

# Computational Semantics

LING 571 — Deep Processing for NLP

November 2, 2020

# Assignments

- HW4
  - Delay in grade posting due to computer problems.
  - Use `check_hwX.sh` before submitting!

# Announcements

- HW5: your grammar should use rules and features that are linguistically motivated (e.g. number, gender, aspect, animacy, ....)
- Consider grammars for the following suite of examples:
  - This sentence is grammatical.
  - \*This grammatical sentence is.
- The following is not an acceptable grammar (you would lose some points):
  - $S[+grammatical] \rightarrow \text{'This sentence is grammatical.'}$
  - $S[-grammatical] \rightarrow \text{'This grammatical sentence is.'}$

# NP of the Week

The screenshot shows the top navigation bar of The Guardian website. On the left, there's a yellow 'guardian' logo with a small orange icon next to it, followed by the text 'by readers'. To the right are links for 'Search jobs', 'Sign in', 'Search' (with a magnifying glass icon), and 'US edition' (with a dropdown arrow). The main title 'The Guardian' is prominently displayed in large white letters. Below the main title, there's a horizontal menu with categories: 'World', 'Sport', 'Culture', 'Lifestyle', and 'More' (with a dropdown arrow). Underneath this menu, there's a secondary row of links: 'Business', 'Football', 'Environment', 'UK politics', 'Education', 'Society', 'Science', 'Tech', 'Global development', and 'Obituaries'.

## Trade department's Brexit soy sauce price cut claim prompts backlash

<https://www.theguardian.com/politics/2020/oct/28/dfids-brexit-soy-sauce-price-cut-claim-prompts-backlash-bake-off>

# Varieties of Entailment in the News

# Presuppositions, etc

Behold Trump's pre-election secret weapon: Nigel Farage, 'king of Europe'

# Presuppositions, etc

- “I present to you the King of Europe, Nigel Farage” — Trump (paraphrased)
  - *presupposes that there is a king of Europe*

Behold Trump's pre-election secret weapon: Nigel Farage, 'king of Europe'

# Presuppositions, etc

- “I present to you the King of Europe, Nigel Farage” — Trump (paraphrased)
  - *presupposes that there is a king of Europe*
- Consider two sentences:
  - “The King of Europe is here today.”
  - “The King of Europe is NOT here today.”
  - From both, it follows that there is a King of Europe.

Behold Trump's pre-election secret weapon: Nigel Farage, 'king of Europe'

# Presuppositions, etc

- “I present to you the King of Europe, Nigel Farage” — Trump (paraphrased)
  - *presupposes that there is a king of Europe*
- Consider two sentences:
  - “The King of Europe is here today.”
  - “The King of Europe is NOT here today.”
  - From both, it follows that there is a King of Europe.
- Contrast:
  - “We are talking on Zoom right now.”
  - “We are NOT talking on Zoom right now.”
  - The former, but not the latter, entails that we are talking right now.

Behold Trump's pre-election secret weapon: Nigel Farage, 'king of Europe'

# Presuppositions, etc

- “I present to you the King of Europe, Nigel Farage” — Trump (paraphrased)
  - *presupposes* that there *is* a king of Europe
- Consider two sentences:
  - “The King of Europe is here today.”
  - “The King of Europe is NOT here today.”
  - From both, it follows that there is a King of Europe.
- Contrast:
  - “We are talking on Zoom right now.”
  - “We are NOT talking on Zoom right now.”
  - The former, but not the latter, entails that we are talking right now.
- Presuppositions (that there is a king) “project out” from negation (and other operators, like questions, conditionals, etc). Standard logical entailments do not.
- Presuppositions must be true in order for a sentence to be true or false at all.

Behold Trump's pre-election secret weapon: Nigel Farage, 'king of Europe'

# (Scalar) Implicatures

# (Scalar) Implicatures

- “Some conferences were cancelled this year.”
  - Seems to entail: “Not all conferences were cancelled this year.”
  - But: can follow with “In fact, all of them were!” (In jargon: the implicature can be cancelled.)

# (Scalar) Implicatures

- “Some conferences were cancelled this year.”
  - Seems to entail: “Not all conferences were cancelled this year.”
  - But: can follow with “In fact, all of them were!” (In jargon: the implicature can be cancelled.)
- Conversational implicature: inferences that a speaker would tend to draw assuming a cooperative and knowledgable speaker.

# (Scalar) Implicatures

- “Some conferences were cancelled this year.”
  - Seems to entail: “Not all conferences were cancelled this year.”
  - But: can follow with “In fact, all of them were!” (In jargon: the implicature can be cancelled.)
- Conversational implicature: inferences that a speaker would tend to draw assuming a cooperative and knowledgable speaker.
- In this example: speaker could have said “All conferences were cancelled.” Since they did not, assume that it is false.
  - Common examples of scales: {some, all}, {or, and}, {may, must}, ...

# (Scalar) Implicatures

- “Some conferences were cancelled this year.”
  - Seems to entail: “Not all conferences were cancelled this year.”
  - But: can follow with “In fact, all of them were!” (In jargon: the implicature can be cancelled.)
- Conversational implicature: inferences that a speaker would tend to draw assuming a cooperative and knowledgable speaker.
- In this example: speaker could have said “All conferences were cancelled.” Since they did not, assume that it is false.
  - Common examples of scales: {some, all}, {or, and}, {may, must}, ...
- Trump’s doctor when he was at the hospital:
  - Press: “Has he ever been on supplemental oxygen?”
  - Doc: “He hasn’t had supplemental oxygen today or yesterday.”

# Presupposition, Entailment, Implicature?

# Presupposition, Entailment, Implicature?

- “Several students were told that the exam will be postponed.”

# Presupposition, Entailment, Implicature?

- “Several students were told that the exam will be postponed.”
- There is an exam.

# Presupposition, Entailment, Implicature?

- “Several students were told that the exam will be postponed.”
  - There is an exam.
  - A student was told that the exam will be postponed.

# Presupposition, Entailment, Implicature?

- “Several students were told that the exam will be postponed.”
  - There is an exam.
  - A student was told that the exam will be postponed.
  - The exam will be postponed.

# Presupposition, Entailment, Implicature?

- “Several students were told that the exam will be postponed.”
  - There is an exam.
  - A student was told that the exam will be postponed.
  - The exam will be postponed.
  - Not every student was told that the exam will be postponed.

# An Interesting Example

**A top baseball prospect's Southern California scholarship was lost to the pandemic**

<https://www.washingtonpost.com/road-to-recovery/2020/11/02/tank-espalin-usc-indiana-baseball/>

# An Interesting Example

**A top baseball prospect's Southern California scholarship was lost to the pandemic**

<https://www.washingtonpost.com/road-to-recovery/2020/11/02/tank-espalin-usc-indiana-baseball/>

“A prospect’s scholarship”: presupposes there is a scholarship

Rest of headline: there is no more scholarship

Complex compositional interaction between tense and presupposition

# An Interesting Example

**A top baseball prospect's Southern California scholarship was lost to the pandemic**

<https://www.washingtonpost.com/road-to-recovery/2020/11/02/tank-espalin-usc-indiana-baseball/>

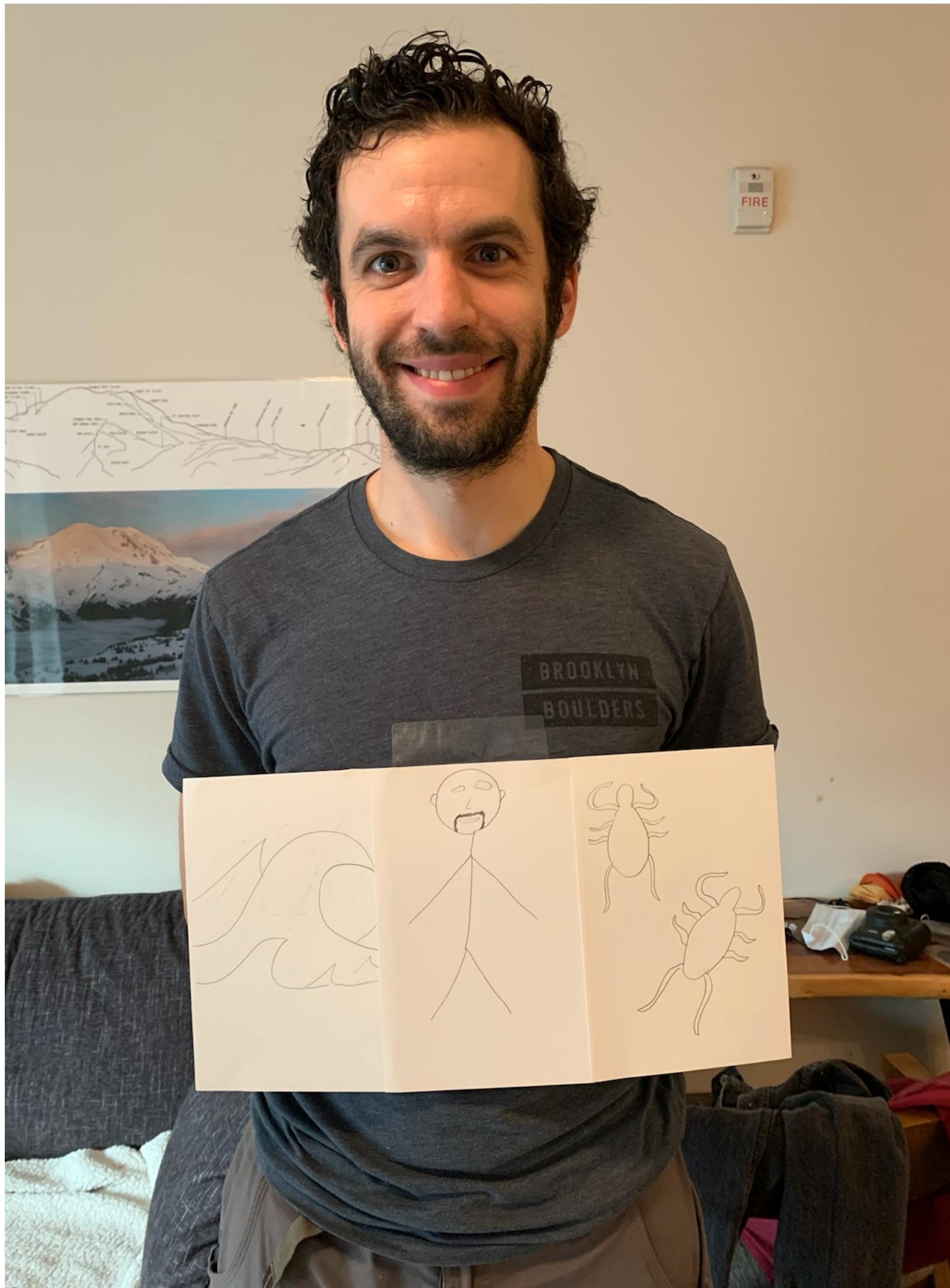
“A prospect’s scholarship”: presupposes there is a scholarship

Rest of headline: there is no more scholarship

Complex compositional interaction between tense and presupposition

Poll!

# Guess My “Costume”!



It's topical!  
Each picture is a syllable;  
together the form one word.

# Roadmap

- First-order Logic: Syntax and Semantics
- Inference + Events
- Rule-to-rule Model
  - More lambda calculus

# FOL Syntax + Semantics

# Example Meaning Representation

- A non-stop flight that serves Pittsburgh:

$$\exists x \text{ } Flight(x) \wedge Serves(x, \text{Pittsburgh}) \wedge \text{Non-stop}(x)$$

# FOL Syntax Summary

<b>Formula</b>	$\rightarrow$	<i>AtomicFormula</i>	<b>Connective</b>	$\rightarrow$	$\wedge \mid \vee \mid \Rightarrow$
		<i>Formula Connective Formula</i>	<b>Quantifier</b>	$\rightarrow$	$\forall \mid \exists$
		<i>Quantifier Variable, ... Formula</i>	<b>Constant</b>	$\rightarrow$	<i>VegetarianFood</i>   <i>Maharani</i>   ...
		$\neg$ <i>Formula</i>	<b>Variable</b>	$\rightarrow$	<i>x</i>   <i>y</i>   ...
		( <i>Formula</i> )	<b>Predicate</b>	$\rightarrow$	<i>Serves</i>   <i>Near</i>   ...
<b>AtomicFormula</b>	$\rightarrow$	<i>Predicate(Term,...)</i>	<b>Function</b>	$\rightarrow$	<i>LocationOf</i>   <i>CuisineOf</i>   ...
<b>Term</b>	$\rightarrow$	<i>Function(Term,...)</i>			
		<i>Constant</i>			
		<i>Variable</i>			

J&M p. 556 (3rd ed. 16.3)

# Model-Theoretic Semantics

- A “model” represents a particular state of the world
- Our language has **logical** and **non-logical** elements.
  - **Logical:** Symbols, operators, quantifiers, etc
  - **Non-Logical:** Names, properties, relations, etc

# Denotation

- Every **non-logical** element points to a fixed part of the model

# Denotation

- Every **non-logical** element points to a fixed part of the model
- **Objects** — elements in the domain, denoted by *terms*
  - *John, Farah, fire engine, dog, stop sign*

# Denotation

- Every **non-logical** element points to a fixed part of the model
- **Objects** — elements in the domain, denoted by *terms*
  - *John, Farah, fire engine, dog, stop sign*
- **Properties** — sets of elements
  - **red**: *{fire hydrant, apple, ...}*

# Denotation

- Every **non-logical** element points to a fixed part of the model
- **Objects** — elements in the domain, denoted by *terms*
  - *John, Farah, fire engine, dog, stop sign*
- **Properties** — sets of elements
  - **red**: *{fire hydrant, apple, ...}*
- **Relations** — *sets of tuples of elements*
  - **CapitalCity**: *{(Washington, Olympia), (Yamoussokro, Cote d'Ivoire), (Ulaanbaatar, Mongolia), ...}*

# Sample Domain $\mathcal{D}$

via J&M, p. 554

## Objects

Matthew, Franco, Katie, Caroline	$a, b, c, d$
Frasca, Med, Rio	$e, f, g$
Italian, Mexican, Eclectic	$h, i, j$

# Sample Domain $\mathcal{D}$

via J&M, p. 554

## Objects

Matthew, Franco, Katie, Caroline	$a, b, c, d$
Frasca, Med, Rio	$e, f, g$
Italian, Mexican, Eclectic	$h, i, j$

## Properties

<b>Noisy</b>	Frasca, Med, and Rio are noisy	<b>Noisy</b> = $\{e, f, g\}$
--------------	--------------------------------	------------------------------

# Sample Domain $\mathcal{D}$

via J&M, p. 554

## Objects

Matthew, Franco, Katie, Caroline	$a, b, c, d$
Frasca, Med, Rio	$e, f, g$
Italian, Mexican, Eclectic	$h, i, j$

## Properties

<b>Noisy</b>	Frasca, Med, and Rio are noisy	<b>Noisy</b> = $\{e, f, g\}$
--------------	--------------------------------	------------------------------

## Relations

<b>Likes</b>	Matthew likes the Med Katie likes the Med and Rio Franco likes Frasca Caroline likes the Med and Rio	<b>Likes</b> = $\{ \langle a, f \rangle, \langle c, f \rangle, \langle c, g \rangle, \langle b, e \rangle, \langle d, f \rangle, \langle d, g \rangle \}$
--------------	---	---

# Sample Domain $\mathcal{D}$

via J&M, p. 554

## Objects

Matthew, Franco, Katie, Caroline	$a, b, c, d$
Frasca, Med, Rio	$e, f, g$
Italian, Mexican, Eclectic	$h, i, j$

## Properties

<b>Noisy</b>	Frasca, Med, and Rio are noisy	<b>Noisy</b> = $\{e, f, g\}$
--------------	--------------------------------	------------------------------

## Relations

<b>Likes</b>	Matthew likes the Med Katie likes the Med and Rio Franco likes Frasca Caroline likes the Med and Rio	<b>Likes</b> = $\{ \langle a, f \rangle, \langle c, f \rangle, \langle c, g \rangle, \langle b, e \rangle, \langle d, f \rangle, \langle d, g \rangle \}$
--------------	---	---

<b>Serves</b>	Med serves eclectic Rio serves Mexican Frasca serves Italian	<b>Serves</b> = $\{ \langle c, f \rangle, \langle f, i \rangle, \langle e, h \rangle \}$
---------------	--	--

# Inference + Events

(last Wednesday's slides)

# Rule-to-Rule Model

# Recap

- **Meaning Representation**
  - Can represent meaning in natural language in many ways
  - We are focusing on First-Order Logic (FOL)

# Recap

- **Meaning Representation**
  - Can represent meaning in natural language in many ways
  - We are focusing on First-Order Logic (FOL)
- **Principle of compositionality**
  - The meaning of a complex expression is a function of the meaning of its parts

# Recap

- **Meaning Representation**
  - Can represent meaning in natural language in many ways
  - We are focusing on First-Order Logic (FOL)
- **Principle of compositionality**
  - The meaning of a complex expression is a function of the meaning of its parts
- **Lambda Calculus**
  - $\lambda$ -expressions denote functions
  - Can be nested
  - Reduction = function application

# Semantics Reflects Syntax

# Chiasmus: Syntax affects Semantics!



*Bowie playing Tesla*

*The Prestige* (2006)



*Tesla playing Bowie*

*SpaceX Falcon Heavy Test Launch* (2/6/2018)

# Chiasmus: Syntax affects Semantics!

- “Never let a fool kiss you or a kiss fool you” (Grothe, 2002)
- “Then you should say what you mean,” the March Hare went on.

“I do,” Alice hastily replied; “at least—at least I mean what I say—that’s the same thing, you know.”

“Not the same thing a bit!” said the Hatter. “Why, you might just as well say that ‘I see what I eat’ is the same thing as ‘I eat what I see’!”

“You might just as well say,” added the March Hare,  
“that ‘I like what I get’ is the same thing as ‘I get what I like’!”

“You might just as well say,” added the Dormouse, which seemed to be talking in his sleep,  
“that ‘I breathe when I sleep’ is the same thing as ‘I sleep when I breathe’!”

—Alice in Wonderland, Lewis Carroll

# Ambiguity & Models

- “*Every Tesla is powered by a battery.*” — Ambiguous!

# Ambiguity & Models

- “*Every Tesla is powered by a battery.*” — Ambiguous!
- $\forall x. \text{Tesla}(x) \Rightarrow (\exists(y). \text{Battery}(y) \wedge \text{Powers}(y, x))$

# Ambiguity & Models

- “*Every Tesla is powered by a battery.*” — Ambiguous!
  - $\forall x. \text{Tesla}(x) \Rightarrow (\exists(y). \text{Battery}(y) \wedge \text{Powers}(y, x))$
  - $\exists(y). \text{Battery}(y) \wedge (\forall x. \text{Tesla}(x) \Rightarrow \text{Powers}(y, x))$

# Ambiguity & Models

- “*Every Tesla is powered by a battery.*” — Ambiguous!
  - $\forall x. \text{Tesla}(x) \Rightarrow (\exists(y). \text{Battery}(y) \wedge \text{Powers}(y, x))$
  - $\exists(y). \text{Battery}(y) \wedge (\forall x. \text{Tesla}(x) \Rightarrow \text{Powers}(y, x))$
- Every Tesla is not hurtling toward Mars.

