Computational Semantics

LING 571 — Deep Processing for NLP

October 28, 2019

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.’

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

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

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
- **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
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={ ⟨a,f⟩ , ⟨c,f⟩ , ⟨c,g⟩ , ⟨b,e⟩ , ⟨d,f⟩ , ⟨d,g⟩ }
	Serves	Med serves eclectic Rio serves Mexican Frasca serves Italian	Serves={ ⟨c,f⟩ , ⟨f,i⟩ , ⟨e,h⟩ }

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)
- **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."

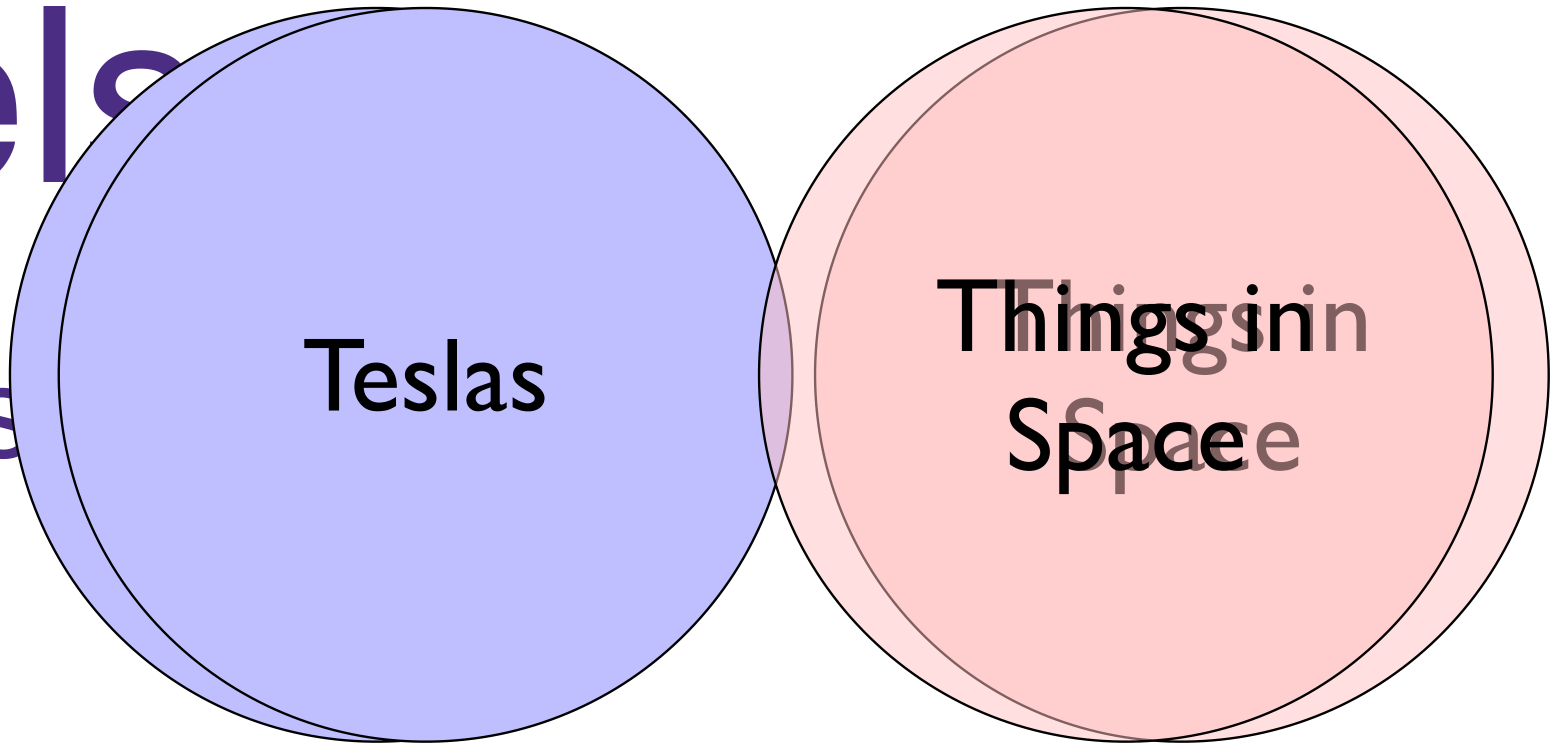
  "Not the same thing a bit!" said the Hatter. "Why, you might just as well say that 'I see what I eat' is the same thing as 'I eat what I see'!"

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

- 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!

