red" $= \lambda P\ \lambda x(red(x)\ \wedge\ P(x))$
- Any issues?
  - Non-intersective adjectives (e.g. 'skillful', 'alleged', 'fake')

Definite Article

- $a = \lambda P.\lambda Q.\exists x(P(x) \wedge Q(x))$
- $the = \lambda P.\lambda Q.\exists x(P(x) \wedge \forall y(\mathrm{P}(y) \Leftrightarrow x{=}y) \wedge \mathrm{Q}(x)))$
- Roughly: “The P Q”: there is a *unique* P, which is also Q
  - Unique: *x* is P, and anything else that is also P is equal to *x*

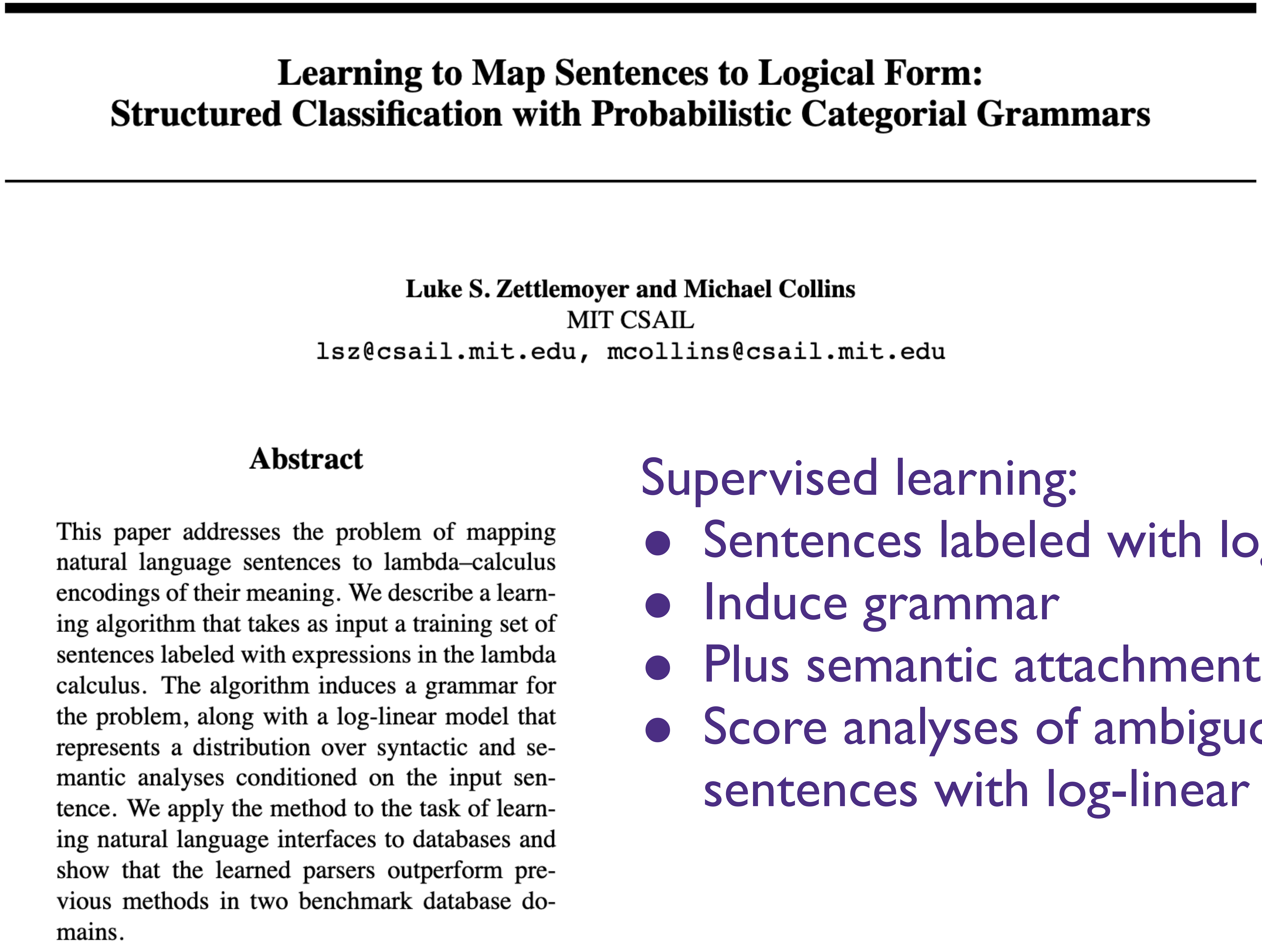
Definite Article

- $the = \lambda P.\lambda Q.\exists x(P(x) \wedge \forall y(\mathrm{P}(y) \Leftrightarrow x{=}y) \wedge \mathrm{Q}(x)))$
  - Bertrand Russel, "*On Denoting*" (1905).
  - The definite article isn't exactly the same as a constant (like "*John*")
  - Rather, it picks out a set of items from a set (the generic NN), and makes a strong assertion:
    A) The book arrived.
    B) A book arrived.
    - A $\vDash$ B, but B $\nvDash$ A

Definite Article + Presupposition

- “The slides for Monday are amazing.”
  - ~> there are slides for Monday.
- “The slides for Monday are not amazing.”
  - ~> there are slides for Monday.
- The P Q: *presupposes* that there is a unique P, does not *assert* it [Strawson 1950, …]
  - If there is no P, “The P Q” is *neither true nor false*

Learning Semantic Parsers



Learning to Map Sentences to Logical Form: Structured Classification with Probabilistic Categorial Grammars

Luke S. Zettlemoyer and Michael Collins
MIT CSAIL
lsz@csail.mit.edu, mcollins@csail.mit.edu

Abstract

This paper addresses the problem of mapping natural language sentences to lambda–calculus encodings of their meaning. We describe a learning algorithm that takes as input a training set of sentences labeled with expressions in the lambda calculus. The algorithm induces a grammar for the problem, along with a log-linear model that represents a distribution over syntactic and semantic analyses conditioned on the input sentence. We apply the method to the task of learning natural language interfaces to databases and show that the learned parsers outperform previous methods in two benchmark database domains.

Supervised learning:

- Sentences labeled with logical forms
- Induce grammar
- Plus semantic attachments
- Score analyses of ambiguous sentences with log-linear model