# Ambiguity & Models

- “*Every Tesla is powered by a battery.*” — Ambiguous!
  - $\forall x. \text{Tesla}(x) \Rightarrow (\exists(y). \text{Battery}(y) \wedge \text{Powers}(y, x))$
  - $\exists(y). \text{Battery}(y) \wedge (\forall x. \text{Tesla}(x) \Rightarrow \text{Powers}(y, x))$
- Every Tesla is not hurtling toward Mars.
  - $\forall x. \text{Tesla}(x) \Rightarrow \neg(\text{HurtlingTowardMars}(x))$

# Ambiguity & Models

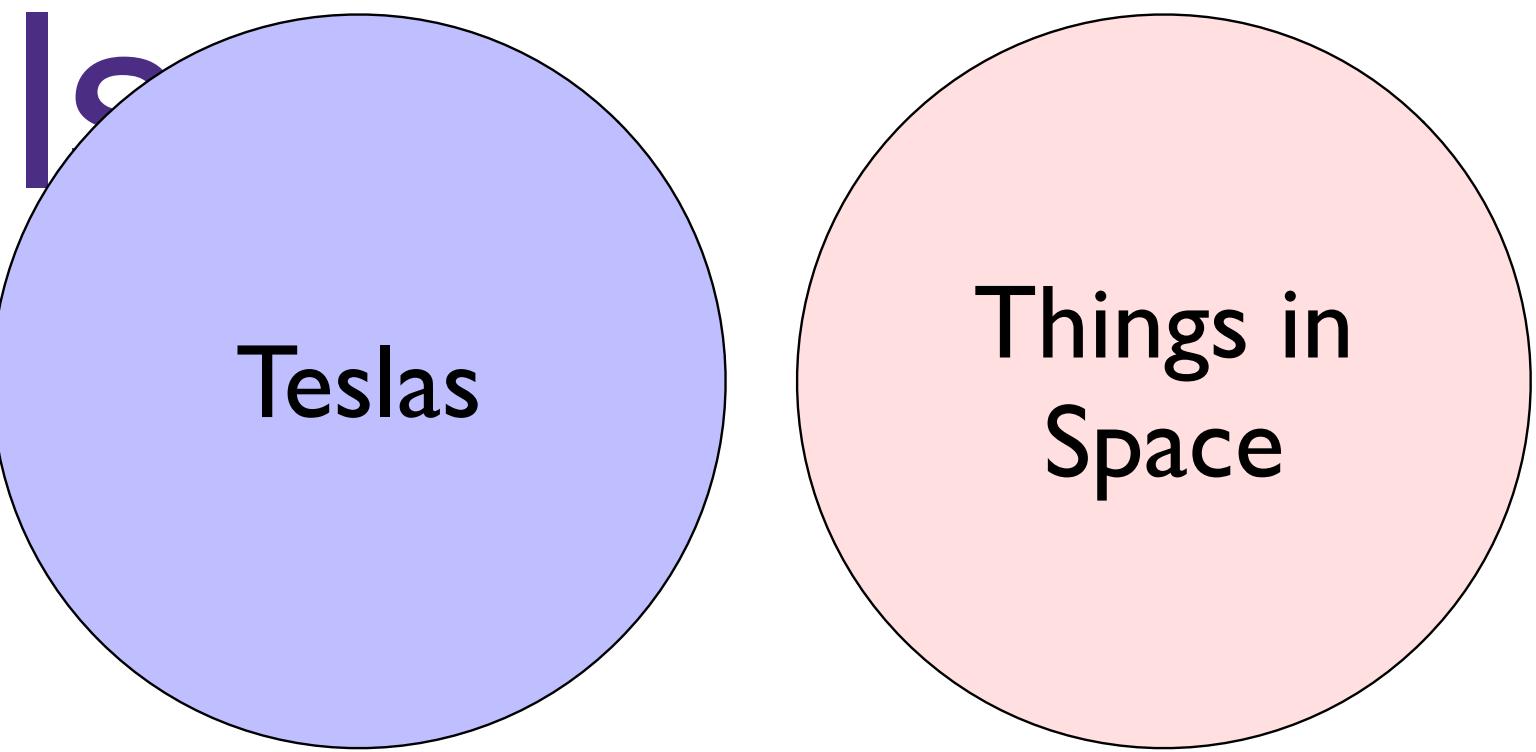
- “*Every Tesla is powered by a battery.*” — Ambiguous!
  - $\forall x. \text{Tesla}(x) \Rightarrow (\exists(y). \text{Battery}(y) \wedge \text{Powers}(y, x))$
  - $\exists(y). \text{Battery}(y) \wedge (\forall x. \text{Tesla}(x) \Rightarrow \text{Powers}(y, x))$
- Every Tesla is not hurtling toward Mars.
  - $\forall x. \text{Tesla}(x) \Rightarrow \neg(\text{HurtlingTowardMars}(x))$
  - $\neg\forall x. (\text{Tesla}(x) \Rightarrow (\text{HurtlingTowardMars}(x)))$

# Ambiguity & Models

- “*Every Tesla is powered by a battery.*” — Ambiguous!
  - $\forall x. Tesla(x) \Rightarrow (\exists(y). Battery(y) \wedge Powers(y, x))$
  - $\exists(y). Battery(y) \wedge (\forall x. Tesla(x) \Rightarrow Powers(y, x))$
- Every Tesla is not hurtling toward Mars.
  - $\forall x. Tesla(x) \Rightarrow \neg(HurtlingTowardMars(x))$
  - $\neg\forall x. (Tesla(x) \Rightarrow (HurtlingTowardMars(x)))$
  - $[\exists(x). (Tesla(x) \wedge \neg HurtlingTowardMars(x))]$

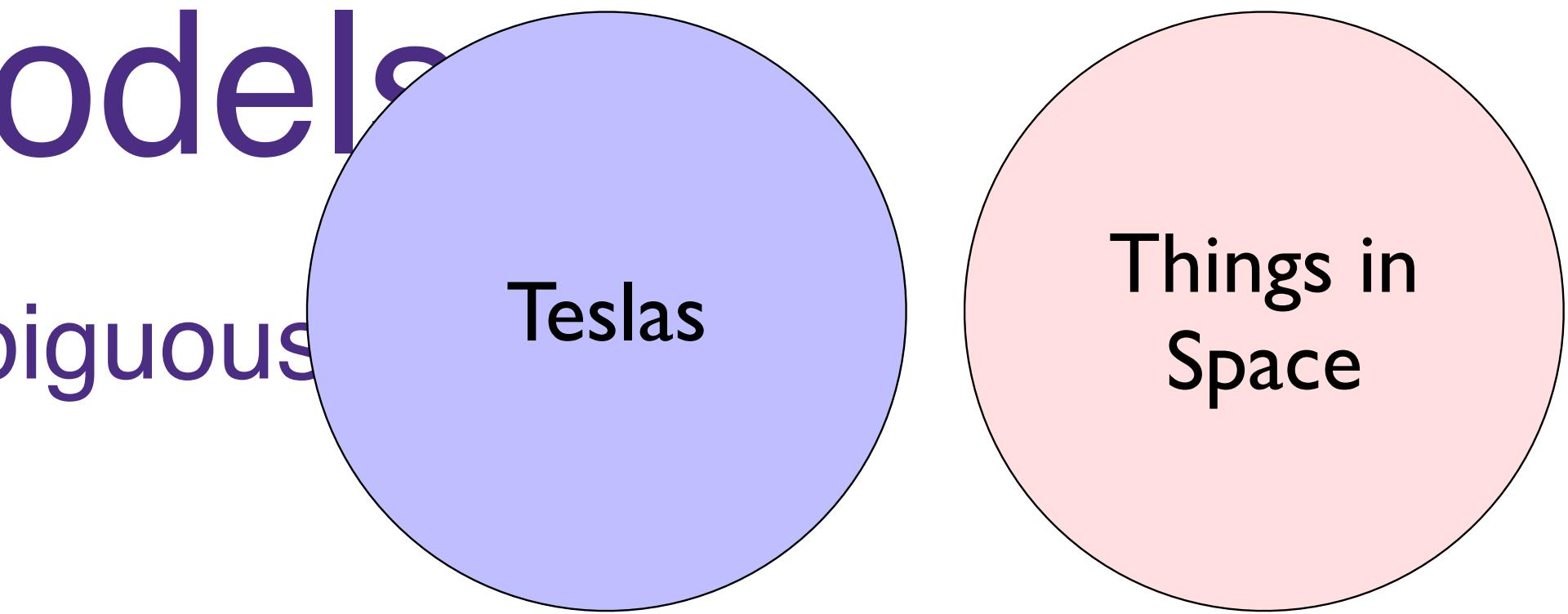
# Ambiguity & Models

- “*Every Tesla is powered by a battery.*” — Ambiguous
  - $\forall x. Tesla(x) \Rightarrow (\exists(y). Battery(y) \wedge Powers(y, x))$
  - $\exists(y). Battery(y) \wedge (\forall x. Tesla(x) \Rightarrow Powers(y, x))$
- Every Tesla is not hurtling toward Mars.
  - $\forall x. Tesla(x) \Rightarrow \neg(HurtlingTowardMars(x))$
  - $\neg\forall x. (Tesla(x) \Rightarrow (HurtlingTowardMars(x)))$
  - $[\exists(x). (Tesla(x) \wedge \neg HurtlingTowardMars(x))]$



# Ambiguity & Models

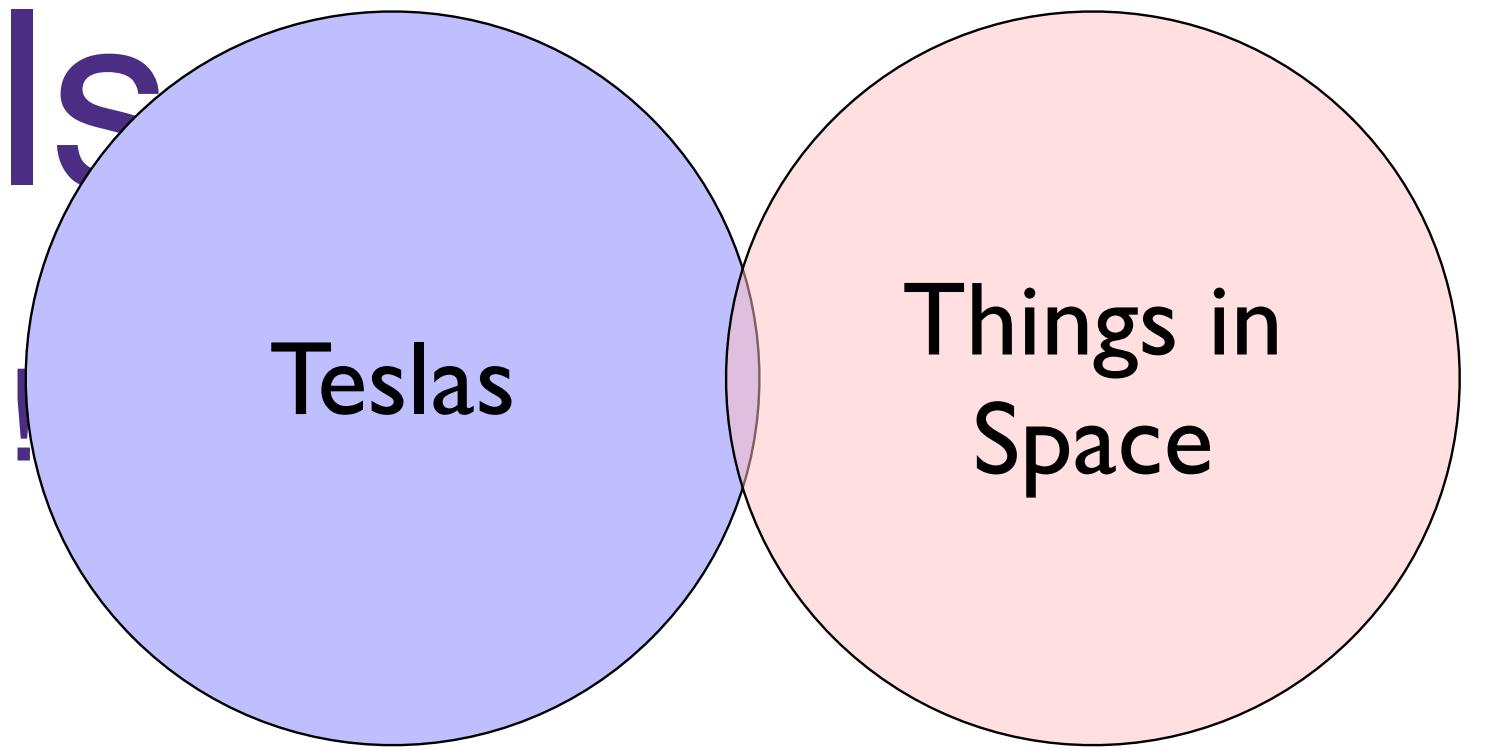
- “*Every Tesla is powered by a battery.*” — Ambiguous
  - $\forall x. \text{Tesla}(x) \Rightarrow (\exists(y). \text{Battery}(y) \wedge \text{Powers}(y, x))$
  - $\exists(y). \text{Battery}(y) \wedge (\forall x. \text{Tesla}(x) \Rightarrow \text{Powers}(y, x))$
- Every Tesla is not hurtling toward Mars.
  - $\forall x. \text{Tesla}(x) \Rightarrow \neg(\text{HurtlingTowardMars}(x))$
  - $\neg\forall x. (\text{Tesla}(x) \Rightarrow (\text{HurtlingTowardMars}(x)))$
  - $[\exists(x). (\text{Tesla}(x) \wedge \neg\text{HurtlingTowardsMars}(x))]$



$\exists(x). (\text{Tesla}(x) \wedge \text{HurtlingTowardsMars}(x))$

# Ambiguity & Models

- “*Every Tesla is powered by a battery.*” — Ambiguous!
- $\forall x. \text{Tesla}(x) \Rightarrow (\exists(y). \text{Battery}(y) \wedge \text{Powers}(y, x))$
- $\exists(y). \text{Battery}(y) \wedge (\forall x. \text{Tesla}(x) \Rightarrow \text{Powers}(y, x))$
- Every Tesla is not hurtling toward Mars.
- $\forall x. \text{Tesla}(x) \Rightarrow \neg(\text{HurtlingTowardMars}(x))$
- $\neg\forall x. (\text{Tesla}(x) \Rightarrow (\text{HurtlingTowardMars}(x)))$
- $[\exists(x). (\text{Tesla}(x) \wedge \neg \text{HurtlingTowardsMars}(x))]$

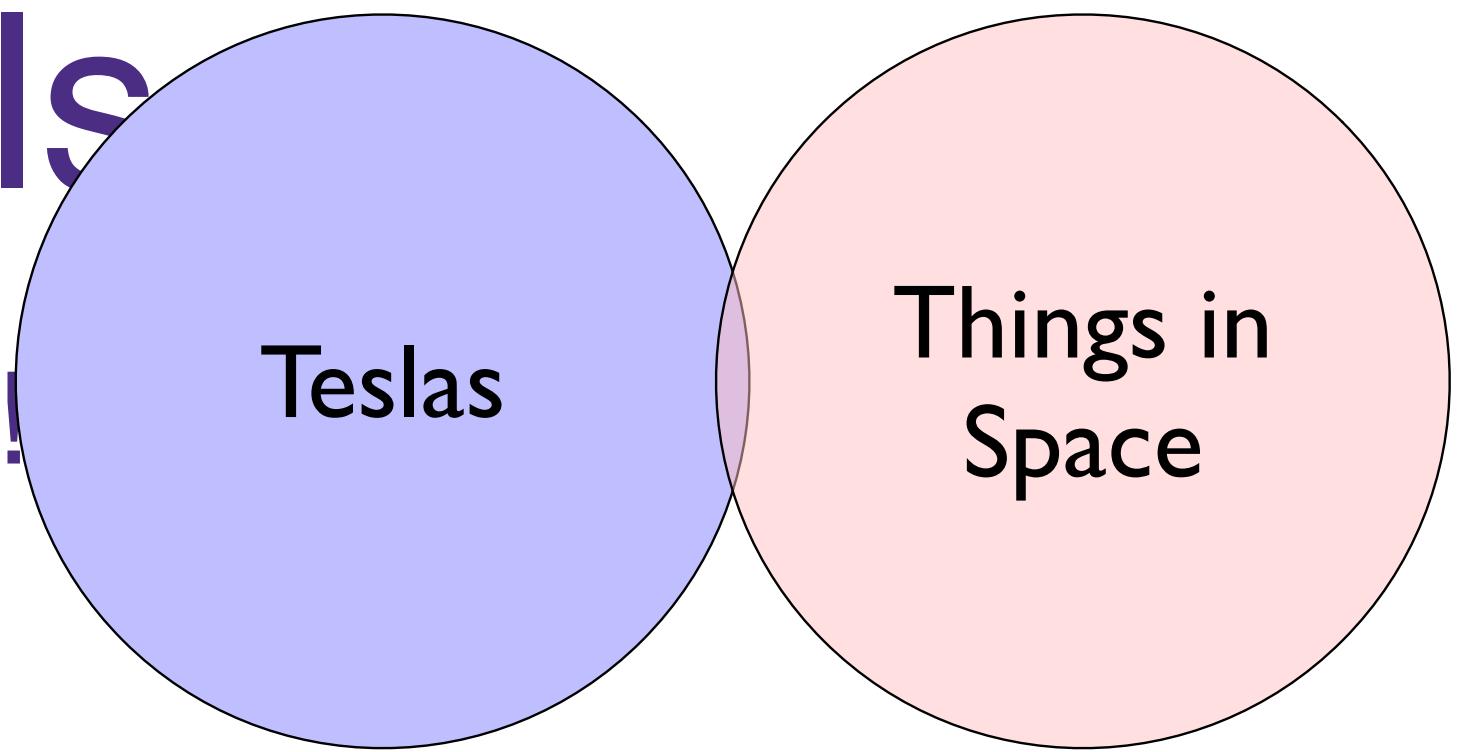


$\exists(x). (\text{Tesla}(x) \wedge \text{HurtlingTowardsMars}(x))$

# Ambiguity & Models

- “*Every Tesla is powered by a battery.*” — Ambiguous!

- $\forall x. \text{Tesla}(x) \Rightarrow (\exists(y). \text{Battery}(y) \wedge \text{Powers}(y, x))$
- $\exists(y). \text{Battery}(y) \wedge (\forall x. \text{Tesla}(x) \Rightarrow \text{Powers}(y, x))$



- Every Tesla is not hurtling toward Mars.

- ~~$\forall x. \text{Tesla}(x) \Rightarrow \neg(\text{HurtlingTowardMars}(x))$~~
- $\neg\forall x. (\text{Tesla}(x) \Rightarrow (\text{HurtlingTowardMars}(x)))$
- $[\exists(x). (\text{Tesla}(x) \wedge \neg\text{HurtlingTowardsMars}(x))]$



$\exists(x). (\text{Tesla}(x) \wedge \text{HurtlingTowardsMars}(x))$

# Scope Ambiguity

- Potentially  $O(n!)$  scope interpretations (“scopings”)
  - Where  $n$ =number of scope-taking operators.
    - (*every, a, all, no, modals, negations, conditionals, ...*)
- Different interpretations correspond to different syntactic parses!

Poll!