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

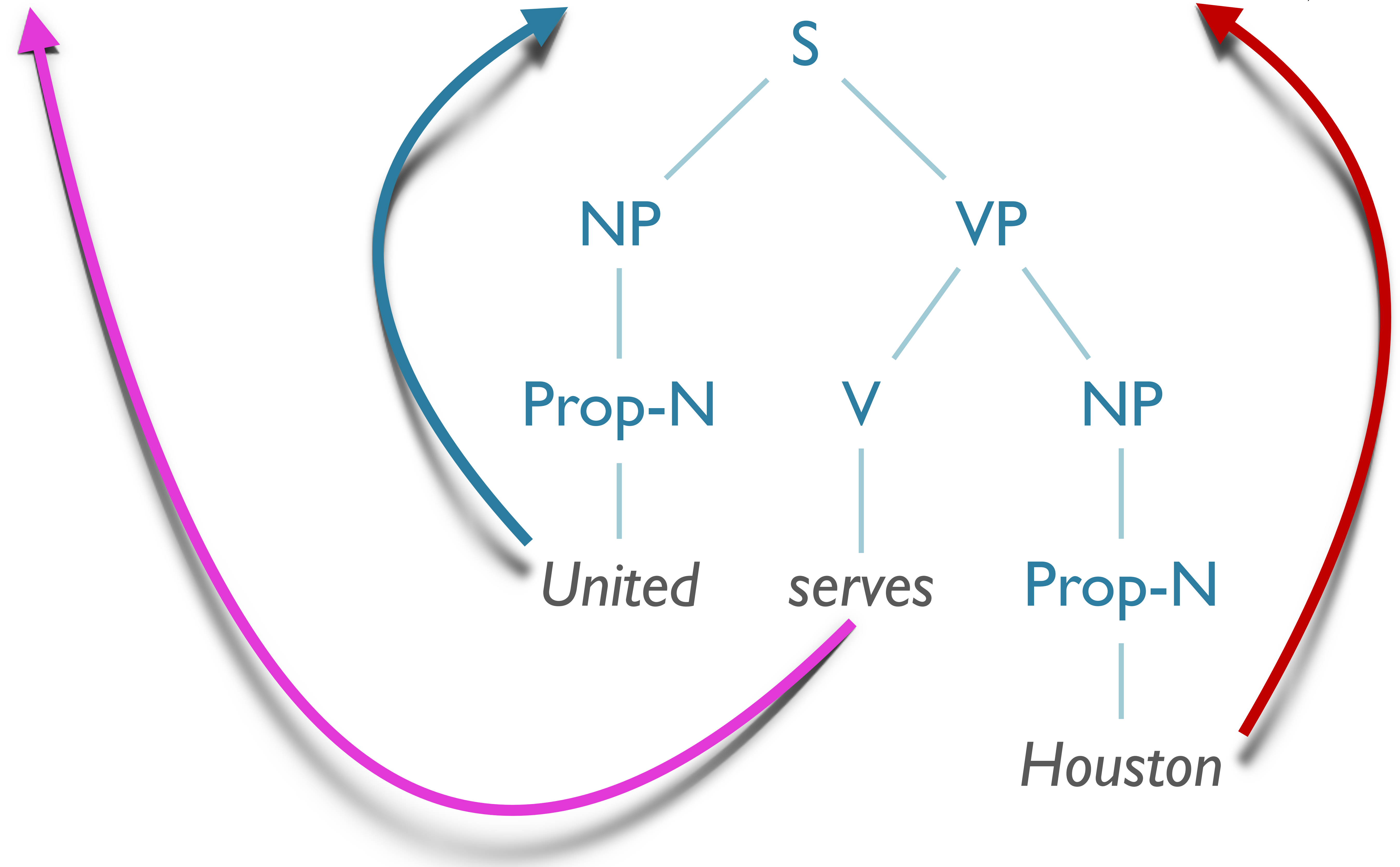
2. Integrate Directly into Grammar

- 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*

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



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:

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

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'
```

'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!

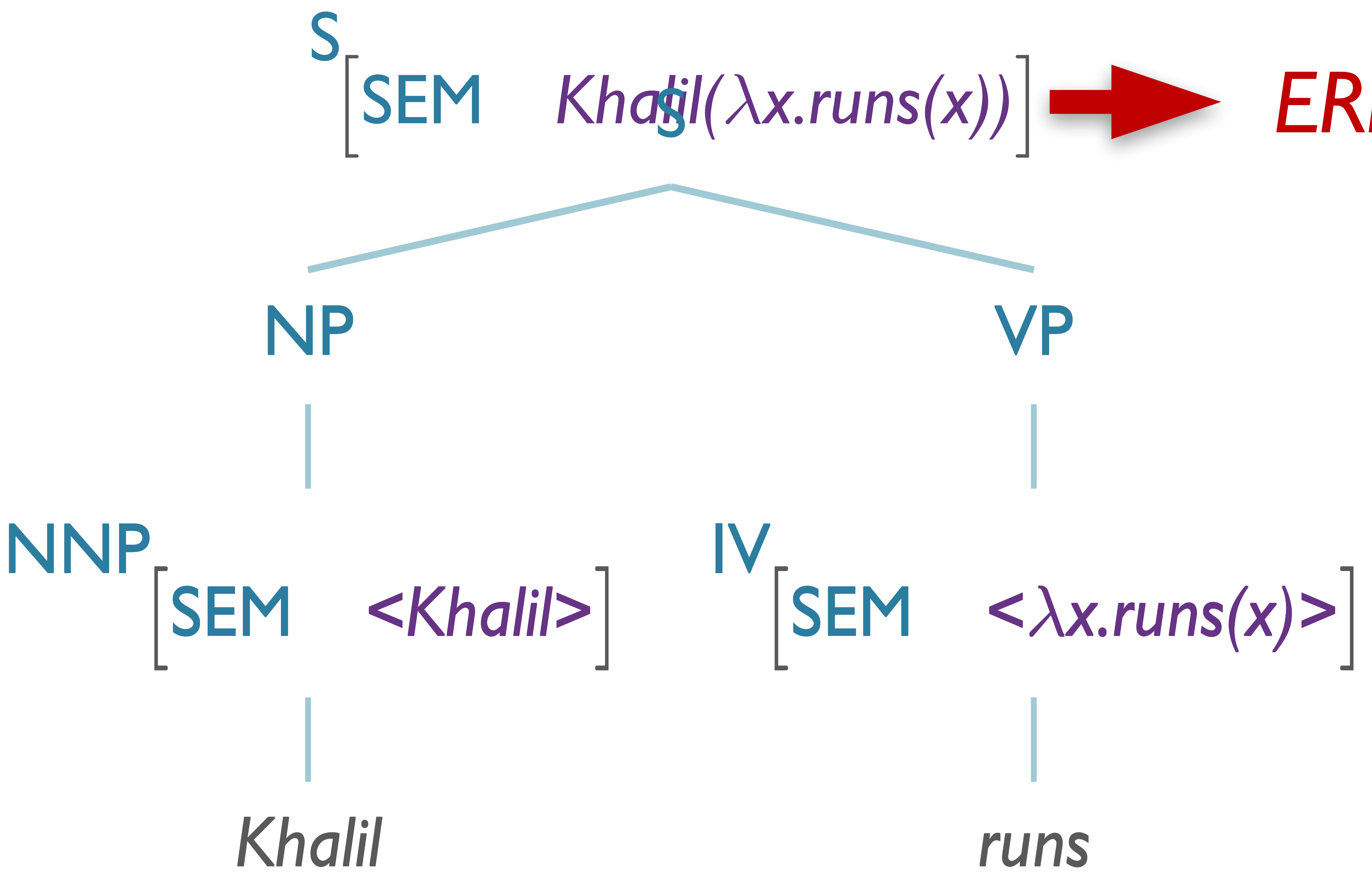
Defining Representations

- 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'`
- However, we want to apply our λ-closures left-to-right consistently.

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



ERROR: Constant "Khalil" is not a function!

Proper Nouns & Intransitive Verbs

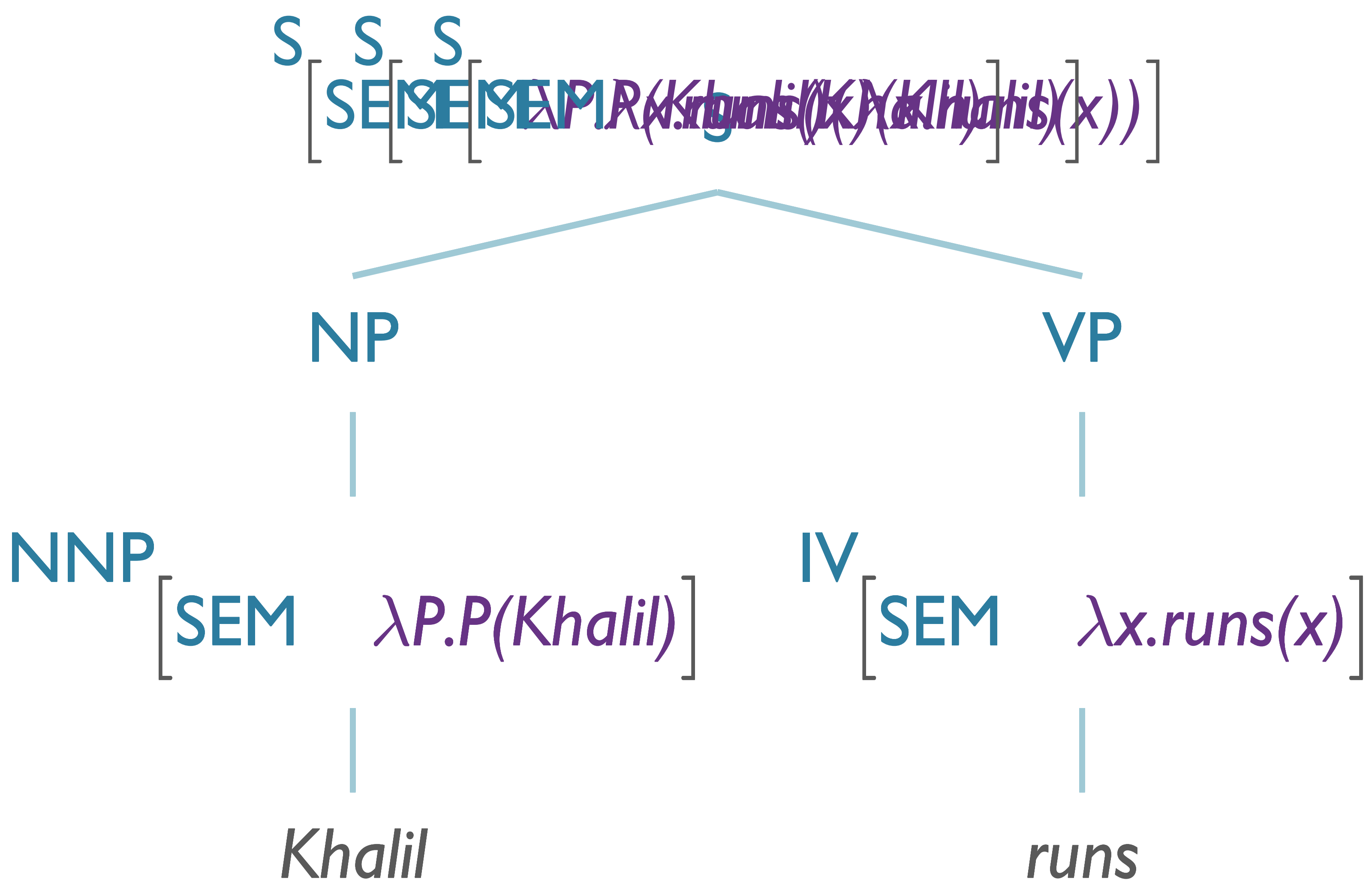
- Instead, we use a ***dummy predicate:***
  - $\lambda Q.Q(\text{Khalil})$

- “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?]
```



