Computational Semantics

LING 571 — Deep Processing for NLP

Shane Steinert-Threlkeld

Semantics in the News

https://apnews.com/article/supreme-court-mandatory-minimum-sentencing-drug-crimes-235b5dd23cf70bead9f8f23d659a572d

Semantics in the News

offenders. A defendant satisfies § 3553(f)(1), as amended, if he "does not have-(A) more than 4 criminal history points, excluding any criminal history points resulting from a 1-point offense, as determined under the sentencing guidelines; (B) a prior 3-point offense, as determined under the sentencing guidelines; *and* (C) a prior 2-point violent offense, as determined under the sentencing guidelines." 18 U.S.C. § 3553(f)(1) (emphasis added).

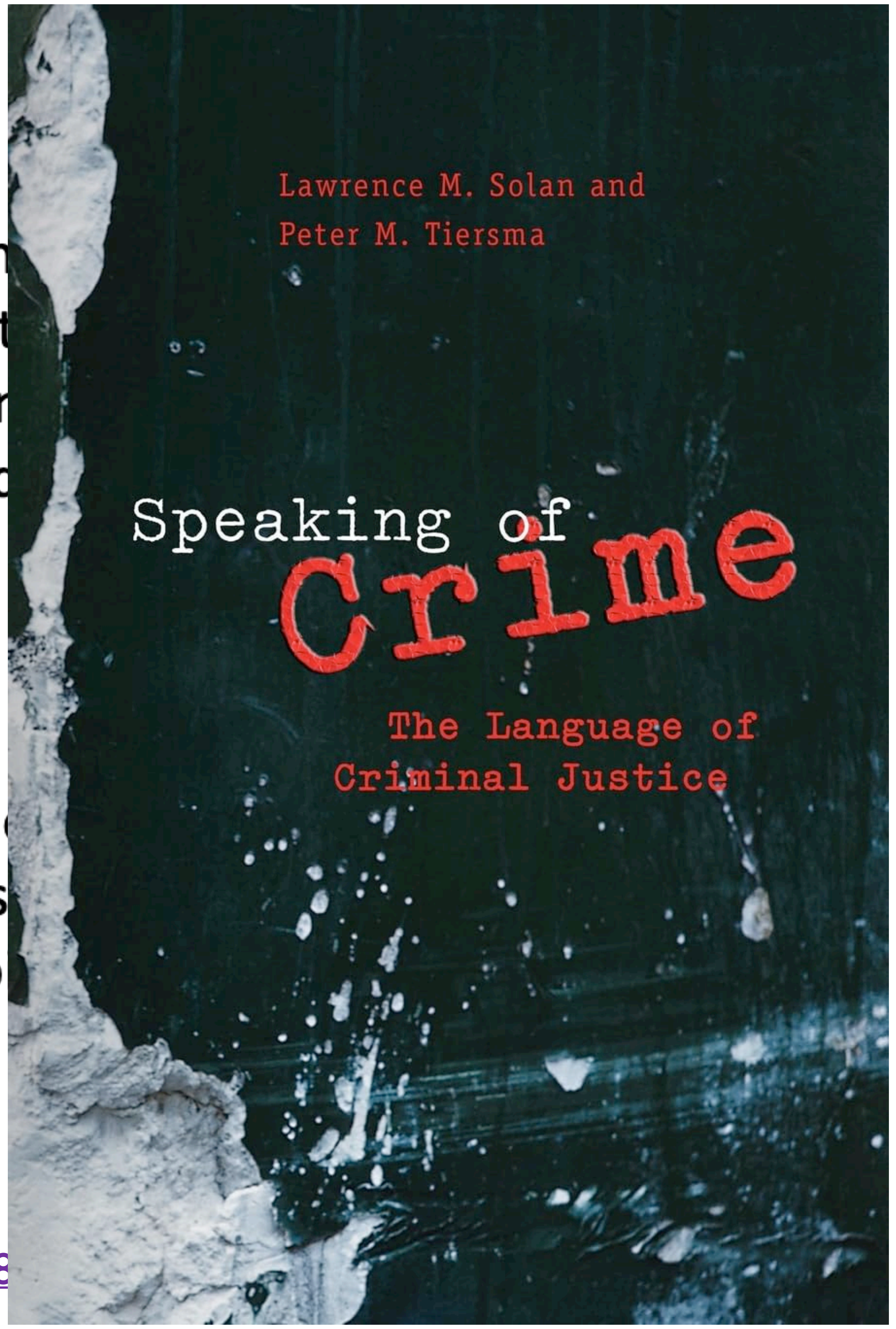
The question presented is whether the "and" in 18 U.S.C. § 3553(f)(1) means "and," so that a defendant satisfies the provision so long as he does not have (A) more than 4 criminal history points, (B) a 3-point offense, *and* (C) a 2-point offense (as the Ninth Circuit holds), or whether the "and" means "or," so that a defendant satisfies the provision so long as he does not have (A) more than 4 criminal history points, (B) a 3- point offense, *or* (C) a 2-point violent offense (as the Seventh and Eighth Circuits hold).

https://apnews.com/article/supreme-court-mandatory-minimum-sentencing-drug-crimes-235b5dd23cf70bead9f8f23d659a572d

Semantics in the News

offenders. A defendant satisfies § 3553(f)(1), as amended, if he "does not have-(A) m criminal history points, excluding any criminal history points resulting from a 1-point determined under the sentencing guidelines; (B) a prior 3-point offense, as determin the sentencing guidelines; *and* (C) a prior 2-point violent offense, as determined und sentencing guidelines." 18 U.S.C. § 3553(f)(1) (emphasis added).

The question presented is whether the "and" in 18 U.S.C. § 3553(f)(1) means that a defendant satisfies the provision so long as he does not have (A) more than 4 history points, (B) a 3-point offense, *and* (C) a 2-point offense (as the Ninth Circuit h whether the "and" means "or," so that a defendant satisfies the provision so long as not have (A) more than 4 criminal history points, (B) a 3- point offense, *or* (C) a 2-po offense (as the Seventh and Eighth Circuits hold).



https://apnews.com/article/supreme-court-mandatory-minimum-sentencing-drug-crimes-235b5dd23cf70bead9f8

Python Nugget of the Week

- Sequential logical operators:

```
if (
    # if lhs is not in the cell, or if the current logprob is greater than the current logprob
    # add a backpointer to the cell
    # because "or" operates sequentially, the lhs lookup in the right disjunct is guaranteed
    # to not throw a KeyError
    lhs not in chart[i][j]
    or chart[i][j][lhs].logprob < current_logprob
):
    chart[i][j][lhs] = Backpointer(
        lhs, k, rhs[0], rhs[1], current_logprob
    )
```

- An alternative here: *defaultdict* with default values for BP
- "or" also therefore useful for conditional assignment (e.g. `x = False or 42`)

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 with COVID-19:
  - 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

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**:

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

FOL Syntax Summary

Formula	→	*AtomicFormula*	***Connective***	→	∧ | ∨ | ⇒
	|	*Formula Connective Formula*	***Quantifier***	→	∀ | ∃
	|	*Quantifier Variable, … Formula*	***Constant***	→	*VegetarianFood* | *Maharani* | …
	|	¬ *Formula*	***Variable***	→	*x* | *y* | …
	|	*(Formula)*	***Predicate***	→	*Serves* | *Near* | …
AtomicFormula	→	*Predicate(Term,…)*	***Function***	→	*LocationOf* | *CuisineOf* | …
Term	→	*Function(Term,…)*			
	|	*Constant*			
	|	*Variable*			

J&M p. 556 (3rd ed. F.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

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 $\mathcal{D}$

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***={ $\langle a,f \rangle$, $\langle c,f \rangle$, $\langle c,g \rangle$, $\langle b,e \rangle$, $\langle d,f \rangle$, $\langle d,g \rangle$ }
Serves	Med serves eclectic Rio serves Mexican Frasca serves Italian	***Serves***={ $\langle c,f \rangle$, $\langle f,i \rangle$, $\langle e,h \rangle$ }

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

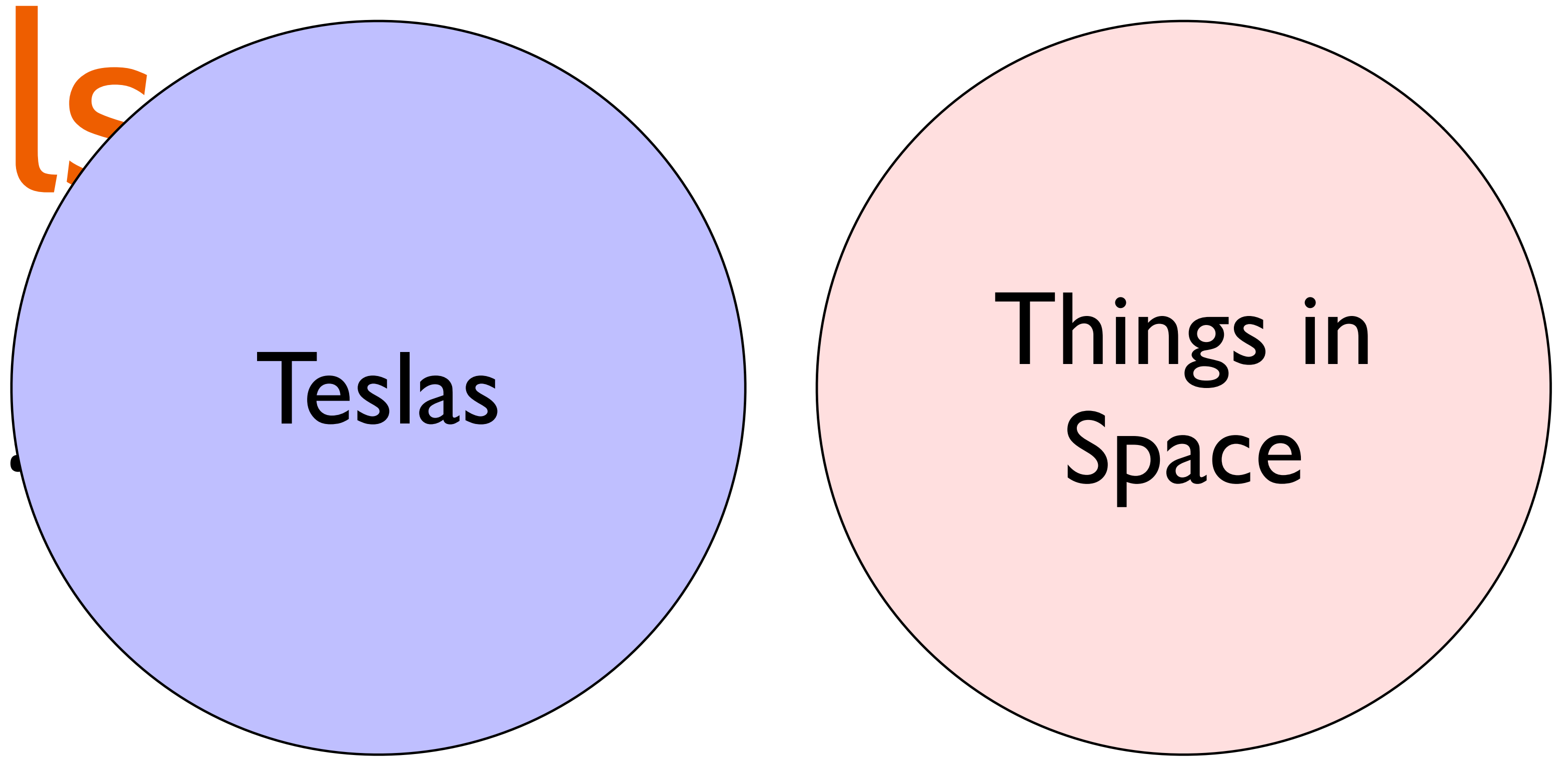
Ambiguity & Models

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

State of known Universe: 02/05/2018

Ambiguity & Models

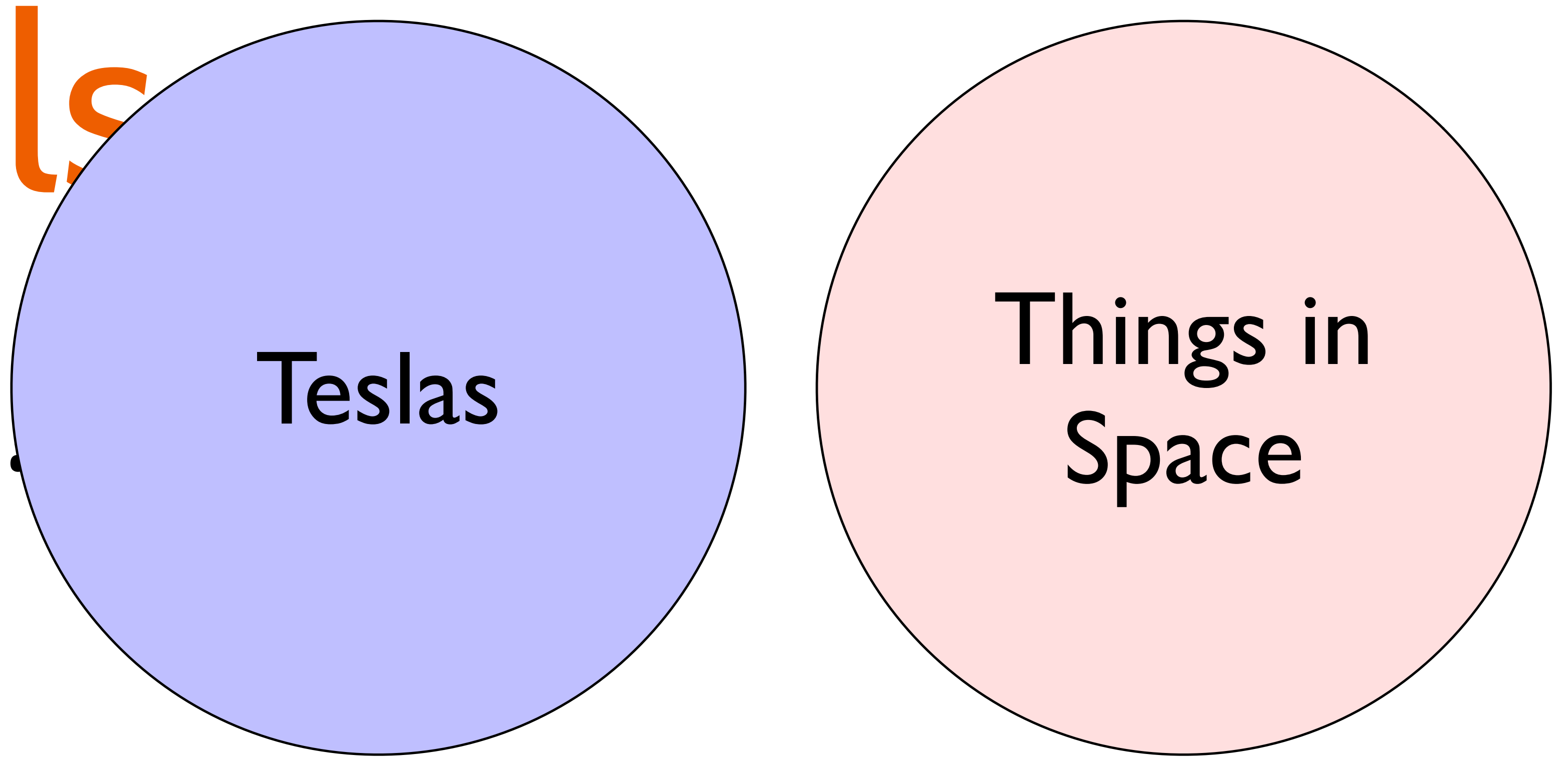


- *"Every Tesla is powered by a battery."* — Ambiguous.
  - $\forall x. Tesla(x) \Rightarrow (\exists(y). Battery(y) \wedge Powers(y, x))$
  - $\exists(y). Battery(y) \wedge (\forall x. Tesla(x) \Rightarrow Powers(y, x))$