Zettlemoyer and Collins 2005

Learning from Denotations

Liang, Jordan, and Klein Learning Dependency-Based Compositional Semantics

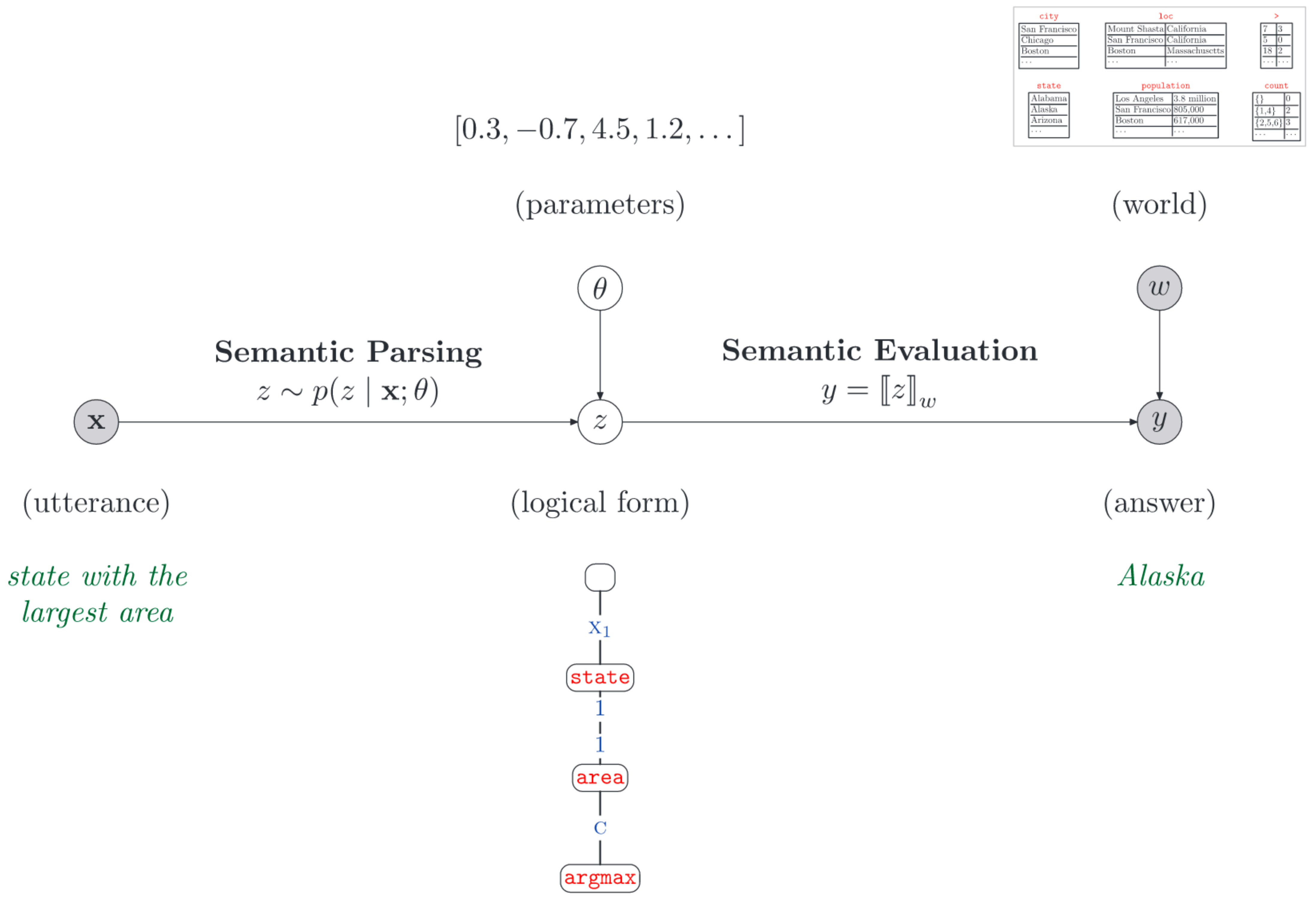


Figure 2
Our statistical methodology consists of two steps: (i) semantic parsing ($p(z \mid \mathbf{x}; \theta)$): an utterance $\mathbf{x}$ is mapped to a logical form z by drawing from a log-linear distribution parametrized by a vector θ; and (ii) evaluation ($[[z]]_w$): the logical form z is evaluated with respect to the world w (database of facts) to deterministically produce an answer y. The figure also shows an example configuration of the variables around the graphical model. Logical forms z are represented as labeled trees. During learning, we are given w and (x, y) pairs (shaded nodes) and try to infer the latent logical forms z and parameters θ.

Learn semantic representations as *latent variables* for downstream task (QA, conversation, …)

Liang et al 2011

Resources

- Datasets
  - General:
    - Abstract Meaning Representations: LDC2017T10
    - Minimal Recursion Semantics: DeepBank
  - SQL:
    - Spider: https://yale-lily.github.io/spider
    - SParC: https://yale-lily.github.io/sparc

Resources: Knowledge Graphs

- R.I.P. Freebase
  - Used by Google Knowledge Graph, then bought and killed
  - [they have an API with 100,000 queries/day for free]
- BUT: data moved to Wikidata

Lexical Semantics

Lexical Semantics

- Thus far: $\mathrm{POS} \rightarrow \mathit{Word}\ \{\mathrm{sem}\}$
  - Can compose larger semantic formulae bottom-up this way
  - …but we haven't really discussed what a "word" is, semantically.
- Lexical semantics:
  - How do we formally discuss what a "word" is?
  - How do we relate words to one another?
  - How do we differentiate/relate linked senses?

What is a Plant?

