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] -> ‘This sentence is grammatical.’
  - S[-grammatical] -> ‘This grammatical sentence is.’

NP of the Week

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

Behold Trump's pre-election secret weapon: Nigel Farage, '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.

(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

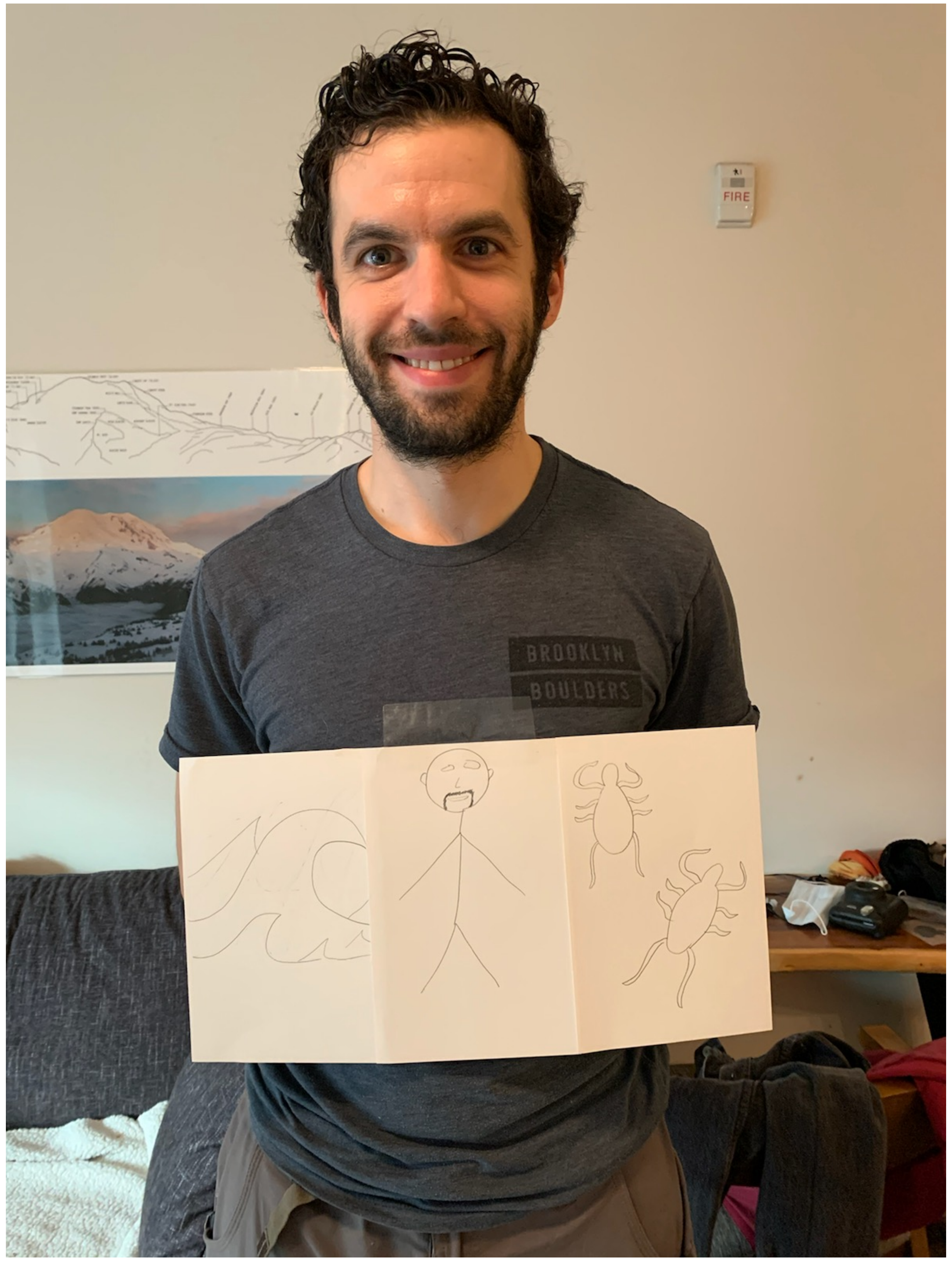
A top baseball prospect's Southern California scholarship was lost to the pandemic

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**:

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

FOL Syntax Summary

$$
\begin{array}{lcl}
\textbf{Formula} & \rightarrow & \textit{AtomicFormula} \\
 & | & \textit{Formula Connective Formula} \\
 & | & \textit{Quantifier Variable, } \ldots \textit{ Formula} \\
 & | & \neg \textit{ Formula} \\
 & | & (\textit{Formula}) \\
\textbf{AtomicFormula} & \rightarrow & \textit{Predicate}(\textit{Term}, \ldots) \\
\textbf{Term} & \rightarrow & \textit{Function}(\textit{Term}, \ldots) \\
 & | & \textit{Constant} \\
 & | & \textit{Variable}
\end{array}
$$

$$
\begin{array}{lcl}
\textbf{Connective} & \rightarrow & \land \mid \lor \mid \Rightarrow \\
\textbf{Quantifier} & \rightarrow & \forall \mid \exists \\
\textbf{Constant} & \rightarrow & \textit{VegetarianFood} \mid \textit{Maharani} \mid \ldots \\
\textbf{Variable} & \rightarrow & x \mid y \mid \ldots \\
\textbf{Predicate} & \rightarrow & \textit{Serves} \mid \textit{Near} \mid \ldots \\
\textbf{Function} & \rightarrow & \textit{LocationOf} \mid \textit{CuisineOf} \mid \ldots
\end{array}
$$

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 $\mathscr{D}$

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

Noisy	Frasca, Med, and Rio are noisy	***Noisy***={*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

Noisy	Frasca, Med, and Rio are noisy	***Noisy***=$\{e,f,g\}$

Relations

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

Sample Domain $\mathscr{D}$

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

Noisy	Frasca, Med, and Rio are noisy	***Noisy***=$\{e,f,g\}$

Relations

Likes	Matthew likes the Med Katie likes the Med and Rio Franco likes Frasca Caroline likes the Med and Rio	***Likes***=$\{ \langle a,f \rangle, \langle c,f \rangle, \langle c,g \rangle, \langle b,e \rangle, \langle d,f \rangle, \langle d,g \rangle \}$
Serves	Med serves eclectic Rio serves Mexican Frasca serves Italian	***Serves***=$\{ \langle c,f \rangle, \langle f,i \rangle, \langle e,h \rangle \}$

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**
  - λ-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 Carrol

Ambiguity & Models

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

Ambiguity & Models

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

Ambiguity & Models

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

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.

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.
  - $\forall x. Tesla(x) \Rightarrow \neg(HurtlingTowardMars(x))$

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.
    - $\forall x. Tesla(x) \Rightarrow \neg(HurtlingTowardMars(x))$
    - $\neg\forall \mathrm{x}.(Tesla(x) \Rightarrow (HurtlingTowardMars(x)))$

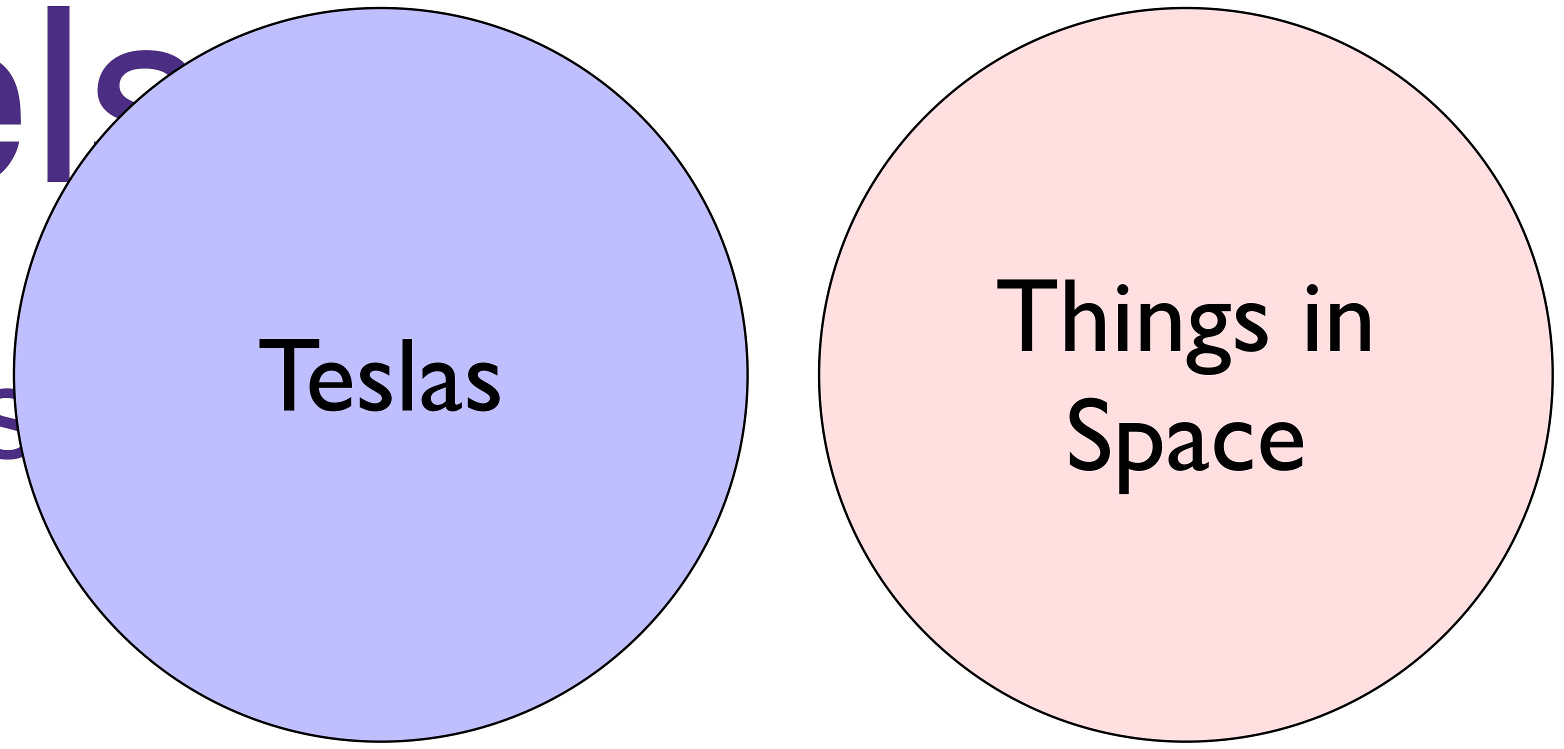
Ambiguity & Models

- *"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.
  - $\forall x. Tesla(x) \Rightarrow \neg(HurtlingTowardMars(x))$
  - $\neg\forall \mathrm{x}.(Tesla(x) \Rightarrow (HurtlingTowardMars(x)))$
    - $[\exists(\mathrm{x}).(\mathrm{Tesla}(\mathrm{x}) \wedge \neg\mathrm{HurtlingTowardsMars}(\mathrm{x}))]$

State of known Universe: 02/05/2018

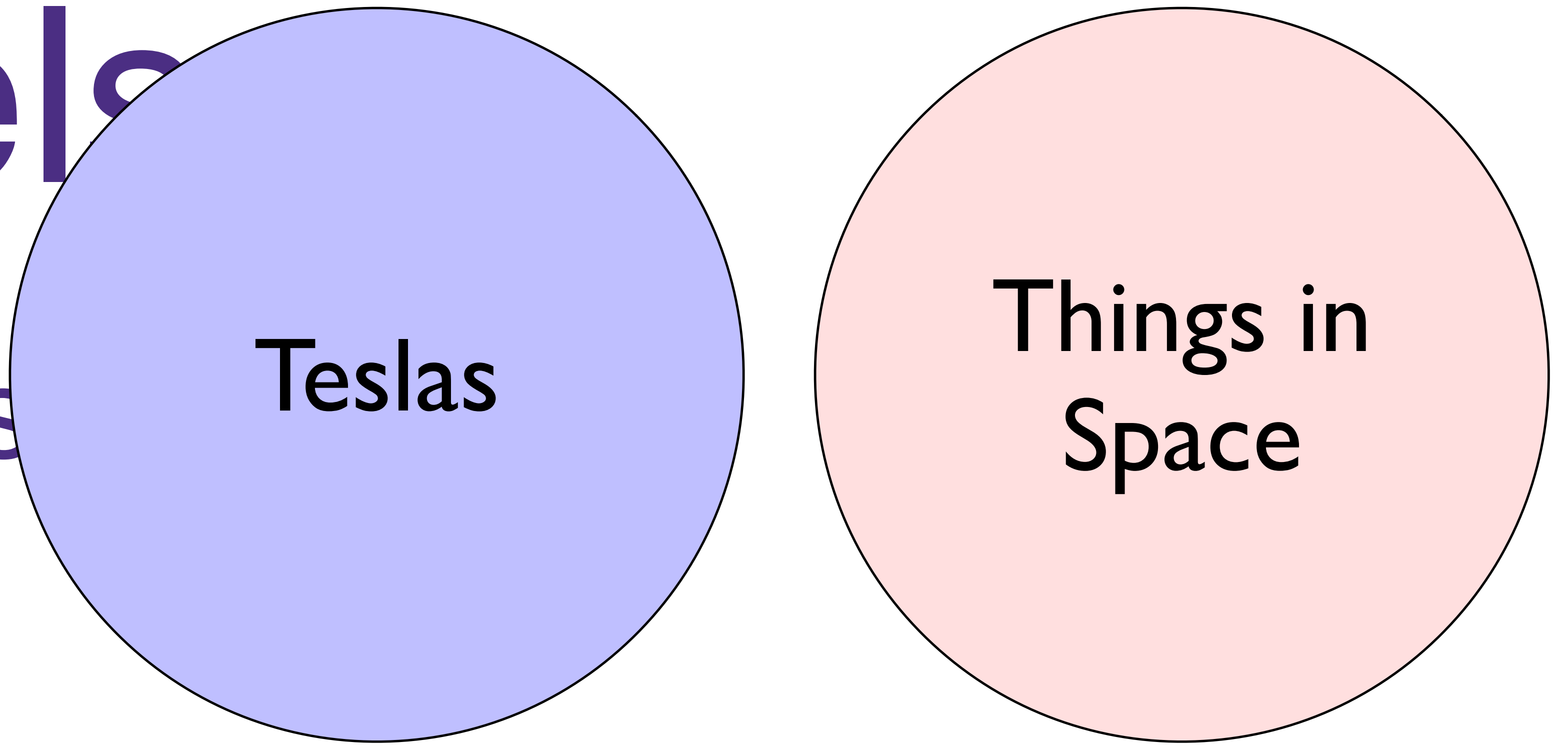
Ambiguity & Models



- *"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.
  - $\forall x. Tesla(x) \Rightarrow \neg(HurtlingTowardMars(x))$
  - $\neg\forall \mathrm{x}.(Tesla(x) \Rightarrow (HurtlingTowardMars(x)))$
    - $[\exists(\mathrm{x}).(\mathrm{Tesla}(\mathrm{x}) \wedge \neg\mathrm{HurtlingTowardsMars}(\mathrm{x}))]$