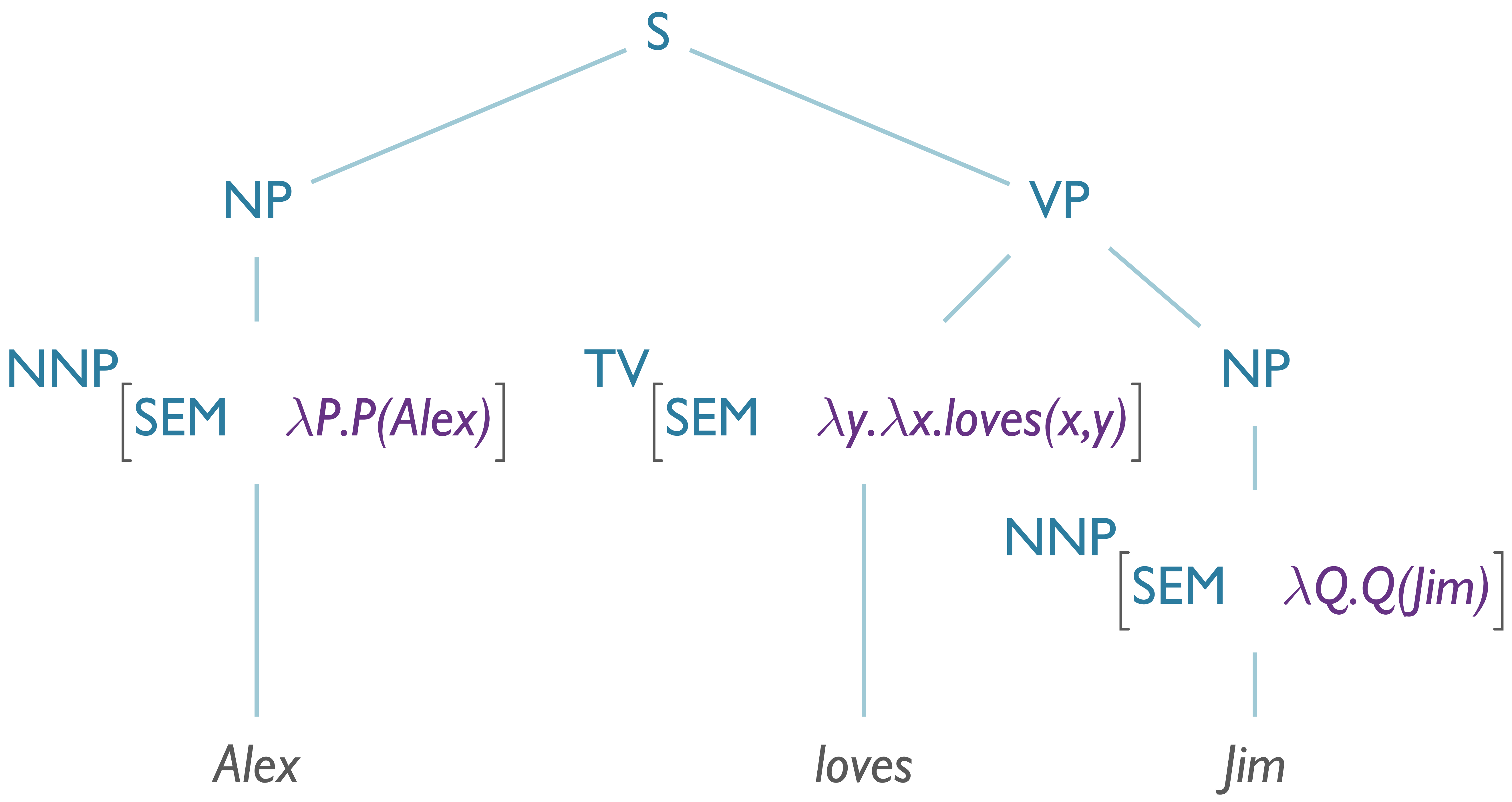
Transitive Verbs

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.

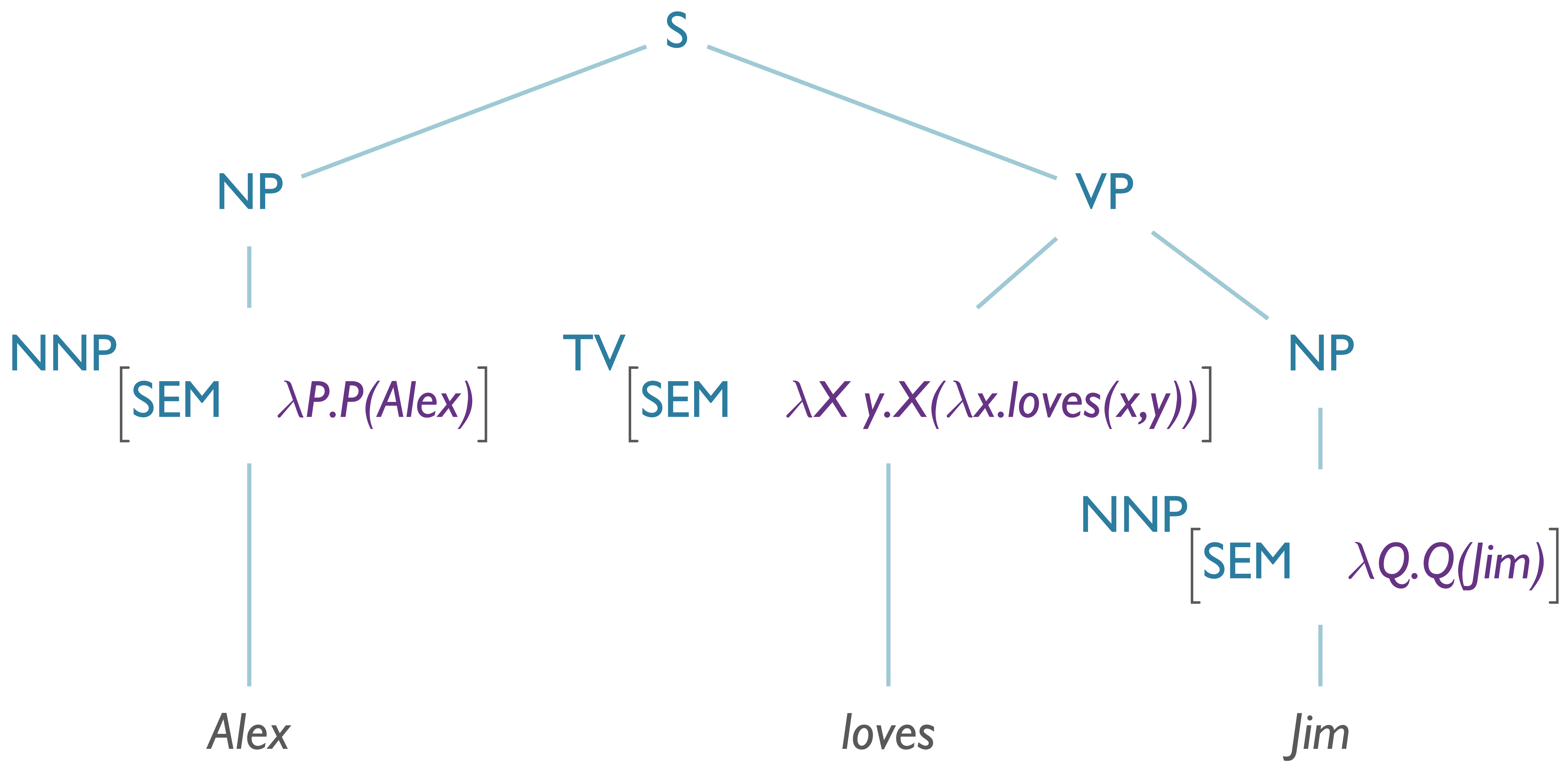


Transitive Verbs

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

λP takes (λy.(loves(Jim,y))