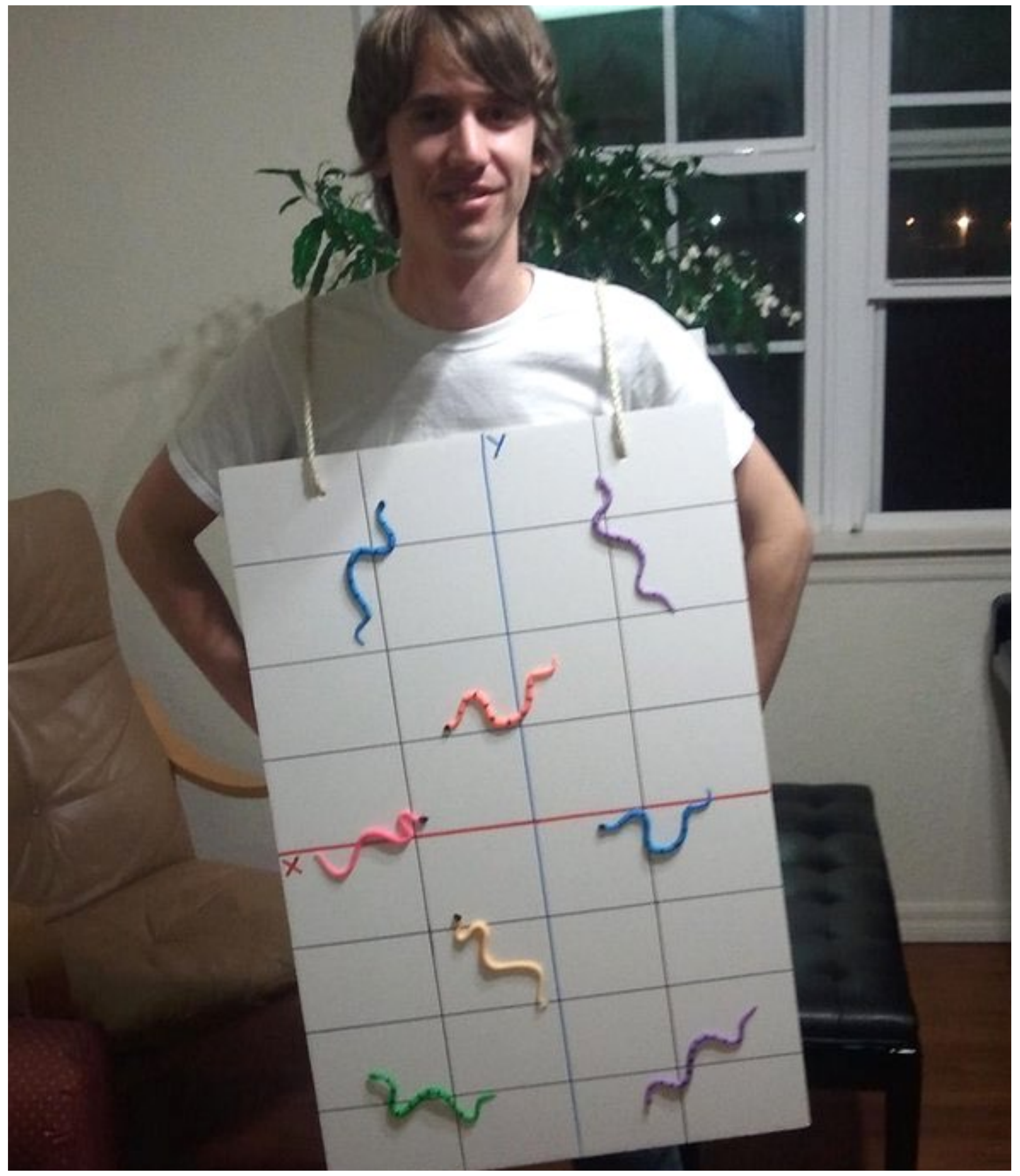
- There are more kinds of **plants** and animals in the rainforests than anywhere else on Earth. Over half of the millions of known species of **plants** and animals live in the rainforest. Many are found nowhere else. There are even **plants** and animals in the rainforest that we have not yet discovered.

- The Paulus company was founded in 1938. Since those days the product range has been the subject of constant expansions and is brought up continuously to correspond with the state of the art. We're engineering, manufacturing, and commissioning world-wide ready-to-run **plants** packed with our comprehensive know-how.