Ambiguity & Models



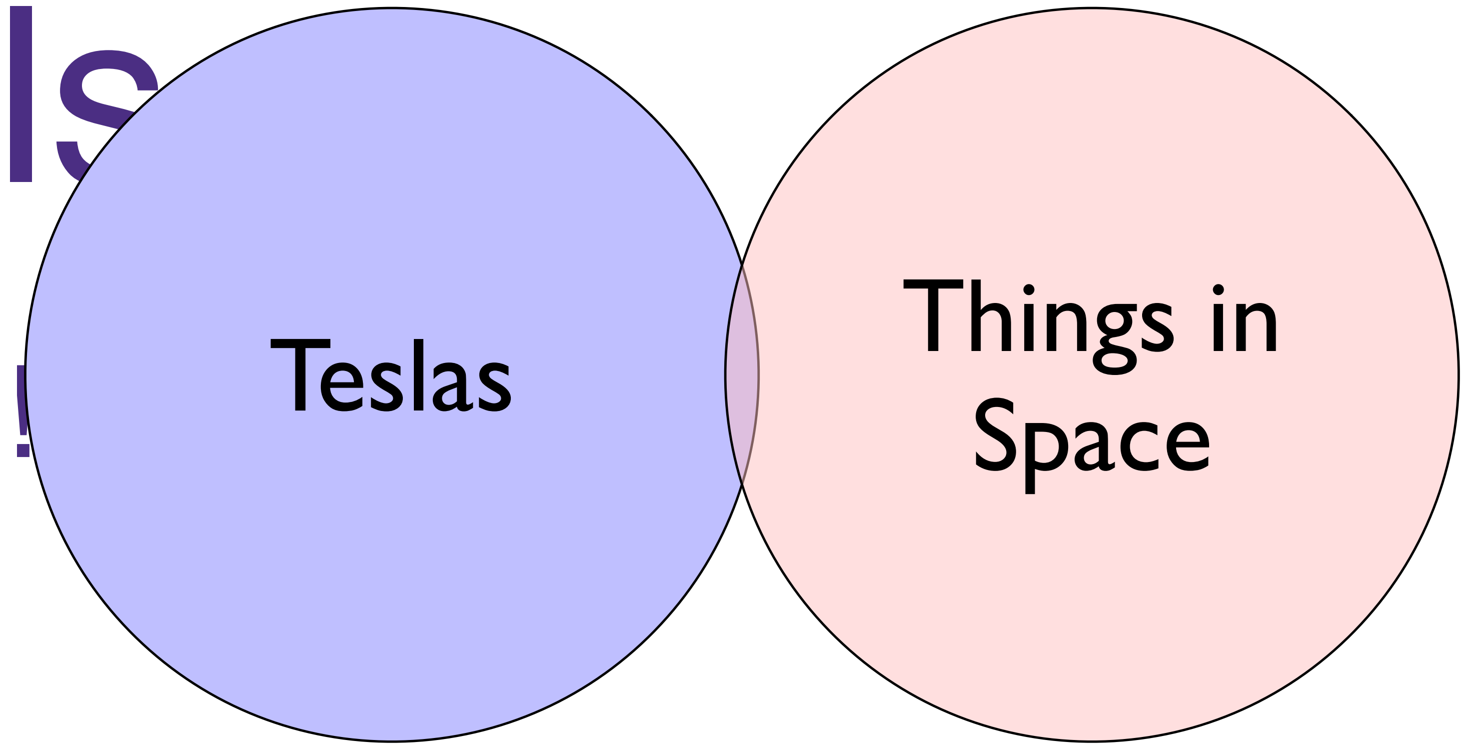
- *"Every Tesla is powered by a battery."* — Ambiguous
  - $\forall x. Tesla(x) \Rightarrow (\exists(y). Battery(y) \wedge Powers(y, x))$
  - $\exists(y). Battery(y) \wedge (\forall x. Tesla(x) \Rightarrow Powers(y, x))$

- Every Tesla is not hurtling toward Mars.
  - $\forall x. Tesla(x) \Rightarrow \neg(HurtlingTowardMars(x))$
  - $\neg\forall \mathrm{x}.(Tesla(x) \Rightarrow (HurtlingTowardMars(x)))$
    - $[\exists(\mathrm{x}).(\mathrm{Tesla}(\mathrm{x}) \wedge \neg\mathrm{HurtlingTowardsMars}(\mathrm{x}))]$

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

State of known Universe: 02/06/2018

Ambiguity & Models



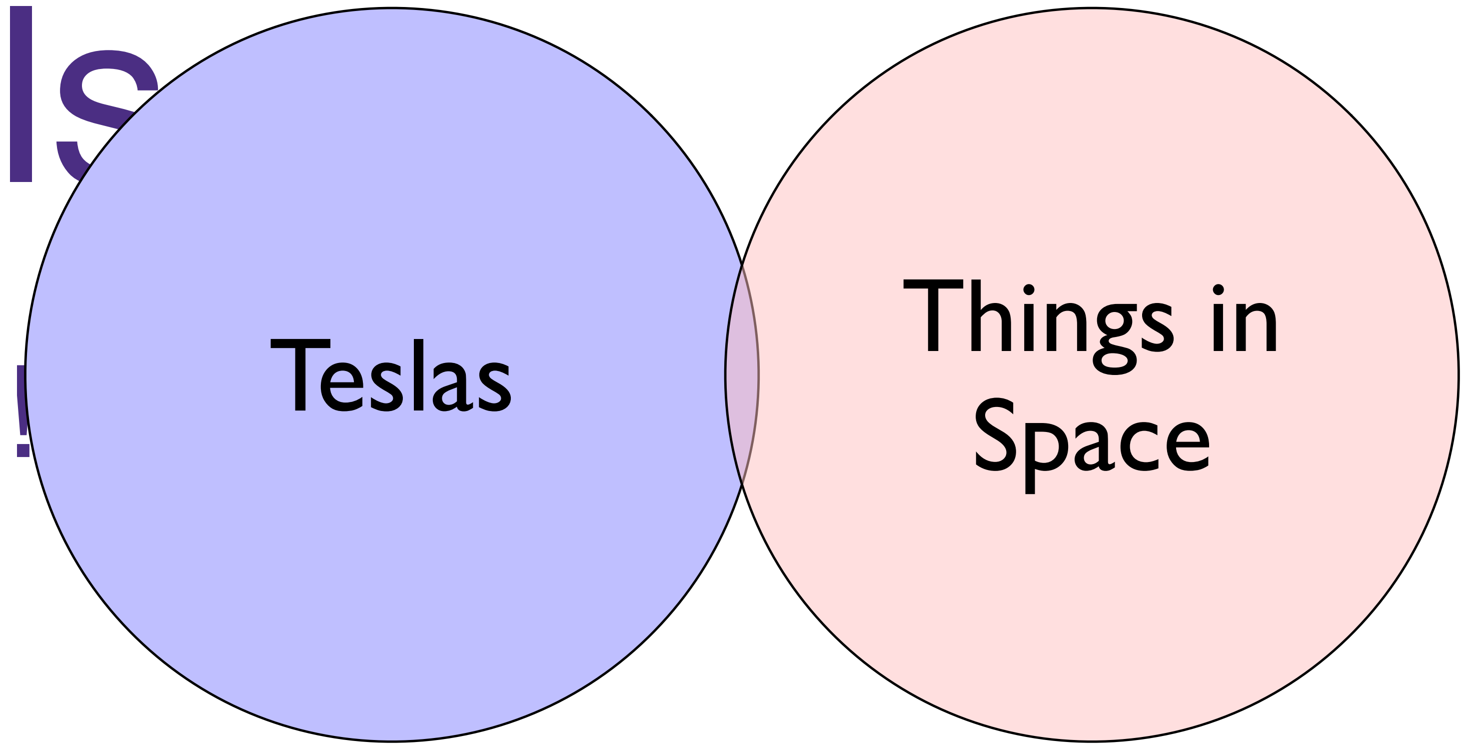
- *"Every Tesla is powered by a battery."* — Ambiguous!
  - $\forall x. Tesla(x) \Rightarrow (\exists(y). Battery(y) \wedge Powers(y, x))$
  - $\exists(y). Battery(y) \wedge (\forall x. Tesla(x) \Rightarrow Powers(y, x))$

- Every Tesla is not hurtling toward Mars.
  - $\forall x. Tesla(x) \Rightarrow \neg(HurtlingTowardMars(x))$
  - $\neg \forall x. (Tesla(x) \Rightarrow (HurtlingTowardMars(x)))$
    - $[\exists(x). (Tesla(x) \wedge \neg HurtlingTowardsMars(x))]$