# Integrating Semantics into Syntax

## 1. Pipeline System

- Feed parse tree and sentence to semantic analyzer
- How do we know which pieces of the semantics link to which part of the analysis?
- Need detailed information about sentence, parse tree
- Infinitely many sentences & parse trees
- Semantic mapping function per parse tree → intractable

# Integrating Semantics into Syntax

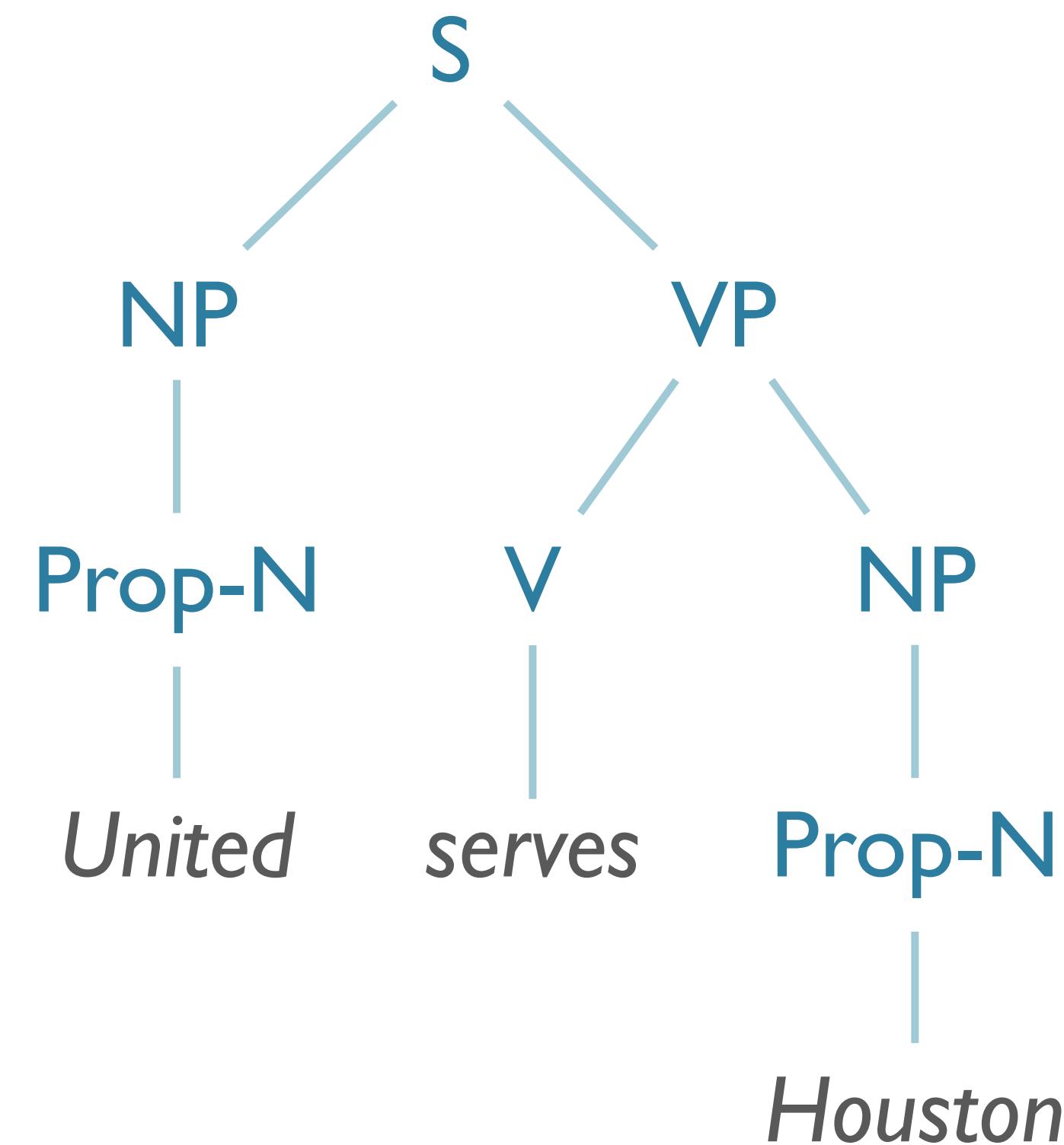
# Integrating Semantics into Syntax

## 2. Integrate Directly into Grammar

- This is the “rule-to-rule” approach we’ve been implicitly examining and will now make more explicit
- Tie semantics to finite components of grammar (rules & lexicon)
- Augment grammar rules with semantic info
  - a.k.a. “attachments” — specify how RHS elements compose to LHS

# Simple Example

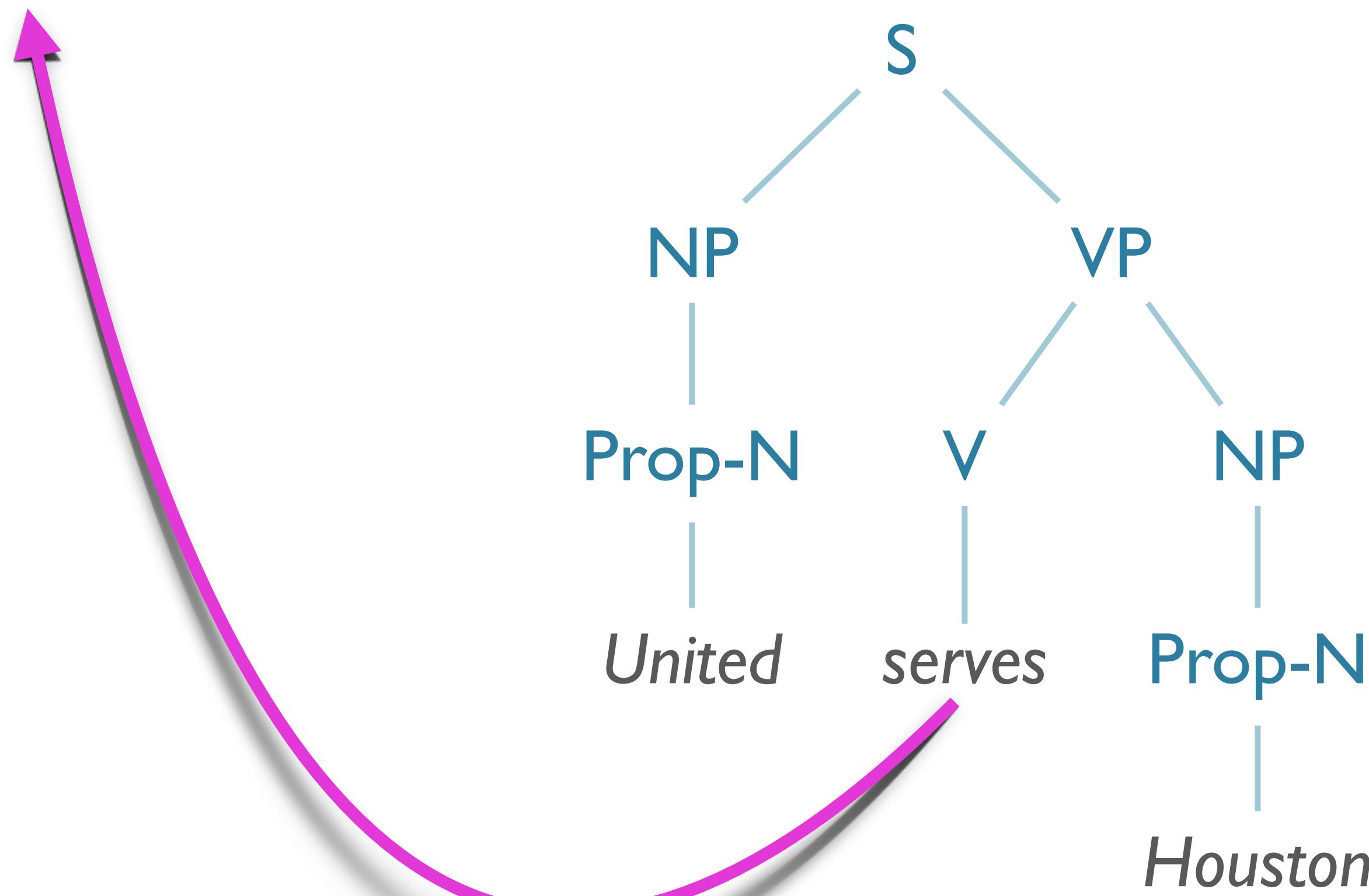
- *United serves Houston*



# Simple Example

- *United serves Houston*

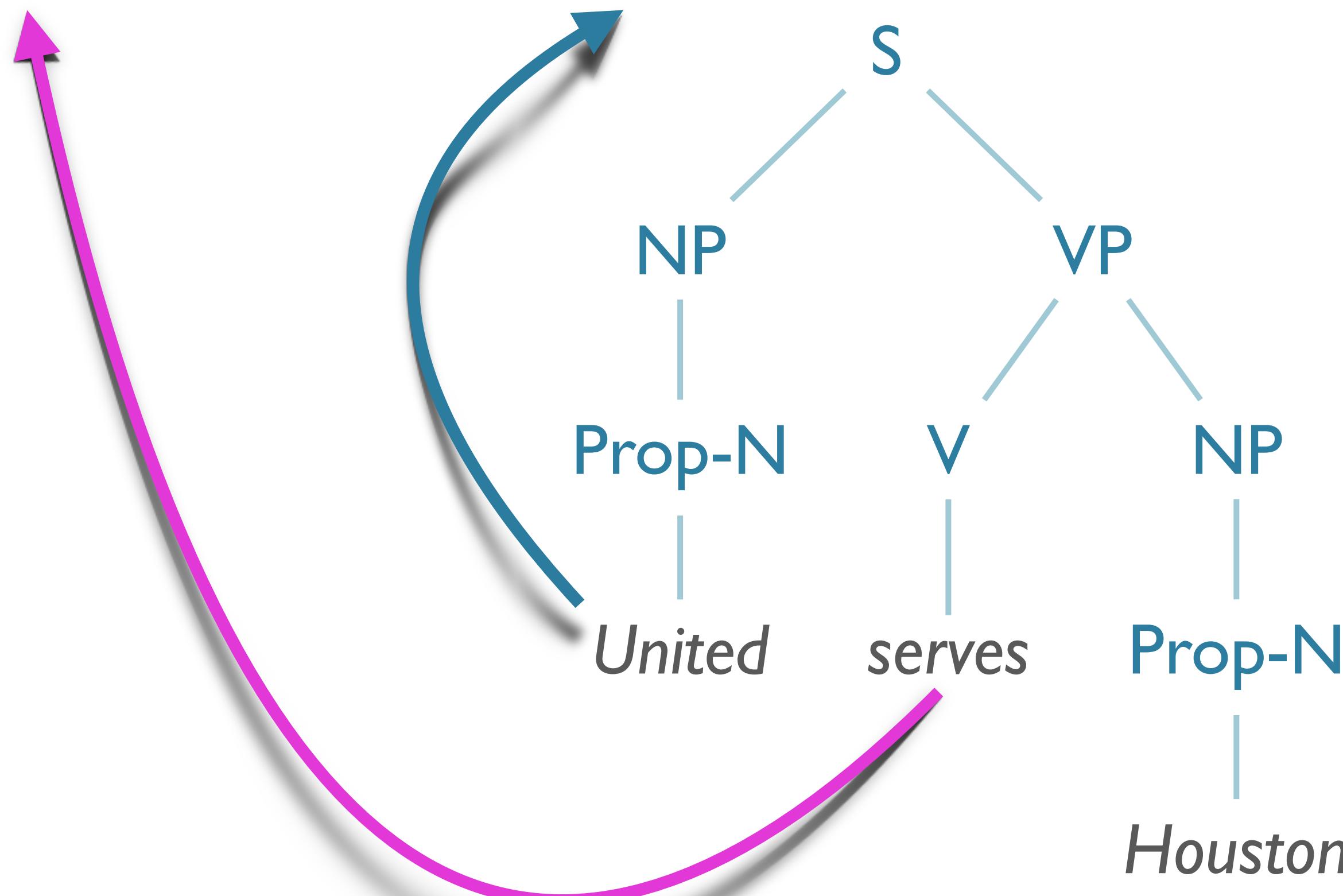
$\exists e (\text{Serving}(e) \wedge$



# Simple Example

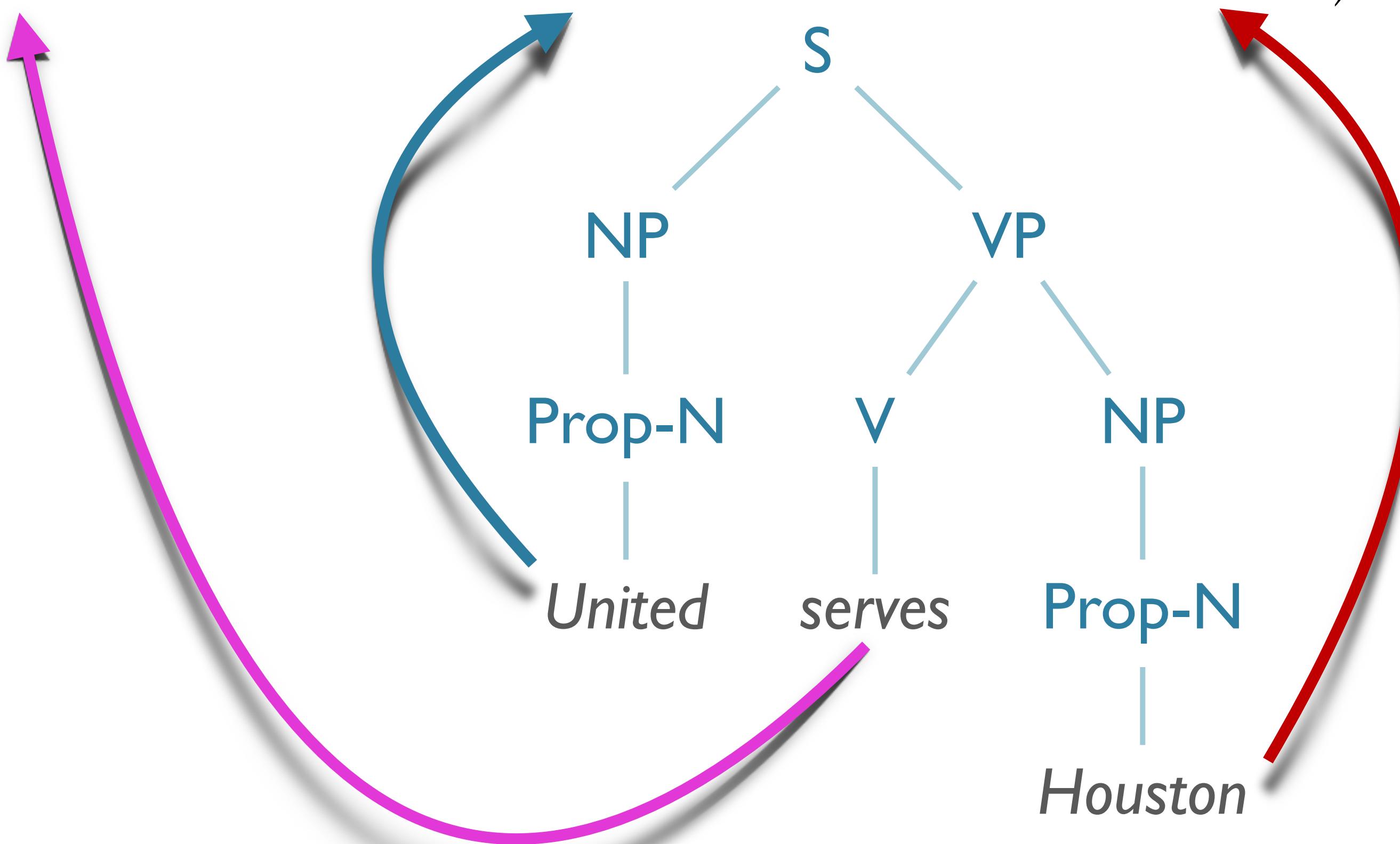
- *United serves Houston*

$\exists e (\text{Serving}(e) \wedge \text{Server}(e, \text{United}) \wedge$



# Simple Example

- *United serves Houston*

$$\exists e (\text{Serving}(e) \wedge \text{Server}(e, \text{United}) \wedge \text{Served}(e, \text{Houston}))$$


# Rule-to-rule Model

- **Lambda Calculus and the Rule-to-Rule Hypothesis**
  - $\lambda$ -expressions can be attached to grammar rules
  - used to compute meaning representations from syntactic trees based on the principle of compositionality
  - Go up the tree, using reduction (function application) to compute meanings at non-terminal nodes

# Semantic Attachments

- Basic Structure:

$$A \rightarrow a_1, \dots, a_n \underbrace{\{f(a_j.\text{sem}, \dots a_k.\text{sem})\}}_{\text{Semantic Function}}$$

- In NLTK syntax (more later):

$A \rightarrow a_1 \dots a_n [ \text{SEM}=< f( ?a_j \cdot \text{sem} \dots ) > ]$

# Attachments as SQL!

NLTK book, ch. 10

```
>>> nltk.data.show_cfg('grammars/book_grammars/sql0.fcfg')
% start S
S[SEM=(?np + WHERE + ?vp)] -> NP[SEM=?np] VP[SEM=?vp]
VP[SEM=(?v + ?pp)] -> IV[SEM=?v] PP[SEM=?pp]
VP[SEM=(?v + ?ap)] -> IV[SEM=?v] AP[SEM=?ap]
NP[SEM=(?det + ?n)] -> Det[SEM=?det] N[SEM=?n]
PP[SEM=(?p + ?np)] -> P[SEM=?p] NP[SEM=?np]
AP[SEM=?pp] -> A[SEM=?a] PP[SEM=?pp]
NP[SEM='Country="greece"'] -> 'Greece'
NP[SEM='Country="china"'] -> 'China'
Det[SEM='SELECT'] -> 'Which' | 'What'
N[SEM='City FROM city_table'] -> 'cities'
IV[SEM=''] -> 'are'
A[SEM=''] -> 'located'
P[SEM=''] -> 'in'
```

# Attachments as SQL!

NLTK book, ch. 10

```
>>> nltk.data.show_cfg('grammars/book_grammars/sql0.fcfg')
% start S
S[SEM=(?np + WHERE + ?vp)] -> NP[SEM=?np] VP[SEM=?vp]
VP[SEM=(?v + ?pp)] -> IV[SEM=?v] PP[SEM=?pp]
VP[SEM=(?v + ?ap)] -> IV[SEM=?v] AP[SEM=?ap]
NP[SEM=(?det + ?n)] -> Det[SEM=?det] N[SEM=?n]
PP[SEM=(?p + ?np)] -> P[SEM=?p] NP[SEM=?np]
AP[SEM=?pp] -> A[SEM=?a] PP[SEM=?pp]
NP[SEM='Country="greece"'] -> 'Greece'
NP[SEM='Country="china"'] -> 'China'
Det[SEM='SELECT'] -> 'Which' | 'What'
N[SEM='City FROM city_table'] -> 'cities'
IV[SEM=''] -> 'are'
A[SEM=''] -> 'located'
P[SEM=''] -> 'in'
```

*'What cities are located in China'*

parses[0]: SELECT City FROM city\_table WHERE Country="china"

# Semantic Attachments: Options

- Why not use SQL? Python?
  - Arbitrary power but hard to map to logical form
  - No obvious relation between syntactic, semantic elements
- Why Lambda Calculus?
  - First Order Predicate Calculus (FOPC) + function application is highly expressive, integrates well with syntax
  - Can extend our existing feature-based model, using unification
  - Can ‘translate’ FOL to target / task / downstream language (e.g. SQL)

# Semantic Analysis Approach

- Semantic attachments:
  - Each CFG production gets semantic attachment
- Semantics of a phrase is function of combining the children
  - Complex functions need to have parameters
  - *Verb* → ‘arrived’
    - Intransitive verb, so has one argument: *subject*
    - ...but we don’t have this available at the preterminal level of the tree!