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

State of known Universe: 02/05/2018

Ambiguity & Models



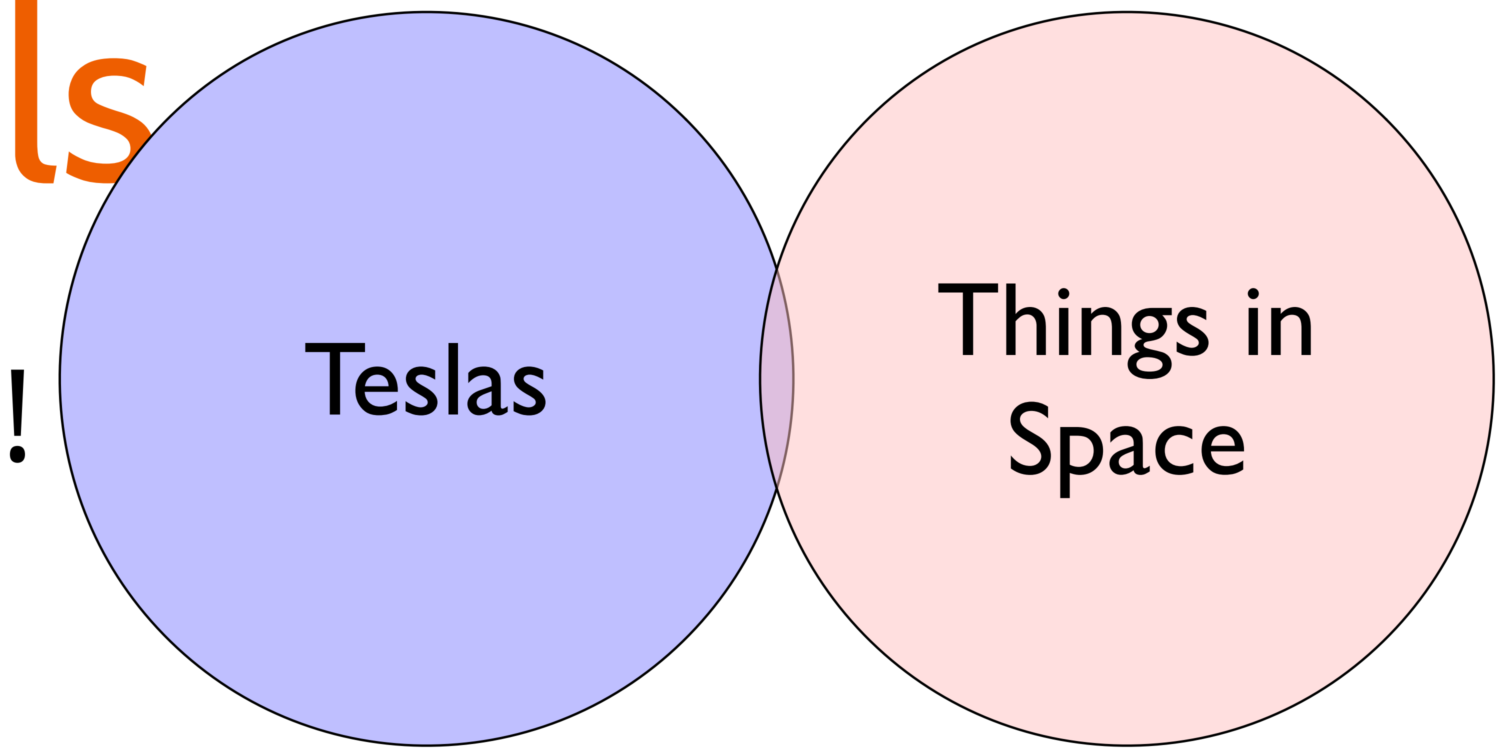
- *"Every Tesla is powered by a battery."* — Ambiguous.
  - $\forall x.\,Tesla(x) \Rightarrow (\exists(y).\,Battery(y) \wedge Powers(y, x))$
  - $\exists(y).\,Battery(y) \wedge (\forall x.\,Tesla(x) \Rightarrow Powers(y, x))$

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

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

State of known Universe: 02/06/2018

Ambiguity & Models



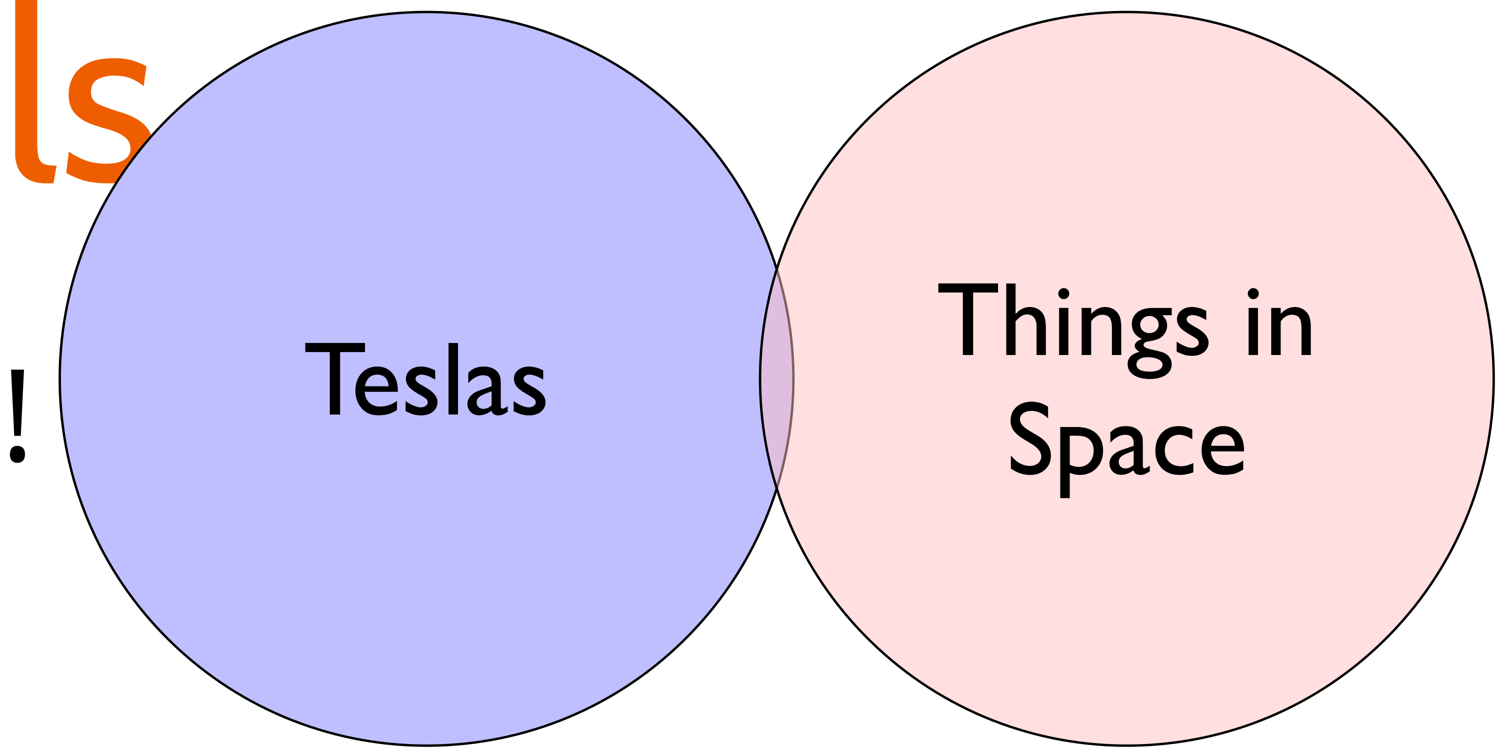
- *"Every Tesla is powered by a battery."* — Ambiguous!
  - $\forall x. Tesla(x) \Rightarrow (\exists(y). Battery(y) \wedge Powers(y, x))$
  - $\exists(y). Battery(y) \wedge (\forall x. Tesla(x) \Rightarrow Powers(y, x))$

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

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

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

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

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!

Ambiguity of the Week

Ambiguity of the Week

- Derivative of an alleged Groucho Marx-ism:

Ambiguity of the Week

- Derivative of an alleged Groucho Marx-ism:
- In the US, a woman gives birth every fifteen minutes.

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.

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.

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.)

Scope Ambiguity in the News

Scope Ambiguity in the News

- “Boston voters have elected City Councilor Michelle Wu as mayor, the city's first woman and person of color elected to the post.”
  - Source: https://www.npr.org/2021/11/02/1051720391/boston-mayor-michelle-wu-elected

Scope Ambiguity in the News

- "Boston voters have elected City Councilor Michelle Wu as mayor, the city's first woman and person of color elected to the post."
  - Source: https://www.npr.org/2021/11/02/1051720391/boston-mayor-michelle-wu-elected
- What do people think this says about Wu?

Scope Ambiguity in the News

- "Boston voters have elected City Councilor Michelle Wu as mayor, the city's first woman and person of color elected to the post."
  - Source: https://www.npr.org/2021/11/02/1051720391/boston-mayor-michelle-wu-elected
- What do people think this says about Wu?
- What's a scope ambiguity here?

Scope Ambiguity in the News

- “Boston voters have elected City Councilor Michelle Wu as mayor, the city's first woman and person of color elected to the post.”
  - Source: https://www.npr.org/2021/11/02/1051720391/boston-mayor-michelle-wu-elected
- What do people think this says about Wu?
- What’s a scope ambiguity here?
- ‘first’ > ‘and’ vs ‘and’ > ‘first’
  - Intended is actually the latter: first woman *and* first POC

Scope Ambiguity in the News

- “Boston voters have elected City Councilor Michelle Wu as mayor, the city's first woman and person of color elected to the post.”
  - Source: https://www.npr.org/2021/11/02/1051720391/boston-mayor-michelle-wu-elected
- What do people think this says about Wu?
- What’s a scope ambiguity here?
- ‘first’ > ‘and’ vs ‘and’ > ‘first’
  - Intended is actually the latter: first woman *and* first POC
- [sidebar: Stanford Parser totally botches it]

Scope Ambiguity in the News

- “Boston voters have elected City Councilor Michelle Wu as mayor, the city's first woman and person of color elected to the post.”
  - Source: https://www.npr.org/2021/11/02/1051720391/boston-mayor-michelle-wu-elected
- What do people think this says about Wu?
- What’s a scope ambiguity here?
- ‘first’ > ‘and’ vs ‘and’ > ‘first’
  - Intended is actually the latter: first woman *and* first POC
- [sidebar: [Stanford Parser](#) totally botches it]

Parse

```
(ROOT
  (S
    (NP (NNP Boston) (NNS voters))
    (VP (VBP have)
      (VP (VBN elected)
        (NP
          (NP
            (NP (NNP City) (NNP Councilor) (NNP Michelle) (NNP Wu))
            (PP (IN as)
              (NP (NN mayor))))
          (, ,)
          (NP
            (NP (DT the) (NN city) (POS 's))
            (JJ first) (NN woman))
          (CC and)
          (NP
            (NP (NN person))
            (PP (IN of)
              (NP
                (NP (NN color))
                (VP (VBN elected)
                  (PP (IN to)
                    (NP (DT the) (NN post))))))))))
    (. .)))
```