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

State of known Universe: 02/06/2018

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

$\exists(x). (Tesla(x) \wedge HurtlingTowardsMars(x))$

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

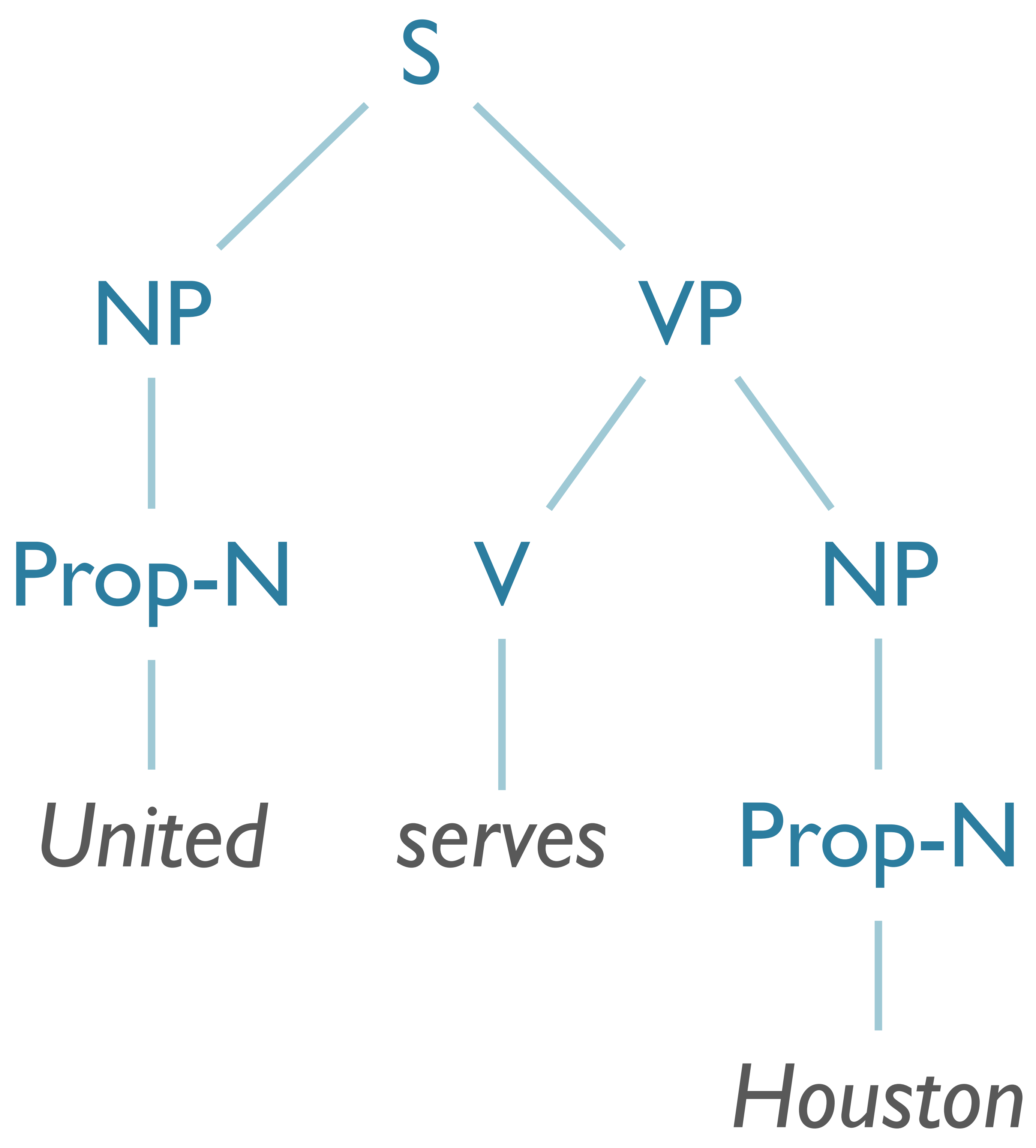
Integrating Semantics into Syntax

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

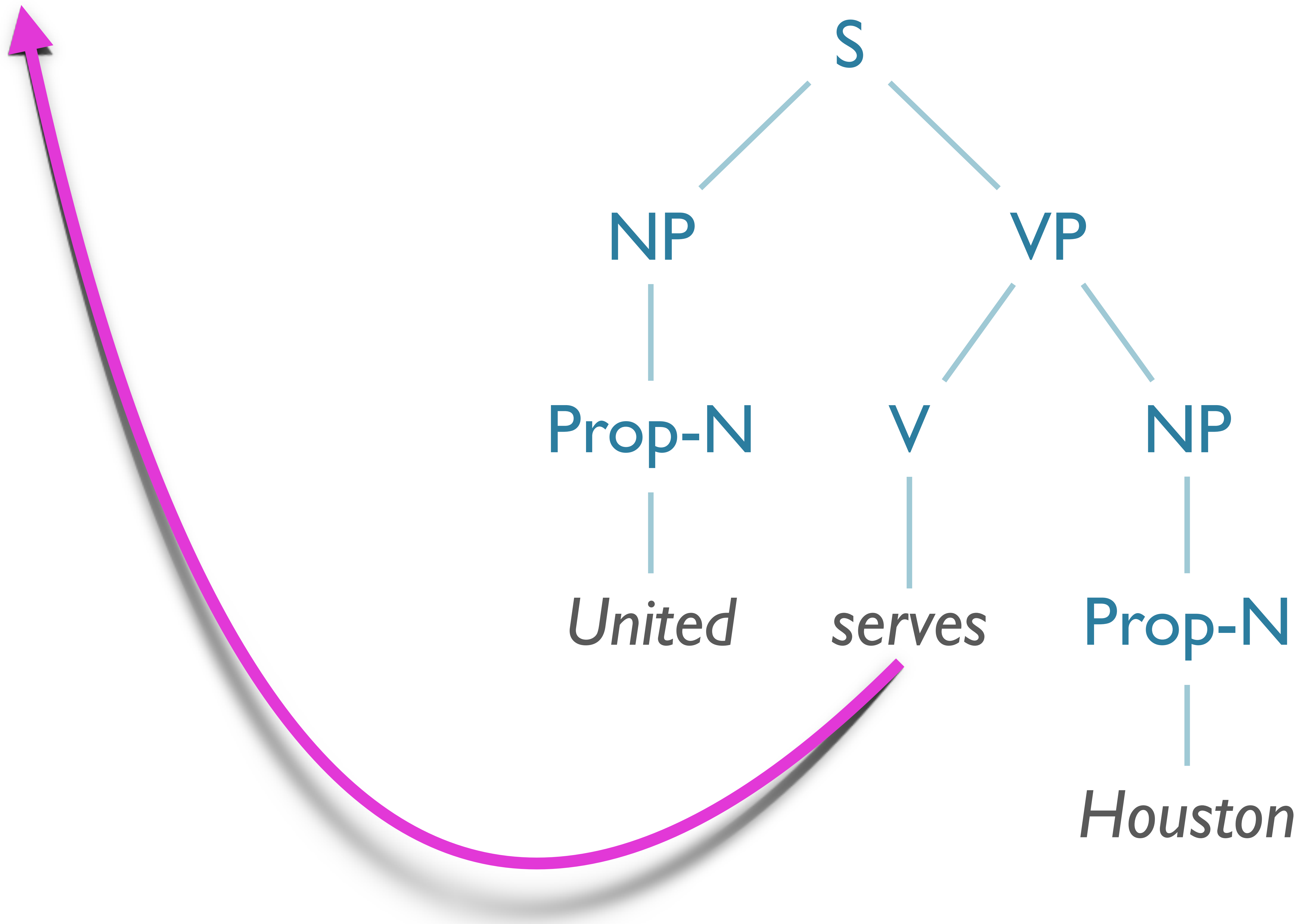
- *United serves Houston*



Simple Example

- *United serves Houston*

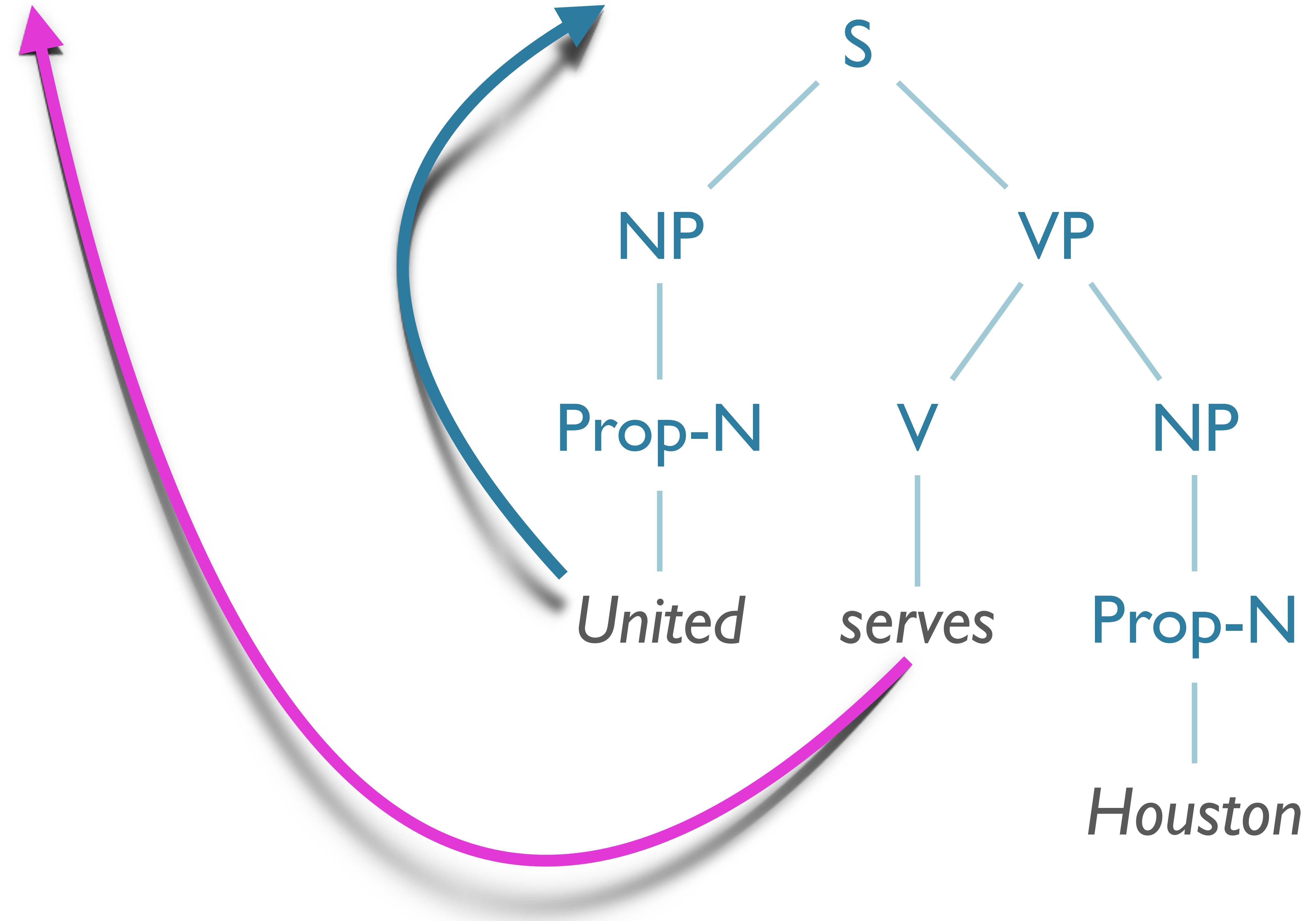
$\exists e\left(Serving(e) \wedge\right.$



Simple Example

- *United serves Houston*

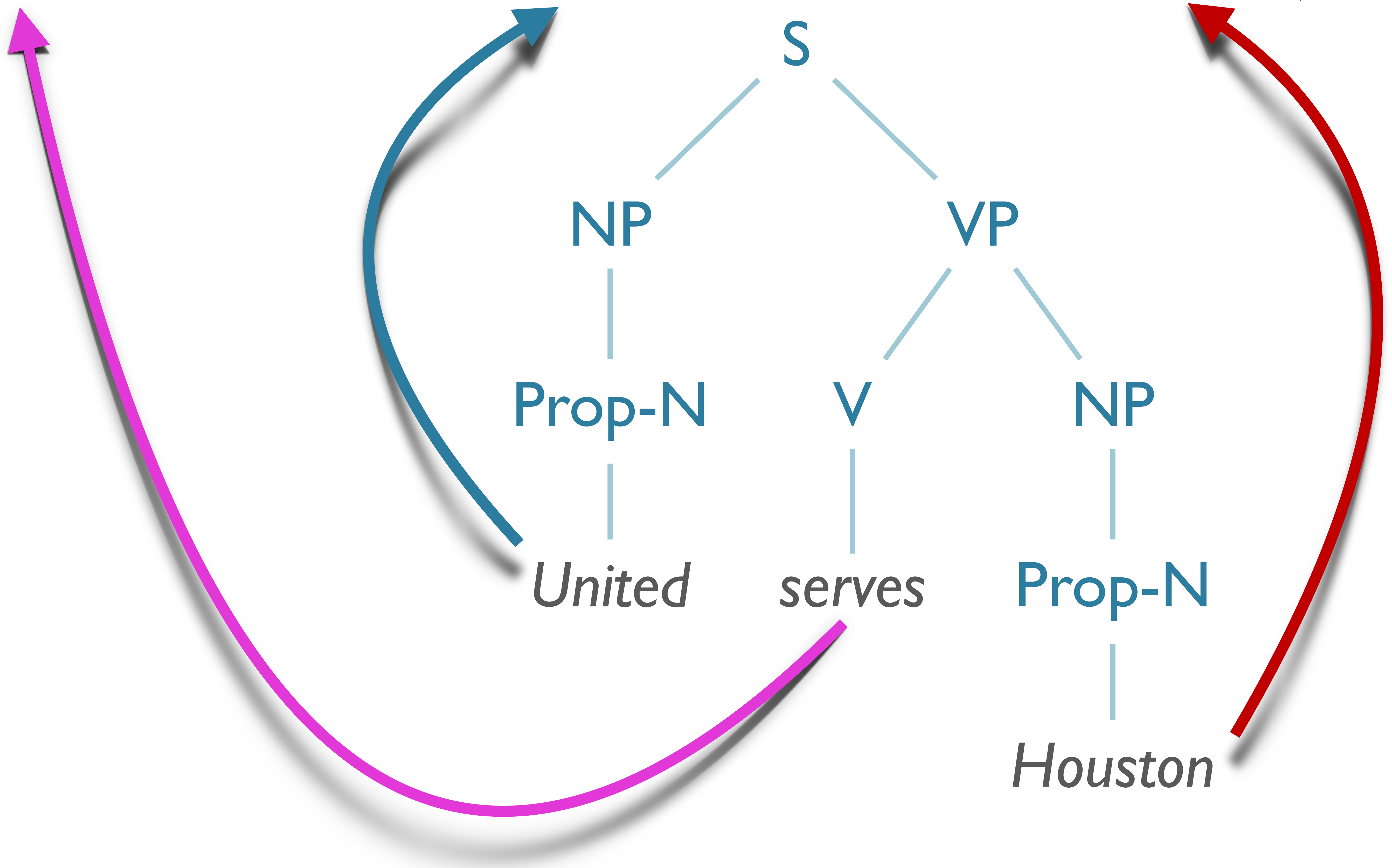
$\exists e\Big(Serving(e) \wedge Server(e, United) \wedge$



Simple Example

- *United serves Houston*

$$\exists e\left(Serving(e) \wedge Server(e, United) \wedge Served(e, Houston)\right)$$



Rule-to-rule Model

- **Lambda Calculus** and the **Rule-to-Rule Hypothesis**
  - λ-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:

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

- In NLTK syntax (more later):

```
A → a₁ … aₙ[SEM=<f(?aⱼ.sem …)>]
```

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
  - $Verb \rightarrow \text{'arrived'}$
    - 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=<Khalil>] → ‘Khalil’

Proper Nouns & Intransitive Verbs

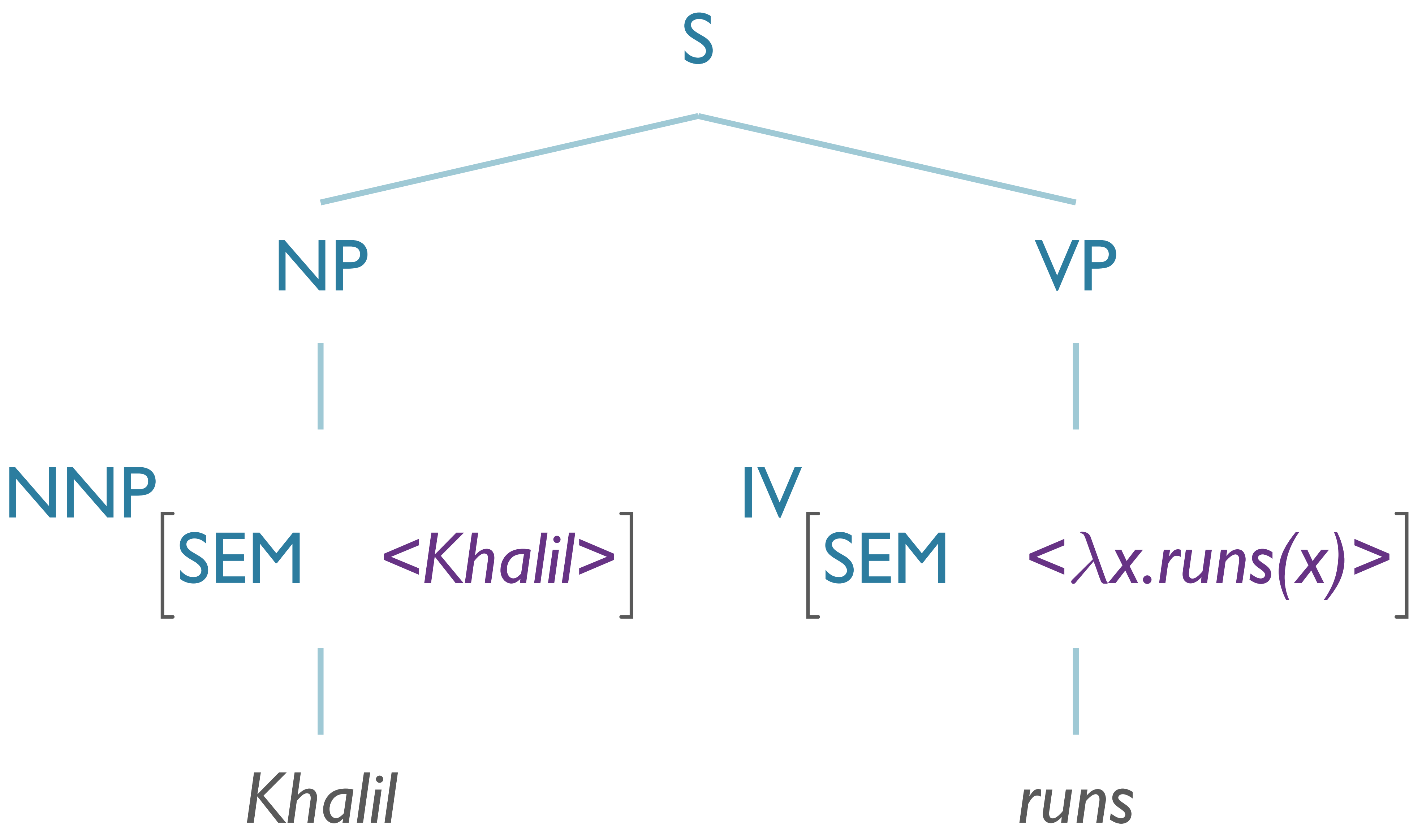
- Our instinct for names is to just use the constant:
  - `NNP[SEM=<Khalil>] → 'Khalil'`
- However, we want to apply our λ-closures left-to-right consistently.

```
S[SEM=np?(vp?)] → NP[SEM=np?] VP[SEM=vp?]
```

Proper Nouns & Intransitive Verbs

- Our instinct for names is to just use the constant:
  - NNP[SEM=<Khalil>] → ‘Khalil’
- However, we want to apply our λ-closures left-to-right consistently.