λy takes (Alex)

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))`

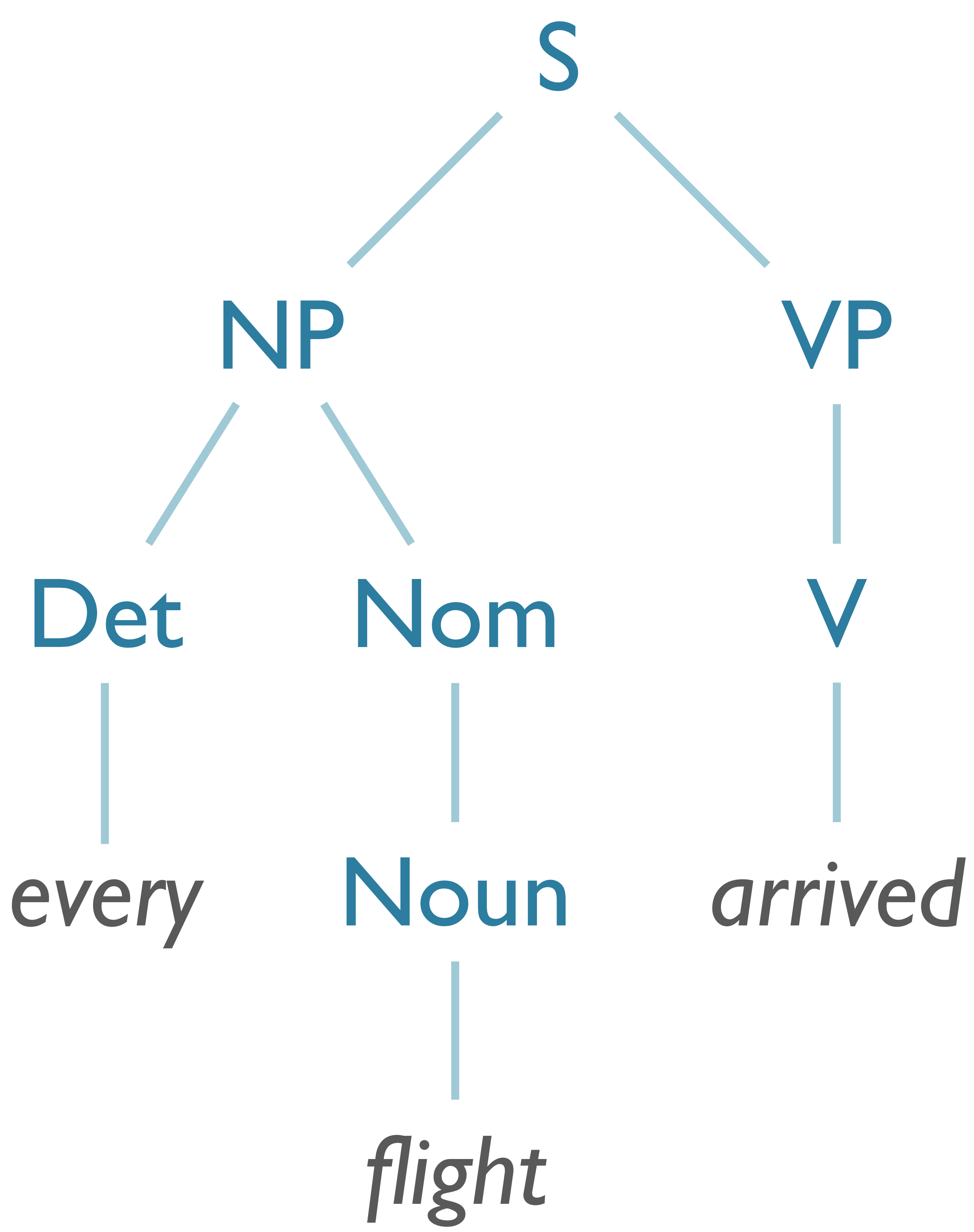
Quantifiers & Scope

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 = $\forall x\ Flight(x)$ ✗
    - "Everything is a flight"
- Instead, we want:
  - $\forall \boldsymbol{x}\ Flight(\boldsymbol{x}) \Rightarrow Q(\boldsymbol{x})$
    - "if a thing is a flight, then *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*” 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…
  - $\lambda P.\lambda Q.\exists x\ P(x) \wedge Q(x)$