Integrating Semantics into Syntax

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

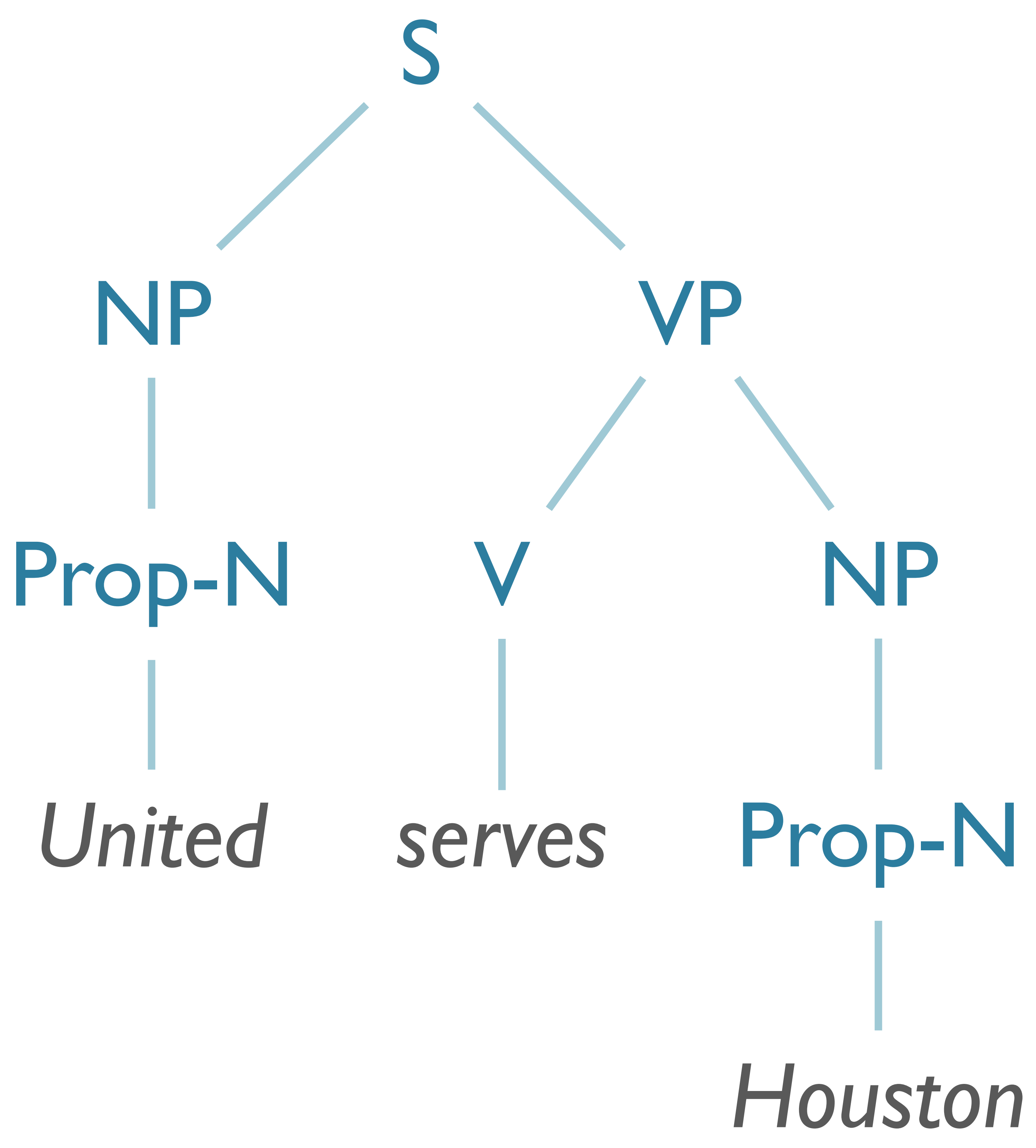
Integrating Semantics into Syntax

Integrating Semantics into Syntax

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

Simple Example

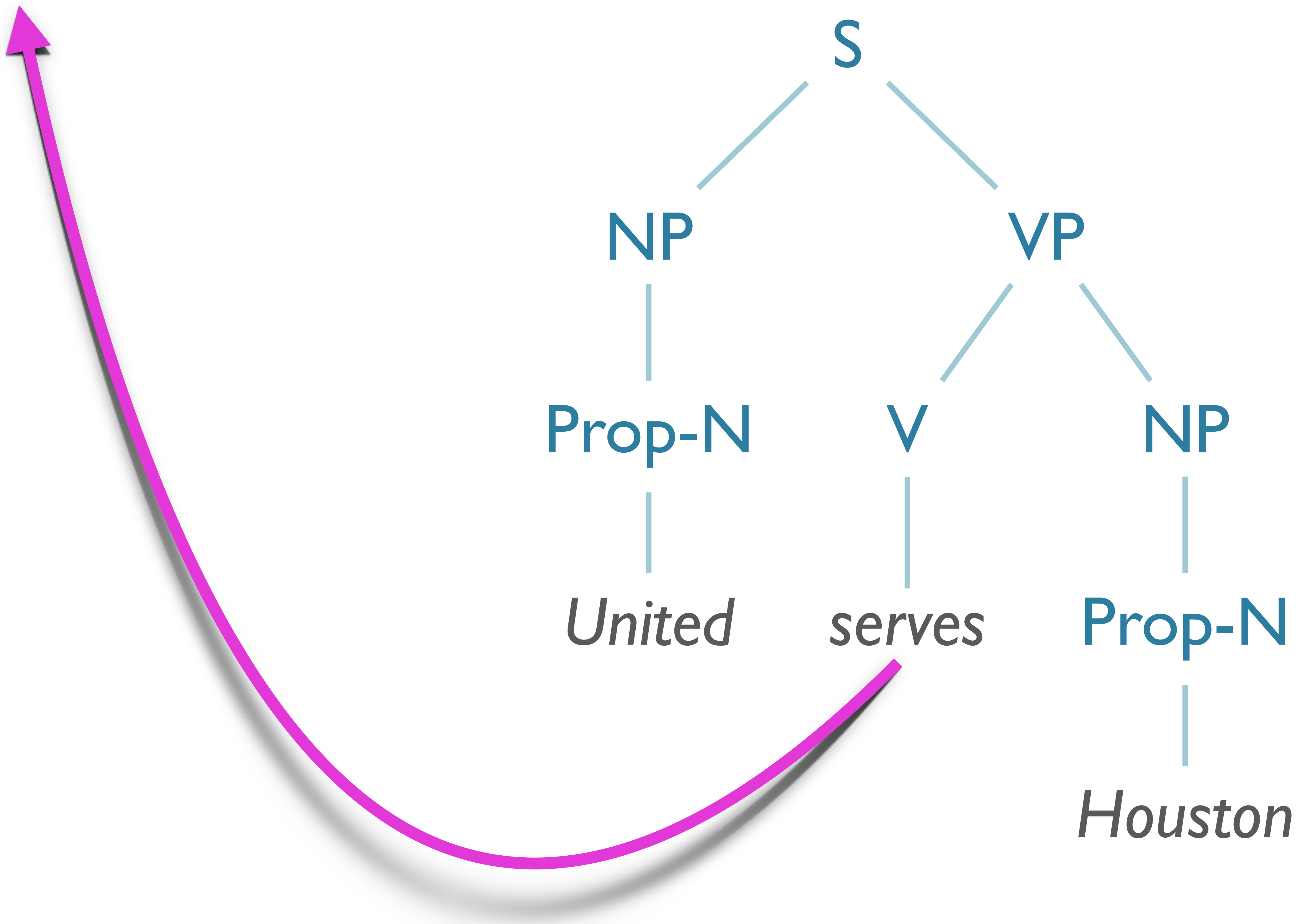
- *United serves Houston*



Simple Example

- *United serves Houston*

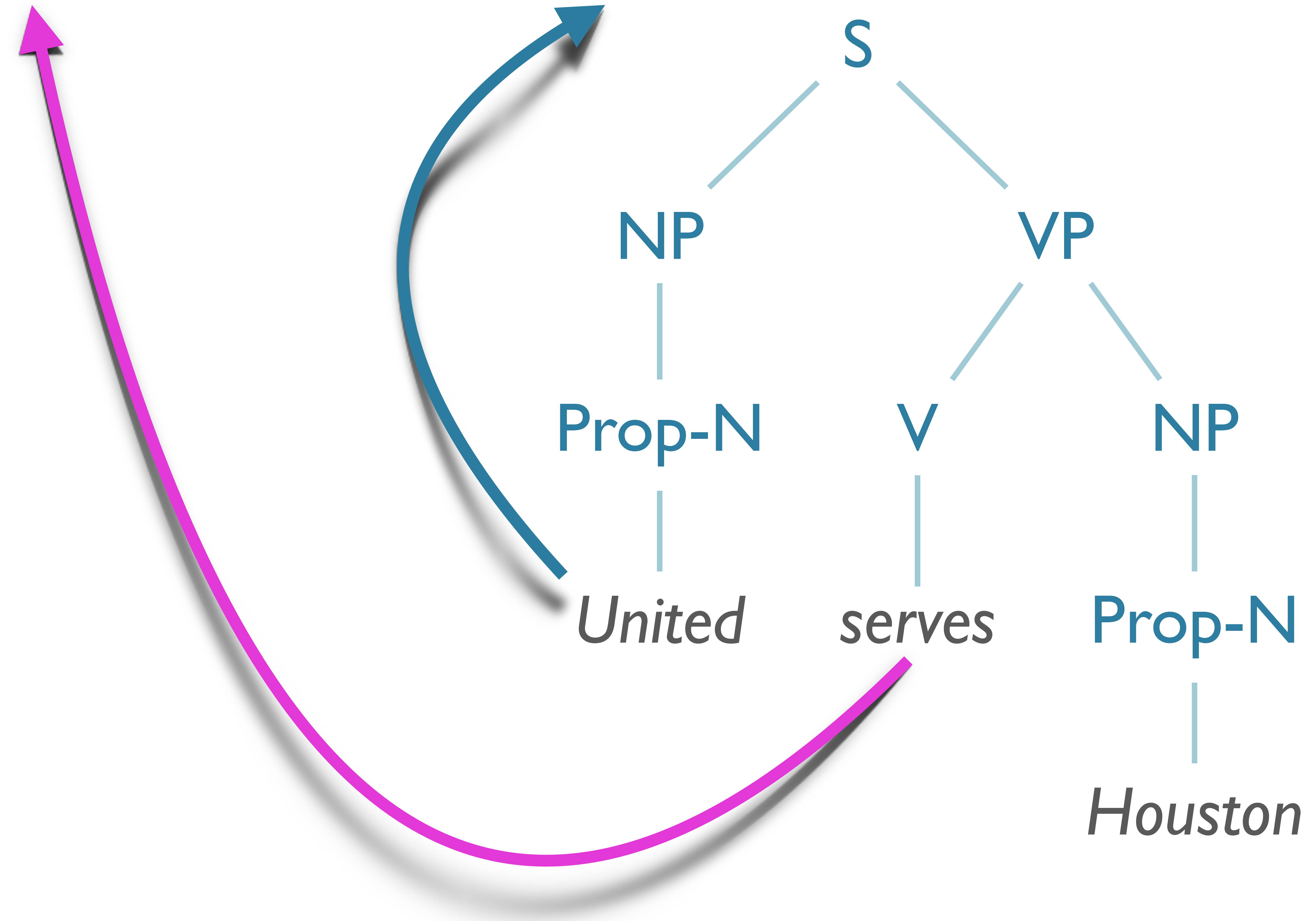
  $\exists e \left(\textcolor{magenta}{Serving}(e) \land \right.$



Simple Example

- *United serves Houston*

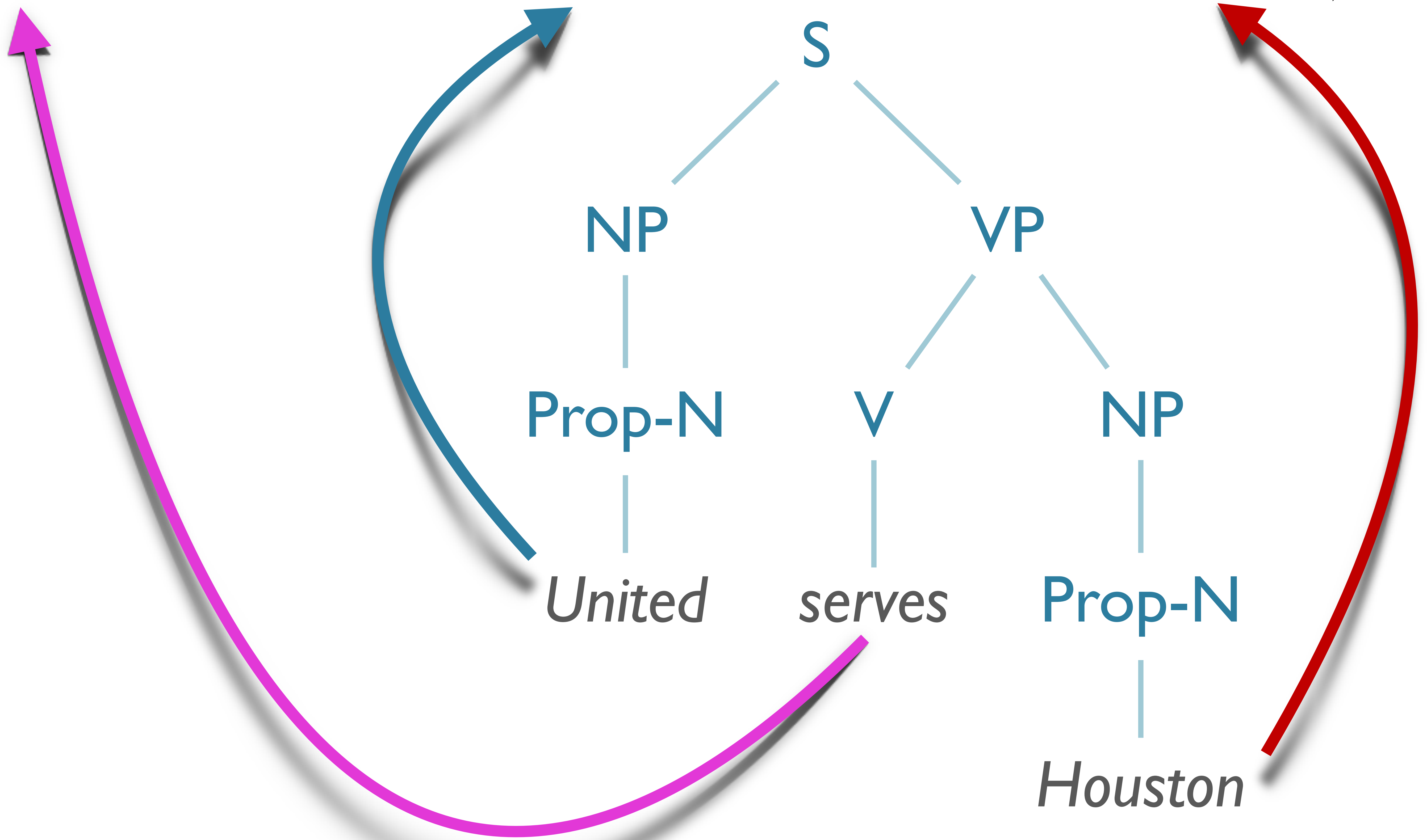
$\exists e\big(Serving(e) \land Server(e, United) \land$



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:

$$A \rightarrow a_1, \ldots, a_n \{f(a_j.\text{sem}, \ldots\ a_k.\text{sem})\}$$

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* → ‘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