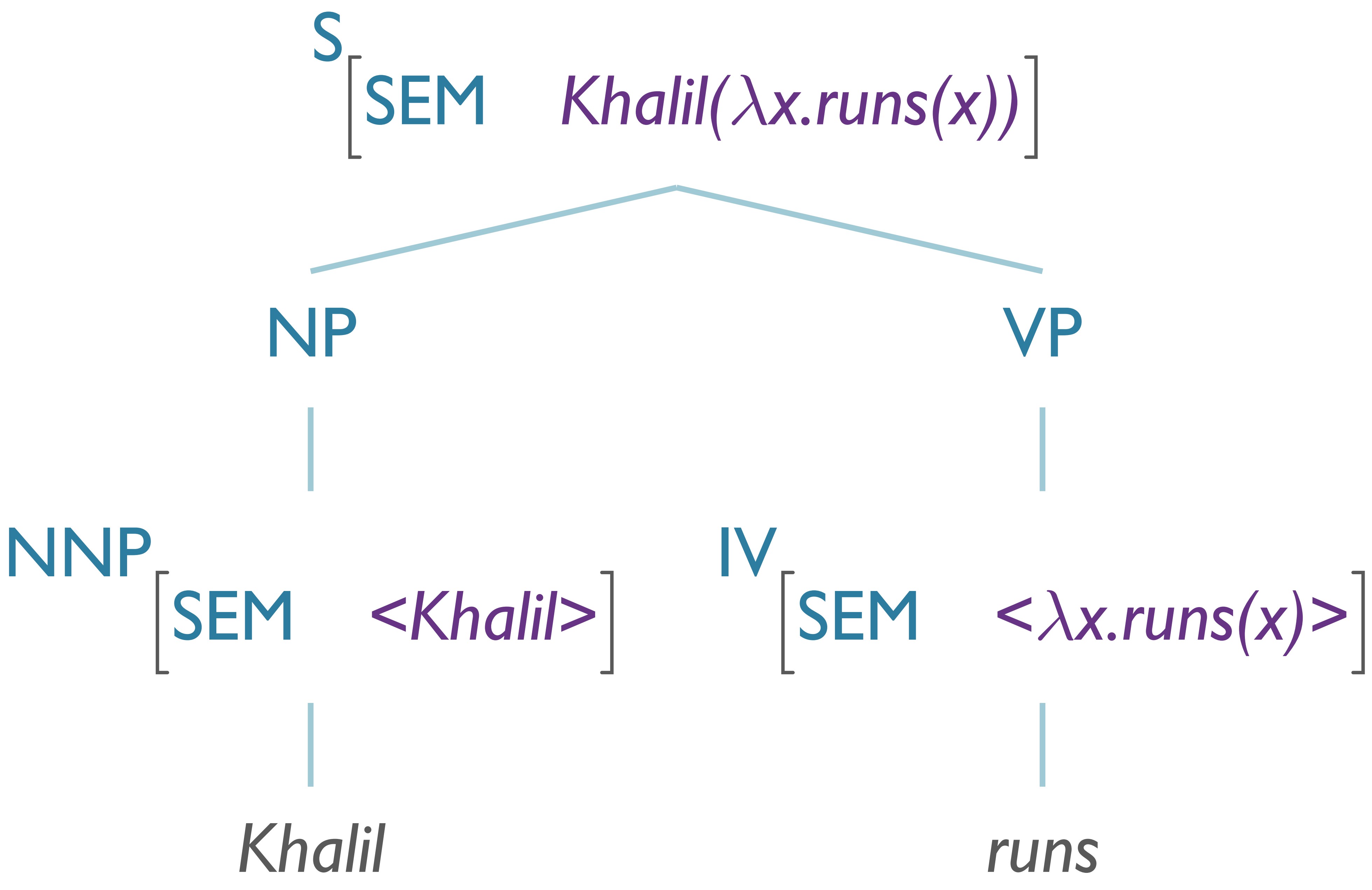
S[SEM=np?(vp?)] → NP[SEM=np?] VP[SEM=vp?]



Proper Nouns & Intransitive Verbs

- Our instinct for names is to just use the constant:
  - NNP[SEM=<Khalil>] → ‘Khalil’
- However, we want to apply our λ-closures left-to-right consistently.

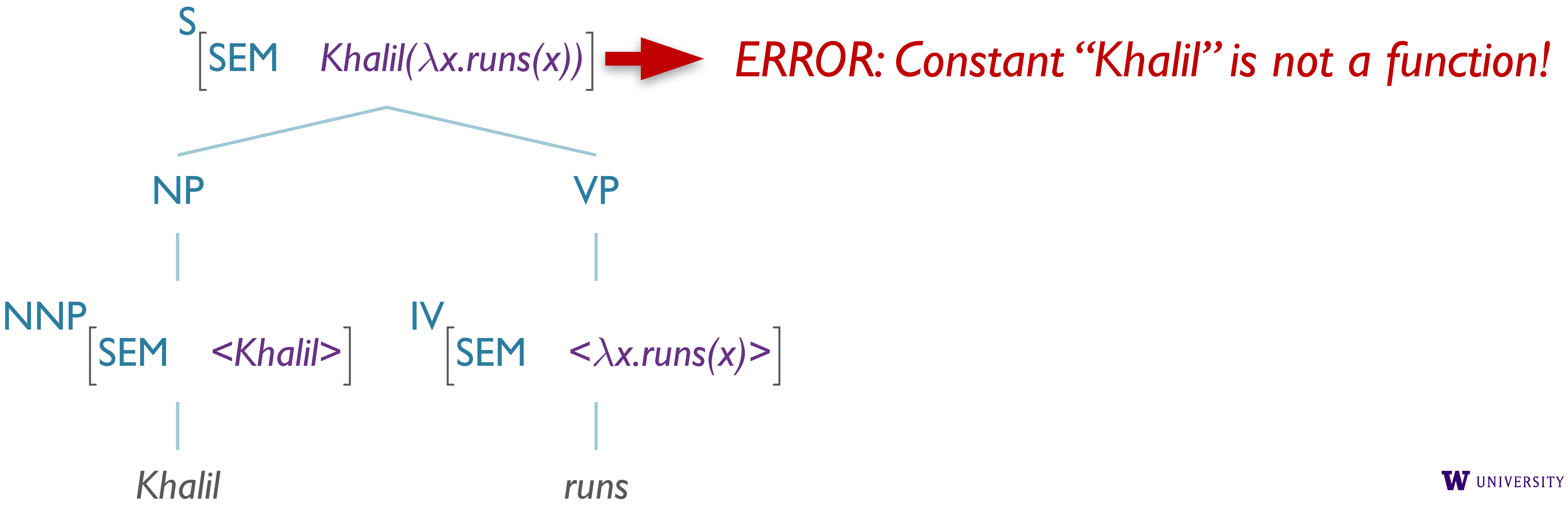
S[SEM=np?(vp?)] → NP[SEM=np?] VP[SEM=vp?]



Proper Nouns & Intransitive Verbs

- Our instinct for names is to just use the constant:
  - NNP[SEM=<Khalil>] → ‘Khalil’
- However, we want to apply our λ-closures left-to-right consistently.

```
S[SEM=np?(vp?)] → NP[SEM=np?] VP[SEM=vp?]
```



Proper Nouns & Intransitive Verbs

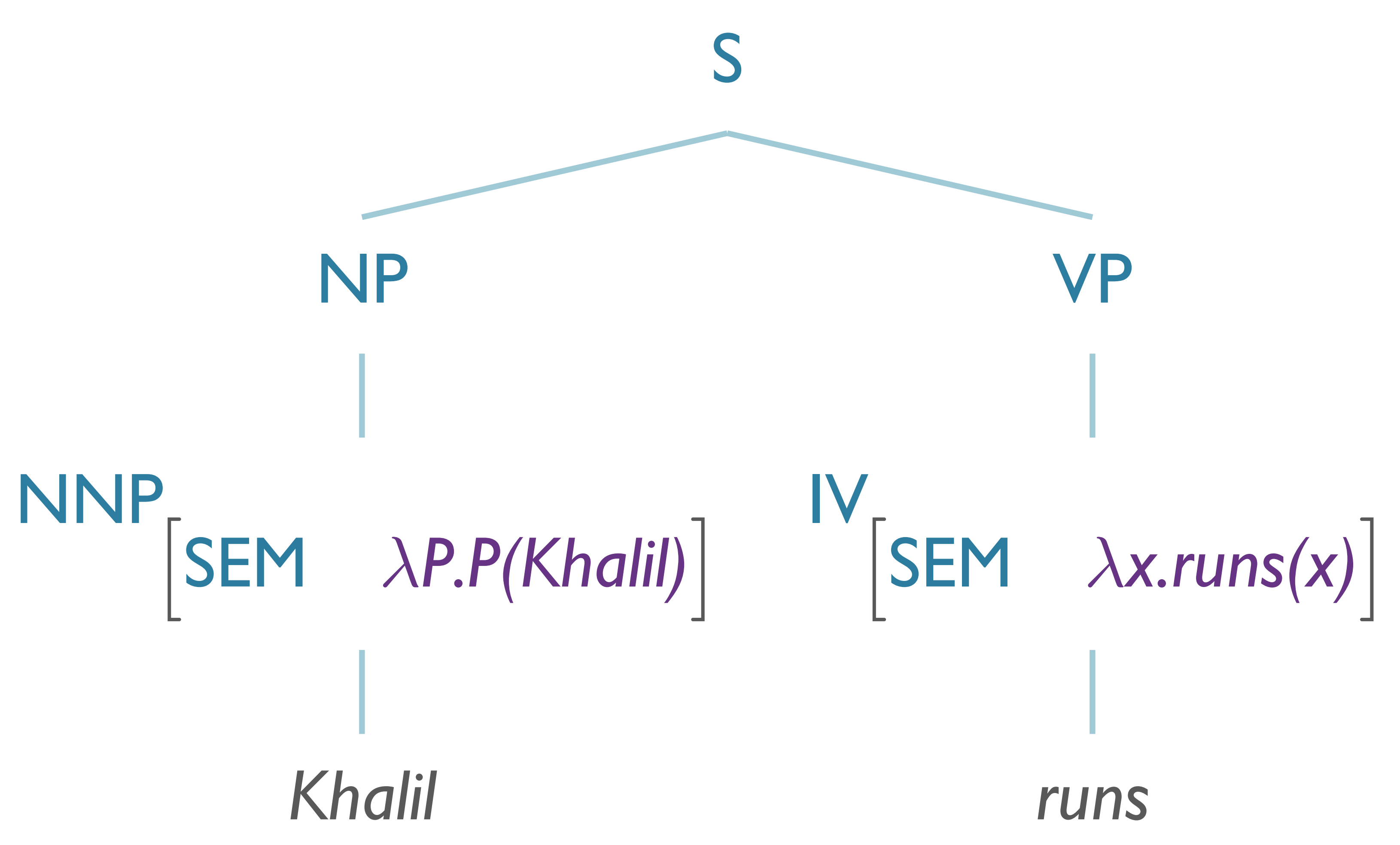
- Instead, we use a ***dummy predicate:***
  - $\lambda Q.Q(Khalil)$

- *"Generalizing to the worst case" (cf. Montague; Partee on type-shifting)*

Proper Nouns & Intransitive Verbs

- With the dummy predicate:
  - `NNP[SEM=<\P.P(Khalil)>] → 'Khalil'`

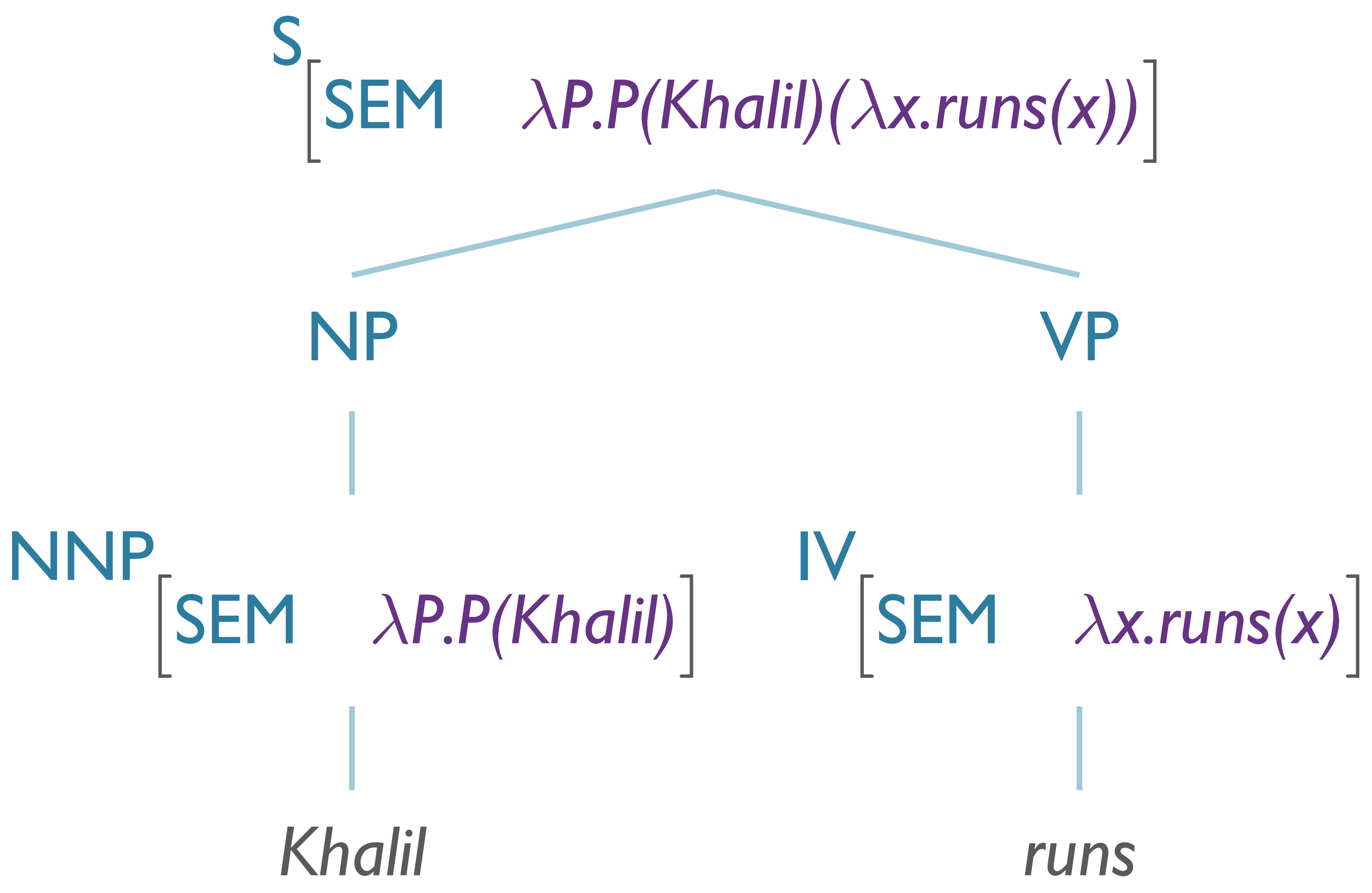
```
S[SEM=np?(vp?)] → NP[SEM=np?] VP[SEM=vp?]
```