Lexical Semantics

…by way of dad-joke Halloween costumes. 🎃

A Ceiling Fan



Snakes on a Plane

Sources of Confusion

Homonymy

Polysemy

Synonymy

Antonymy

[Hypo/Hyper]-nymy

Sources of Confusion: Homonymy

- Words have same form but different meanings
  - Generally same POS, but unrelated meaning
    - bank_1 (side of river)
    - bank_2 (financial institution)

Sources of Confusion: Homonymy

- Different types of Homonymy:
  - ***Homophones***: same phonology, different orthographic form
    - two
    - to
    - too
  - ***Homographs***: Same orthography, different phonology:
    - "*lead*" (metal)
    - "*lead*" (take somewhere)
- Why do we care?
  - Problem for applications: TTS, ASR transcription, IR

Sources of Confusion: Polysemy

- Multiple RELATED senses
  - e.g. bank: money, organ, blood
- Big issue in lexicography
  - Number of senses
  - Relations between senses
  - Differentiation

Sources of Confusion: Polysemy

- Example: [[*serve*]]
  - *serve breakfast*
  - *serve Philadelphia*
  - *serve time*

Sources of Confusion: Synonymy

- (near) identical meaning
- Substitutability
  - Maintains propositional meaning

Sources of Confusion: Synonymy

- Issues:
  - Also has polysemy!
    - Shades of meaning - other associations
      - *price* vs. *fare*
      - *big* vs. *large*
      - *water* vs. *H_20*
- Collocational constraints
  - e.g. *babbling brook* vs. **babbling river*
- Register:
  - social factors: e.g. politeness, formality

Sources of Confusion: Antonymy

- Opposition
- Typically ends of a scale
  - *fast* vs. *slow*
  - *big* vs. *little*
- Can be hard to distinguish automatically from synonyms

Sources of Confusion: Hyponomy

- `instanceOf(x, y)` relations:
- More General (***hypernym***) vs. more specific (***hyponym***)
  - *dog* vs. *golden retriever*
  - *fruit* vs. *mango*
- Organize as ontology/taxonomy

Word Sense Disambiguation

- Application of lexical semantics
- Goal: given a word ***in context***, identify the appropriate sense
  - e.g. plants and animals in the rainforest
- Crucial for real syntactic & semantic analysis
  - Correct sense can determine
    - Available syntactic structure
    - Available thematic roles, correct meaning…

Robust Disambiguation

- More to semantics than predicate-argument structure
  - Select sense where predicates underconstrain
- Learning approaches
  - Supervised, bootstrapped, unsupervised
- Knowledge-based approaches
  - Dictionaries, taxonomies
- Contexts for sense selection

There are more kinds of **plants** and animals in the rainforests than anywhere else on Earth. Over half of the millions of known species of **plants** and animals live in the rainforest. Many are found nowhere else. There are even **plants** and animals in the rainforest that we have not yet discovered.

Biological Example

The Paulus company was founded in 1938. Since those days the product range has been the subject of constant expansions and is brought up continuously to correspond with the state of the art. We're engineering, manufacturing and commissioning world-wide ready-to-run **plants** packed with our comprehensive know-how. Our Product Range includes pneumatic conveying systems for carbon, carbide, sand, lime and many others. We use reagent injection in molten metal for the…

Industrial Example

Label the First Use of "Plant"

Roadmap

- Lexical Semantics
  - Motivation & Definitions
  - Word Senses
  - Tasks:
    - **Word sense disambiguation**
    - Word sense similarity
  - Distributional Similarity

Disambiguation: Features

- Part of Speech
  - Of word and neighbors
- Morphologically simplified form
- Words in neighborhood
  - How big is “neighborhood?”
  - Is there a single optimal size? Why?

Disambiguation: Features

- (Possibly shallow) Syntactic analysis
  - predicate-argument relations
  - modification (complements)
  - phrases
- Collocation
  - words in specific relation
    - Predicate-Argument, or (+/–)1 word index
- Co-occurrence
  - bag of words

Disambiguation: Evaluation

- Ideally, end-to-end evaluation with WSD component
  - Demonstrate real impact of technique in system
  - Difficult, expensive, still application specific
- Typically intrinsic, sense-based
  - Accuracy, precision, recall
  - SENSEVAL/SEMEVAL: all words, lexical sample

WSD Evaluation

- Baseline:
  - Most frequent sense
- Ceiling:
  - Human inter-rater agreement
    - 75-80% fine
    - 90% coarse

Roadmap

- Lexical Semantics
  - Motivation & Definitions
  - Word Senses
  - Tasks:
    - Word sense disambiguation
    - **Word sense similarity**
  - Distributional Similarity