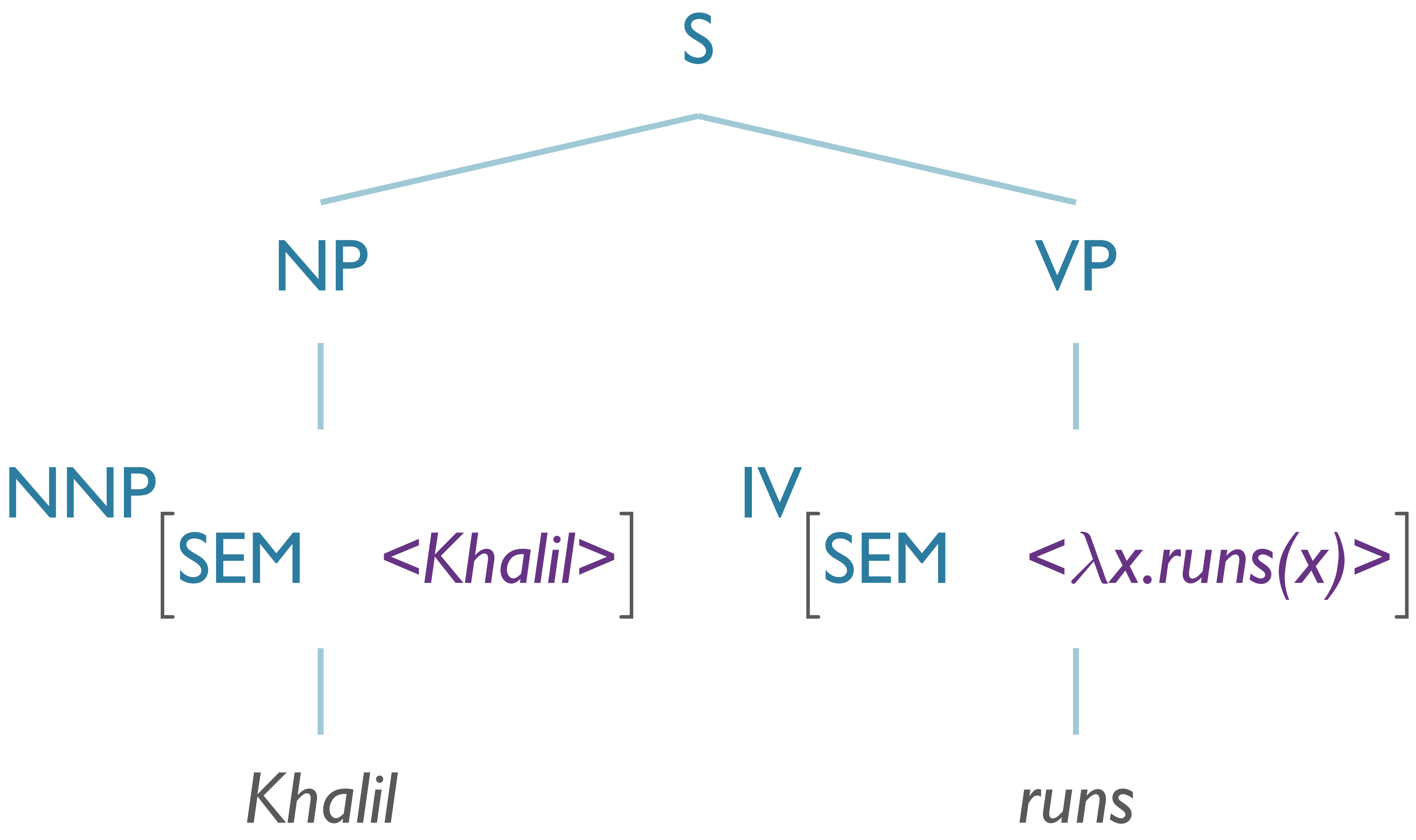
- Our instinct for names is to just use the constant:
  - `NNP[SEM=<Khalil>] → 'Khalil'`
- However, we will want to apply our λ-closures left-to-right consistently.

```
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 will want to apply our λ-closures left-to-right consistently.

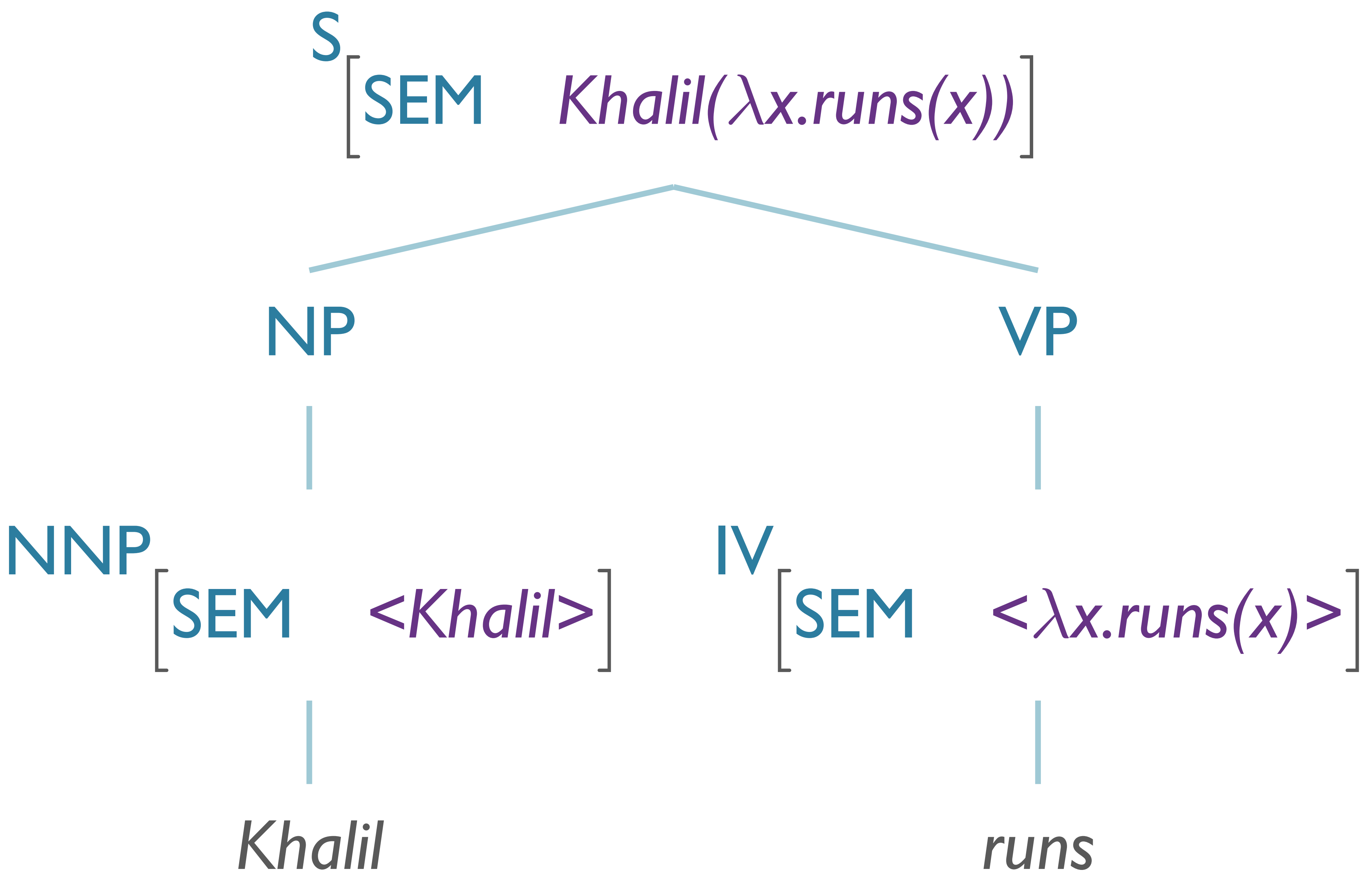
S[SEM=np?(vp?)] → NP[SEM=np?] VP[SEM=vp?]



Proper Nouns & Intransitive Verbs

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

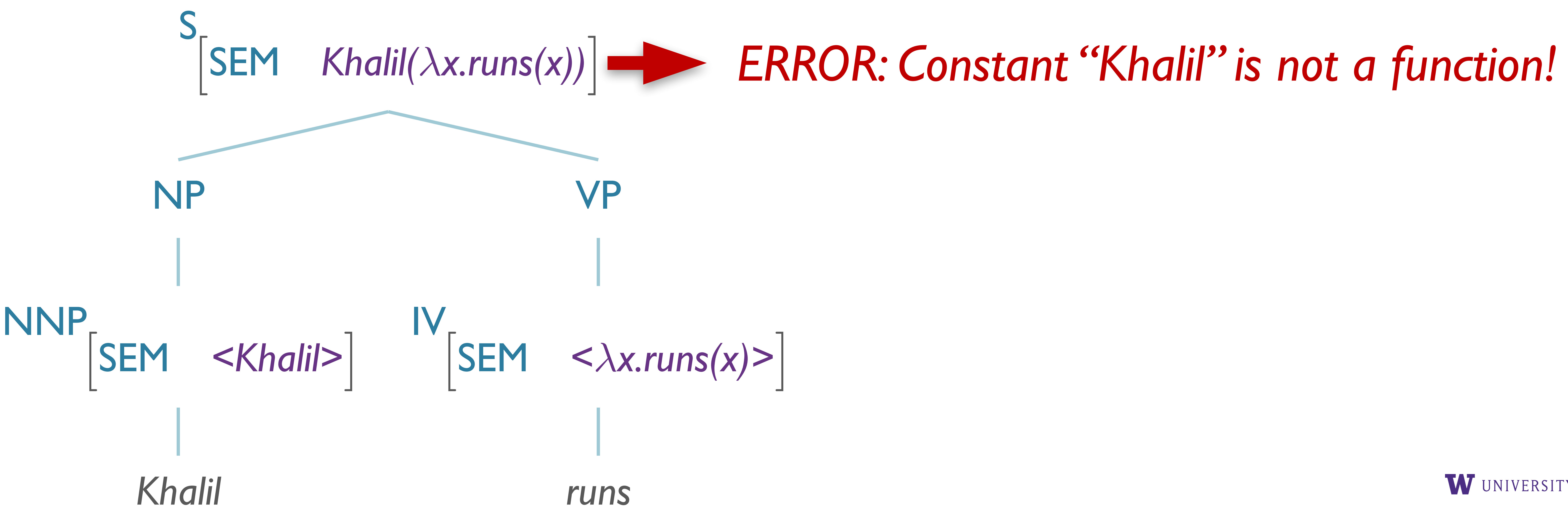
`S[SEM=np?(vp?)] → NP[SEM=np?] VP[SEM=vp?]`



Proper Nouns & Intransitive Verbs

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

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



Proper Nouns & Intransitive Verbs

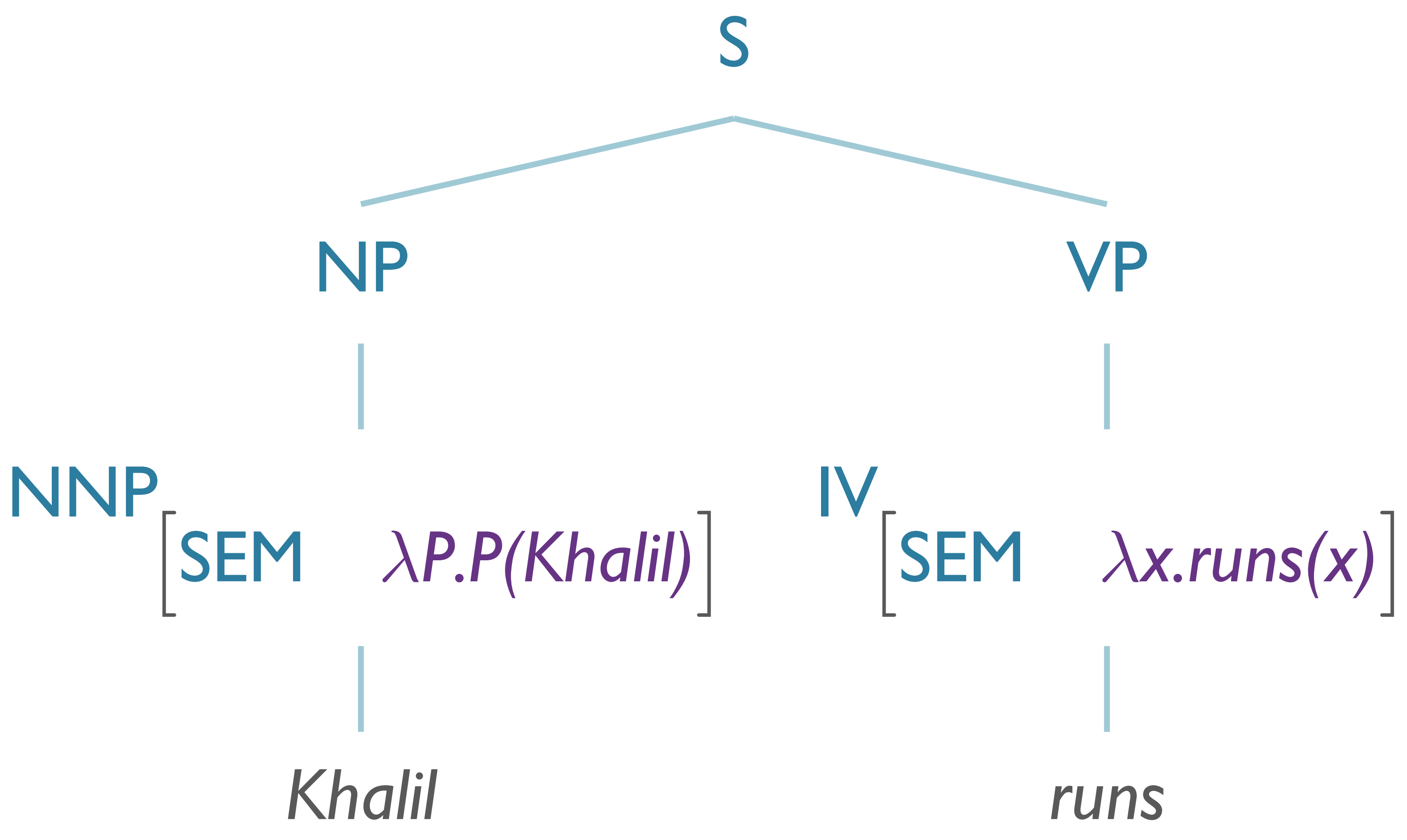
- Instead, we use a ***dummy predicate:***
  - $\boldsymbol{\lambda} Q.Q(\mathit{Khalil})$

- *"Generalizing to the worst case" (cf. Montague; Partee on type-shifting)*
  - I.e.: this move will also be necessary for a uniform semantic treatment of NPs, which can be individual-denoting (like names) or more complex (quantifiers)

Proper Nouns & Intransitive Verbs

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

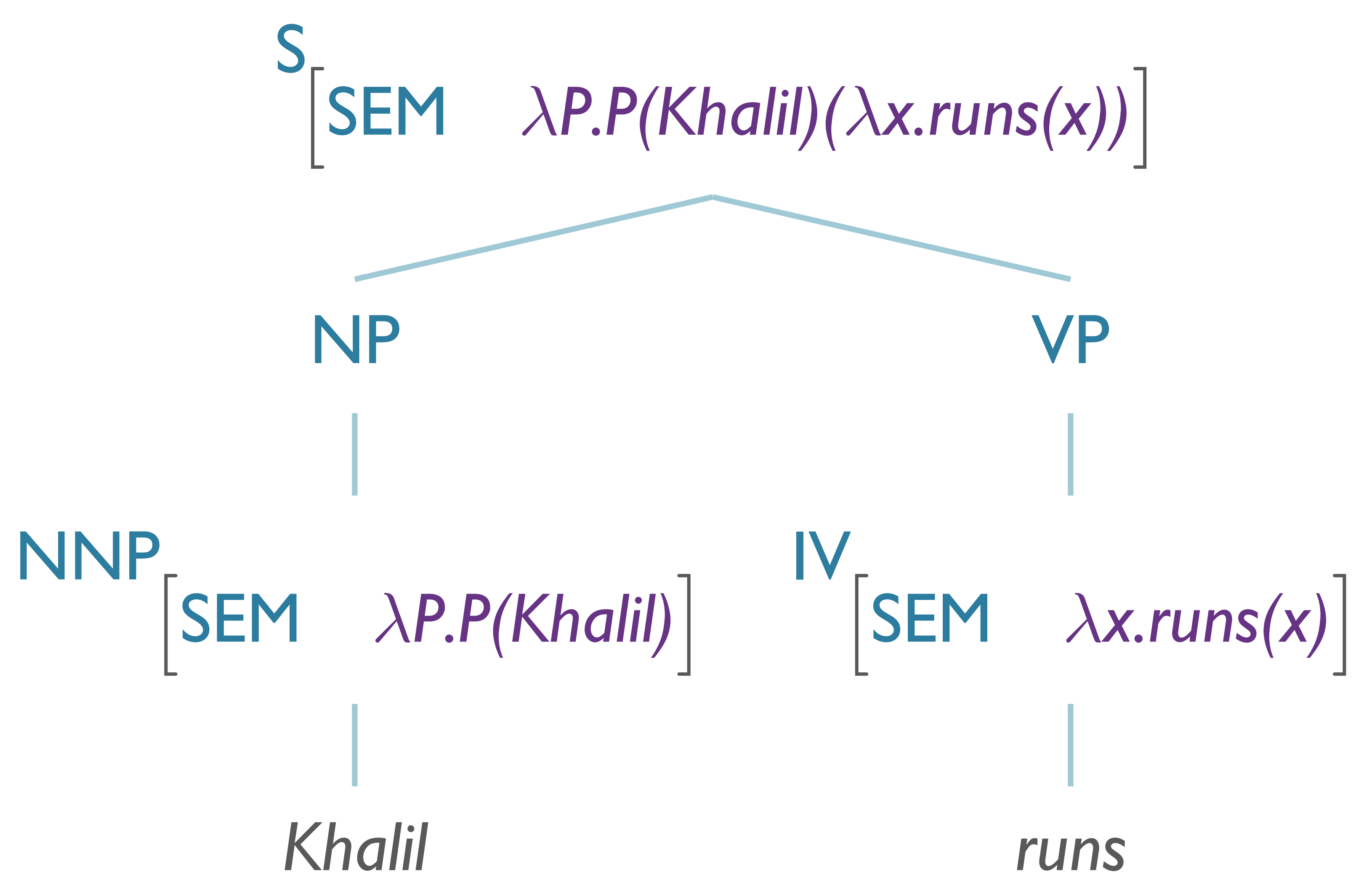
```
S[SEM=np?(vp?)] → NP[SEM=np?] VP[SEM=vp?]
```



Proper Nouns & Intransitive Verbs

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

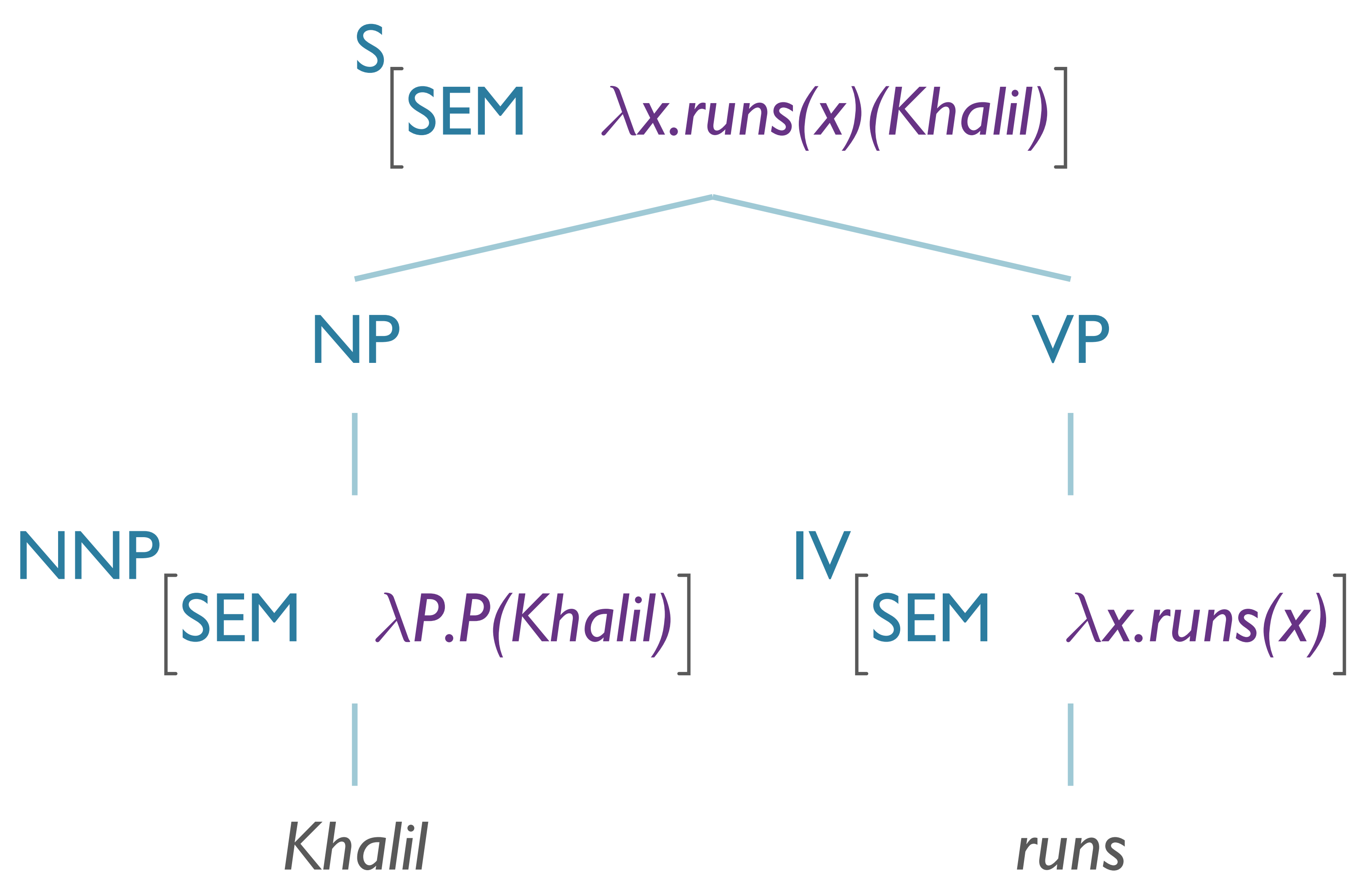
```
S[SEM=np?(vp?)] → NP[SEM=np?] VP[SEM=vp?]
```



Proper Nouns & Intransitive Verbs