Proper Nouns & Intransitive Verbs

- With the dummy predicate:
  - `NNP[SEM=<\P.P(Khalil)>] → ‘Khalil’`

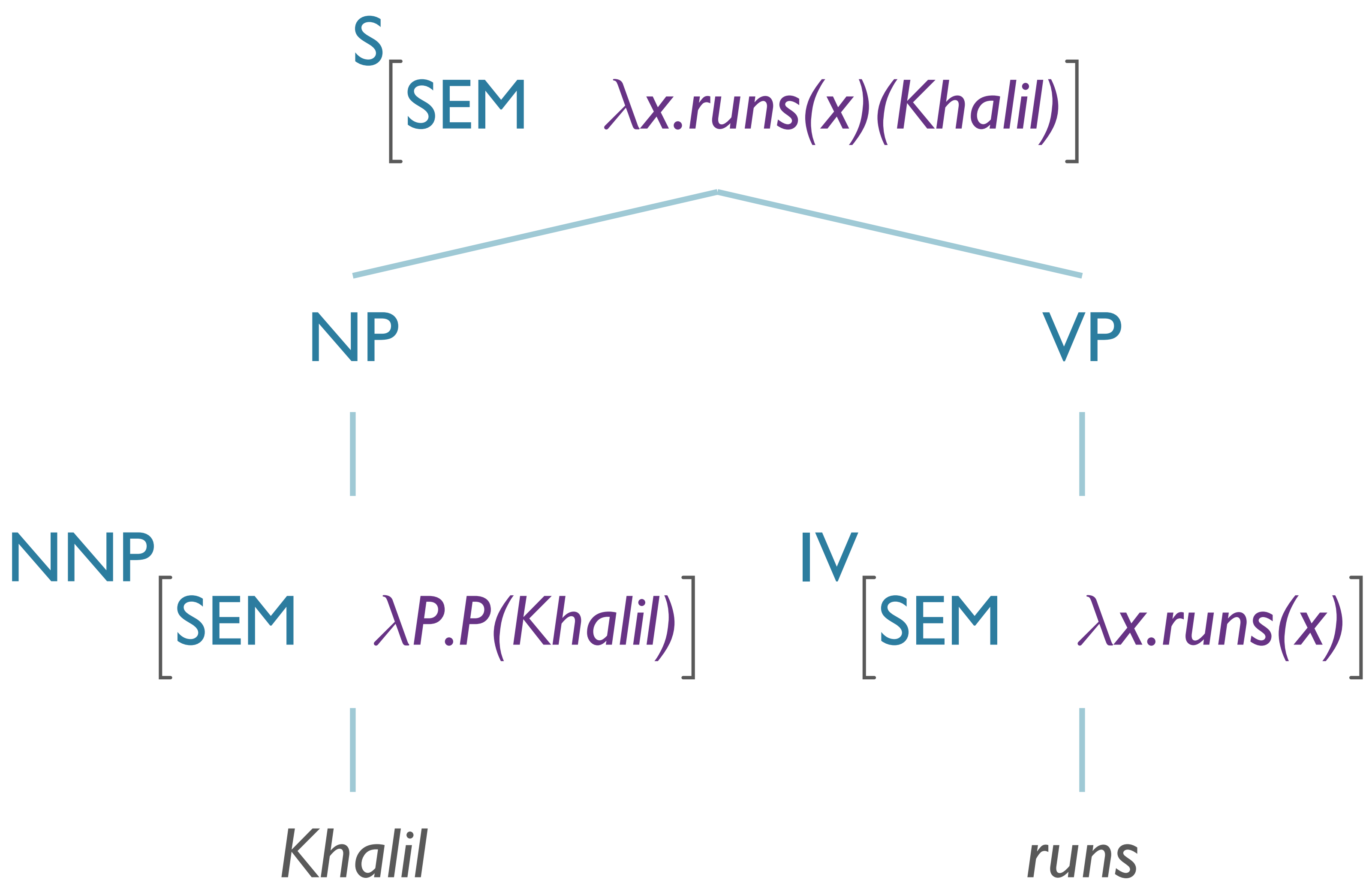
`S[SEM=np?(vp?)] → NP[SEM=np?] VP[SEM=vp?]`



Proper Nouns & Intransitive Verbs

- With the dummy predicate:
  - `NNP[SEM=<\P.P(Khalil)>] → ‘Khalil’`

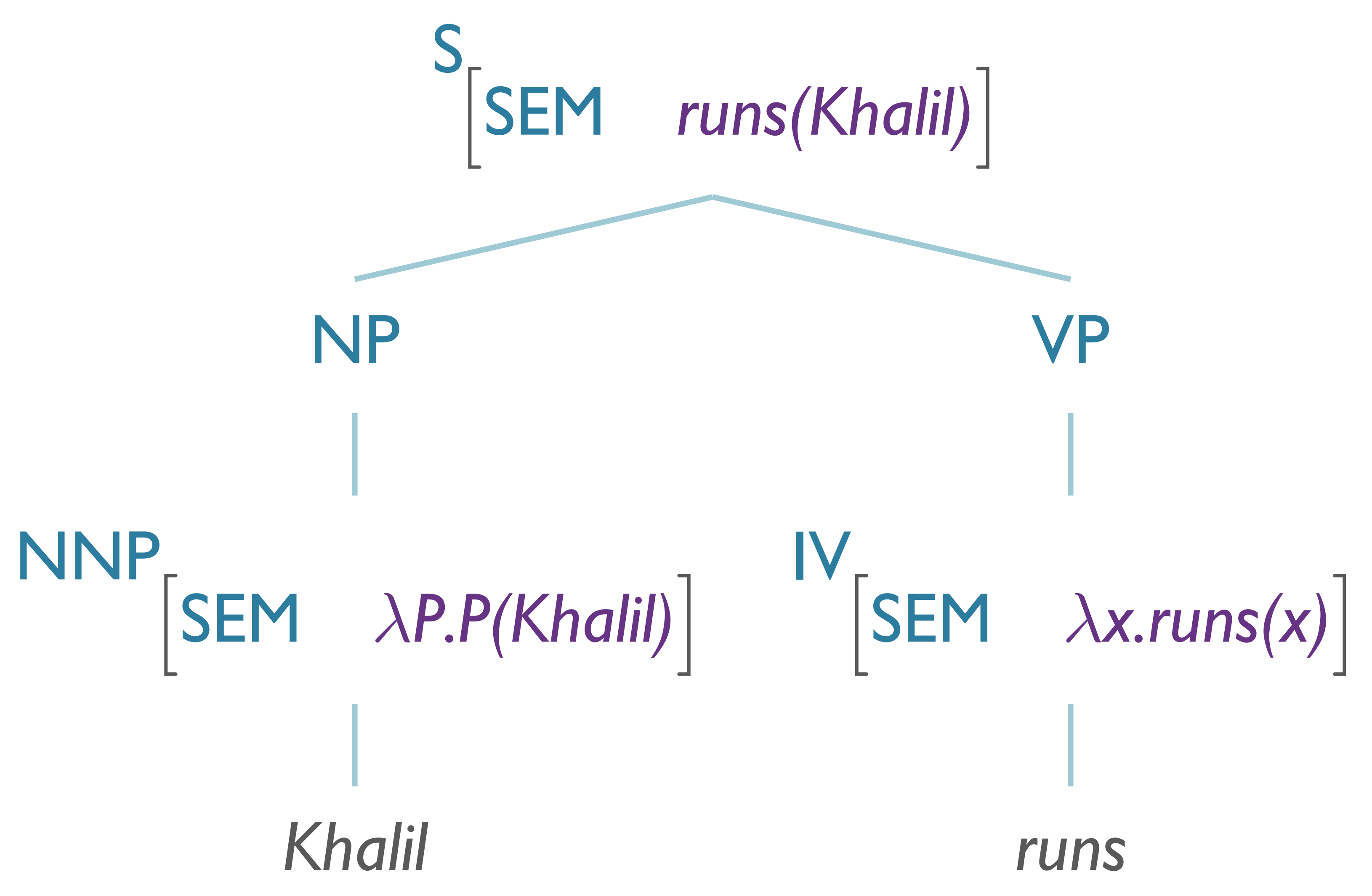
```
S[SEM=np?(vp?)] → NP[SEM=np?] VP[SEM=vp?]
```



Proper Nouns & Intransitive Verbs

- With the dummy predicate:
  - `NNP[SEM=<\P.P(Khalil)>] → 'Khalil'`

```
S[SEM=np?(vp?)] → NP[SEM=np?] VP[SEM=vp?]
```



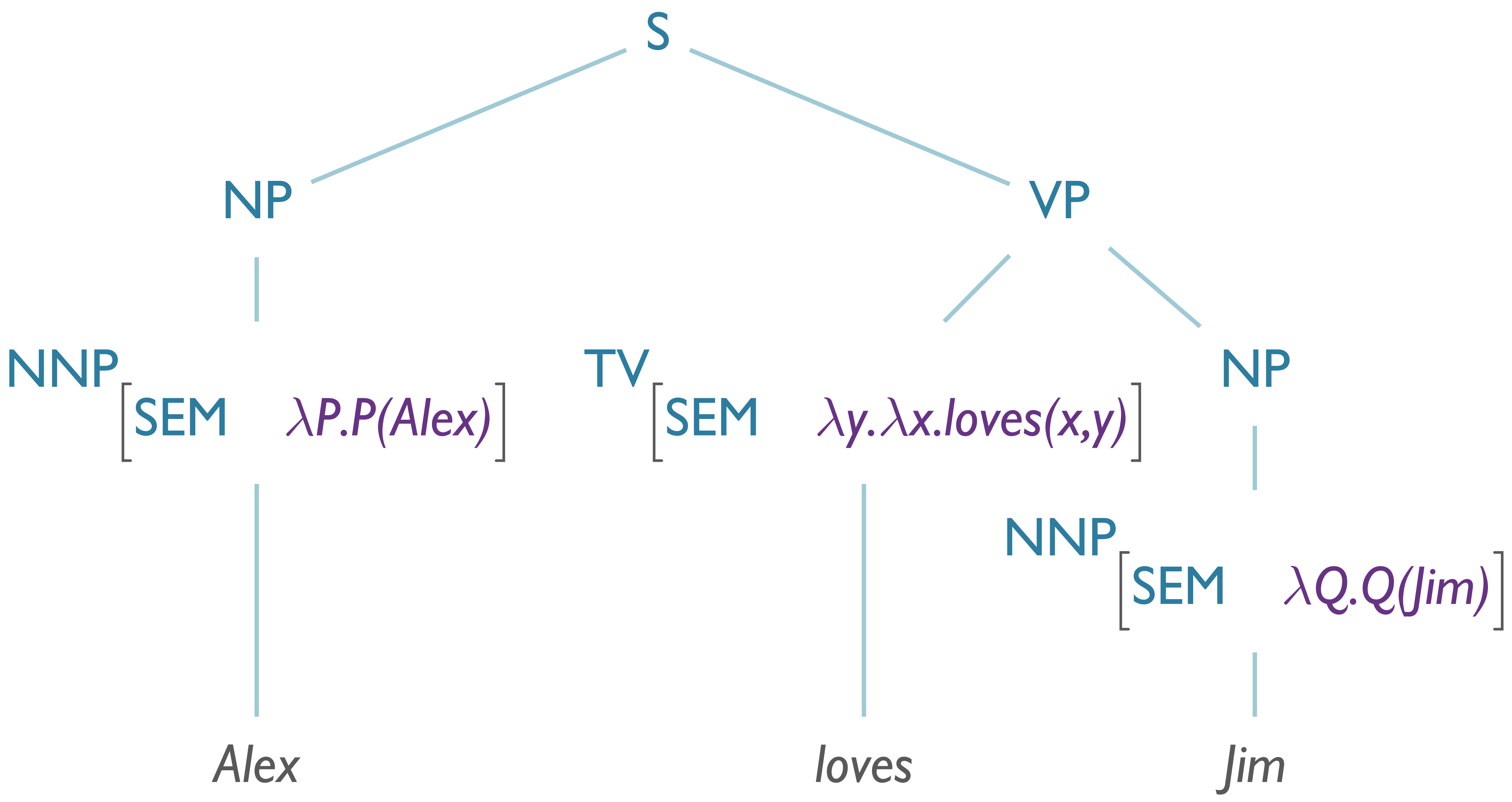
Transitive Verbs

Transitive Verbs

- So, if we want to say "*Alex loves Jim*" we would want `λy.λx.loves(x,y)`
- …but going in linear order, we have one arg to the left and one to the right.
- So, instead:
  - `λX y.X(λx.loves(x,y))`

Transitive Verbs

- So, if we want to say "*Alex loves Jim*" we would want `λy.λx.loves(x,y)`
- …but going in linear order, we have one arg to the left and one to the right.