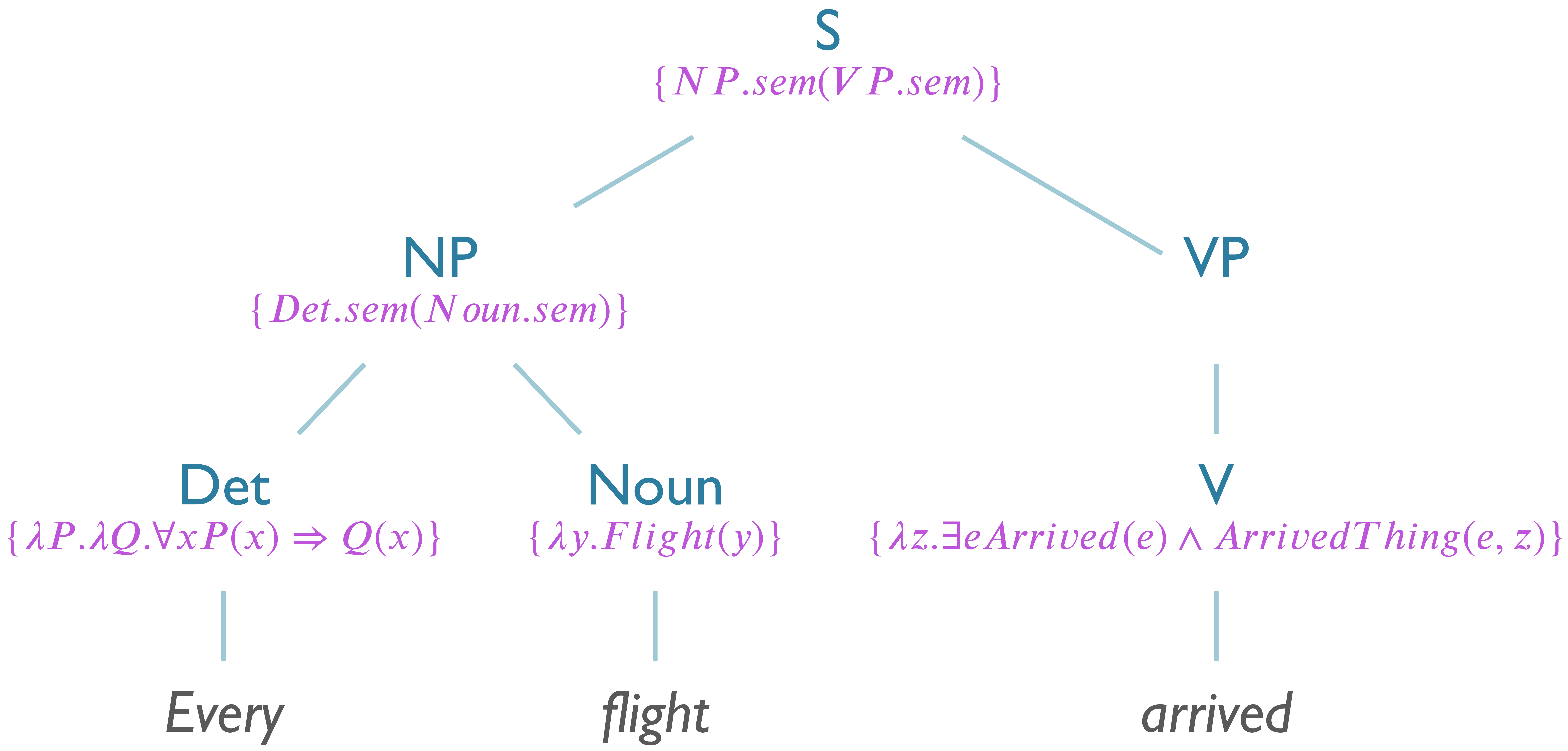
"*The flight arrived*"