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

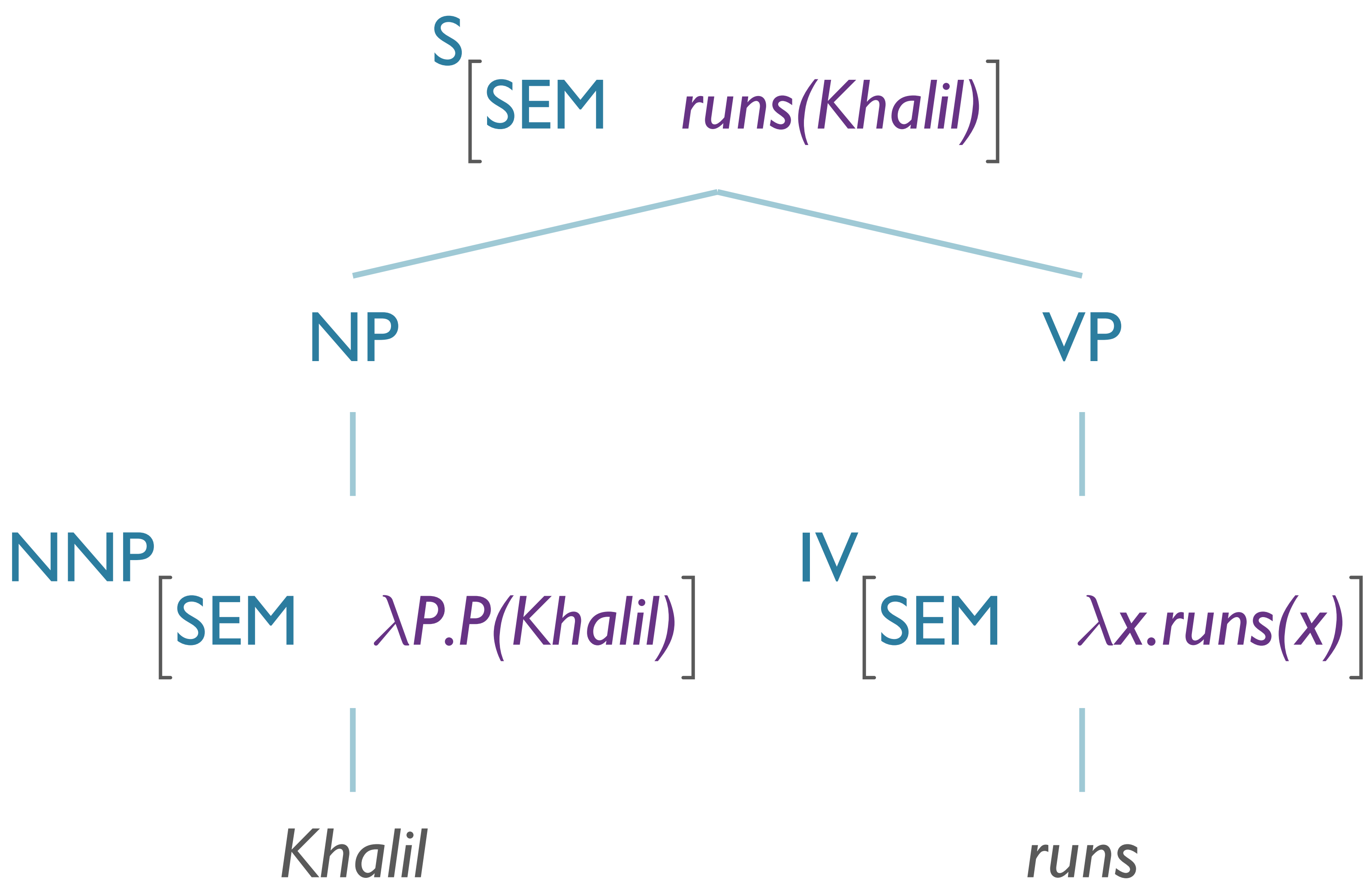
`S[SEM=np?(vp?)] → NP[SEM=np?] VP[SEM=vp?]`



Proper Nouns & Intransitive Verbs

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

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



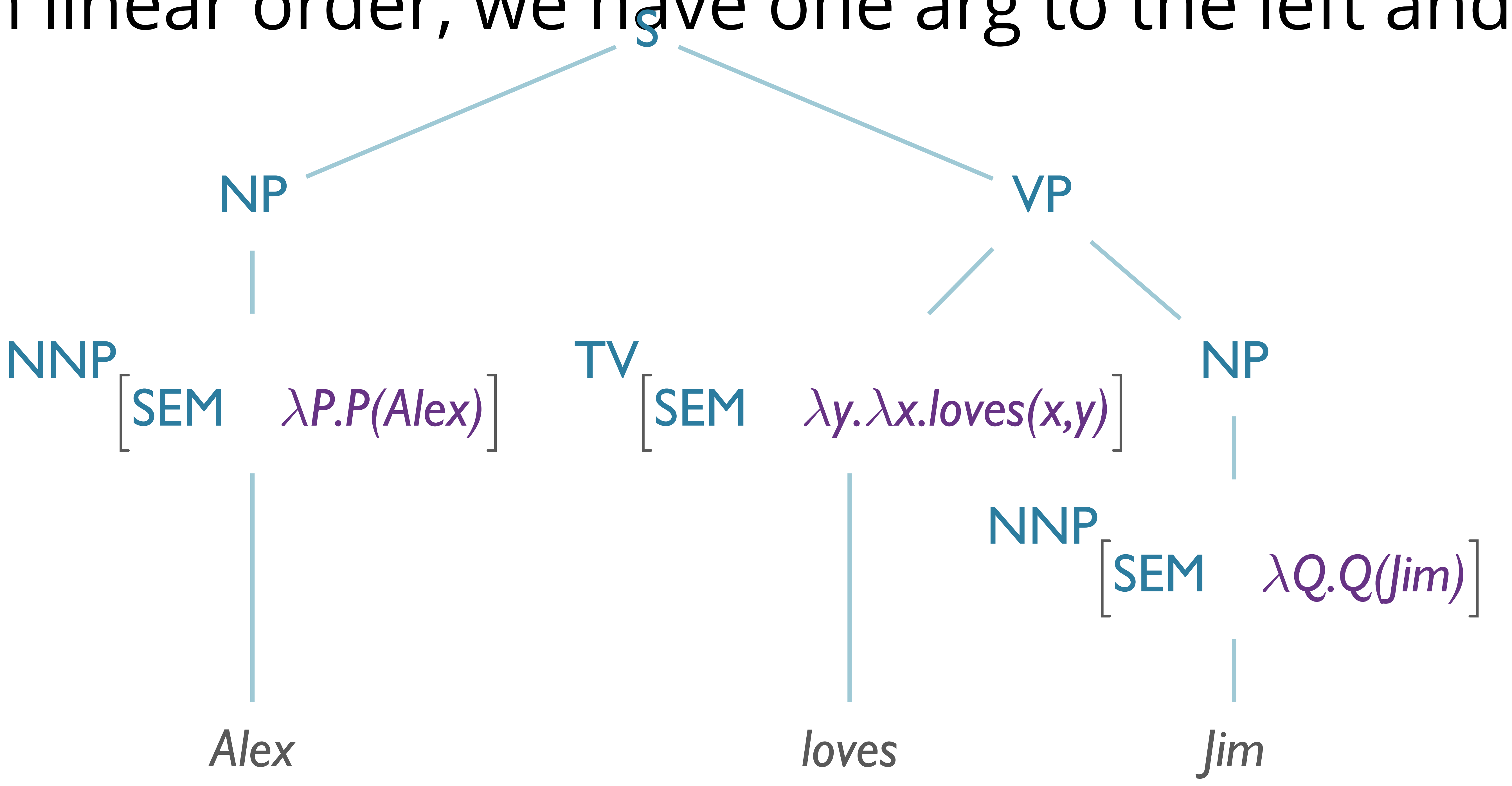
Transitive Verbs

Transitive Verbs

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

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(Jim))`

Transitive Verbs

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

Transitive Verbs

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

Transitive Verbs

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

Transitive Verbs

- TV(NP):
  - `λy.λx.loves(x,y) (λQ.Q(Jim))`
  - `λx.loves(x,λQ.Q(Jim))`
    - → ***Error!*** `We can't reduce Jim.`
- Instead: `λY x.Y(λy.loves(x,y))`
  - ("Continuation-passing")

Transitive Verbs

- TV(NP):
  - `λY x.Y(λy.loves(x,y)) (λQ.Q(Jim))`

Transitive Verbs

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

Transitive Verbs

- TV(NP):
  - `λY x.Y(λy.loves(x,y)) (λQ.Q(Jim))` — `λY takes (λQ.Q(Jim))`
  - `λx.(λQ.Q(Jim)(λy.loves(x,y))` — `λQ takes (λy.loves(x,y))`
  - `λx.(λy.loves(x,y)(Jim))`

Transitive Verbs

- TV(NP):
  - `λY x.Y(λy.loves(x,y)) (λQ.Q(Jim))` — `λY takes (λQ.Q(Jim))`
  - `λx.(λQ.Q(Jim)(λy.loves(x,y))` — `λQ takes (λy.loves(x,y))`
  - `λx.(λy.loves(x,y)(Jim))` — `λy takes (Jim)`
  - `λx.(loves(x,Jim))`

Transitive Verbs

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

`λY takes (λQ.Q(Jim))`

`λQ takes (λy.loves(x,y))`

`λy takes (Jim)`