Transitive Verbs

- TV(NP):
  - `λy.λx.loves(x,y) (λQ.Q(Alex))`

Transitive Verbs

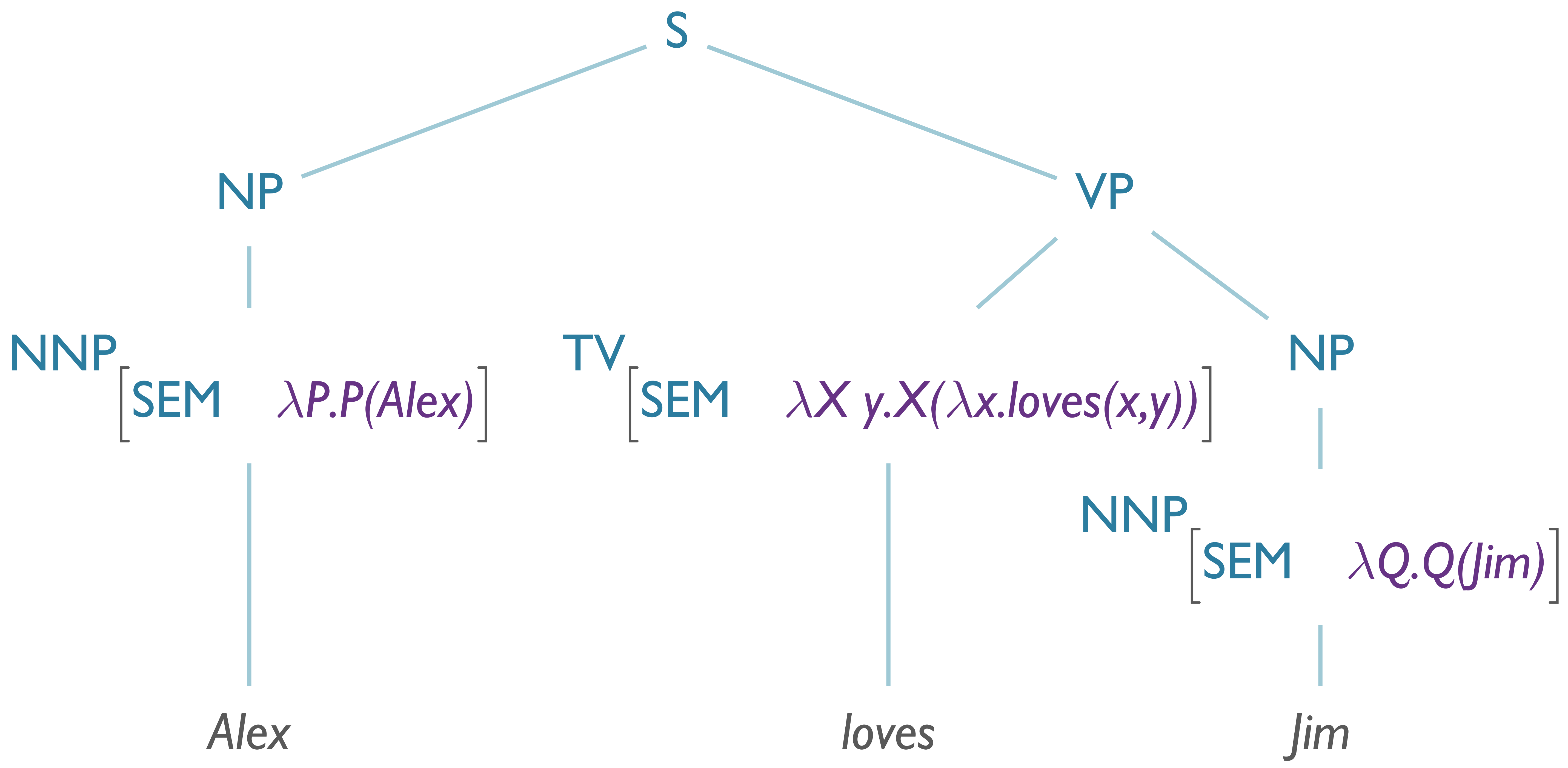
- TV(NP):
  - `λy.λx.loves(x,y) (λQ.Q(Alex))`
  - `λx.loves(x,λQ.Q(Alex))`

Transitive Verbs

- TV(NP):
  - `λy.λx.loves(x,y) (λQ.Q(Alex))`
  - `λx.loves(x,λQ.Q(Alex))`
    - → ***Error!*** We can't reduce Alex.

Transitive Verbs

- Instead: `λX y.X(λx.loves(x,y))`



Transitive Verbs

- TV(NP):
  - `λX y.X(λx.loves(x,y)) (λQ.Q(Jim))`

Transitive Verbs

- TV(NP):
  - `λX y.X(λx.loves(x,y)) (λQ.Q(Jim))` `λX takes (λQ.Q(Jim))`
  - `λy.(λQ.Q(Jim)(λx.loves(x,y))`

Transitive Verbs

- TV(NP):
  - λX y.X(λx.loves(x,y)) (λQ.Q(Jim))
  - λy.(λQ.Q(Jim)(λx.loves(x,y))
  - λy.(λx.loves(x,y)(Jim))

λX takes (λQ.Q(Jim))

λQ takes (λx.loves(x,y))

Transitive Verbs

- TV(NP):
  - `λX y.X(λx.loves(x,y)) (λQ.Q(Jim))`
  - `λy.(λQ.Q(Jim)(λx.loves(x,y))`
  - `λy.(λx.loves(x,y)(Jim))`
  - `λy.(loves(Jim,y))`

`λX takes (λQ.Q(Jim))`

`λQ takes (λx.loves(x,y))`

`λx takes (Jim)`

Transitive Verbs

- TV(NP):
  - `λX y.X(λx.loves(x,y)) (λQ.Q(Jim))`
  - `λy.(λQ.Q(Jim)(λx.loves(x,y))`
  - `λy.(λx.loves(x,y)(Jim))`
  - `λy.(loves(Jim,y))`

`λX takes (λQ.Q(Jim))`

`λQ takes (λx.loves(x,y))`

`λx takes (Jim)`

- NP(VP):
  - `λP.P(Alex)(λy.(loves(Jim,y))`

Transitive Verbs

- TV(NP):
  - λX y.X(λx.loves(x,y)) (λQ.Q(Jim))
  - λy.(λQ.Q(Jim)(λx.loves(x,y))
  - λy.(λx.loves(x,y)(Jim))
  - λy.(loves(Jim,y))

λX takes (λQ.Q(Jim))

λQ takes (λx.loves(x,y))

λx takes (Jim)

- NP(VP):
  - λP.P(Alex)(λy.(loves(Jim,y))
  - λy.(loves(Jim,y)(Alex)

λP takes (λy.(loves(Jim,y)

Transitive Verbs

- TV(NP):
  - λX y.X(λx.loves(x,y)) (λQ.Q(Jim)) — λX takes (λQ.Q(Jim))
  - λy.(λQ.Q(Jim)(λx.loves(x,y)) — λQ takes (λx.loves(x,y))
  - λy.(λx.loves(x,y)(Jim)) — λx takes (Jim)
  - λy.(loves(Jim,y))
- NP(VP):
  - λP.P(Alex)(λy.(loves(Jim,y)) — λP takes (λy.(loves(Jim,y))
  - λy.(loves(Jim,y)(Alex) — λy takes (Alex)
  - loves(Jim,Alex)

Converting to an Event

- "y loves x," Originally:
  - `λx y.x(λx.loves(x,y))`

Converting to an Event

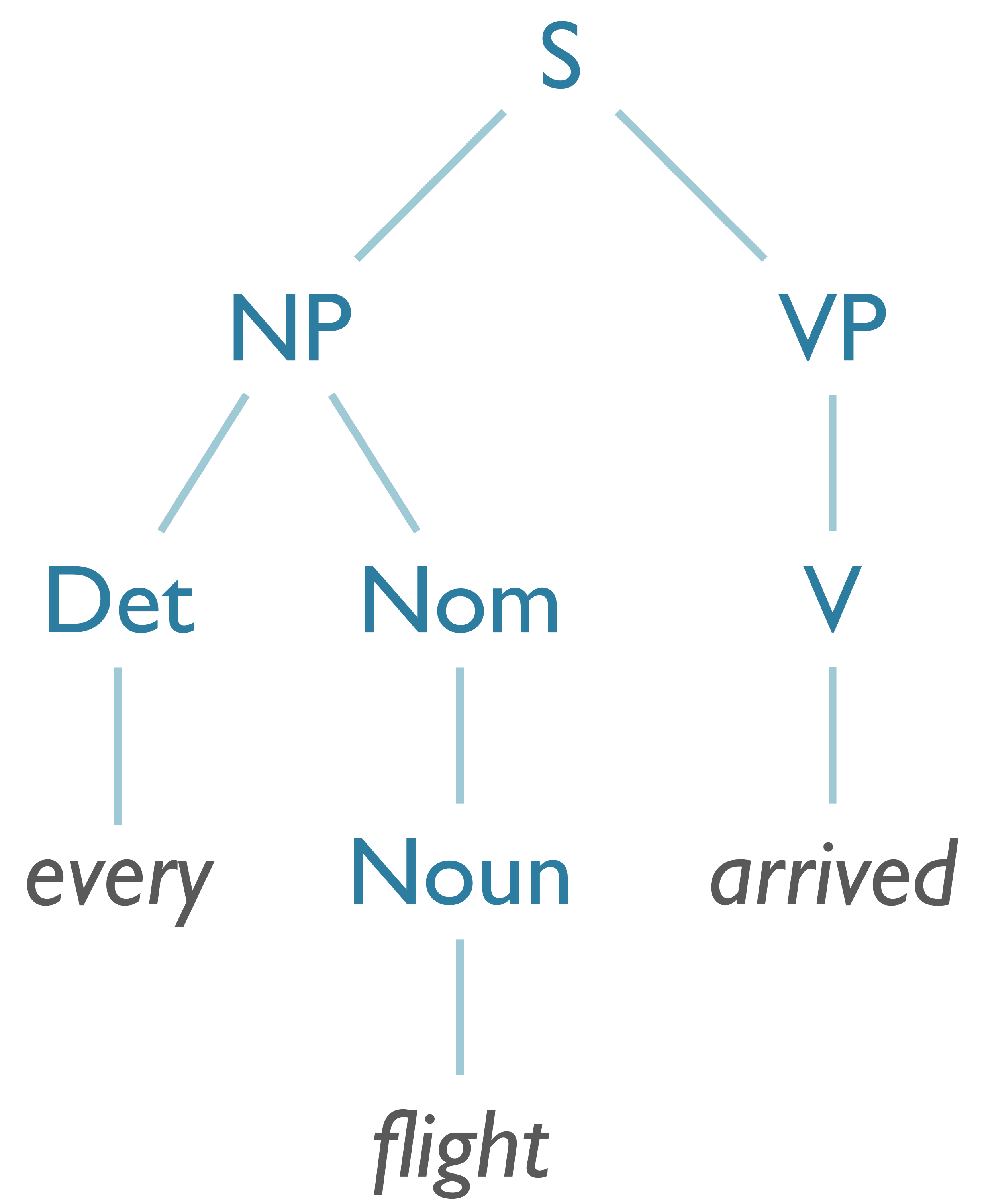
- “y loves x,” Originally:
  - `λX y.X(λx.loves(x,y))`

- as a Neo-Davidsonian event:
  - `λX y.X(λx.∃e love(e) ∧ lover(e,y) ∧ loved(e,x))`

Quantifiers & Scope

Semantic Analysis Example

- Basic model
  - Neo-Davidsonian event-style model
  - Complex quantification



- Example: *Every flight arrived*

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

“*Every flight arrived*”

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

"*Every flight arrived*"

- First intuitive approach:
  - Every flight = $\forall x\ Flight(x)$ ✗
    - "Everything is a flight"

"Every flight arrived"

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

“*Every flight arrived*”

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

"*Every flight arrived*"

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

“*Every flight arrived*”

"*Every flight arrived*"

- "*Every flight*" is:

"*Every flight arrived*"

- "*Every flight*" is:
  - $\lambda Q.\forall x \, Flight(x) \Rightarrow Q(x)$

"*Every flight arrived*"

- "*Every flight*" is:
  - $\lambda Q.\forall x\ Flight(x) \Rightarrow Q(x)$
- …so what is the representation for "**every**"?

"*Every flight arrived*"

- "*Every flight*" is:
  - $\lambda Q.\forall x\, Flight(x) \Rightarrow Q(x)$
- …so what is the representation for "**every**"?
  - $\lambda P.\lambda Q.\forall x\, P(x) \Rightarrow Q(x)$

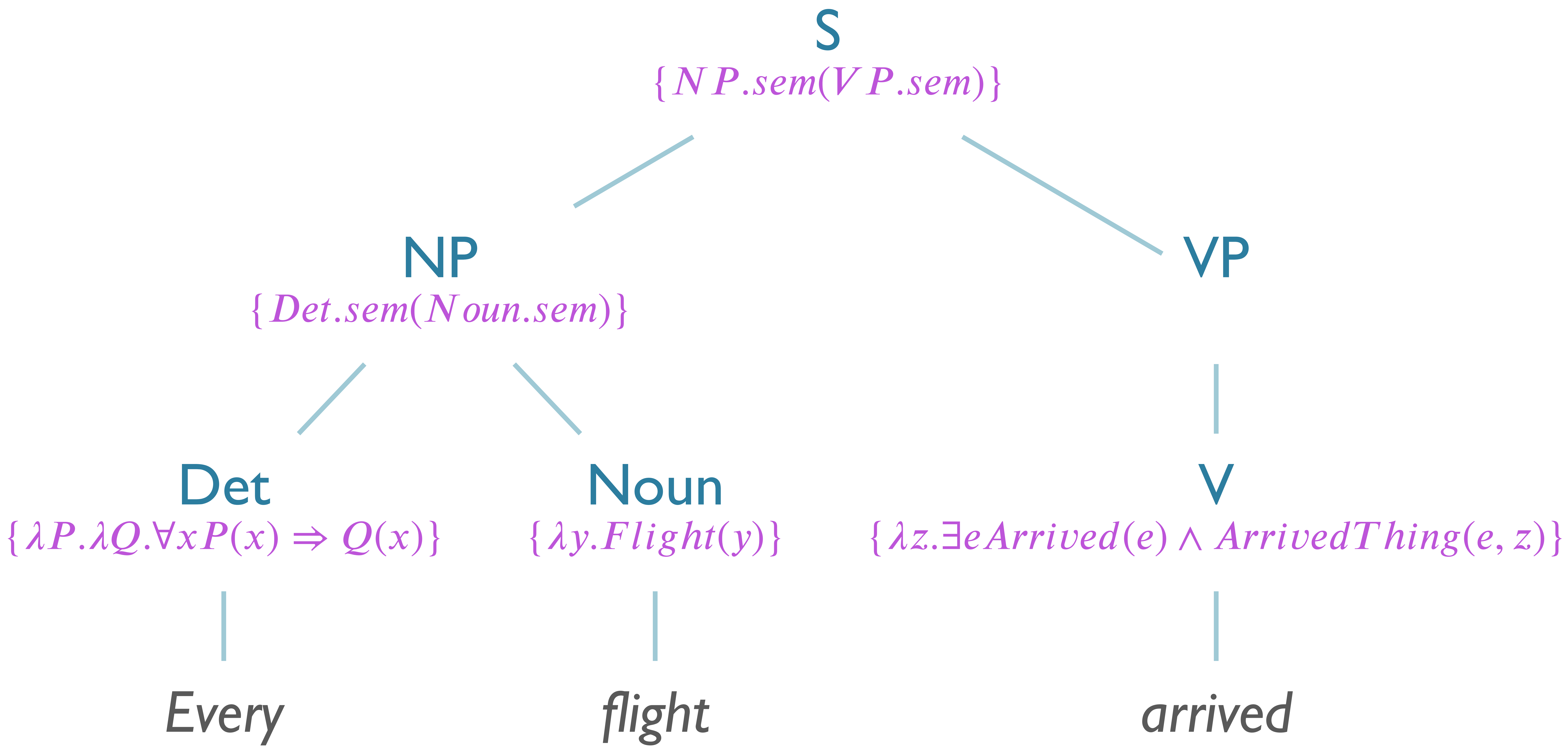
"*A flight arrived*"