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

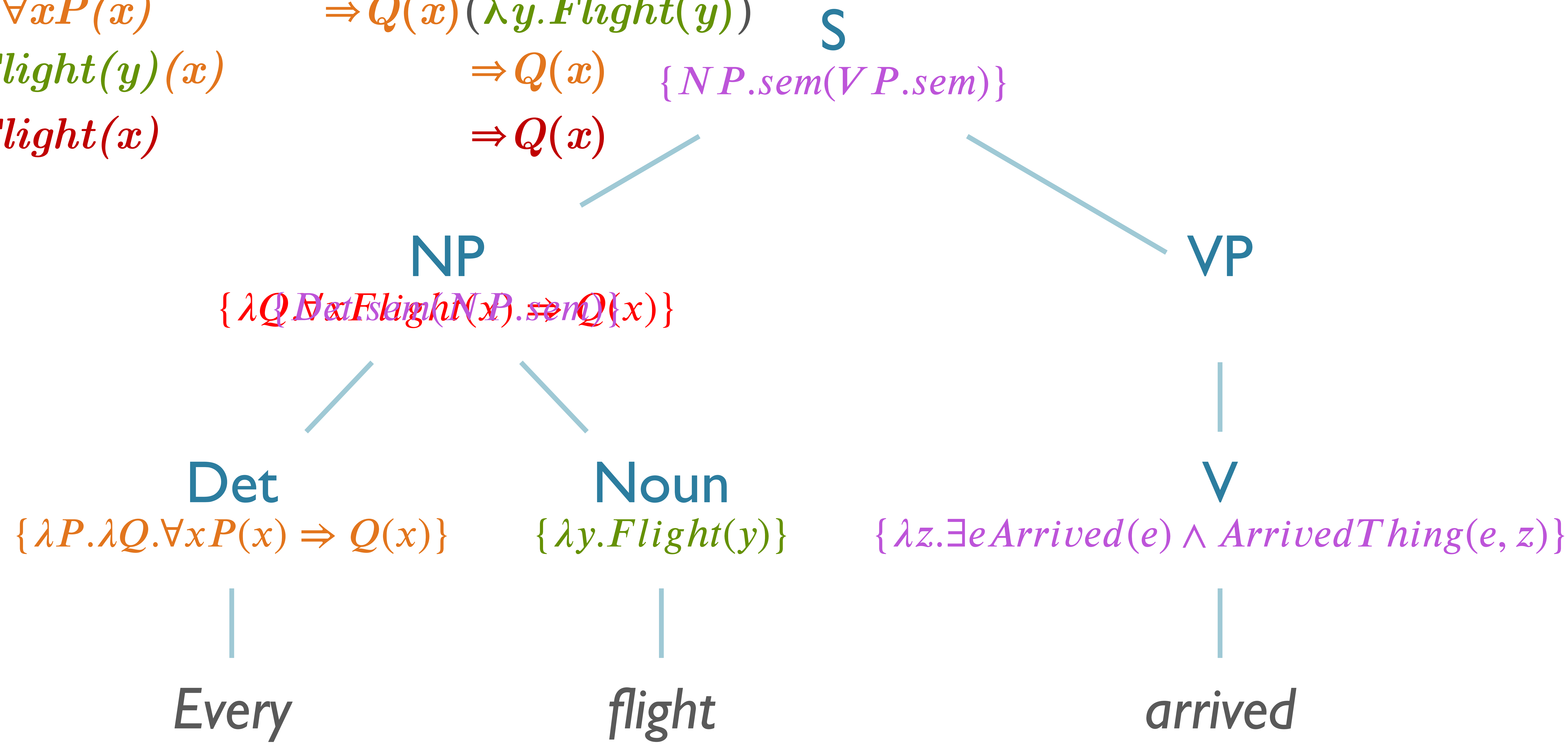
Creating Attachments

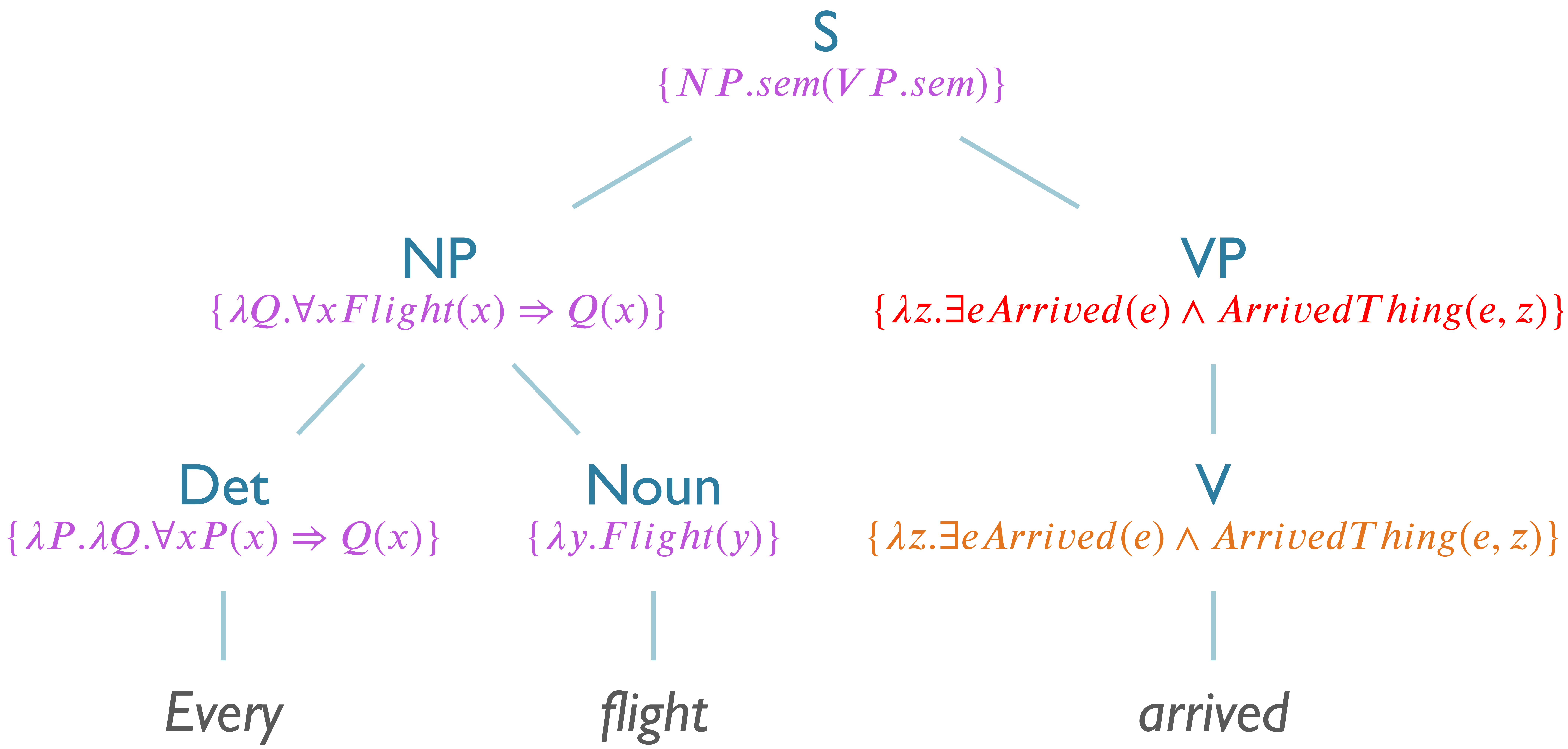
"Every flight arrived"

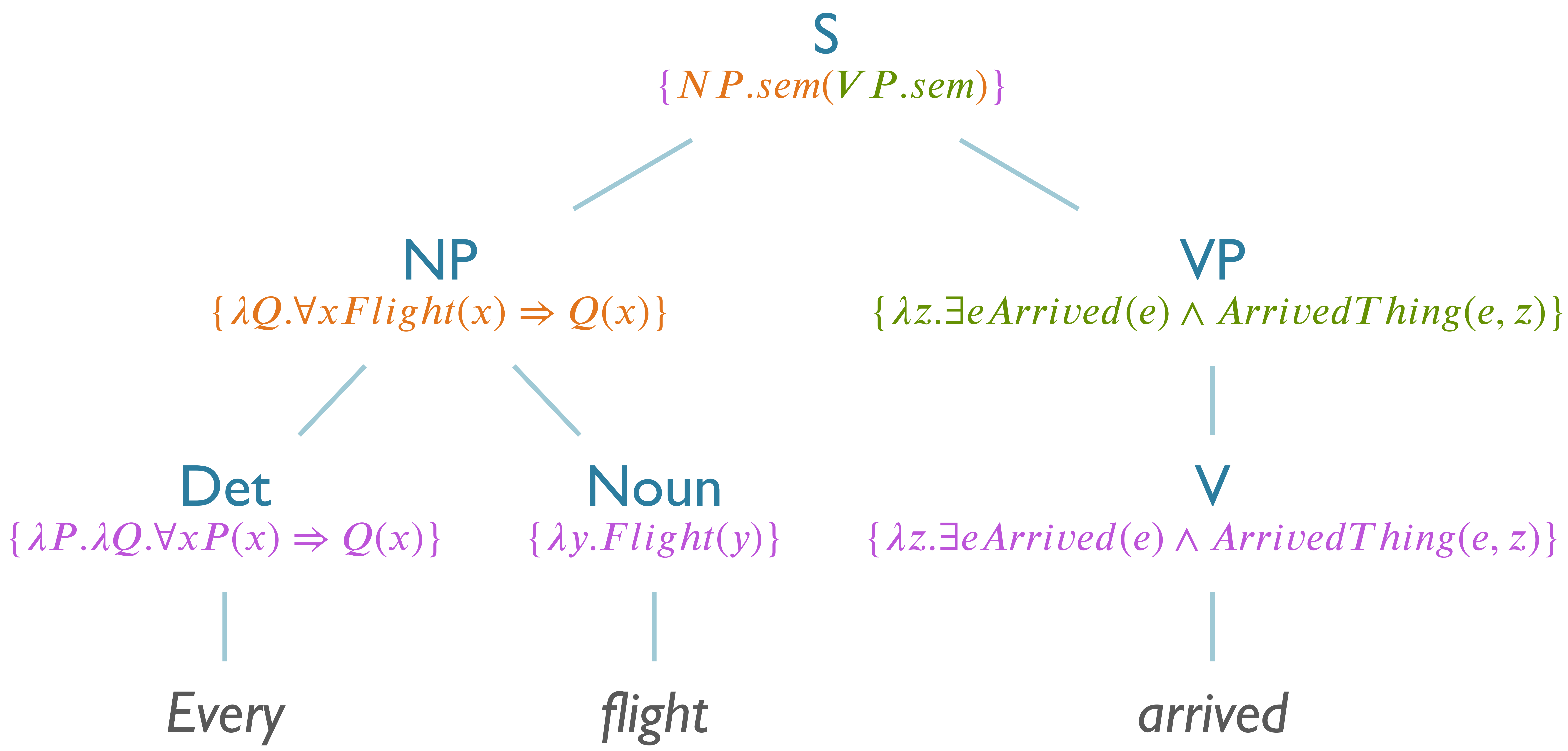
Det	$\rightarrow$ 'Every'	$\{\ \lambda P.\lambda Q.\forall \boldsymbol{x}\ P(\boldsymbol{x}) \Rightarrow Q(\boldsymbol{x})\ \}$
$Noun$	$\rightarrow$ 'flight'	$\{\ \lambda \boldsymbol{x}.Flight(\boldsymbol{x})\ \}$
$Verb$	$\rightarrow$ 'arrived'	$\{\lambda \boldsymbol{y}.\exists \boldsymbol{e} Arrived(\boldsymbol{e})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