- NP(VP):
  - `λP.P(Alex)(λx.(loves(x,Jim)))`

Transitive Verbs

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

Transitive Verbs

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

Converting to an Event

- "x loves y," Originally:
  - `λY x.Y(λy.loves(x,y))`

Converting to an Event

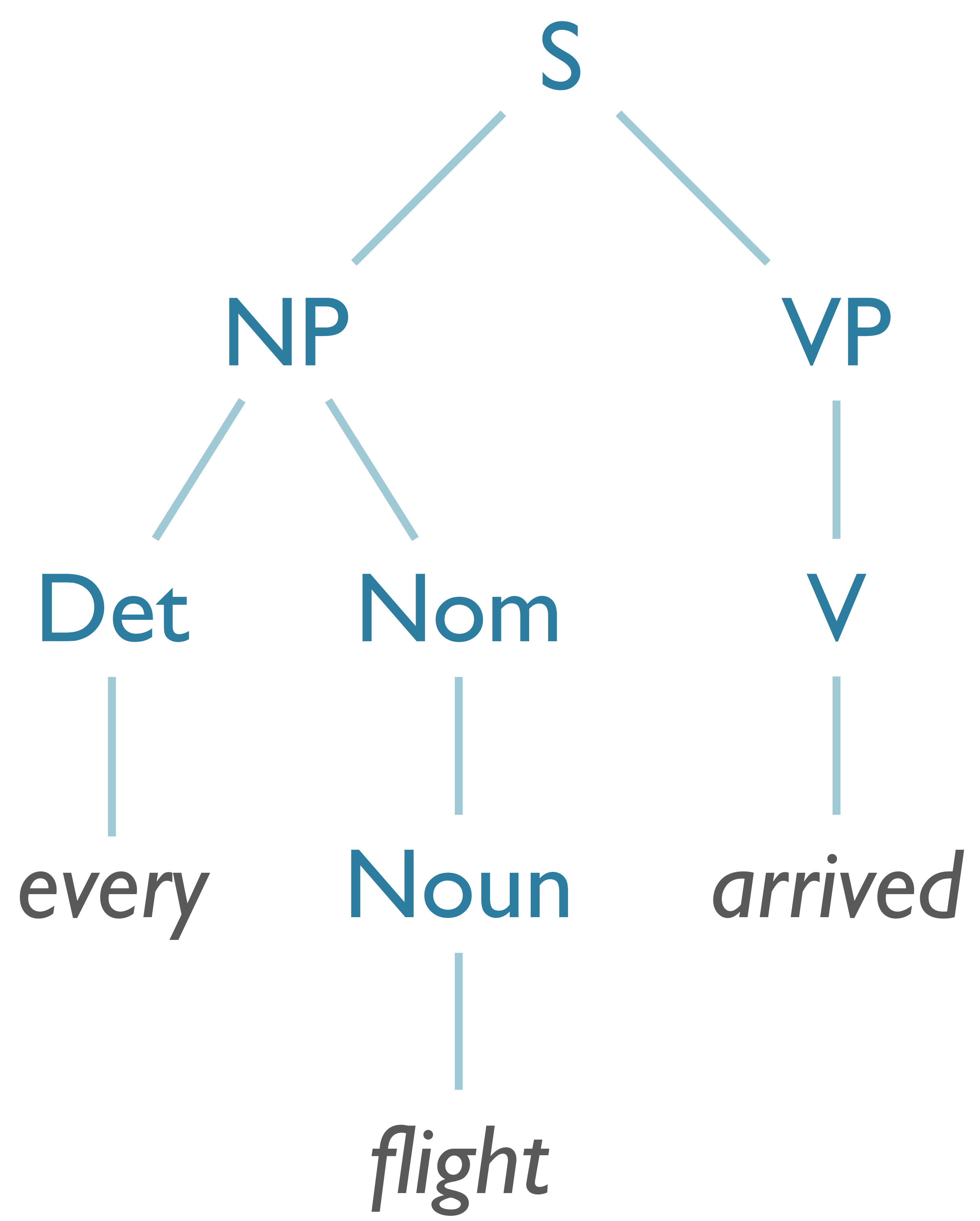
- “x loves y,” Originally:
  - `λY x.Y(λy.loves(x,y))`

- as a Neo-Davidsonian event:
  - `λY x.Y(λy.∃e love(e) ∧ lover(e,x) ∧ loved(e,y))`

Quantifiers & Scope

Semantic Analysis Example

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



- Example: *Every flight arrived*

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

"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:
  - $\forall \mathbf{x}\ Flight(\mathbf{x}) \Rightarrow Q(\mathbf{x})$
    - “if a thing is a flight, then it is *Q*”
  - Since Q isn’t available yet… Dummy predicate!
    - $\lambda Q.\forall \mathbf{x}\ Flight(\mathbf{x}) \Rightarrow Q(\mathbf{x})$

“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:
  - $\lambda \boldsymbol{Q}.\forall \boldsymbol{x}\ \boldsymbol{Flight}(\boldsymbol{x}) \Rightarrow \boldsymbol{Q}(\boldsymbol{x})$
- …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***"?
  - $\lambda P.\lambda Q.\forall x\ P(x) \Rightarrow Q(x)$

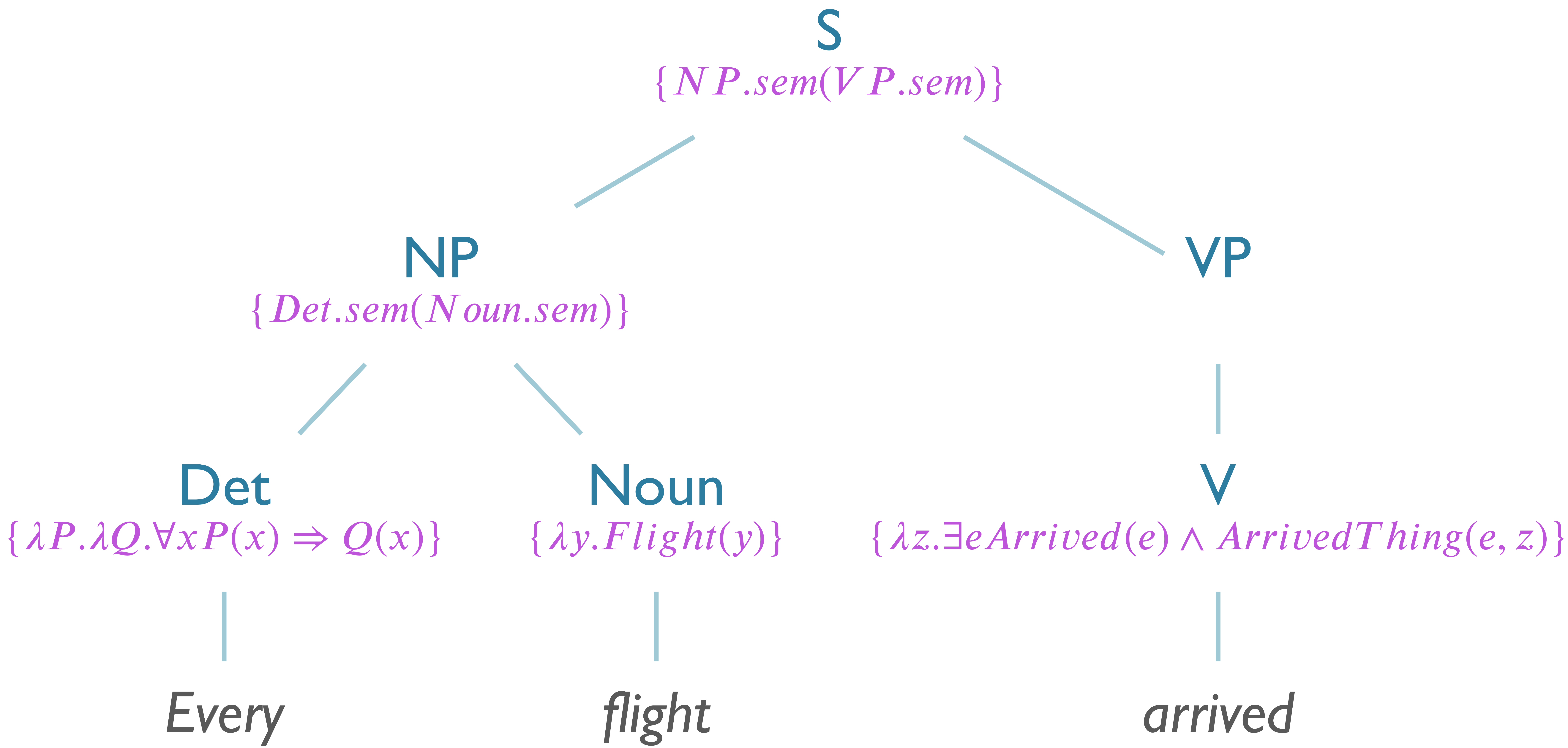
"A flight arrived"

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

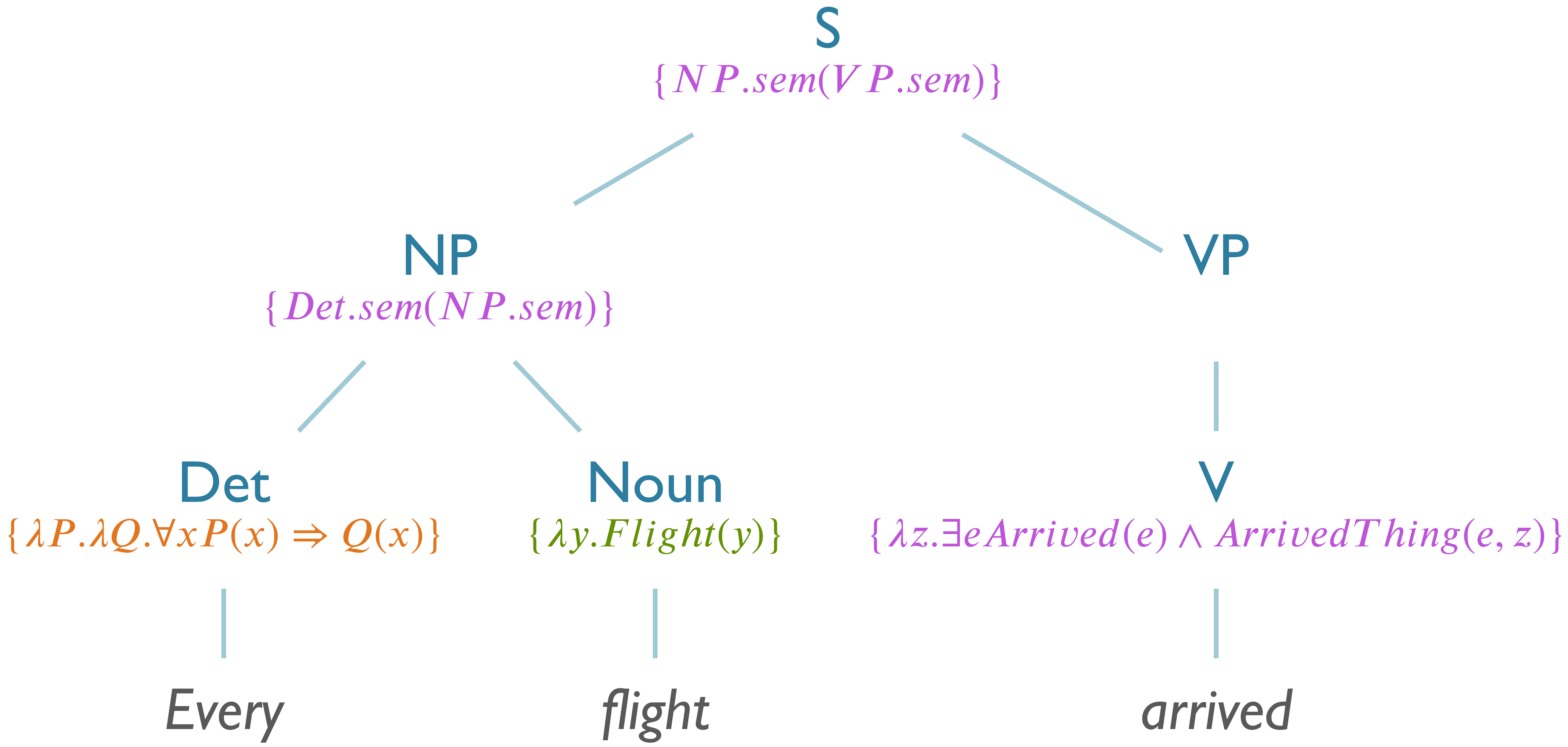
Creating Attachments

"Every flight arrived"

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