Word Sense Similarity

- Synonymy:
  - True propositional substitutability is rare, slippery

- Word similarity (semantic distance)
  - Looser notion, more flexible

Word Sense Similarity

- Appropriate to applications:
  - IR, summarization, MT, essay scoring
    - Don’t need binary +/– synonym decision
    - Want terms/documents that have high ***similarity***
- Approaches:
  - Distributional
  - Thesaurus-based

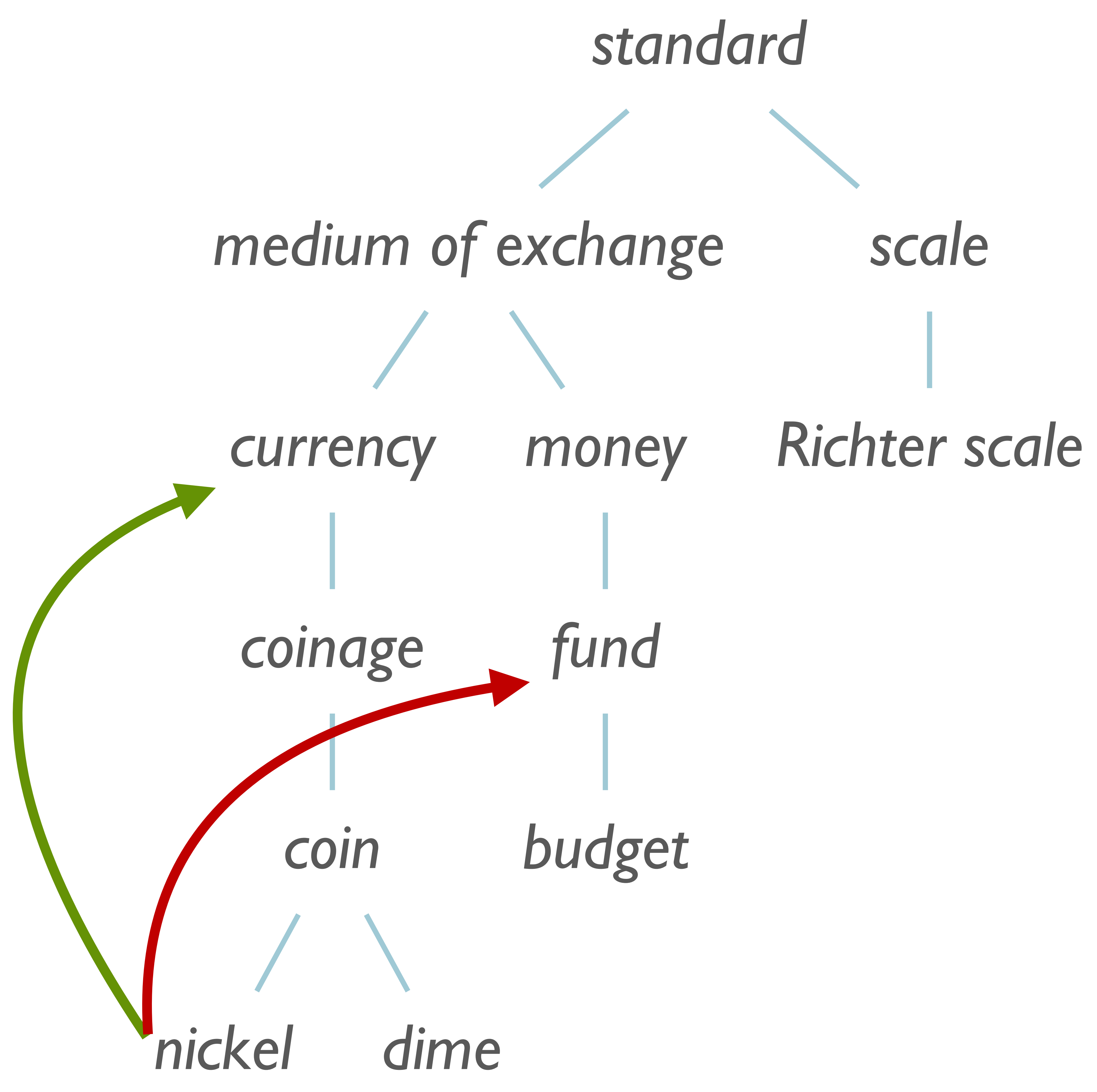
Similarity vs. Relatedness

- Similarity:
  - *car, bicycle*
  - *nickel < coin < currency*

- Related:
  - *car, gasoline*
  - *coin, budget*

Thesaurus-Based:

- Build ontology of senses
  - e.g. WordNet
  - Use distance to infer similarity/relatedness:



Roadmap

- Lexical Semantics
  - Motivation & Definitions
  - Word Senses
  - Tasks:
    - Word sense disambiguation
    - Word sense similarity
  - **Distributional Similarity**

Distributional Similarity

- "You shall know a word by the company it keeps!" (*Firth, 1957*)
  - A bottle of *tezgüino* is on the table.
  - Everybody likes *tezgüino*.
  - *Tezgüino* makes you drunk.
  - We make *tezgüino* from corn.
- Tezguino: corn-based alcoholic beverage. (From *Lin, 1998a*)

Distributional Similarity

- Represent 'company' of word such that similar words will have similar representations
  - 'Company' = context
- Word represented by context feature vector
  - Many alternatives for vector
- Initial representation:
  - 'Bag of words' binary feature vector
  - Feature vector length N, where N is size of vocabulary
    - f_i=1 if $word_i$ within window size w of $word_0$

Context Feature Vector

	arts	boil	data	function	large	sugar	summarized	water
Apricot	0	1	0	0	1	1	0	1
Pineapple	0	1	0	0	1	1	0	1
Digital	0	0	1	1	1	0	1	0
Information	0	0	1	1	1	0	1	0

Distributional Similarity Questions

- What is the right neighborhood?
  - What is the context?
- How should we weight the features?
- How can we compute similarity between vectors?

HW #6

Goals

- Semantics
  - Gain better understanding of semantic representations
  - Develop experience with lambda calculus and FOL
  - Create semantic attachments
  - Understand semantic composition

Compositional Semantics

- **Part 1**:
  - ***Manually*** create target semantic representations
  - Use Neo-Davidsonian event representation
    - e.g. verb representation with event variable, argument conjuncts
  - Can use as test cases for part 2
- **Part 2**:
  - Create semantic attachments to reproduce (NLTK)
  - Add to grammatical rules to derive sentence representations
- Note: Lots of ambiguities (scope, etc)
  - Only need to produce one

Semantics in NLTK

- Grammar files:
  - .fcfg extension
  - Example format in NLTK Book Chapter 10
  - `/corpora/nltk/nltk-data/grammars/book_grammars/simple-sem.fcfg`
  - Note: Not “event-style”
- Parsing:
  - Use `nltk.parse.FeatureChartParser` (or similar)

Semantics in NLTK

- Printing semantic representations:

```
item.label()['SEM'].simplify()
    all x.(dog(x) -> exists e.(barking(e) & barker(e,x)))
```