# Defining Representations

- Proper Nouns
- Intransitive Verbs
- Transitive Verbs
- Quantifiers

# Proper Nouns & Intransitive Verbs

- Our instinct for names is to just use the constant:
- NNP [ SEM=<Khaliil> ] → ‘Khaliil’

# Proper Nouns & Intransitive Verbs

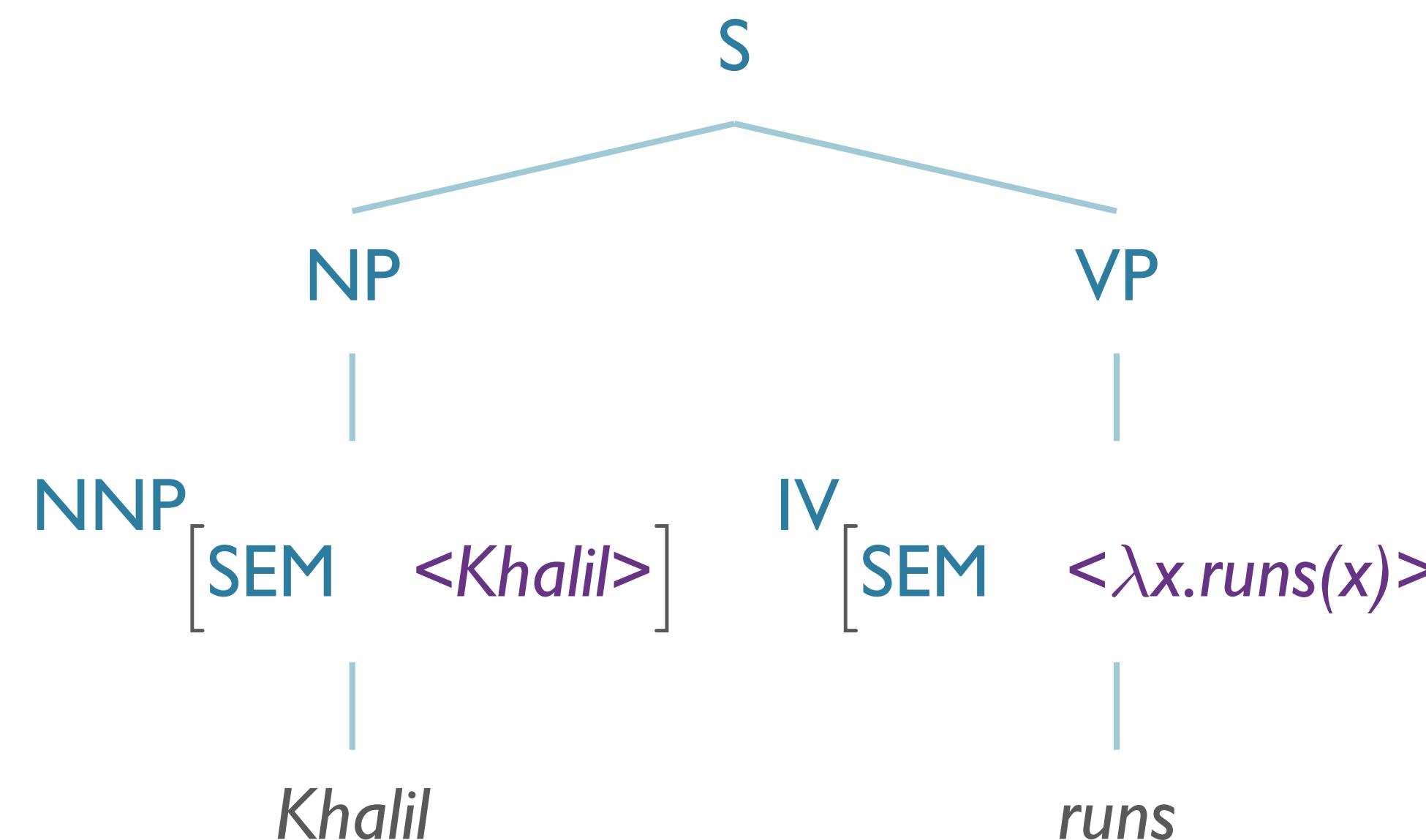
- Our instinct for names is to just use the constant:
- $\text{NNP} [ \text{SEM}=\langle \text{Khalil} \rangle ] \rightarrow ' \text{Khalil}'$
- However, we want to apply our  $\lambda$ -closures left-to-right consistently.

$S [ \text{SEM}=\text{np?} (\text{vp?}) ] \rightarrow \text{NP} [ \text{SEM}=\text{np?} ] \text{ VP} [ \text{SEM}=\text{vp?} ]$

# Proper Nouns & Intransitive Verbs

- Our instinct for names is to just use the constant:
- $\text{NNP} [ \text{SEM}=\langle \text{Khalil} \rangle ] \rightarrow ' \text{Khalil}'$
- However, we want to apply our  $\lambda$ -closures left-to-right consistently.

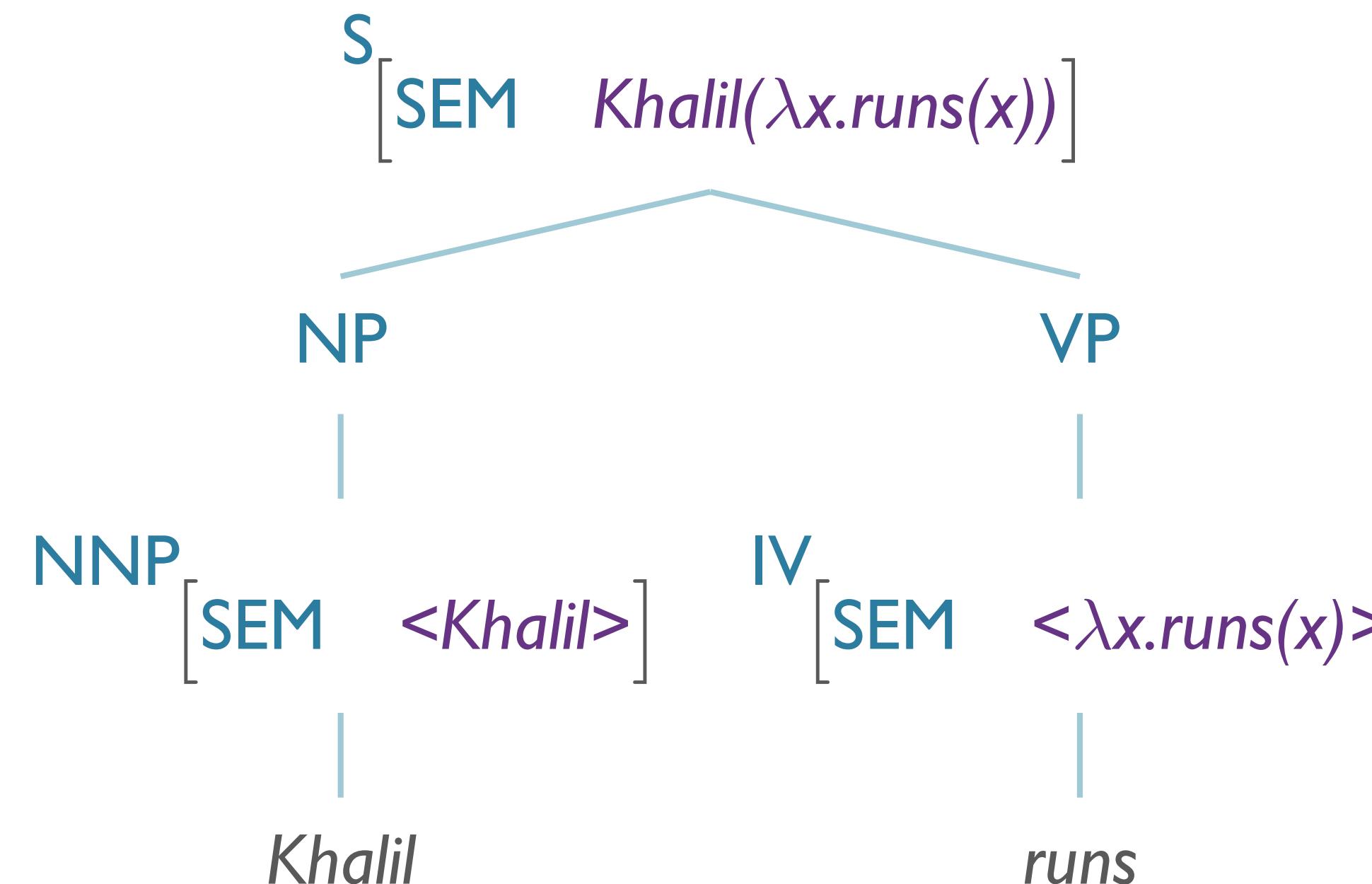
$S [ \text{SEM}=\text{np?} (\text{vp?}) ] \rightarrow \text{NP} [ \text{SEM}=\text{np?} ] \text{ VP} [ \text{SEM}=\text{vp?} ]$



# Proper Nouns & Intransitive Verbs

- Our instinct for names is to just use the constant:
- $\text{NNP} [ \text{SEM}=\langle \text{Khalil} \rangle ] \rightarrow ' \text{Khalil}'$
- However, we want to apply our  $\lambda$ -closures left-to-right consistently.

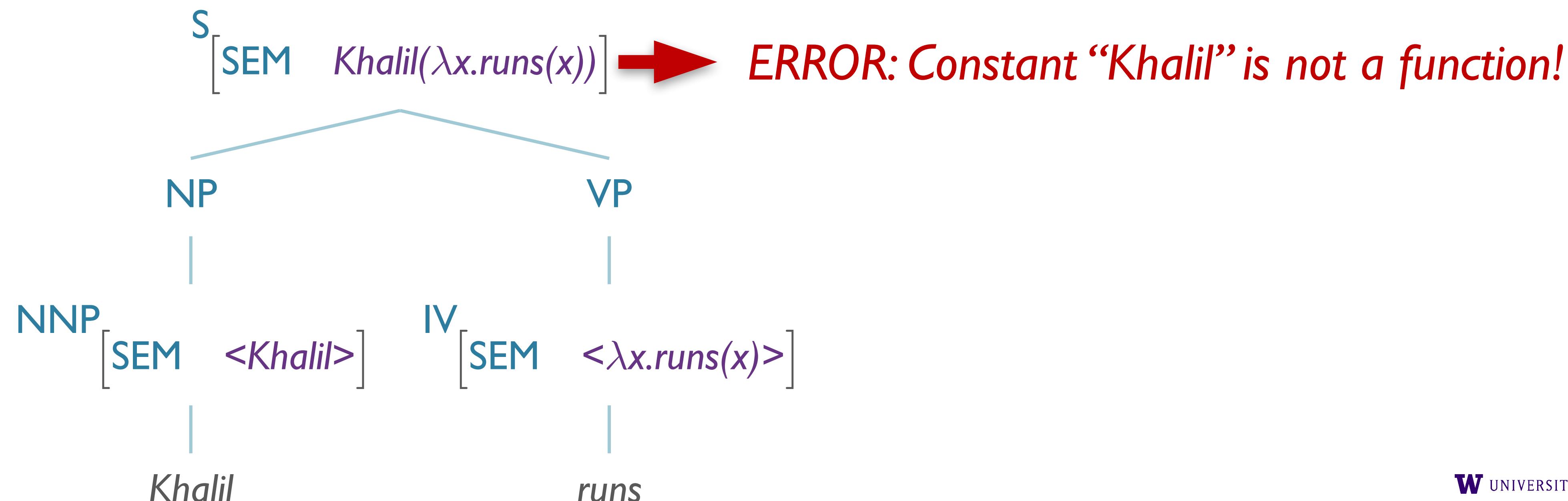
$S [ \text{SEM}=\text{np?} (\text{vp?}) ] \rightarrow \text{NP} [ \text{SEM}=\text{np?} ] \text{ VP} [ \text{SEM}=\text{vp?} ]$



# Proper Nouns & Intransitive Verbs

- Our instinct for names is to just use the constant:
- $\text{NNP} [ \text{SEM}=\langle \text{Khalil} \rangle ] \rightarrow ' \text{Khalil}'$
- However, we want to apply our  $\lambda$ -closures left-to-right consistently.

$S [ \text{SEM}=\text{np?} (\text{vp?}) ] \rightarrow \text{NP} [ \text{SEM}=\text{np?} ] \text{ VP} [ \text{SEM}=\text{vp?} ]$



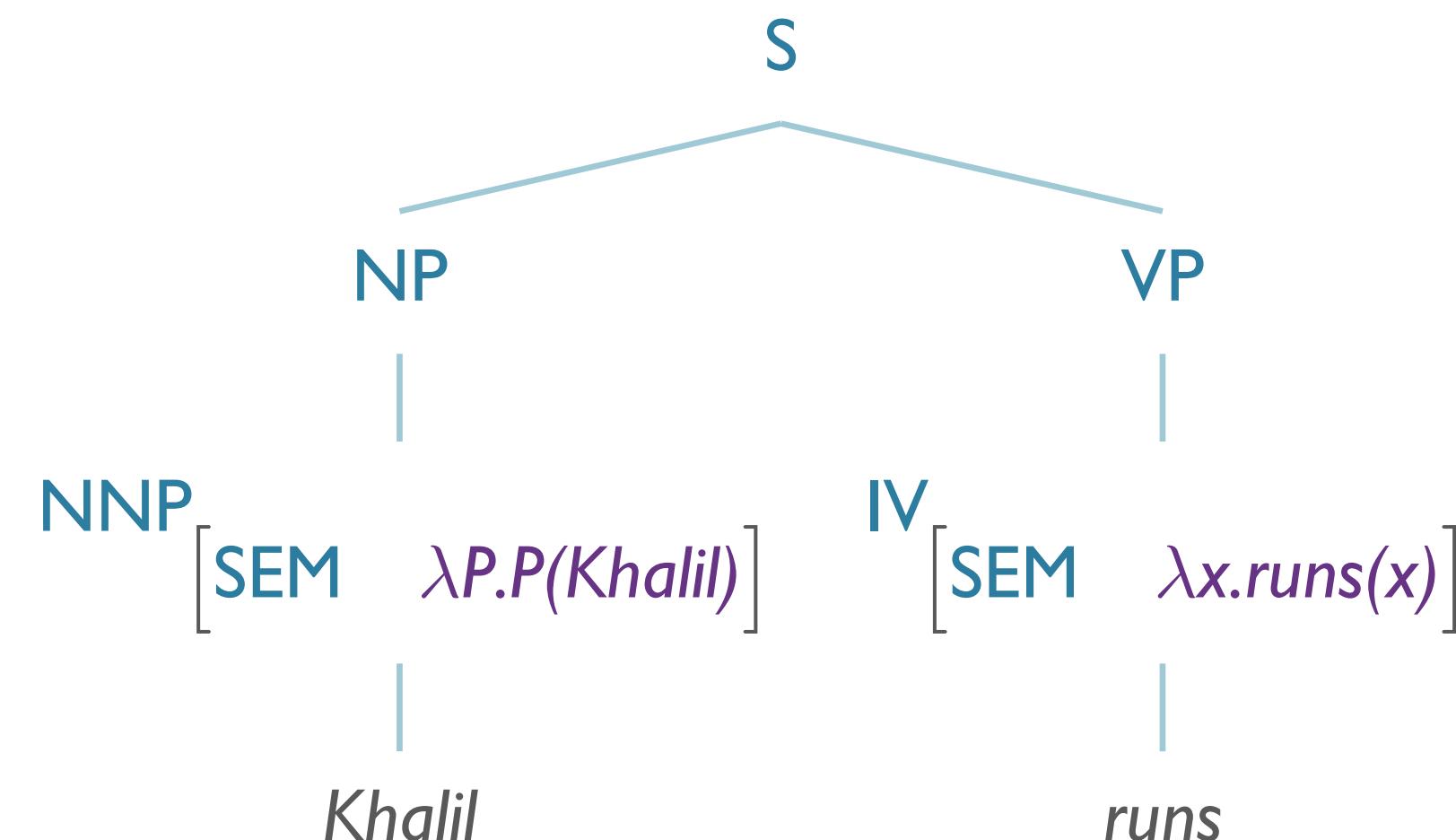
# Proper Nouns & Intransitive Verbs

- Instead, we use a *dummy predicate*:
  - $\lambda Q.Q(Khalil)$
- “Generalizing to the worst case” (cf. Montague; Partee on type-shifting)

# Proper Nouns & Intransitive Verbs

- With the dummy predicate:
- $\text{NNP}[\text{SEM}=<\lambda P.P(\text{Khalil})>] \rightarrow \text{'Khalil'}$

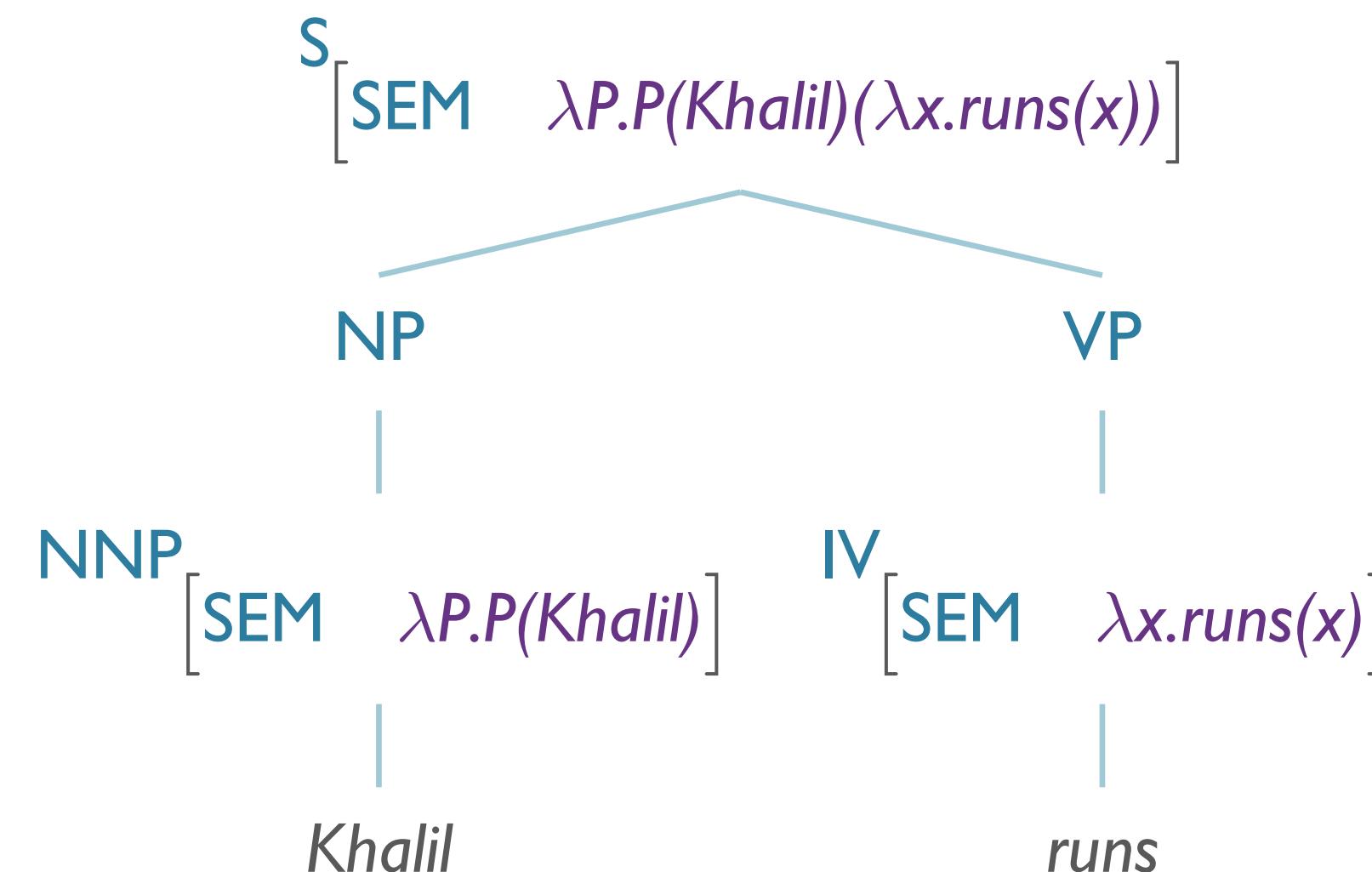
$S[\text{SEM}=\text{np? (vp?)}] \rightarrow \text{NP}[\text{SEM}=\text{np?}] \text{ VP}[\text{SEM}=\text{vp?}]$



# Proper Nouns & Intransitive Verbs

- With the dummy predicate:
- $\text{NNP}[\text{SEM}=<\lambda P.P(\text{Khalil})>] \rightarrow \text{'Khalil'}$