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

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




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

S

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

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

NP

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

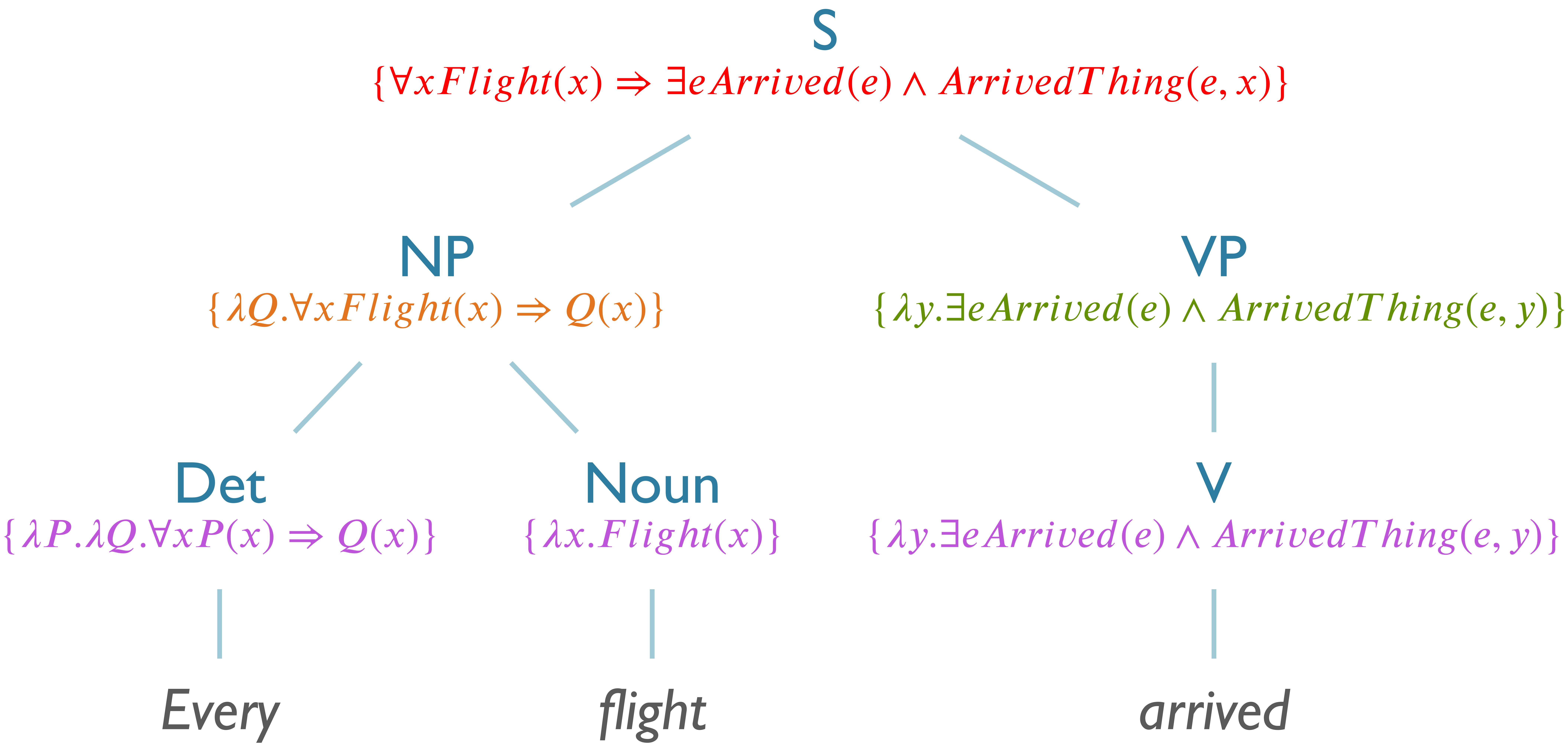
VP

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

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

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

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


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
$\{\lambda y.\exists e Arrived(e) \wedge ArrivedThing(e,y)\}$
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 on Wednesday.

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

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
- 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