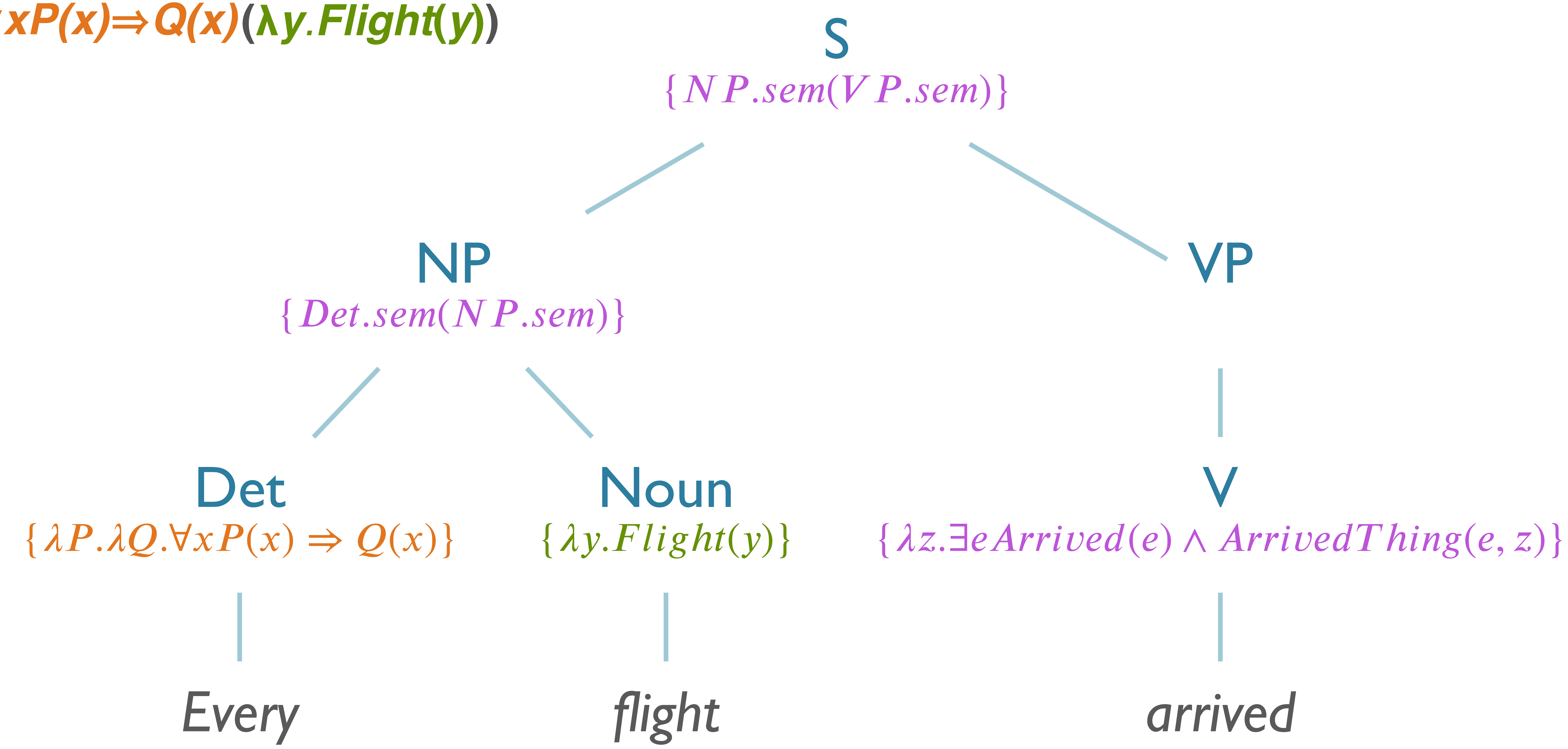

S
{NP.sem(VP.sem)}
NP
{Det.sem(Noun.sem)}
VP
Det
{λP.λQ.∀xP(x) ⇒ Q(x)}
Noun
{λy.Flight(y)}
V
{λz.∃eArrived(e) ∧ ArrivedThing(e, z)}
Every
flight
arrived

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



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

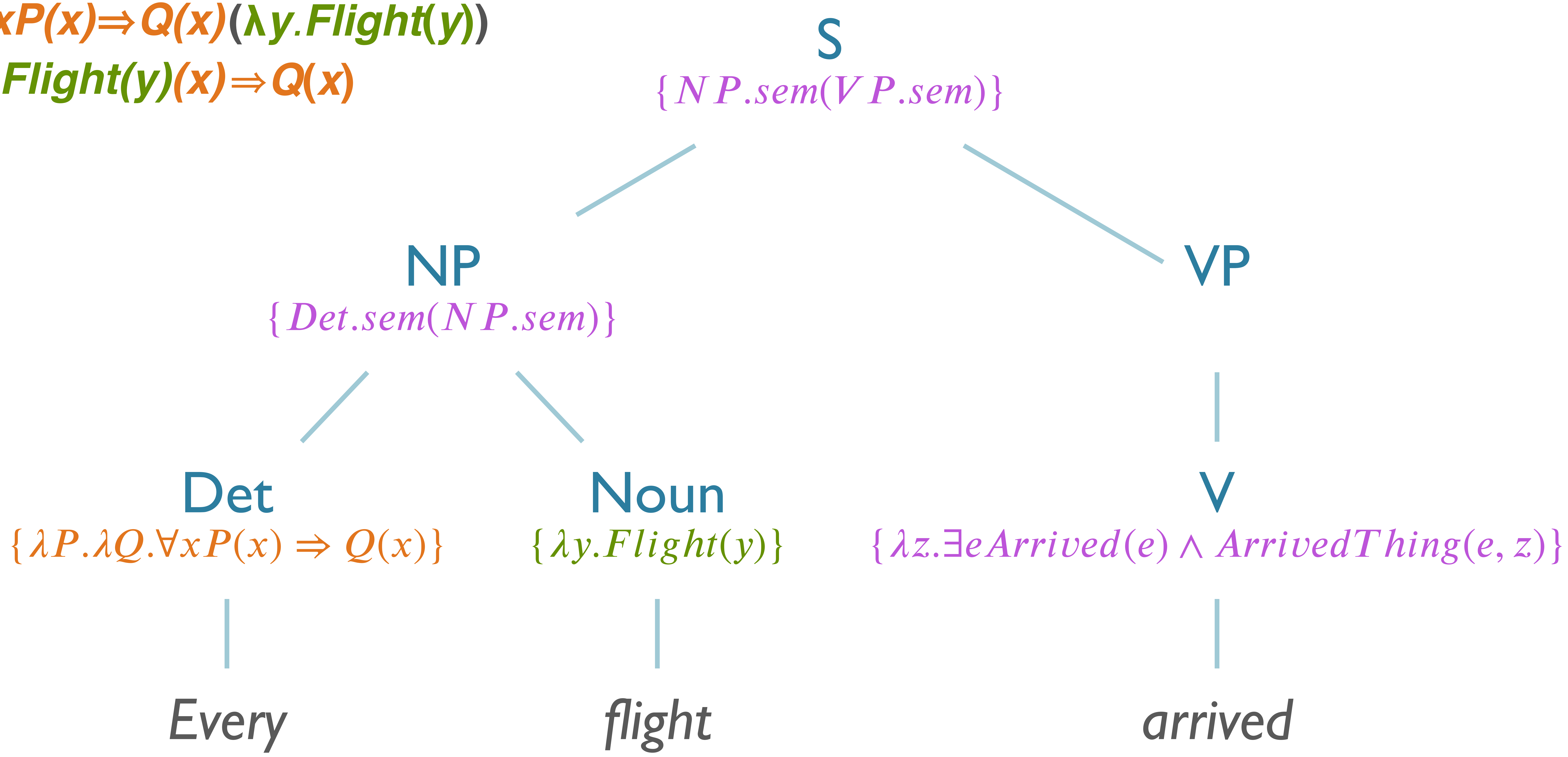
$\lambda P.\lambda Q.\forall x P(x) \Rightarrow Q(x)(\lambda y.Flight(y))$



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

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

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

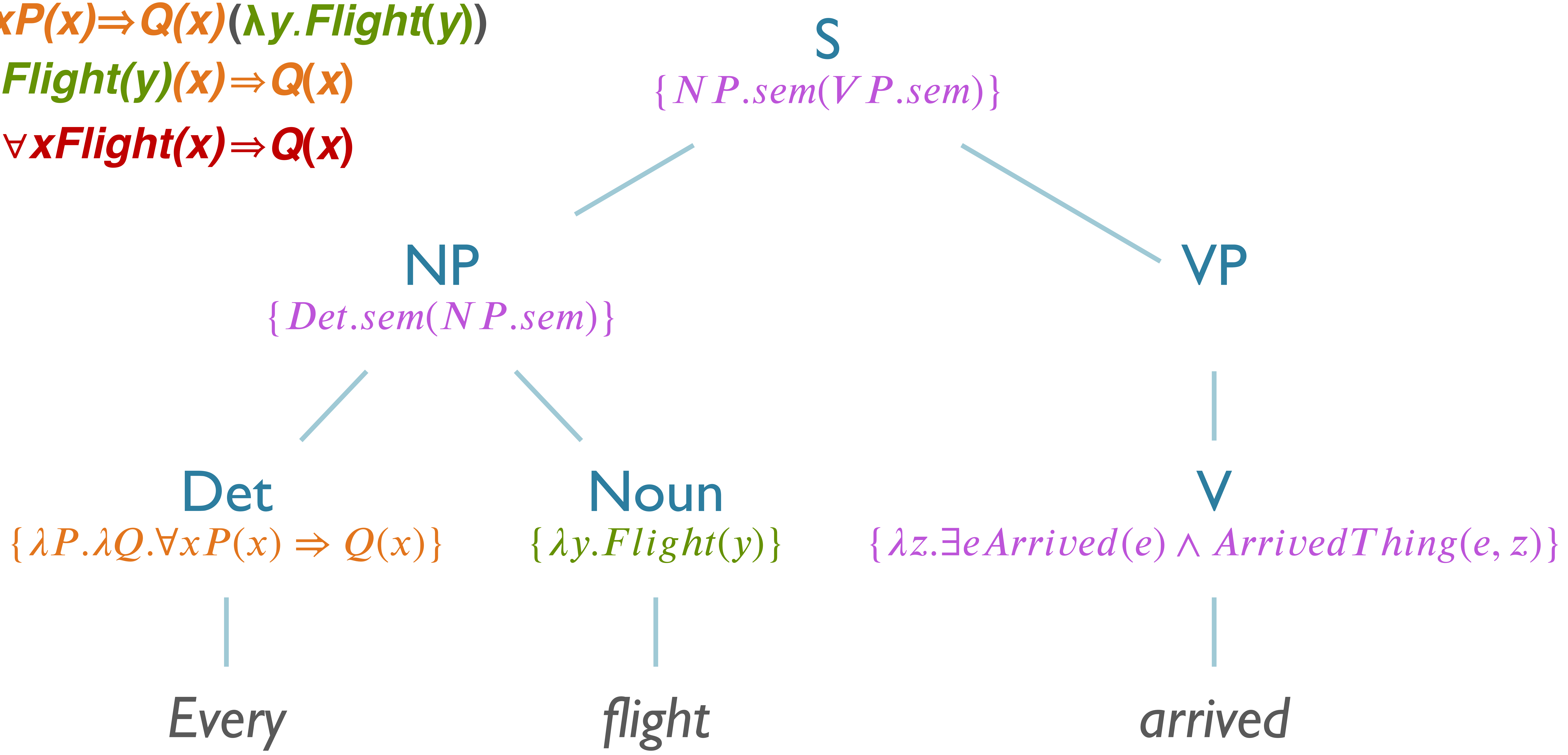


$$NP.sem \rightarrow Det.sem(Noun.sem)$$

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

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

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

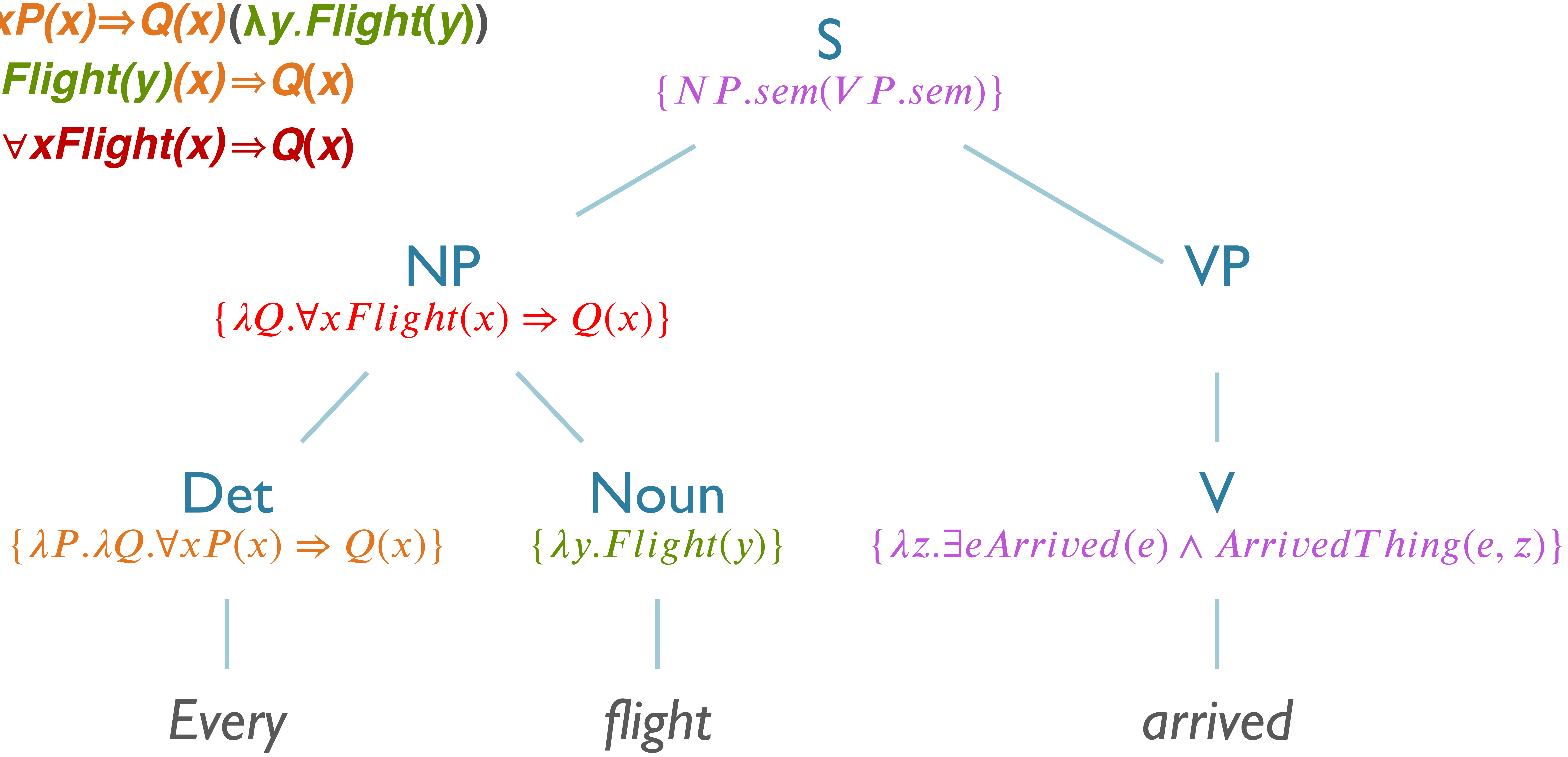


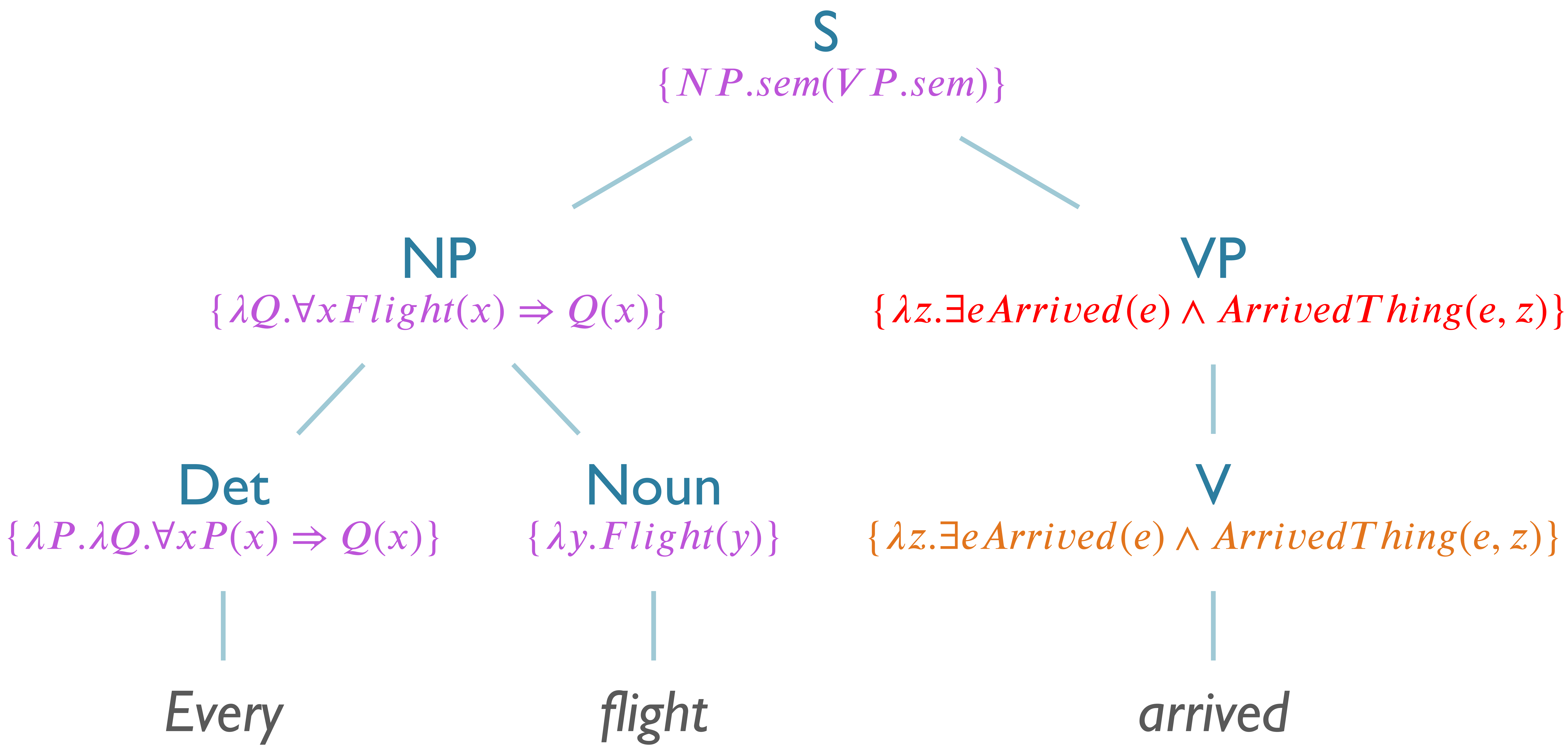
$$NP.sem \rightarrow Det.sem(Noun.sem)$$

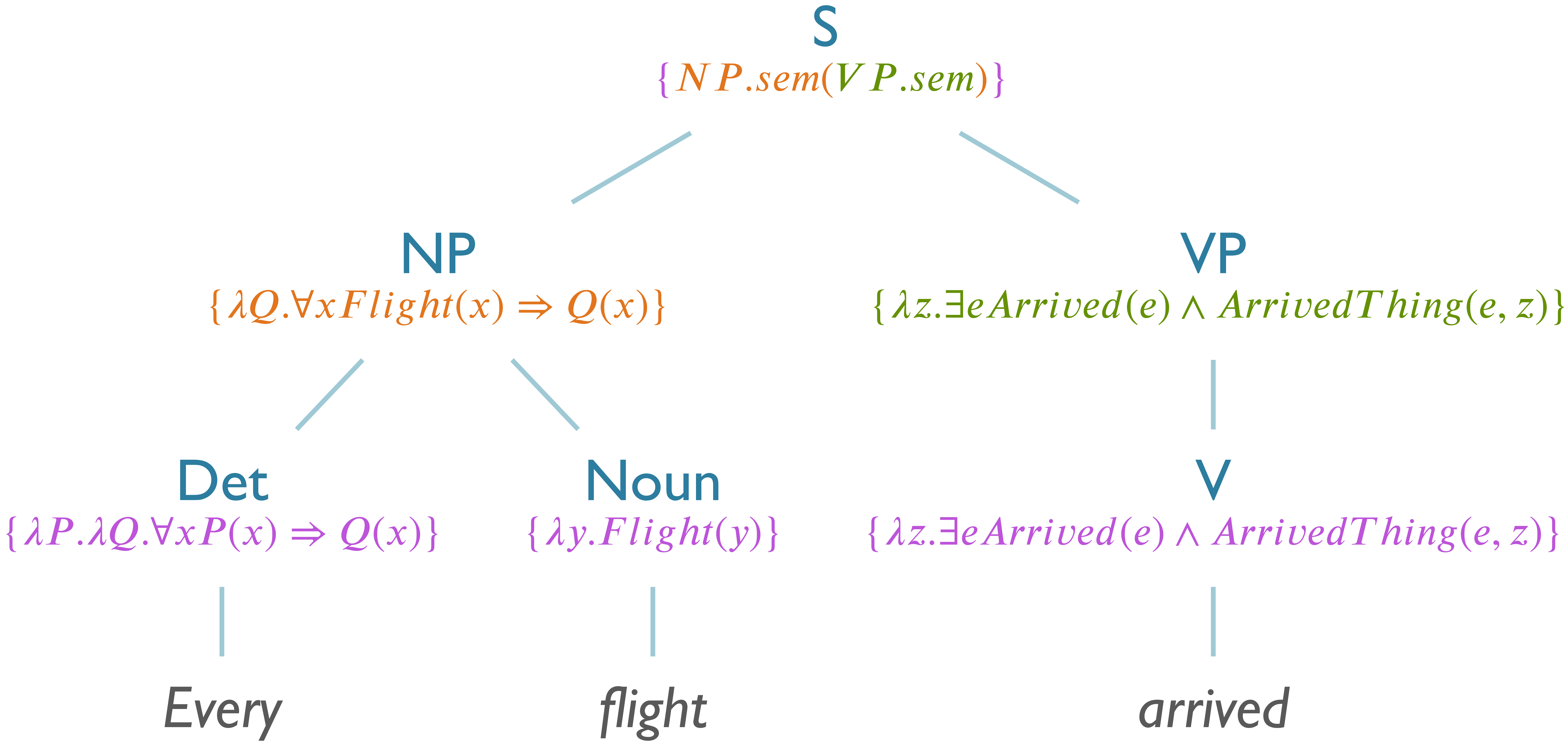
$$\lambda P.\lambda Q.\forall x P(x) \Rightarrow Q(x)(\lambda y.Flight(y))$$

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

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




S
$\{NP.sem(VP.sem)\}$
NP
$\{\lambda Q.\forall x Flight(x) \Rightarrow Q(x)\}$
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

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

NP

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

VP

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

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

S