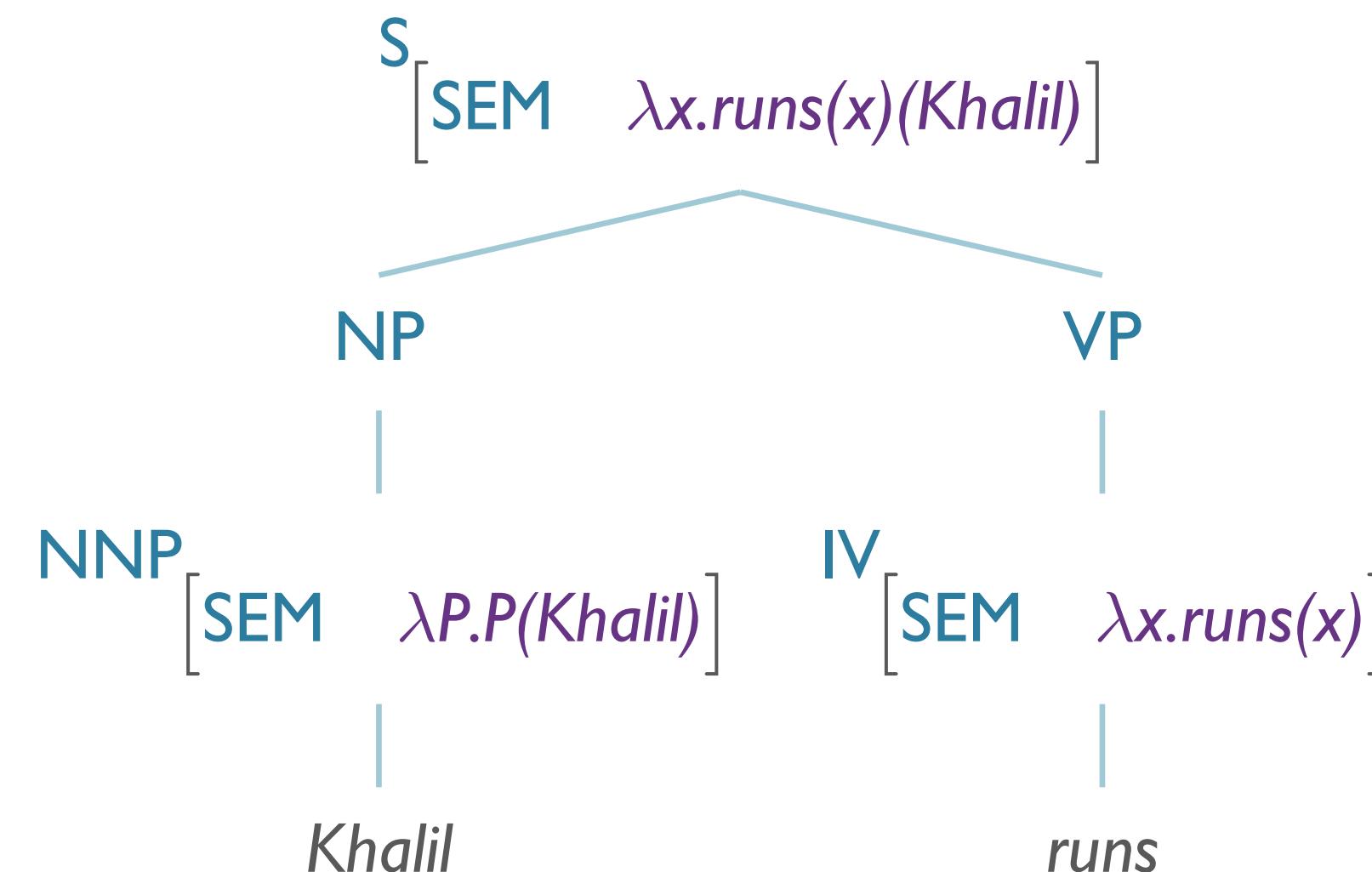
$S[\text{SEM}=\text{np?}(\text{vp?})] \rightarrow \text{NP}[\text{SEM}=\text{np?}] \text{ VP}[\text{SEM}=\text{vp?}]$



# Proper Nouns & Intransitive Verbs

- With the dummy predicate:
- $\text{NNP}[\text{SEM}=<\lambda P.P(\text{Khalil})>] \rightarrow \text{'Khalil'}$

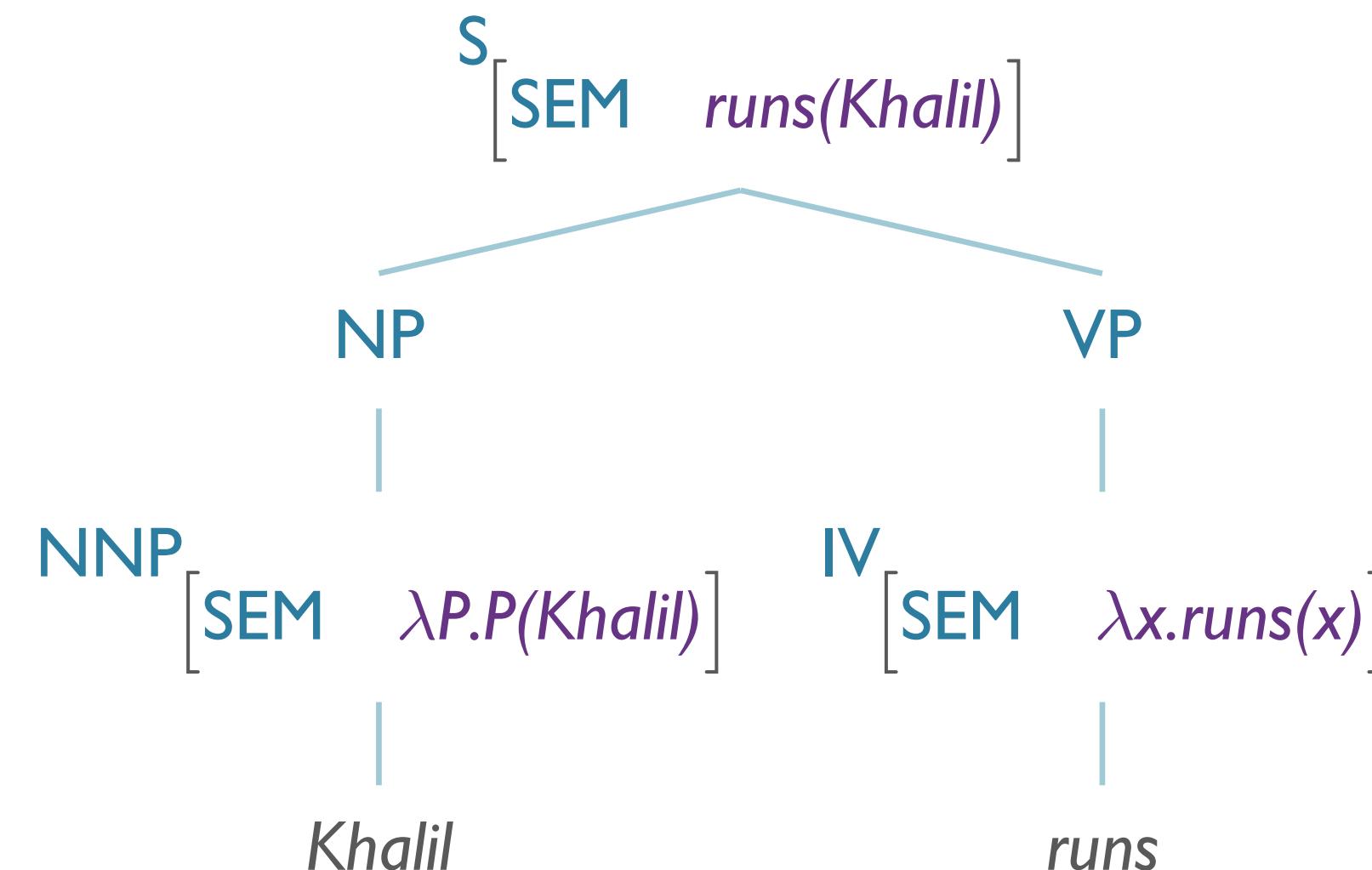
$S[\text{SEM}=\text{np?}(\text{vp?})] \rightarrow \text{NP}[\text{SEM}=\text{np?}] \text{ VP}[\text{SEM}=\text{vp?}]$



# Proper Nouns & Intransitive Verbs

- With the dummy predicate:
- $\text{NNP}[\text{SEM}=<\lambda P.P(\text{Khalil})>] \rightarrow \text{'Khalil'}$

$S[\text{SEM}=\text{np? (vp?)}] \rightarrow \text{NP}[\text{SEM}=\text{np?}] \text{ VP}[\text{SEM}=\text{vp?}]$



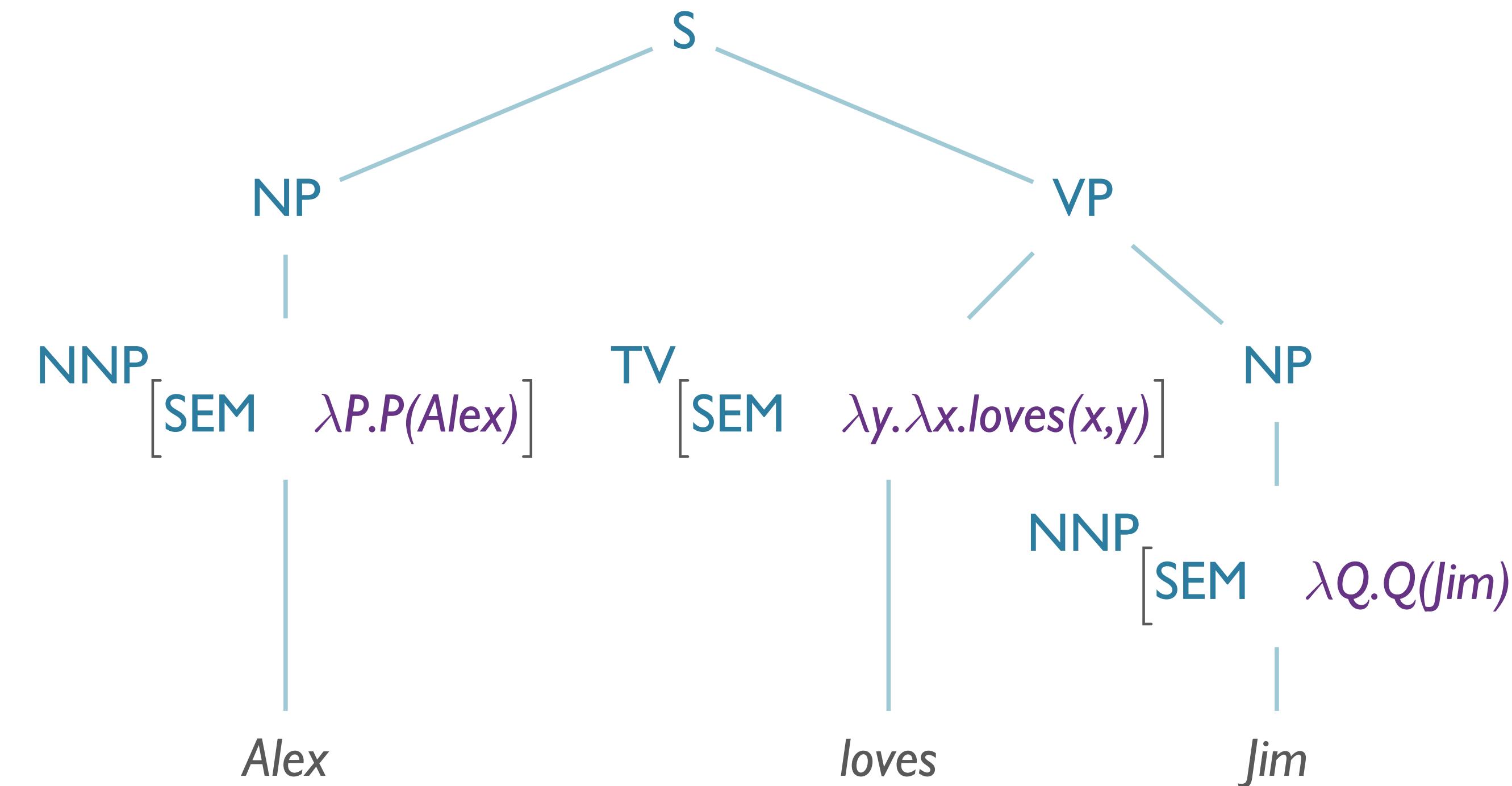
# Transitive Verbs

# Transitive Verbs

- So, if we want to say “*Alex loves Jim*” we would want  $\lambda y. \lambda x. \text{loves}(x, y)$
- ...but going in linear order, we have one arg to the left and one to the right.
- So, instead:
  - $\lambda x \ y. x(\lambda x. \text{loves}(x, y))$

# Transitive Verbs

- So, if we want to say “*Alex loves Jim*” we would want  $\lambda y. \lambda x. \text{loves}(x, y)$
- ...but going in linear order, we have one arg to the left and one to the right.



# Transitive Verbs

- TV(NP):
  - $\lambda y. \lambda x. \text{loves}(x, y) (\lambda Q. Q(\text{Alex}))$

# Transitive Verbs

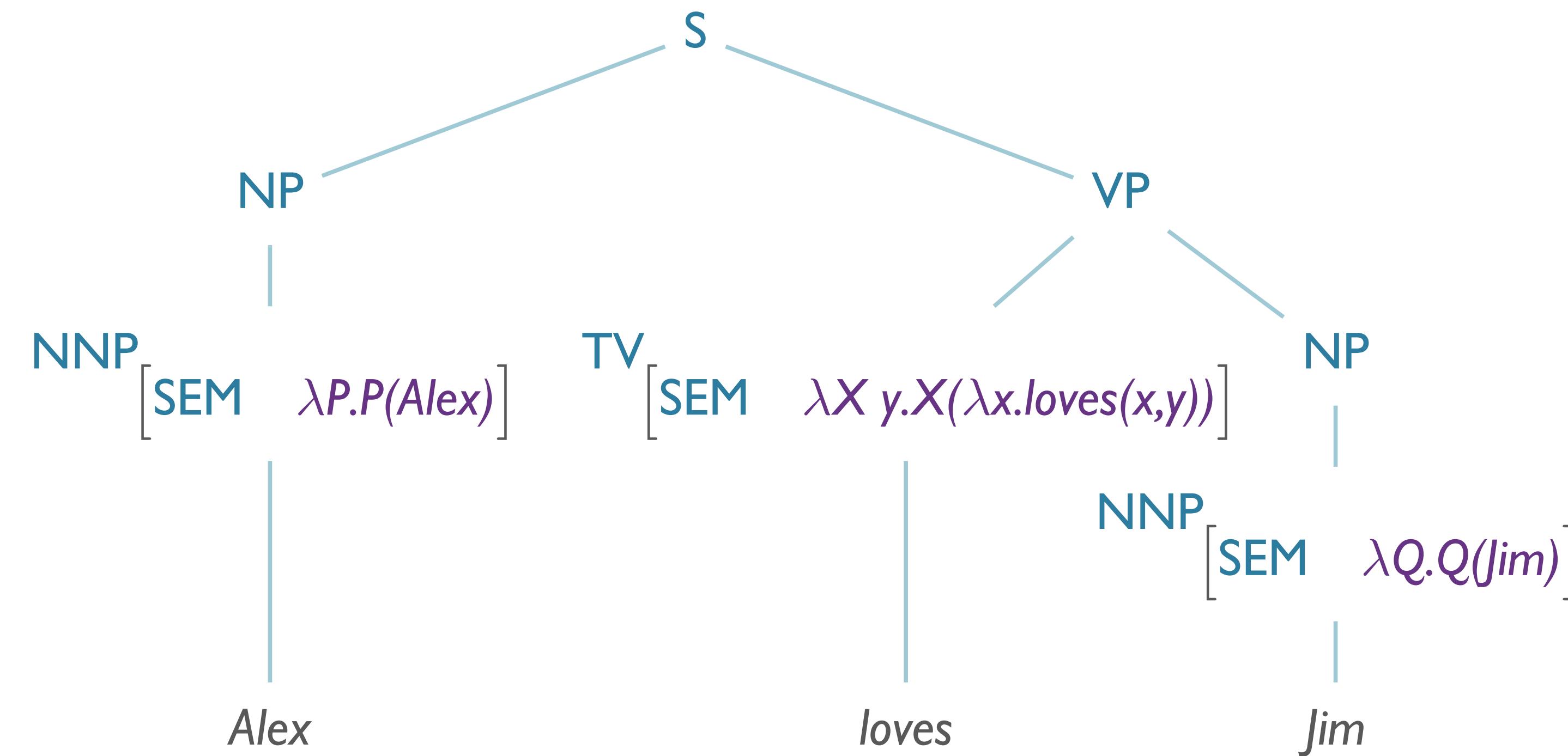
- TV(NP):
  - $\lambda y. \lambda x. \text{loves}(x, y) (\lambda Q. Q(\text{Alex}))$
  - $\lambda x. \text{loves}(x, \lambda Q. Q(\text{Alex}))$

# Transitive Verbs

- TV(NP):
  - $\lambda y. \lambda x. \text{loves}(x, y) (\lambda Q. Q(\text{Alex}))$
  - $\lambda x. \text{loves}(x, \lambda Q. Q(\text{Alex}))$
  - → **Error!** We can't reduce Alex.