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

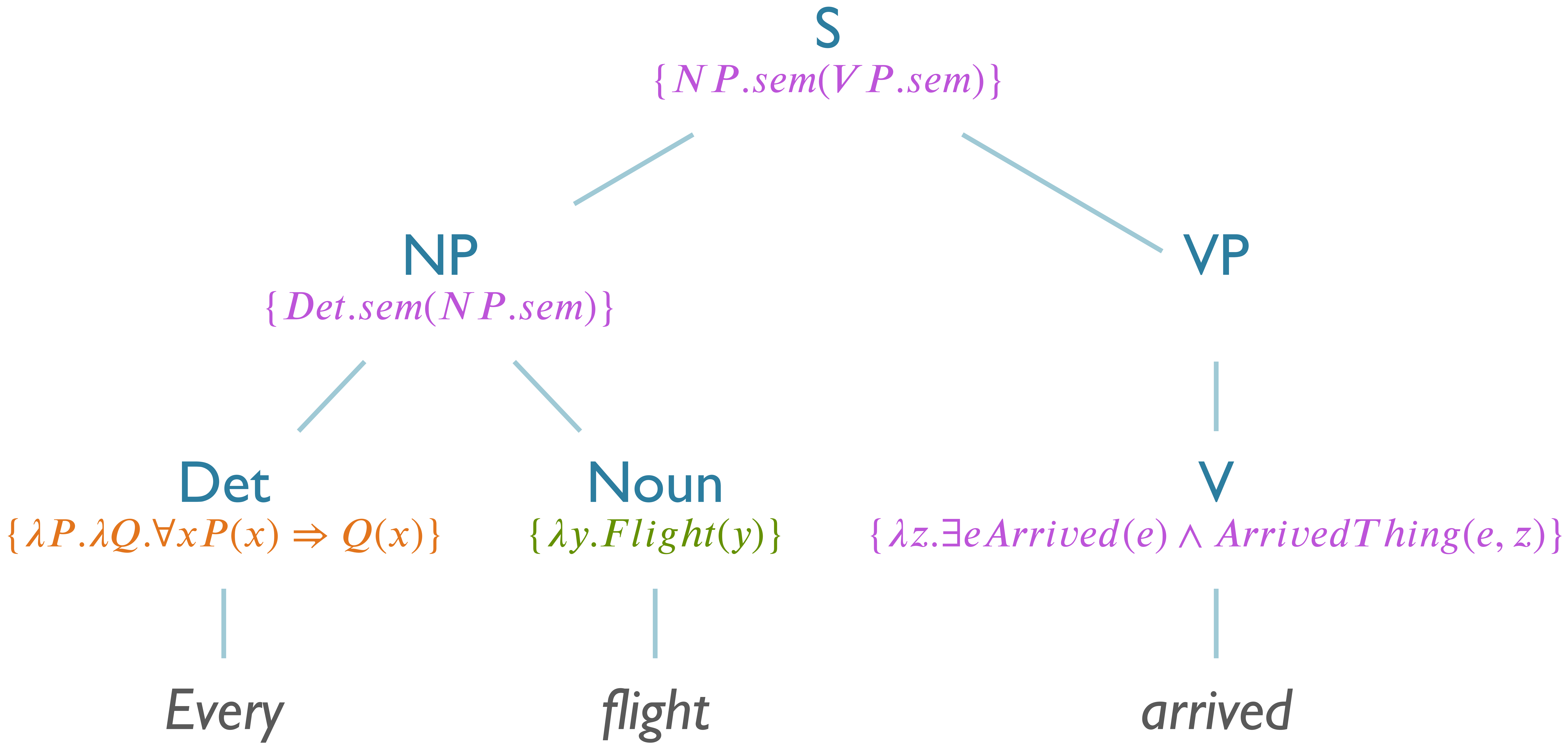
Creating Attachments

"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	$\rightarrow Det\ Nom$	$\{\ Det.sem(Nom.sem)\ \}$

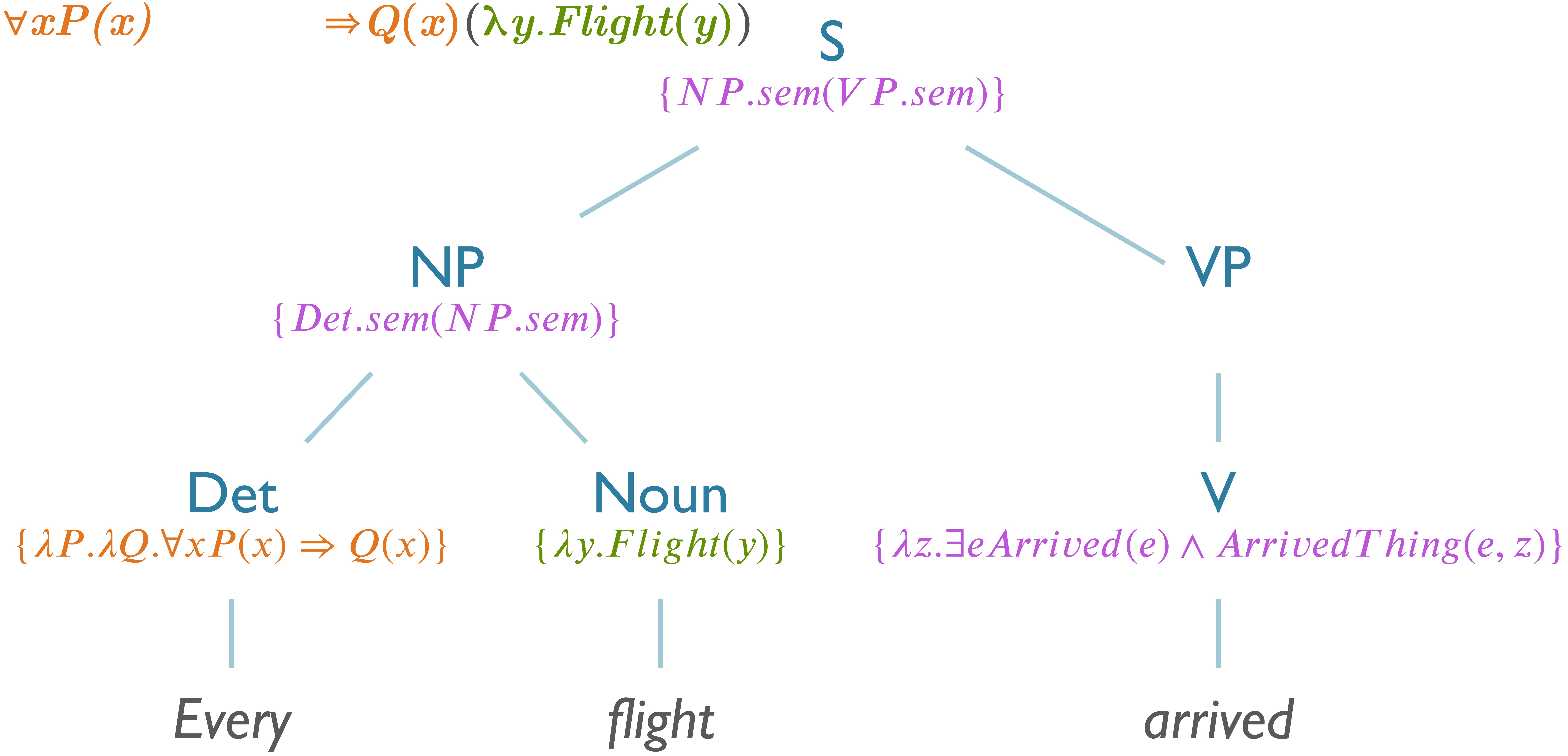

S
{NP.sem(VP.sem)}
NP
{Det.sem(Noun.sem)}
VP
Det
{λP.λQ.∀xP(x) ⇒ Q(x)}
Noun
{λy.Flight(y)}
V
{λz.∃eArrived(e) ∧ ArrivedThing(e, z)}
Every
flight
arrived

$$NP \rightarrow Det.sem(NP.sem)$$



$$NP \rightarrow Det.sem(NP.sem)$$

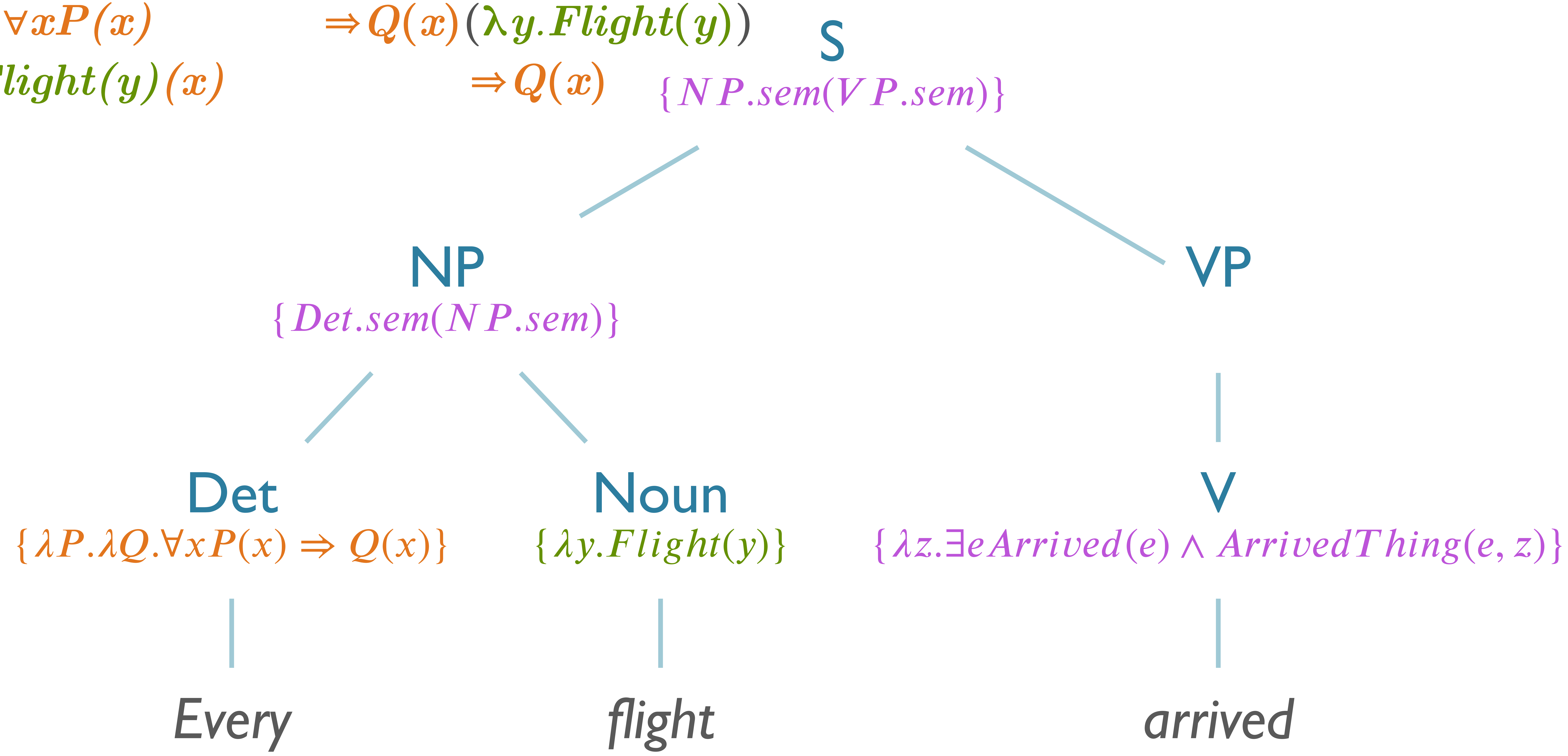
$$\lambda P.\lambda Q.\forall x P(x) \Rightarrow Q(x)(\lambda y.Flight(y))$$