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

NP

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

VP

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

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

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

S

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

NP

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

VP

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

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

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

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

S

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

NP

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

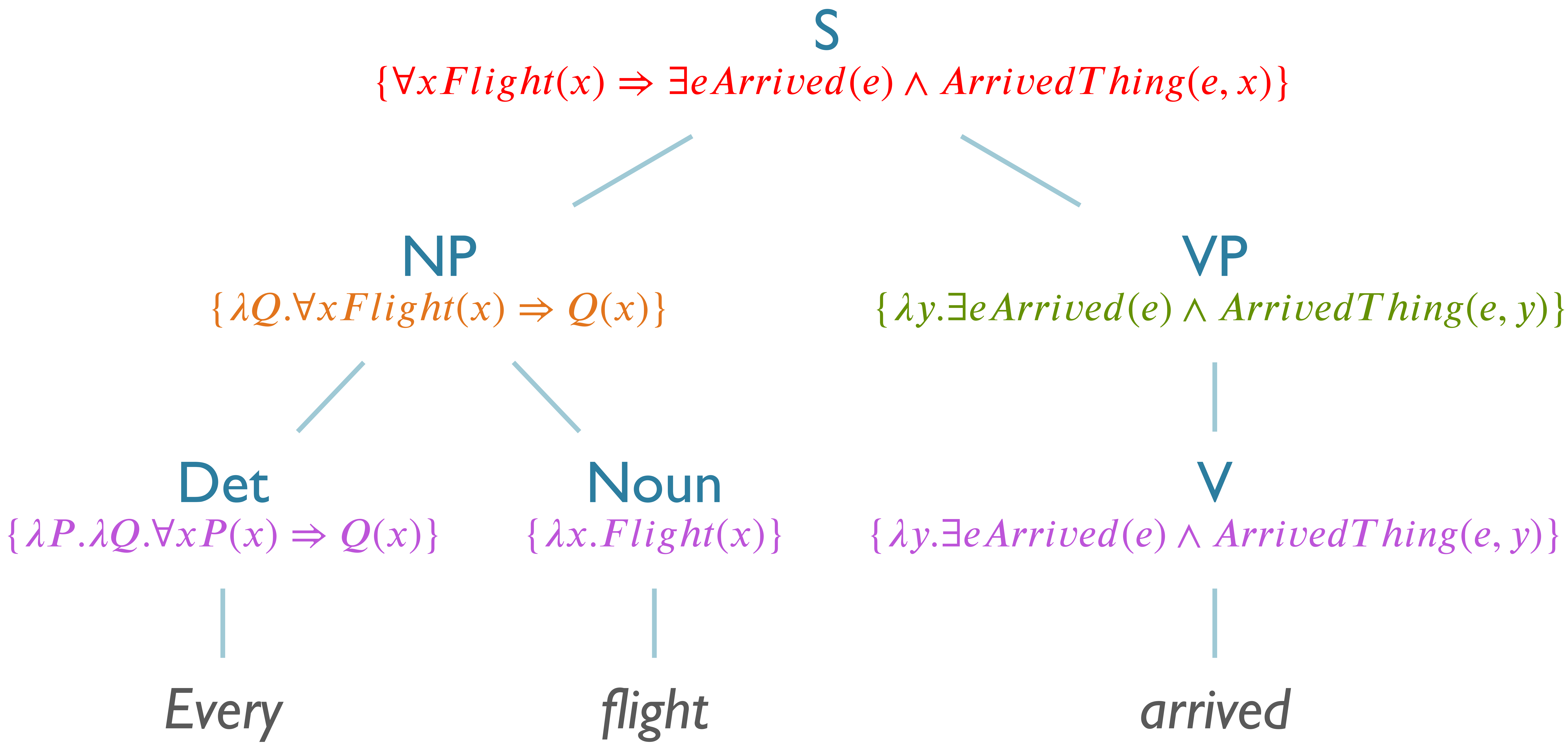
VP

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

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

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

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


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 → *'a'*	{ $\lambda P.\lambda Q.\exists x\ P(x) \wedge Q(x)$ }
Det → *'every'*	{ $\lambda P.\lambda Q.\forall x\ P(x) \Rightarrow Q(x)$ }
NN → *'flight'*	{$\lambda x.Flight(x)$}
NNP → *'John'*	{$\lambda X.X(John)$}
NP → *NNP*	{*NNP.sem*}
S → *NP VP*	{*NP.sem*(*VP.sem*)}
VP → *Verb NP*	{*Verb.sem*(*NP.sem*)}
Verb → *'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
    - Function application

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
    - Function application
    - <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