# Transitive Verbs

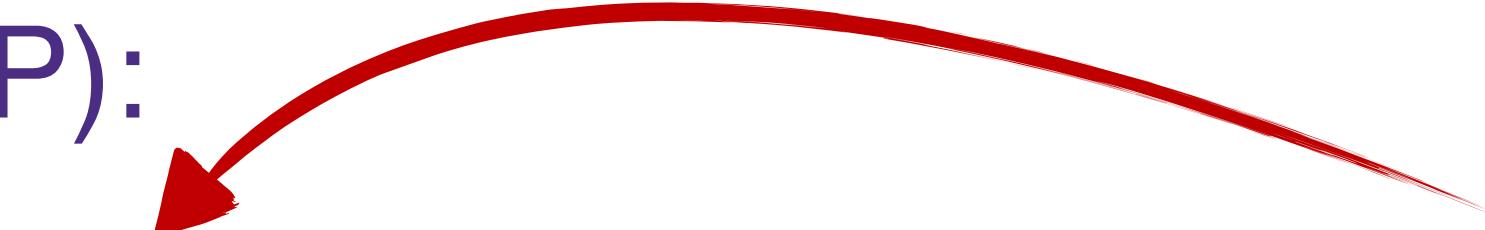
- Instead:  $\lambda x \ y. x(\lambda x. \text{loves}(x, y))$



# Transitive Verbs

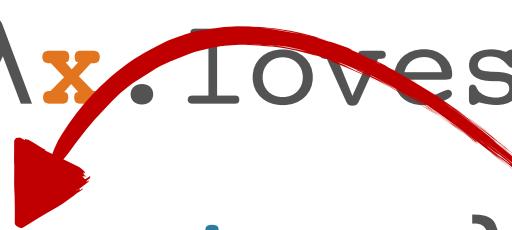
- $\text{TV}(\text{NP})$ :
- $\lambda \text{x} \text{ } \text{y} \text{.} \text{x} (\lambda \text{x} \text{.} \text{loves} (\text{x} \text{,} \text{y})) \text{ } (\lambda \text{Q} \text{.} \text{Q} (\text{Jim}))$

# Transitive Verbs

- $\text{TV}(\text{NP})$ :
    - $\lambda \mathbf{x} \ \mathbf{y}. \mathbf{x} (\lambda \mathbf{x}. \text{loves}(\mathbf{x}, \mathbf{y})) (\lambda \mathbf{Q}. \mathbf{Q}(\mathbf{Jim}))$        $\lambda \mathbf{x}$  takes  $(\lambda \mathbf{Q}. \mathbf{Q}(\mathbf{Jim}))$
    - $\lambda \mathbf{y}. (\lambda \mathbf{Q}. \mathbf{Q}(\mathbf{Jim})) (\lambda \mathbf{x}. \text{loves}(\mathbf{x}, \mathbf{y}))$
- 

# Transitive Verbs

- TV(NP):

- $\lambda x \ y. x(\lambda x. \text{loves}(x, y)) (\lambda Q. Q(\text{Jim}))$        $\lambda x \text{ takes } (\lambda Q. Q(\text{Jim}))$   

- $\lambda y. (\lambda Q. Q(\text{Jim})) (\lambda x. \text{loves}(x, y))$        $\lambda Q \text{ takes } (\lambda x. \text{loves}(x, y))$
- $\lambda y. (\lambda x. \text{loves}(x, y)) (\text{Jim})$

# Transitive Verbs

- TV(NP):

- $\lambda \mathbf{x} \ \mathbf{y}. \mathbf{x}(\lambda \mathbf{x}. \text{loves}(\mathbf{x}, \mathbf{y})) (\lambda \mathbf{Q}. \mathbf{Q}(\mathbf{Jim}))$        $\lambda \mathbf{x}$  takes  $(\lambda \mathbf{Q}. \mathbf{Q}(\mathbf{Jim}))$
- $\lambda \mathbf{y}. (\lambda \mathbf{Q}. \mathbf{Q}(\mathbf{Jim})) (\lambda \mathbf{x}. \text{loves}(\mathbf{x}, \mathbf{y}))$        $\lambda \mathbf{Q}$  takes  $(\lambda \mathbf{x}. \text{loves}(\mathbf{x}, \mathbf{y}))$
- $\lambda \mathbf{y}. (\lambda \mathbf{x}. \text{loves}(\mathbf{x}, \mathbf{y})) (\mathbf{Jim})$        $\lambda \mathbf{x}$  takes  $(\mathbf{Jim})$
- $\lambda \mathbf{y}. (\text{loves}(\mathbf{Jim}, \mathbf{y}))$

# Transitive Verbs

- TV(NP):

- $\lambda \mathbf{x} \ \mathbf{y}. \mathbf{x}(\lambda \mathbf{x}. \text{loves}(\mathbf{x}, \mathbf{y})) \ (\lambda \mathbf{Q}. \mathbf{Q}(\mathbf{Jim}))$        $\lambda \mathbf{x}$  takes  $(\lambda \mathbf{Q}. \mathbf{Q}(\mathbf{Jim}))$
- $\lambda \mathbf{y}. (\lambda \mathbf{Q}. \mathbf{Q}(\mathbf{Jim})) (\lambda \mathbf{x}. \text{loves}(\mathbf{x}, \mathbf{y}))$        $\lambda \mathbf{Q}$  takes  $(\lambda \mathbf{x}. \text{loves}(\mathbf{x}, \mathbf{y}))$
- $\lambda \mathbf{y}. (\lambda \mathbf{x}. \text{loves}(\mathbf{x}, \mathbf{y}) (\mathbf{Jim}))$        $\lambda \mathbf{x}$  takes  $(\mathbf{Jim})$
- $\lambda \mathbf{y}. (\text{loves}(\mathbf{Jim}, \mathbf{y}))$

- NP(VP):

- $\lambda \mathbf{P}. \mathbf{P}(\mathbf{Alex}) (\lambda \mathbf{y}. (\text{loves}(\mathbf{Jim}, \mathbf{y})) )$

# Transitive Verbs

- TV(NP):

- $\lambda \mathbf{x} \ \mathbf{y}. \mathbf{x}(\lambda \mathbf{x}. \text{loves}(\mathbf{x}, \mathbf{y})) \ (\lambda \mathbf{Q}. \mathbf{Q}(\mathbf{Jim}))$        $\lambda \mathbf{x}$  takes  $(\lambda \mathbf{Q}. \mathbf{Q}(\mathbf{Jim}))$
- $\lambda \mathbf{y}. (\lambda \mathbf{Q}. \mathbf{Q}(\mathbf{Jim})) (\lambda \mathbf{x}. \text{loves}(\mathbf{x}, \mathbf{y}))$        $\lambda \mathbf{Q}$  takes  $(\lambda \mathbf{x}. \text{loves}(\mathbf{x}, \mathbf{y}))$
- $\lambda \mathbf{y}. (\lambda \mathbf{x}. \text{loves}(\mathbf{x}, \mathbf{y}) (\mathbf{Jim}))$        $\lambda \mathbf{x}$  takes  $(\mathbf{Jim})$
- $\lambda \mathbf{y}. (\text{loves}(\mathbf{Jim}, \mathbf{y}))$

- NP(VP):

- $\lambda \mathbf{P}. \mathbf{P}(\mathbf{Alex}) (\lambda \mathbf{y}. (\text{loves}(\mathbf{Jim}, \mathbf{y}))$        $\lambda \mathbf{P}$  takes  $(\lambda \mathbf{y}. (\text{loves}(\mathbf{Jim}, \mathbf{y}))$
- $\lambda \mathbf{y}. (\text{loves}(\mathbf{Jim}, \mathbf{y}) (\mathbf{Alex}))$

# Transitive Verbs

- TV(NP):

- $\lambda \mathbf{x} \ \mathbf{y}. \mathbf{x}(\lambda \mathbf{x}. \text{loves}(\mathbf{x}, \mathbf{y})) \ (\lambda \mathbf{Q}. \mathbf{Q}(\mathbf{Jim}))$        $\lambda \mathbf{x}$  takes  $(\lambda \mathbf{Q}. \mathbf{Q}(\mathbf{Jim}))$
- $\lambda \mathbf{y}. (\lambda \mathbf{Q}. \mathbf{Q}(\mathbf{Jim})) (\lambda \mathbf{x}. \text{loves}(\mathbf{x}, \mathbf{y}))$        $\lambda \mathbf{Q}$  takes  $(\lambda \mathbf{x}. \text{loves}(\mathbf{x}, \mathbf{y}))$
- $\lambda \mathbf{y}. (\lambda \mathbf{x}. \text{loves}(\mathbf{x}, \mathbf{y}) (\mathbf{Jim}))$        $\lambda \mathbf{x}$  takes  $(\mathbf{Jim})$
- $\lambda \mathbf{y}. (\text{loves}(\mathbf{Jim}, \mathbf{y}))$

- NP(VP):

- ~~$\lambda \mathbf{P}. \mathbf{P}(\mathbf{Alex}) (\lambda \mathbf{y}. (\text{loves}(\mathbf{Jim}, \mathbf{y})) )$~~
- $\lambda \mathbf{y}. (\text{loves}(\mathbf{Jim}, \mathbf{y}) (\mathbf{Alex}))$        $\lambda \mathbf{P}$  takes  $(\lambda \mathbf{y}. (\text{loves}(\mathbf{Jim}, \mathbf{y})) )$
- $\text{loves}(\mathbf{Jim}, \mathbf{Alex})$        $\lambda \mathbf{y}$  takes  $(\mathbf{Alex})$

# Converting to an Event

- “y loves x,” Originally:
  - $\lambda \text{y} . \lambda \text{x} . \underline{\text{loves}(\text{x}, \text{y})}$

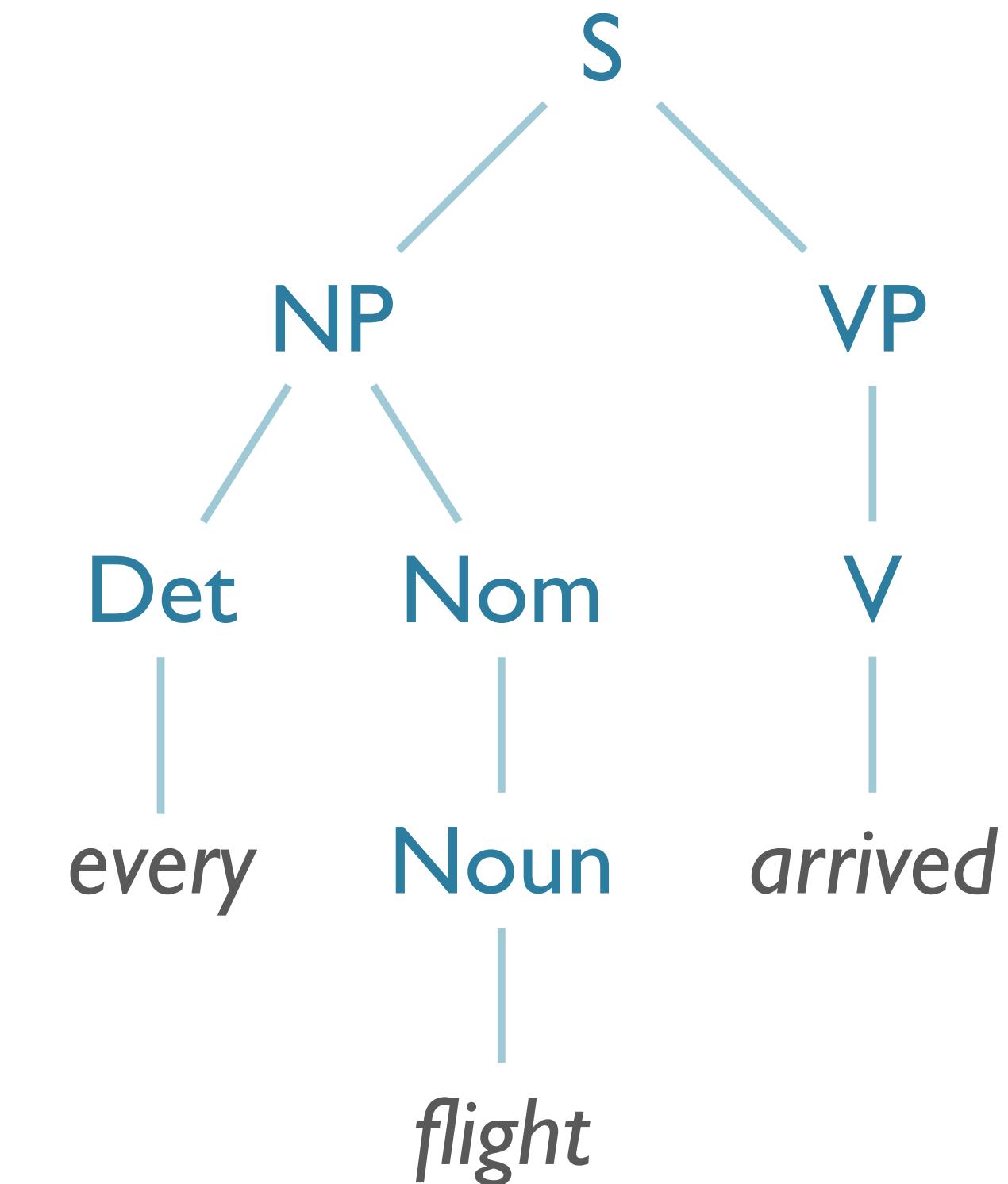
# Converting to an Event

- “y loves x,” Originally:
  - $\lambda \text{x} \text{ } \text{y} \text{.} \text{x} (\lambda \text{x} \text{.} \text{loves} (\text{x} \text{,} \text{y}))$
- as a Neo-Davidsonian event:
  - $\lambda \text{x} \text{ } \text{y} \text{.} \text{x} (\lambda \text{x} \text{.} \exists \text{e} \text{ } \text{love} (\text{e}) \wedge \text{lover} (\text{e} \text{,} \text{y}) \wedge \text{loved} (\text{e} \text{,} \text{x}))$

# Quantifiers & Scope

# Semantic Analysis Example

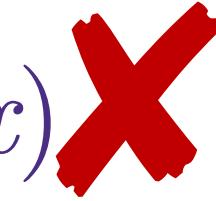
- Basic model
  - Neo-Davidsonian event-style model
  - Complex quantification
- Example: *Every flight arrived*


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

# “*Every flight arrived*”

- First intuitive approach:
- Every flight =  $\forall x \text{ Flight}(x)$

# “*Every flight arrived*”

- First intuitive approach:
  - Every flight =  $\forall x \text{ Flight}(x)$  
  - “Everything is a flight”

# “*Every flight arrived*”

- First intuitive approach:
  - Every flight =  $\forall x \text{ Flight}(x)$  
  - “Everything is a flight”
- Instead, we want:
  - $\forall x \text{ Flight}(x) \Rightarrow Q(x)$

# “*Every flight arrived*”

- First intuitive approach:
  - Every flight =  $\forall x \text{ Flight}(x)$  
  - “Everything is a flight”
- Instead, we want:
  - $\forall x \text{ Flight}(x) \Rightarrow Q(x)$
  - “if a thing is a flight, then it is  $Q$ ”

# “*Every flight arrived*”

- First intuitive approach:
  - Every flight =  $\forall x \text{ Flight}(x)$  
  - “Everything is a flight”
- Instead, we want:
  - $\forall x \text{ Flight}(x) \Rightarrow Q(x)$
  - “if a thing is a flight, then it is  $Q$ ”
  - Since  $Q$  isn’t available yet... Dummy predicate!

# “Every flight arrived”

- First intuitive approach:
  - Every flight =  $\forall x \text{Flight}(x)$  
  - “Everything is a flight”
- Instead, we want:
  - $\forall x \text{Flight}(x) \Rightarrow Q(x)$ 
    - “if a thing is a flight, then it is  $Q$ ”
  - Since  $Q$  isn’t available yet... Dummy predicate!
    - $\lambda Q. \forall x \text{Flight}(x) \Rightarrow Q(x)$

**“Every flight arrived”**

“*Every flight arrived*”

- “*Every flight*” is:

# “*Every* flight arrived”

- “*Every* flight” is:
- $\lambda Q. \forall x \text{Flight}(x) \Rightarrow Q(x)$

# “Every flight arrived”

- “Every flight” is:
  - $\lambda Q. \forall x \text{Flight}(x) \Rightarrow Q(x)$
  - ...so what is the representation for “every”?

# “*Every* flight arrived”

- “*Every* flight” is:
  - $\lambda Q. \forall x \text{Flight}(x) \Rightarrow Q(x)$
- ...so what is the representation for “*every*”?
  - $\lambda P. \lambda Q. \forall x \ P(x) \Rightarrow Q(x)$

# “A flight arrived”

- We just need one item for truth value
  - So, start with  $\exists x \dots$
  - $\lambda P. \lambda Q. \exists x \ P(x) \wedge Q(x)$

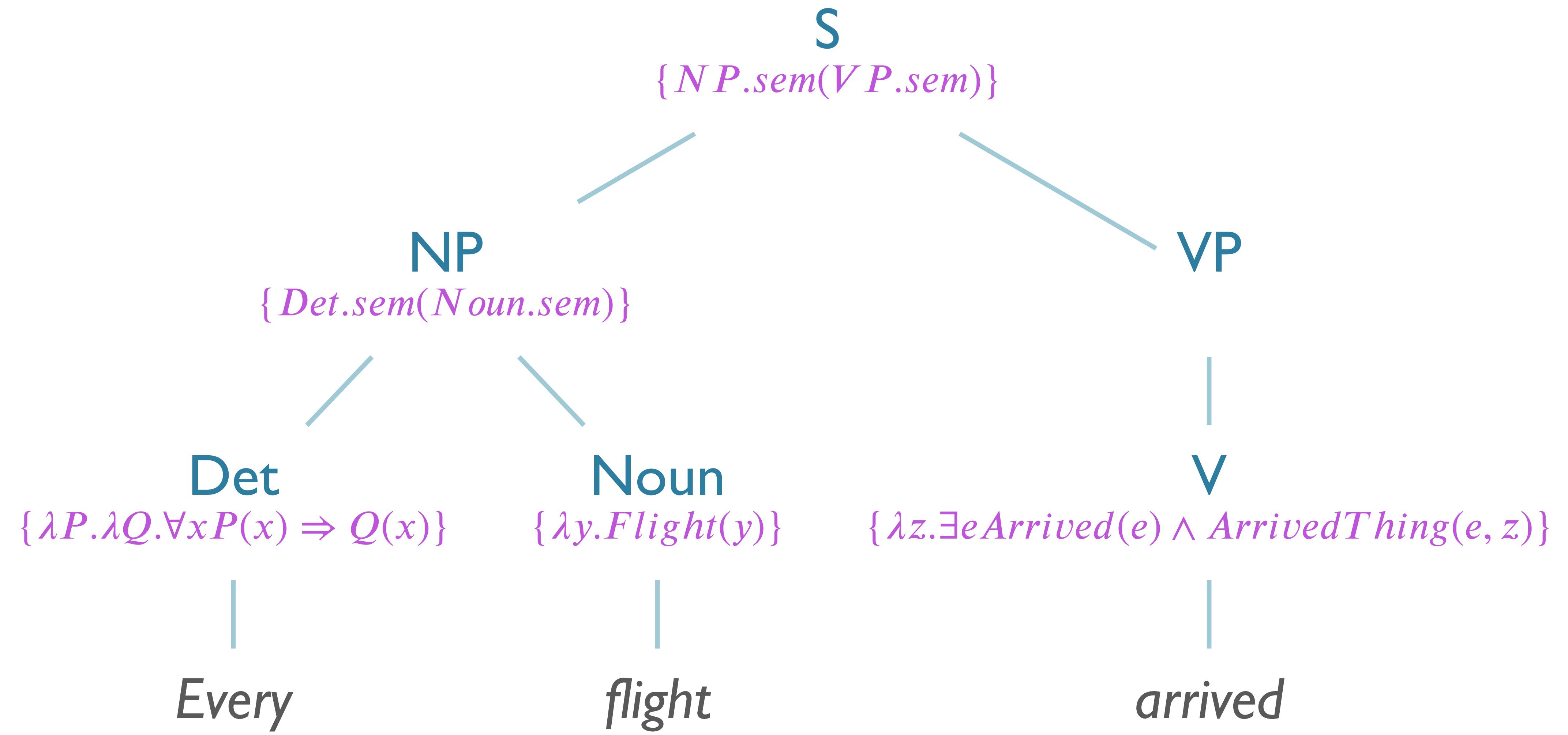
# *“The flight arrived”*

- ...yeah, this turns out to be tricky.
- We'll save it for Wednesday.
- It's not on the homework.