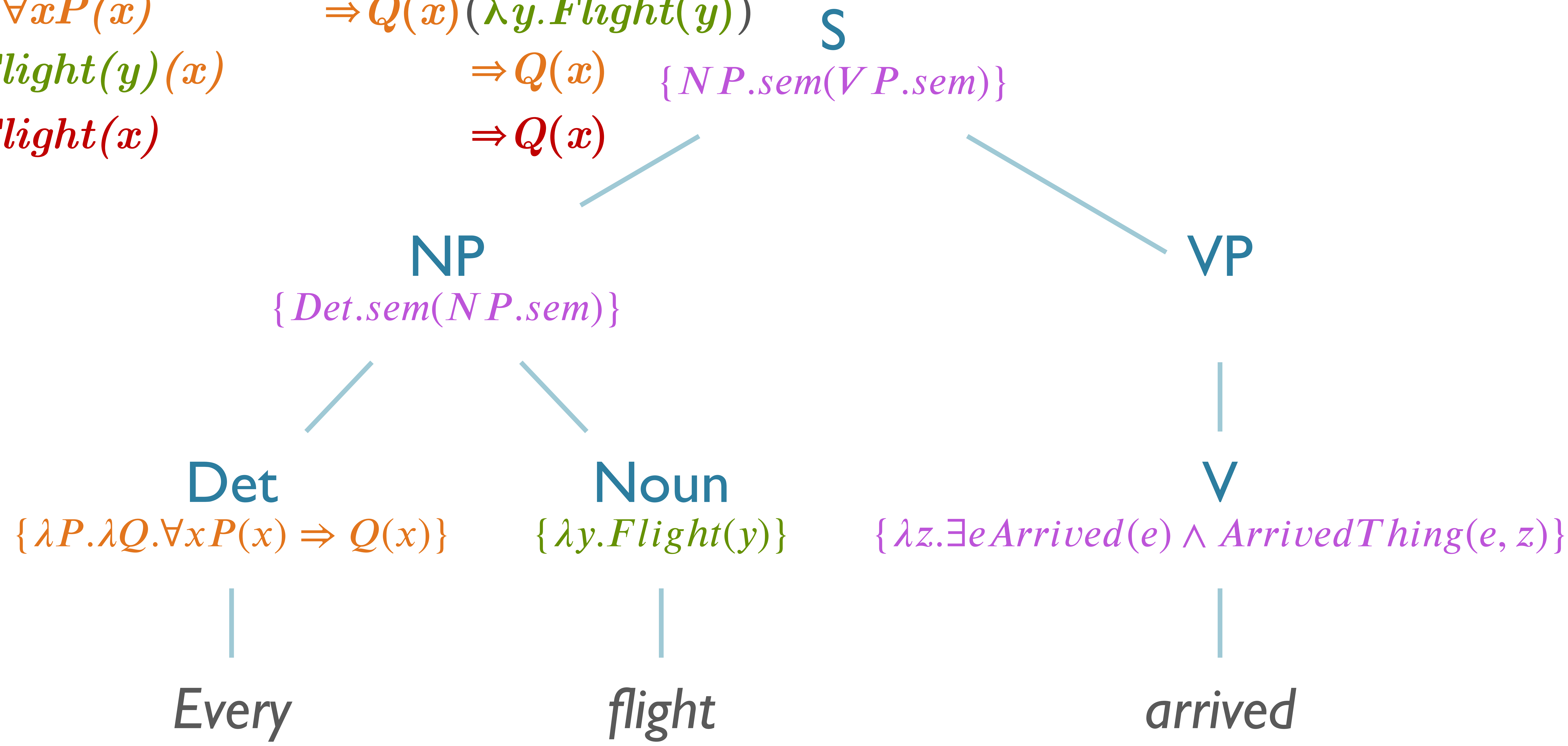
$$NP \rightarrow Det.sem(NP.sem)$$

$$\lambda P.\lambda Q.\forall x P(x) \Rightarrow Q(x)(\lambda y.Flight(y))$$

$$\lambda Q.\forall x \lambda y.Flight(y)(x) \Rightarrow Q(x)$$

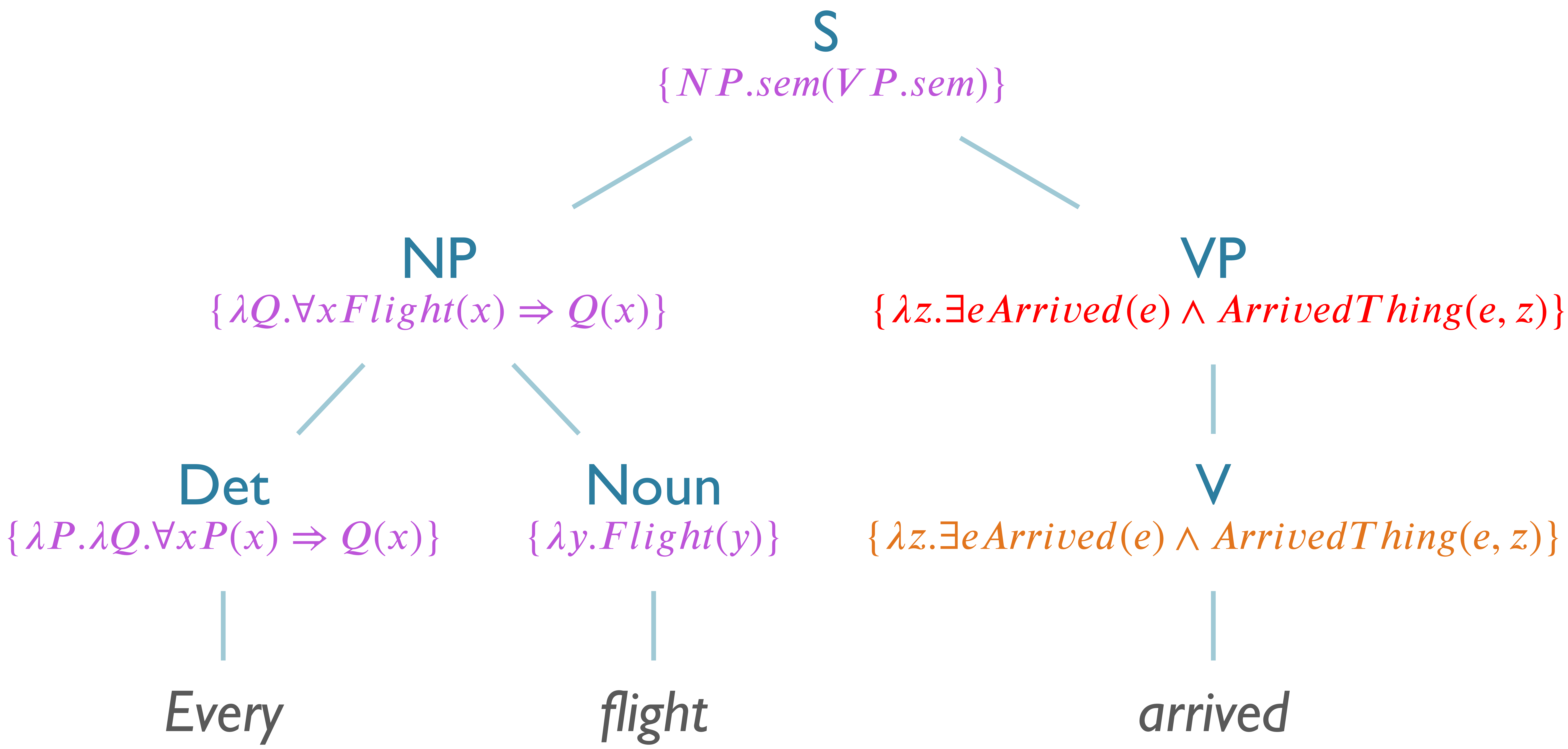


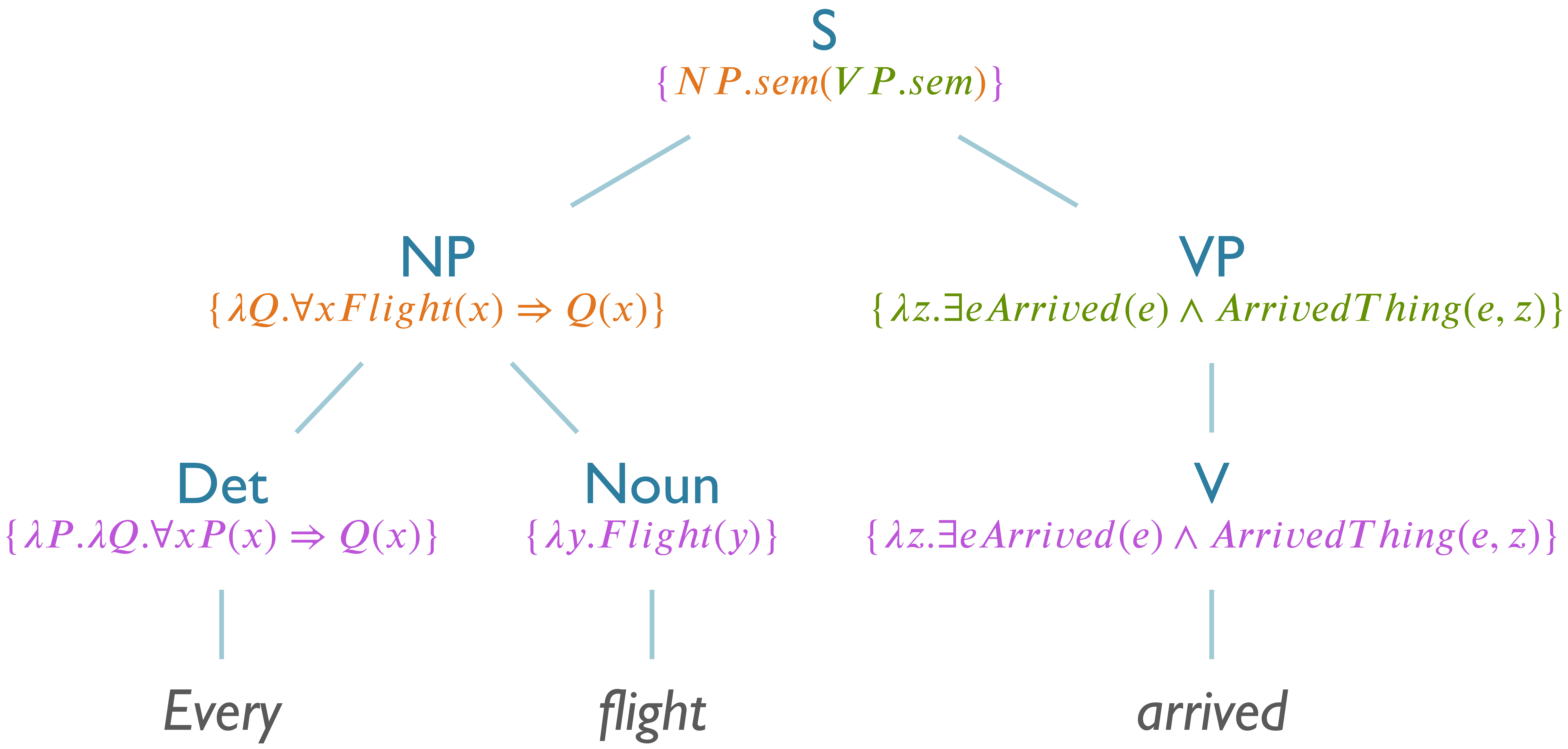
$$\begin{aligned}
NP &\rightarrow Det.sem(NP.sem)\\
\lambda P.\lambda Q.\forall x P(x) &\Rightarrow Q(x)(\lambda y.Flight(y))\\
\lambda Q.\forall x \lambda y.Flight(y)(x) &\Rightarrow Q(x)\\
\lambda Q.\forall x Flight(x) &\Rightarrow Q(x)
\end{aligned}$$



$$NP \rightarrow Det.sem(NP.sem)$$
$$\lambda P.\lambda Q.\forall x P(x) \Rightarrow Q(x)(\lambda y.Flight(y))$$
$$\lambda Q.\forall x \lambda y.Flight(y)(x) \Rightarrow Q(x)$$
$$\lambda Q.\forall x Flight(x) \Rightarrow Q(x)$$

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

S

$\{NP.sem(VP.sem)\}$

NP

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

VP

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

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

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

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

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

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

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

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

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

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

S

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

NP

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

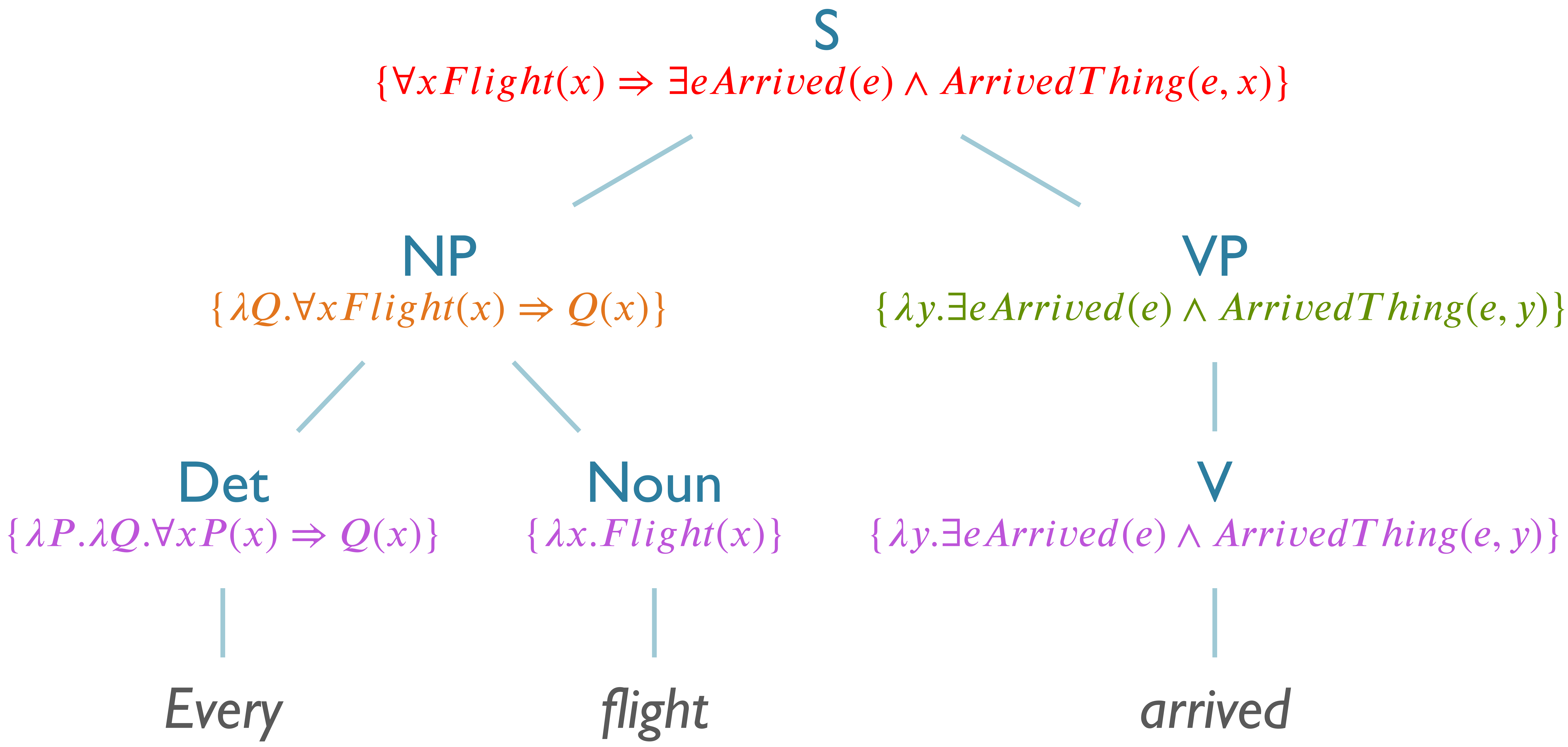
VP

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

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

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

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


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

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

…we'll step through this next time.

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
    - <u>Combine</u> elements, don't introduce new ones

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