- Also `nltk.sem.util.root_semrep(item)`

Semantic attachments in NLTK: Syntax

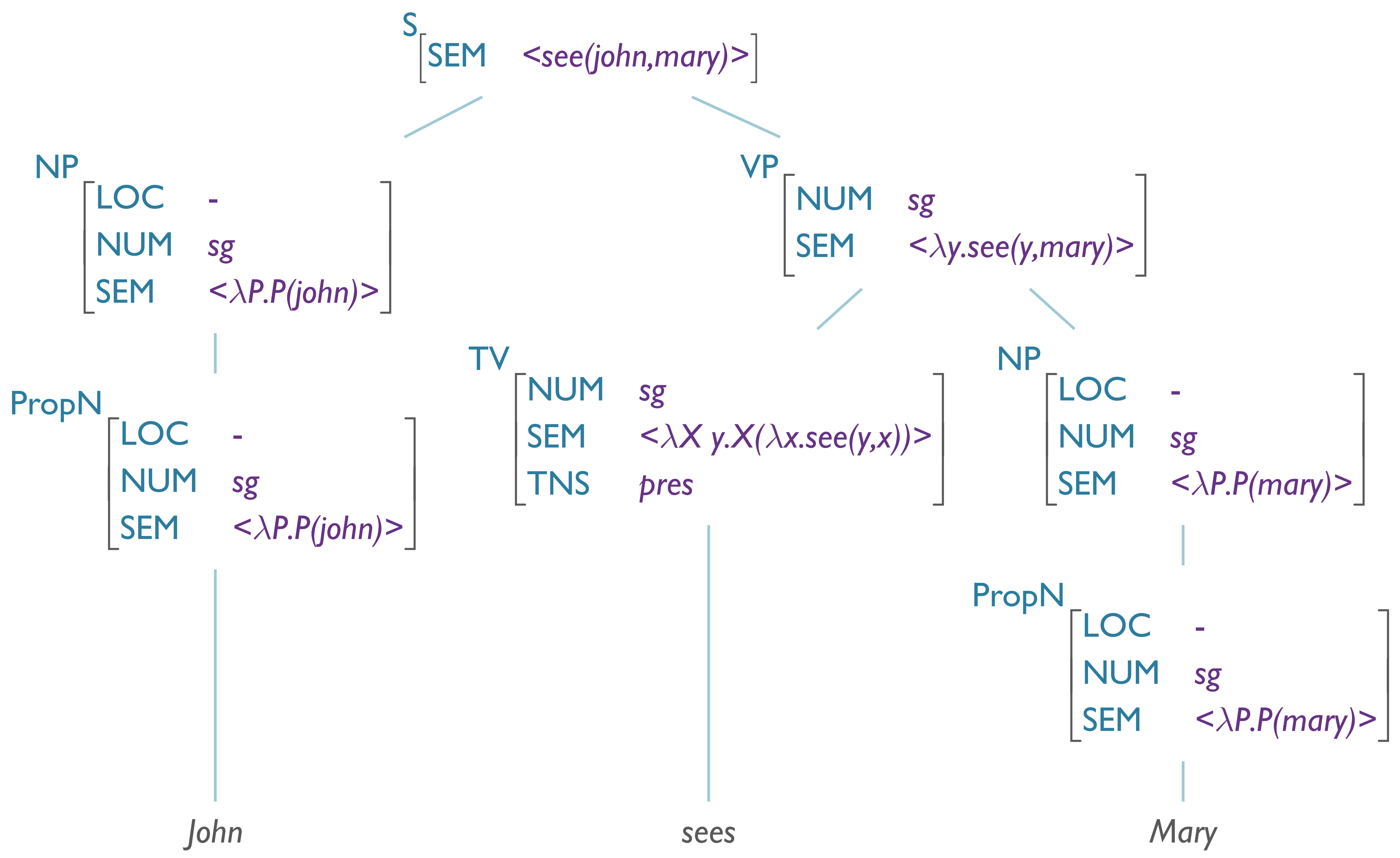
(The programming kind)

- `a,b,e,x`
  - lowercase variables can be arguments:
  - `\x.dog(x)`
- `P,Q,X`
  - uppercase lambda variables are functors
  - `\P.P(john)`

λ	=	`\`
∃	=	`exists`
∀	=	`all`
∧	=	`&`
∨	=	`|`
⇒	=	`->`
¬	=	`-`

More NLTK Logic Format

- Added to typical CFG rules
  - Basic approach similar to HW #5
  - Composing semantics:
    - `S[SEM=<?np(?vp)>] -> NP[SEM=?np] VP[SEM=?vp]`
- Creating lambdas:
  - `IV[SEM=<\x.exists e.(barking(e) & barker(e,x))>] -> ‘barks’`
- Nested lambdas:
  - `\x.\y. Etc → \x y.`
    Can remove ‘.’ between sequences of lambda elements
    Keep ‘.’ between sections: lambdas, quantifiers, body


S
SEM <see(john,mary)>
NP
LOC -
NUM sg
SEM <λP.P(john)>
VP
NUM sg
SEM <λy.see(y,mary)>
PropN
LOC -
NUM sg
SEM <λP.P(john)>
TV
NUM sg
SEM <λX y.X(λx.see(y,x))>
TNS pres
NP
LOC -
NUM sg
SEM <λP.P(mary)>
PropN
LOC -
NUM sg
SEM <λP.P(mary)>
John
sees
Mary