# Creating Attachments

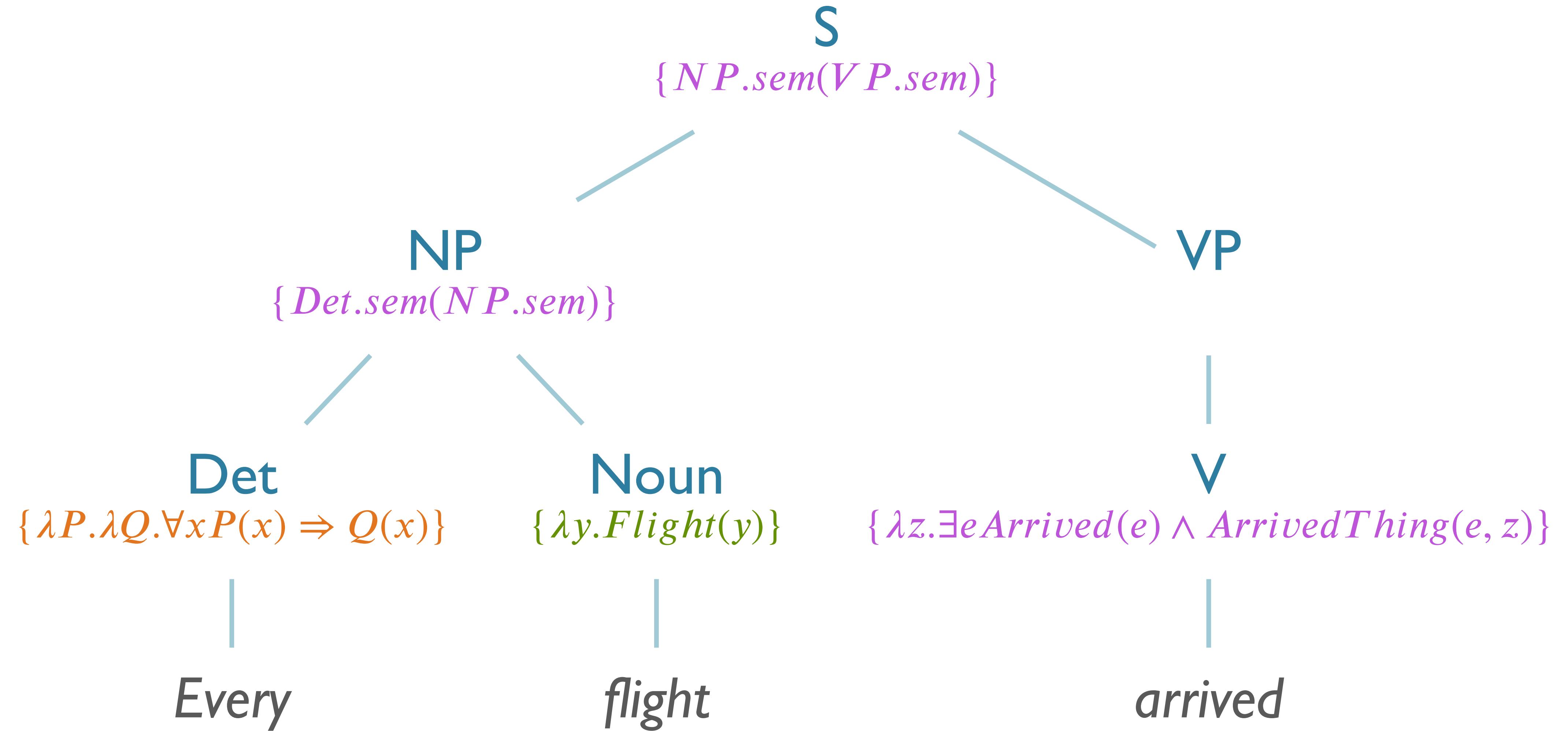
“*Every flight arrived*”

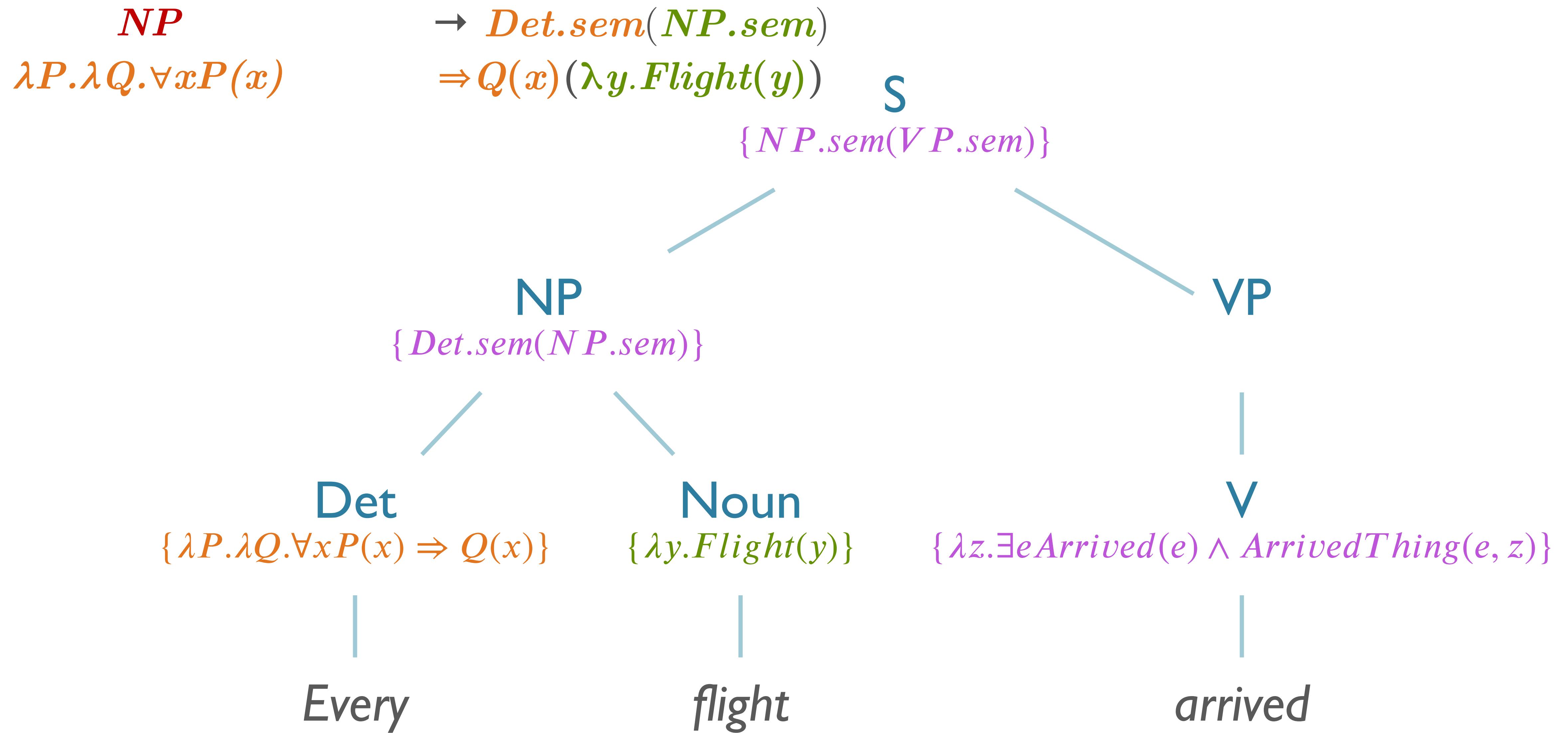
<i>Det</i>	$\rightarrow$ ‘ <i>Every</i> ’	$\{ \lambda P. \lambda Q. \forall x \ P(x) \Rightarrow Q(x) \}$
<i>Noun</i>	$\rightarrow$ ‘ <i>flight</i> ’	$\{ \lambda x. \text{Flight}(x) \}$
<i>Verb</i>	$\rightarrow$ ‘ <i>arrived</i> ’	$\{ \lambda y. \exists e \text{Arrived}(e) \wedge \text{ArrivedThing}(e, y) \}$
<i>VP</i>	$\rightarrow$ <i>Verb</i>	$\{ \text{Verb.sem} \}$
<i>Nom</i>	$\rightarrow$ <i>Noun</i>	$\{ \text{Noun.sem} \}$
<i>S</i>	$\rightarrow$ <i>NP VP</i>	$\{ \text{NP.sem}(\text{VP.sem}) \}$
<i>NP</i>	$\rightarrow$ <i>Det Nom</i>	$\{ \text{Det.sem}(\text{Nom.sem}) \}$

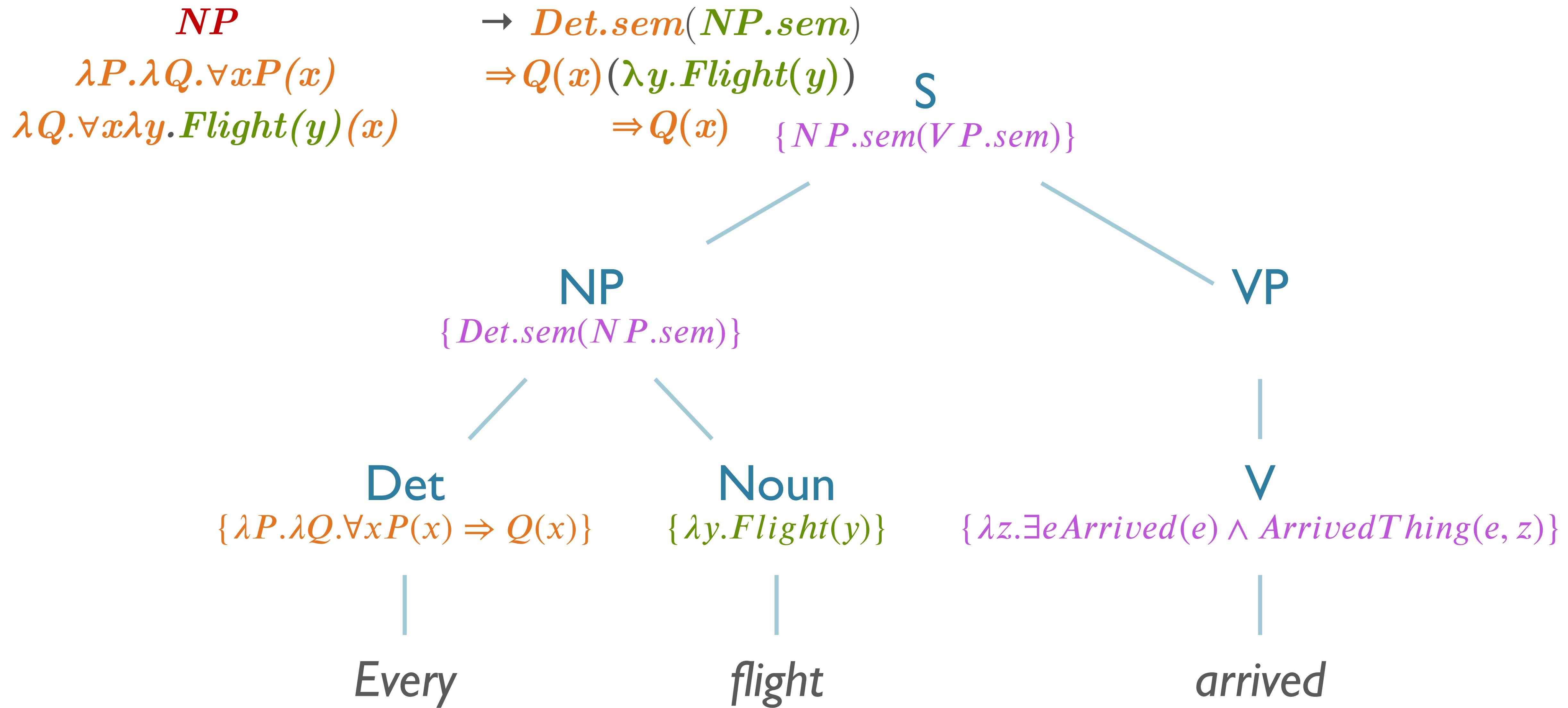


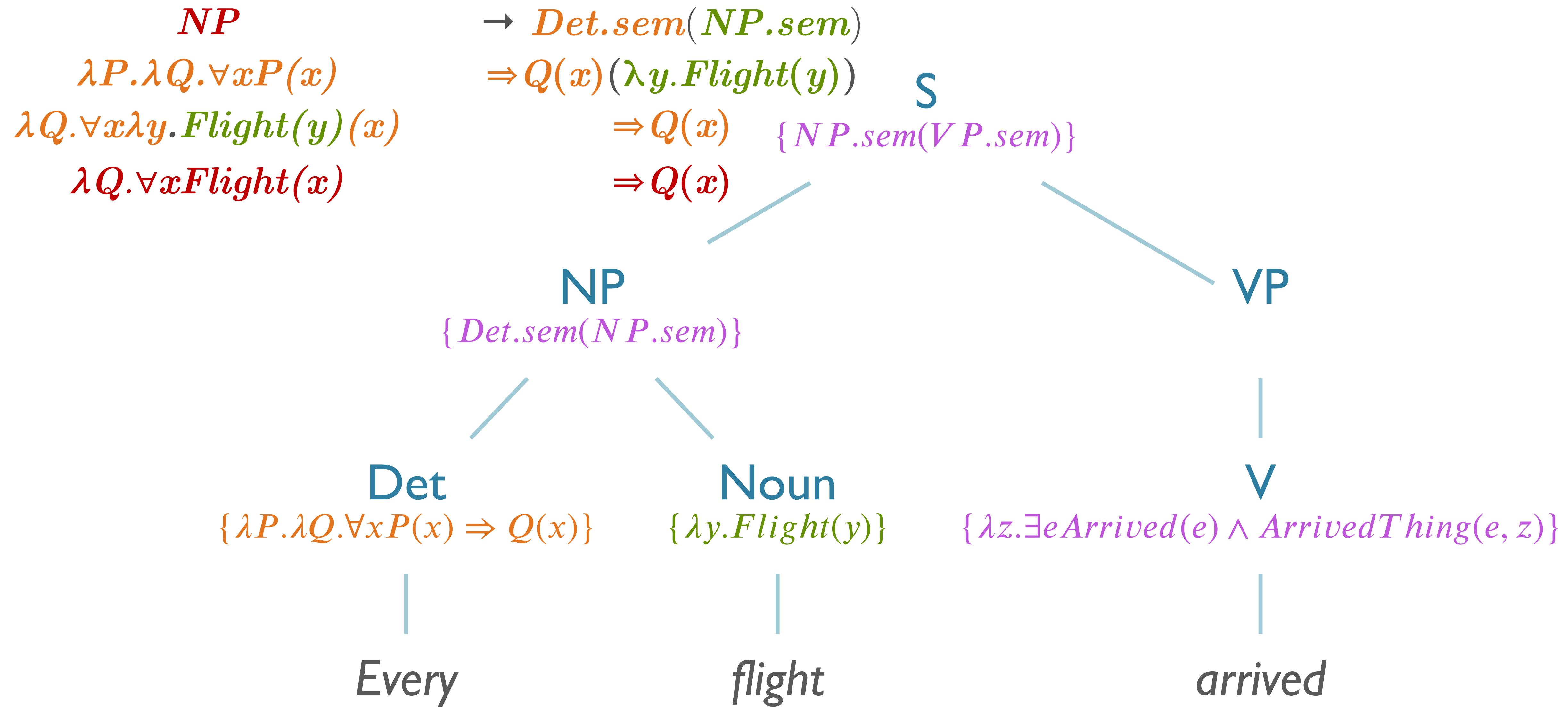
*NP*

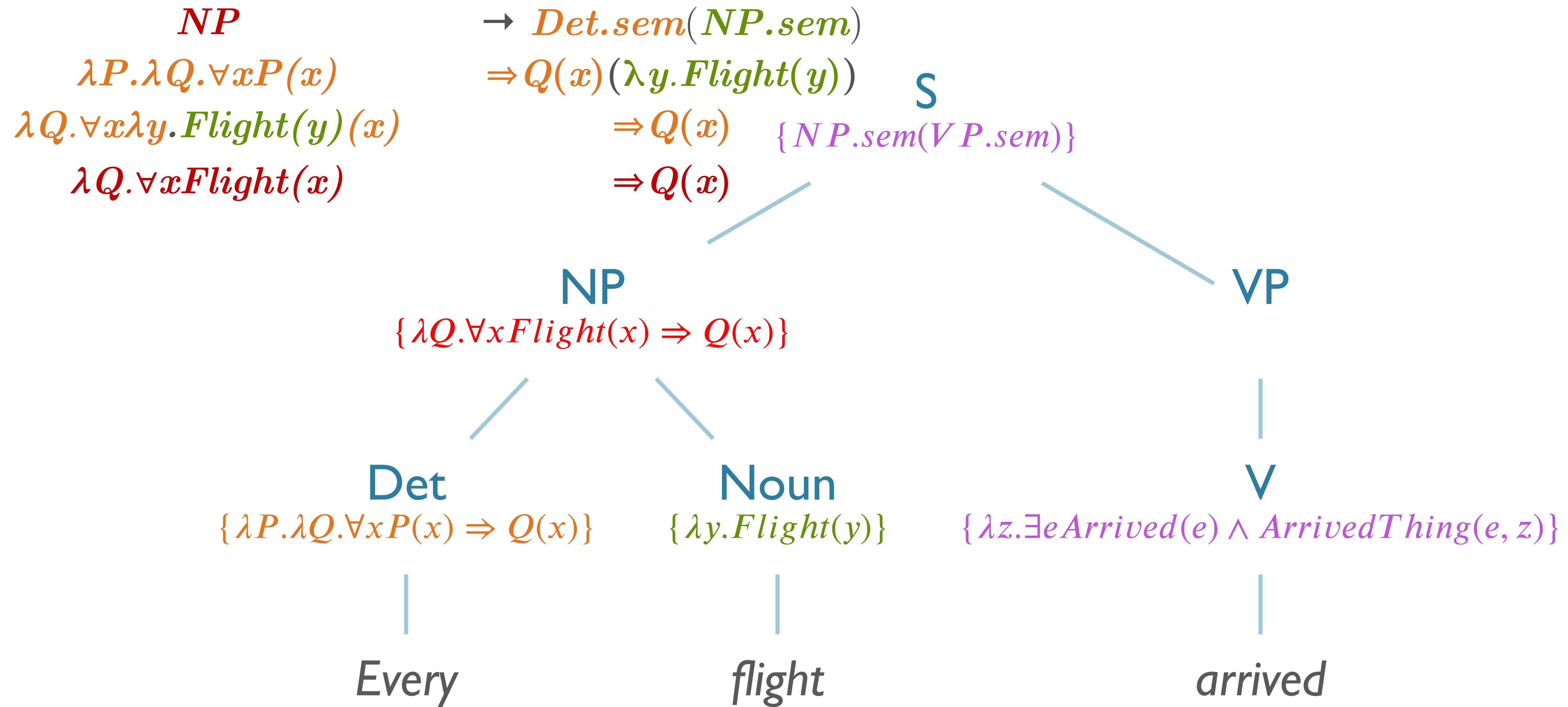
→ *Det.sem(NP.sem)*

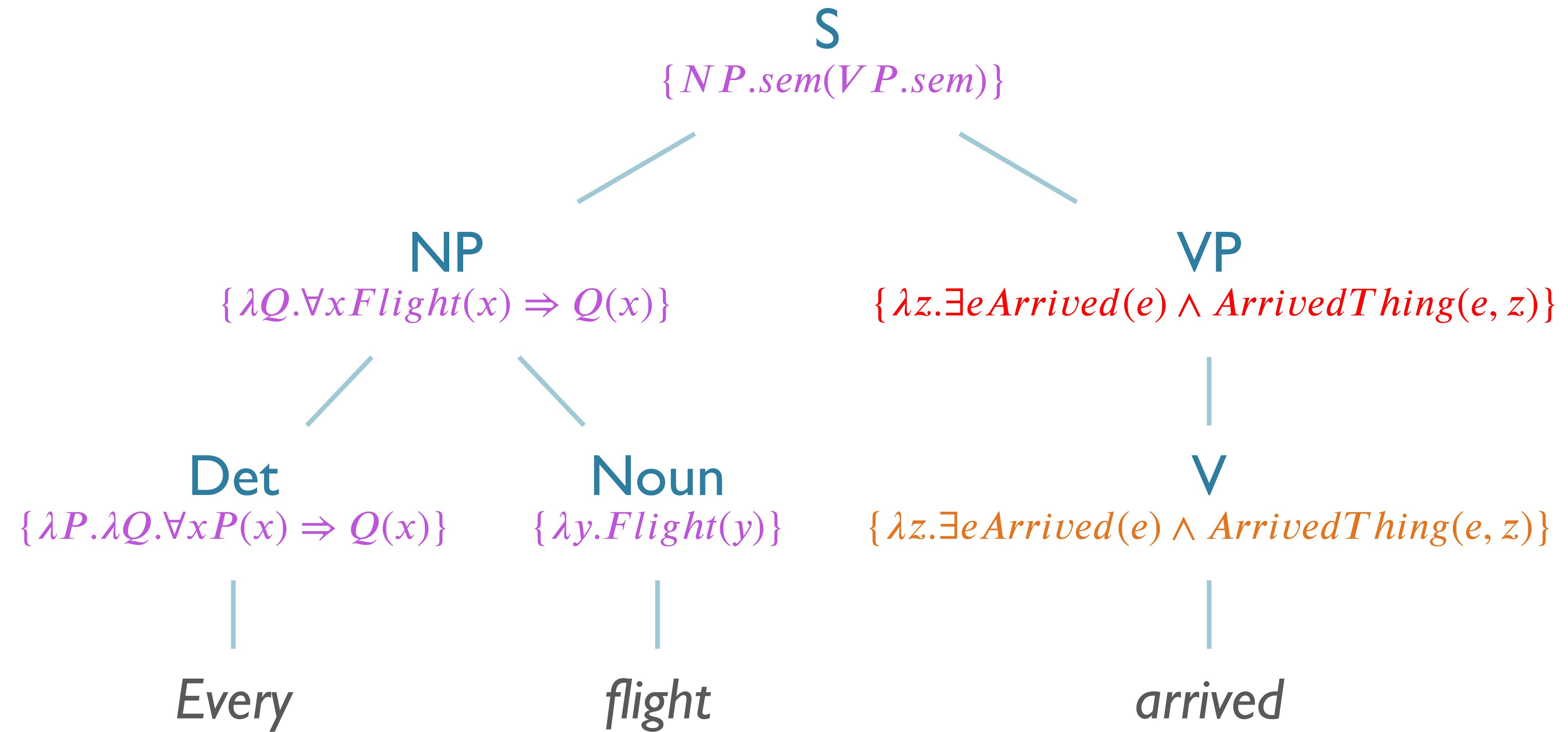


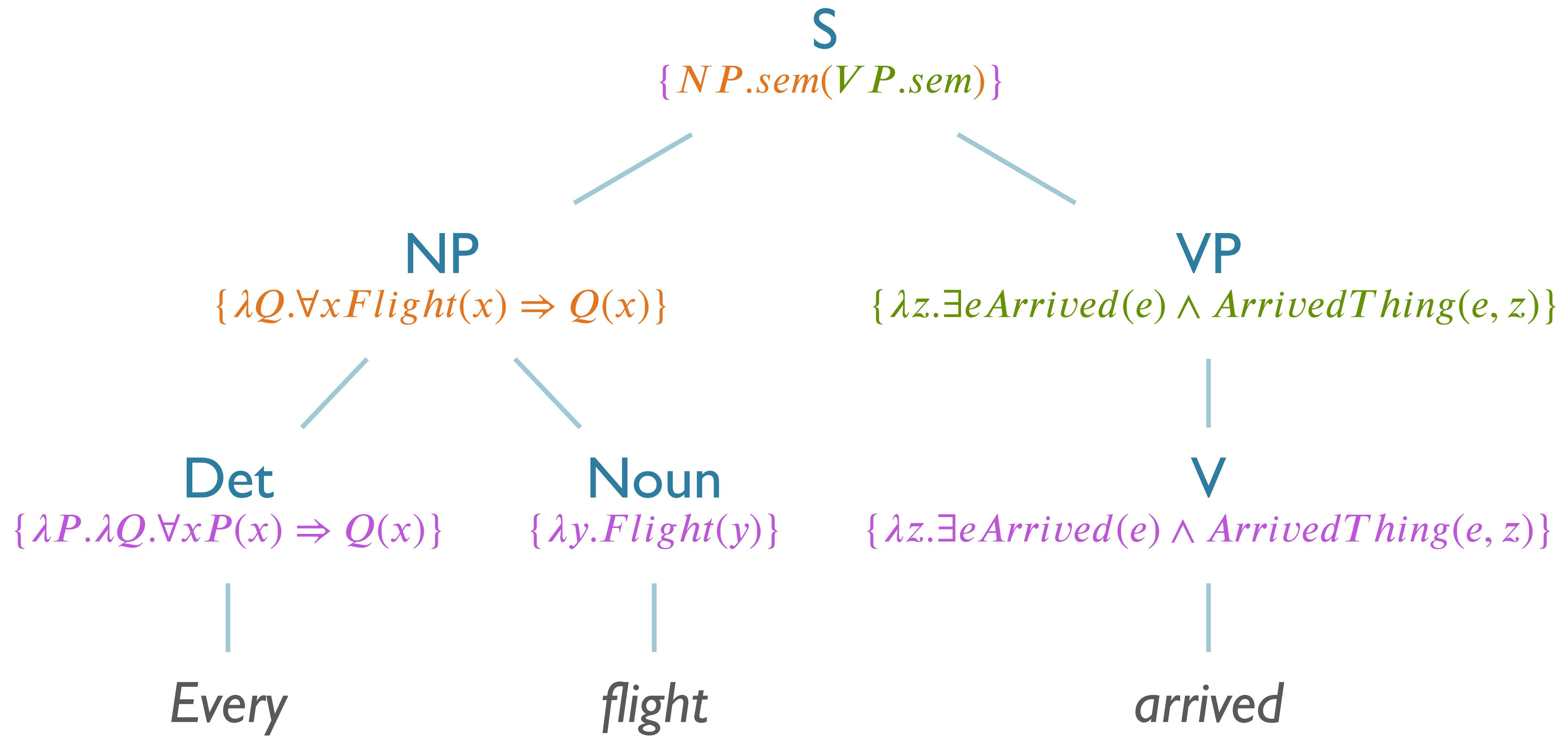


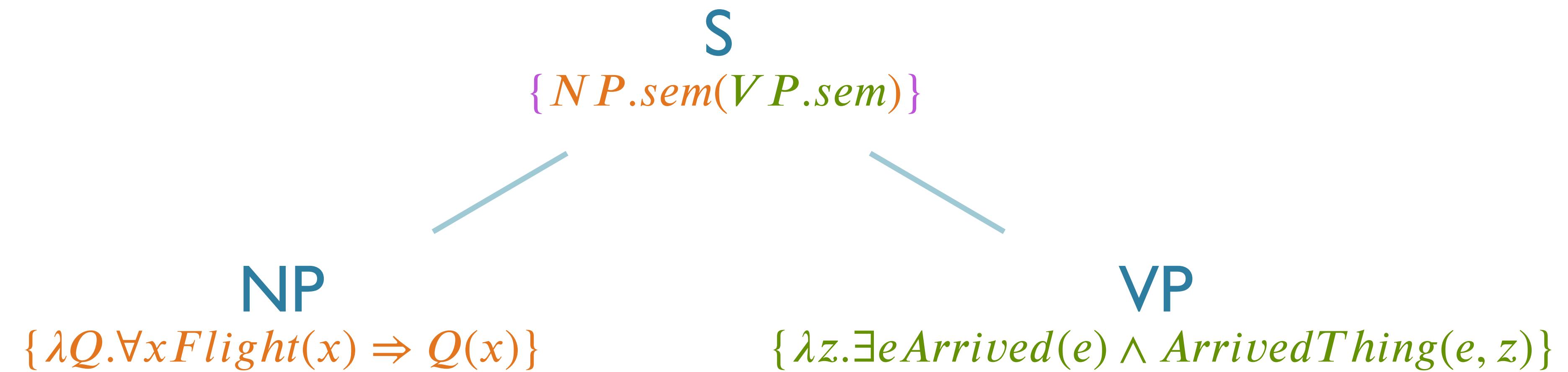












**S**

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

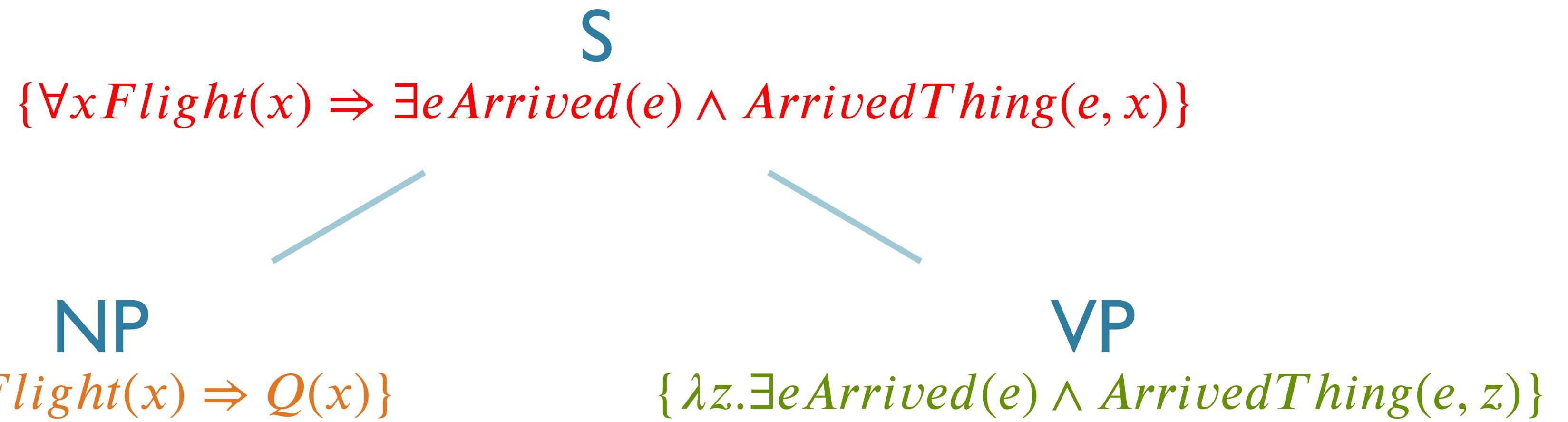


**NP**

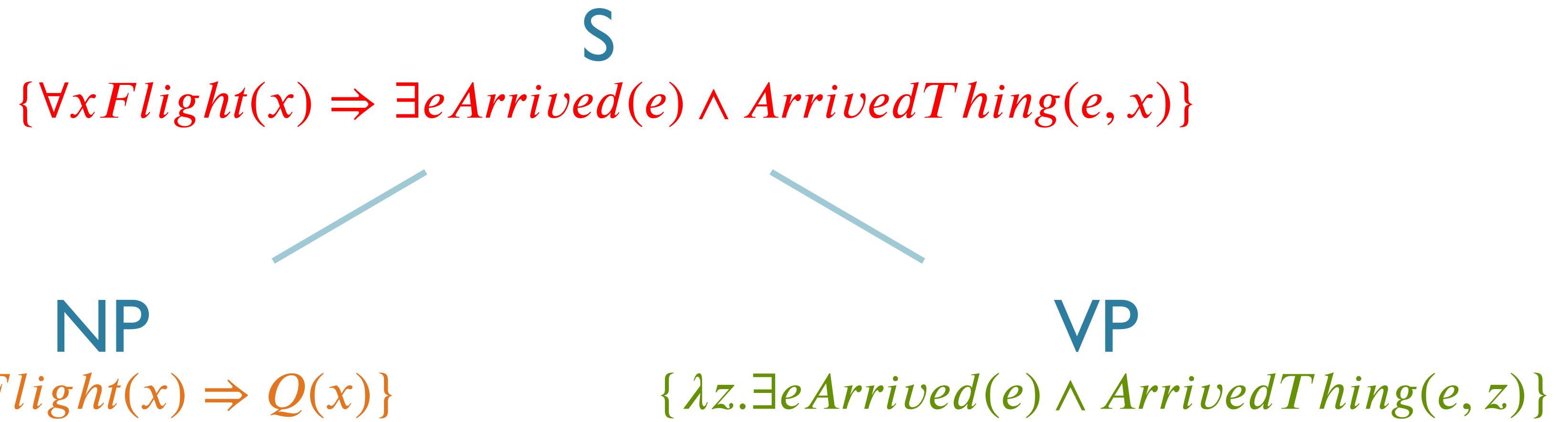
$\{\lambda Q. \forall x Flight(x) \Rightarrow Q(x)\}$

**VP**

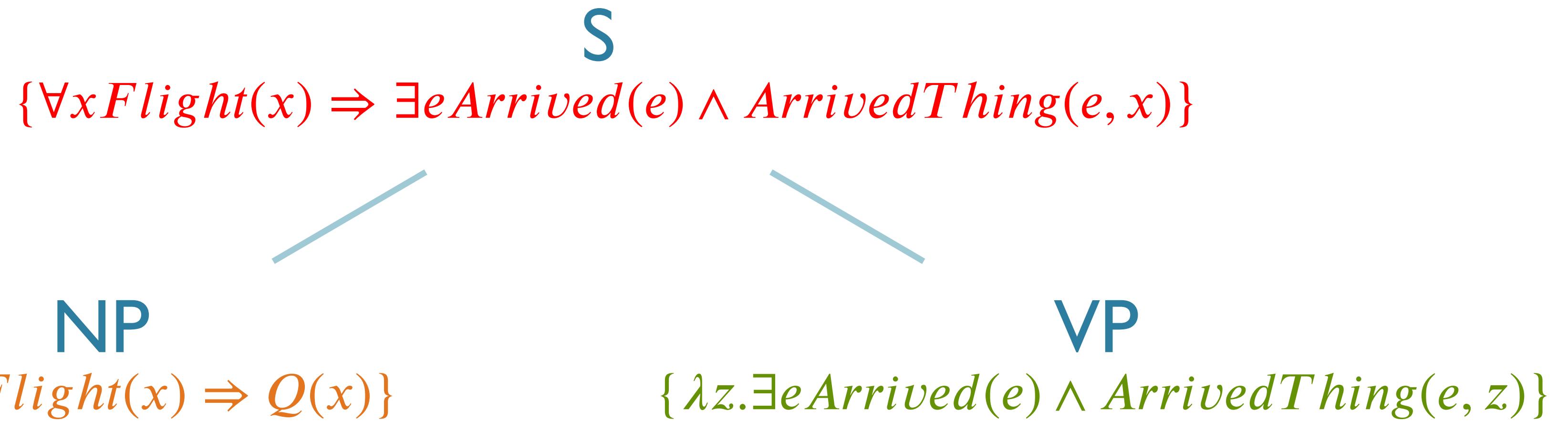
$\{\lambda z. \exists e Arrived(e) \wedge ArrivedThing(e, z)\}$



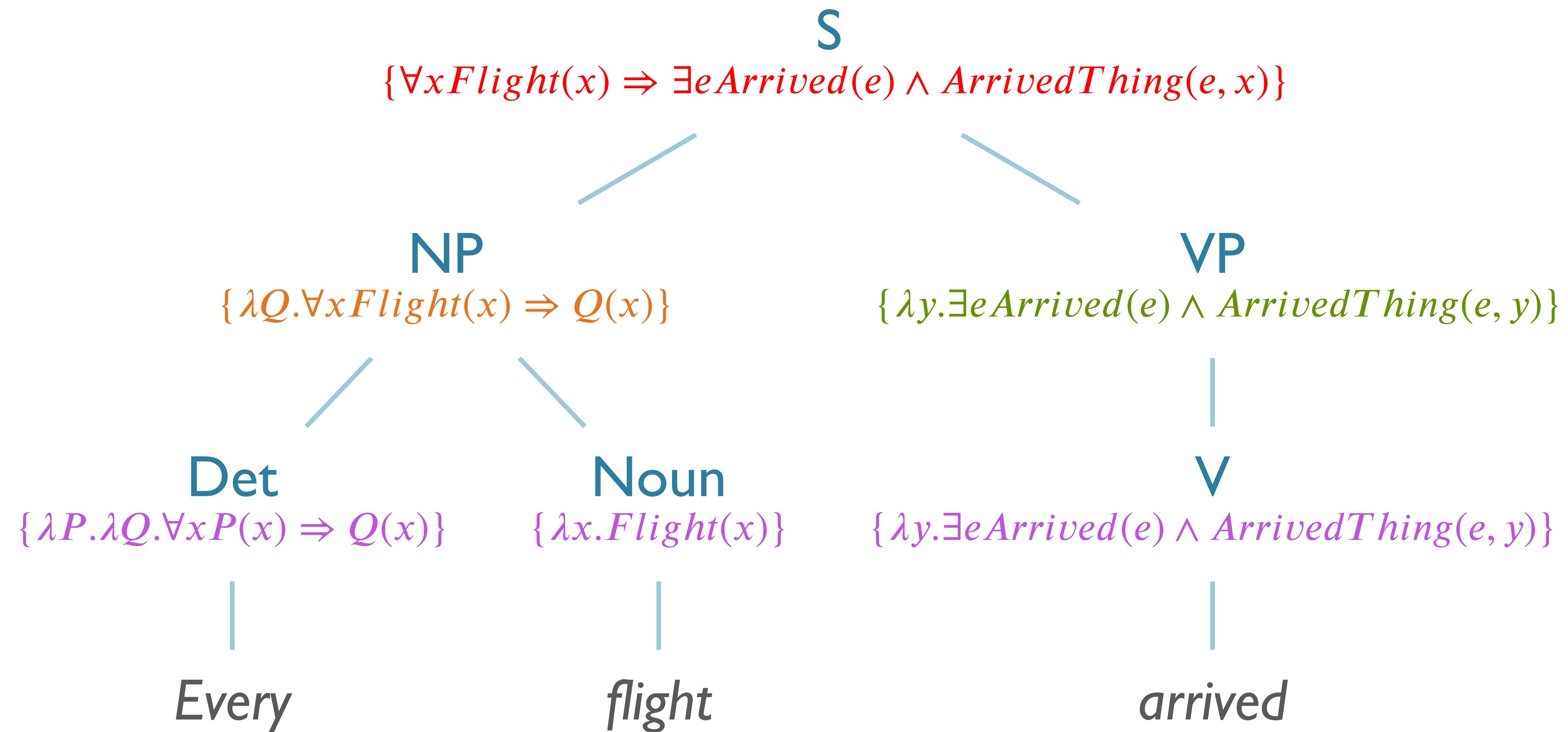
$\lambda Q. \forall x Flight(x) \Rightarrow Q(x)(\lambda z. \exists e Arrived(e) \wedge ArrivedThing(e, z))$



$$\begin{array}{ll}
 \lambda Q. \forall x Flight(x) & \Rightarrow Q(x)(\lambda z. \exists e Arrived(e) \wedge ArrivedThing(e, z)) \\
 \forall x Flight(x) & \Rightarrow \lambda z. \exists e Arrived(e) \wedge ArrivedThing(e, z)(x)
 \end{array}$$



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



# *‘John Booked A Flight’*

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

*...we’ll step through this on Wednesday.*

# Strategy for Semantic Attachments

- General approach:
  - Create complex lambda expressions with lexical items

# Strategy for Semantic Attachments

- General approach:
  - Create complex lambda expressions with lexical items
  - Introduce quantifiers, predicates, terms

# Strategy for Semantic Attachments

- General approach:
  - Create complex lambda expressions with lexical items
  - Introduce quantifiers, predicates, terms
  - Percolate up semantics from child if non-branching

# Strategy for Semantic Attachments

- General approach:
  - Create complex lambda expressions with lexical items
  - Introduce quantifiers, predicates, terms
  - Percolate up semantics from child if non-branching
  - Apply semantics of one child to other through lambda

# Strategy for Semantic Attachments

- General approach:
  - Create complex lambda expressions with lexical items
  - Introduce quantifiers, predicates, terms
  - Percolate up semantics from child if non-branching
  - Apply semantics of one child to other through lambda
  - Combine elements, don't introduce new ones

# Semantics Learning

- Zettlemoyer & Collins ([2005](#), [2007](#), etc); Kate & Mooney ([2007](#))
- Given semantic representation and corpus of parsed sentences
  - Learn mapping from sentences to logical form

# Semantics Learning

- Zettlemoyer & Collins ([2005](#), [2007](#), etc); Kate & Mooney ([2007](#))
- Given semantic representation and corpus of parsed sentences
  - Learn mapping from sentences to logical form
- Similar approaches to:
  - Learning instructions from computer manuals
  - Game play via walkthrough descriptions
  - Robocup/Soccer play from commentary

# Parsing with Semantics

- Implement semantic analysis in parallel with syntactic parsing
- Enabled by this rule-to-rule compositional approach

# Parsing with Semantics

- Implement semantic analysis in parallel with syntactic parsing
  - Enabled by this rule-to-rule compositional approach
- Required modifications
  - Augment grammar rules with semantics field
  - Augment chart states with meaning expression
  - Incrementally compute semantics

# Sidenote: Idioms

- Not purely compositional
  - *kick the bucket* → die
  - *tip of the iceberg* → small part of the entirety
- Handling
  - Mix lexical items with constituents
  - Create idiom-specific construct for productivity
  - Allow non-compositional semantic attachments
- Extremely complex, e.g. metaphor