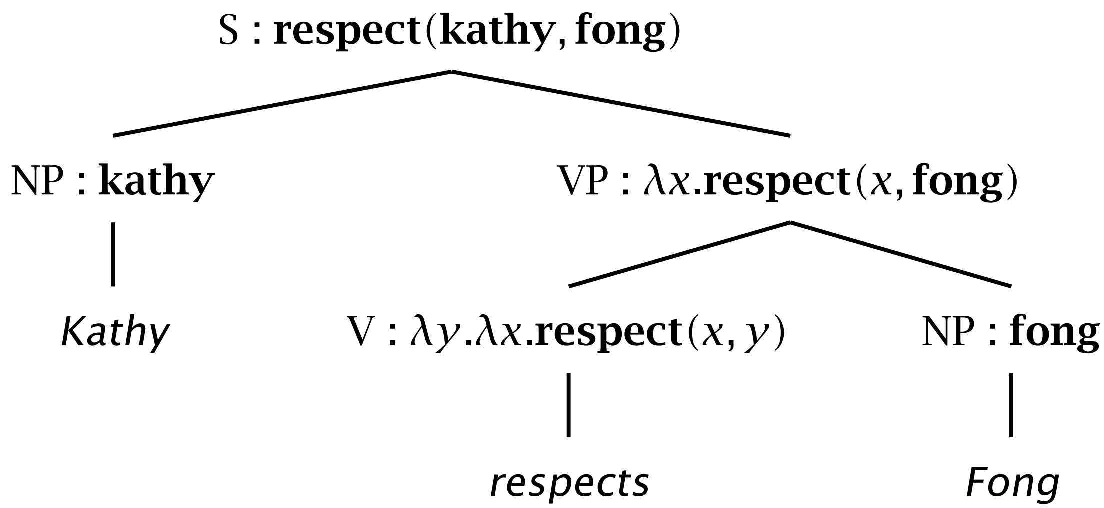
A first example

Lexicon	**Grammar**
Kathy, NP : **kathy**	S : $\beta(\alpha)$ → NP : α VP : β
Fong, NP : **fong**	VP : $\beta(\alpha)$ → V : β NP : α
respects, V : $\lambda y.\lambda x.\mathbf{respect}(x, y)$	VP : β → V : β
runs, V : $\lambda x.\mathbf{run}(x)$	

A first example

- 

- [$_{\text{VP}}$ respects Fong] : $[\lambda y.\lambda x.\mathbf{respect}(x, y)](\mathbf{fong})$

  $= \lambda x.\mathbf{respect}(x, \mathbf{fong})$ [β red.]

  [$_{\text{S}}$ Kathy respects Fong] : $[\lambda x.\mathbf{respect}(x, \mathbf{fong})](\mathbf{kathy})$

  $=$ **respect(kathy,fong)**

Database/knowledgebase interfaces

- Assume that **respect** is a table Respect with two fields respecter and respected
- Assume that **kathy** and **fong** are IDs in the database: k and f
- If we assert *Kathy respects Fong* we might evaluate the form **respect**(**fong**)(**kathy**) by doing an insert operation:

    insert into Respects(respecter, respected) values (k, f)

Database/knowledgebase interfaces

- Below we focus on questions like *Does Kathy respect Fong* for which we will use the relation to ask:

  select 'yes' from Respects where Respects.respecter = k and Respects.respected = f

- We interpret "no rows returned" as 'no' = **0**.

Typed λ calculus (Church 1940)

- Everything has a type (like Java!)
- **Bool** truth values (**0** and **1**)
  Ind individuals
  Ind → **Bool** properties
  Ind → **Ind** → **Bool** binary relations
- **kathy** and **fong** are **Ind**

  run is **Ind** → **Bool**

  respect is **Ind** → **Ind** → **Bool**
- Types are interpreted right associatively.

  respect is **Ind** → (**Ind** → **Bool**)
- We convert a several argument function into embedded unary functions. Referred to as *currying*.

Typed λ calculus (Church 1940)

- Once we have types, we don't need λ variables just to show what arguments something takes, and so we can introduce another operation of the λ calculus:

  η reduction [abstractions can be contracted]

  $\lambda x.(P(x)) \Rightarrow P$

- This means that instead of writing:

  $\lambda y.\lambda x.\mathbf{respect}(x, y)$

  we can just write:

  respect

Typed λ calculus (Church 1940)

- λ extraction allowed over any type (not just first-order)
- β reduction [application]

  $(\lambda x.P(\cdots, x, \cdots))(Z) \Rightarrow P(\cdots, Z, \cdots)$
- η reduction [abstractions can be contracted]

  $\lambda x.(P(x)) \Rightarrow P$
- α reduction [renaming of variables]

Typed λ calculus (Church 1940)

- The first form we introduced is called the β, η long form, and the second more compact representation (which we use quite a bit below) is called the β, η normal form. Here are some examples:
β, η normal form	β, η long form
run	$\lambda x.\mathbf{run}(x)$
$\mathbf{every}^2(\mathbf{kid}, \mathbf{run})$	$\mathbf{every}^2((\lambda x.\mathbf{kid}(x)), (\lambda x.\mathbf{run}(x)))$
$\mathbf{yesterday}(\mathbf{run})$	$\lambda y.\mathbf{yesterday}(\lambda x.\mathbf{run}(x))(y)$

Types of major syntactic categories

- nouns and verb phrases will be properties (**Ind** $\rightarrow$**Bool**)
- noun phrases are **Ind** – though they are commonly type-raised to (**Ind** $\rightarrow$ **Bool**) $\rightarrow$ **Bool**
- adjectives are (**Ind** $\rightarrow$ **Bool**) $\rightarrow$ (**Ind** $\rightarrow$ **Bool**)

  This is because adjectives modify noun meanings, that is properties.
- Intensifiers modify adjectives: e.g, *very* in *a very happy camper*, so they're ((**Ind** $\rightarrow$ **Bool**) $\rightarrow$ (**Ind** $\rightarrow$ **Bool**)) $\rightarrow$ ((**Ind** $\rightarrow$ **Bool**) $\rightarrow$ (**Ind** $\rightarrow$ **Bool**)) [honest!].

A grammar fragment

- $S : \beta(\alpha) \rightarrow NP : \alpha \quad VP : \beta$
  $NP : \beta(\alpha) \rightarrow Det : \beta \quad N' : \alpha$
  $N' : \beta(\alpha) \rightarrow Adj : \beta \quad N' : \alpha$
  $N' : \beta(\alpha) \rightarrow N' : \alpha \quad PP : \beta$
  $N' : \beta \rightarrow N : \beta$
  $VP : \beta(\alpha) \rightarrow V : \beta \quad NP : \alpha$
  $VP : \beta(\gamma)(\alpha) \rightarrow V : \beta \quad NP : \alpha \quad NP : \gamma$
  $VP : \beta(\alpha) \rightarrow VP : \alpha \quad PP : \beta$
  $VP : \beta \rightarrow V : \beta$
  $PP : \beta(\alpha) \rightarrow P : \beta \quad NP : \alpha$

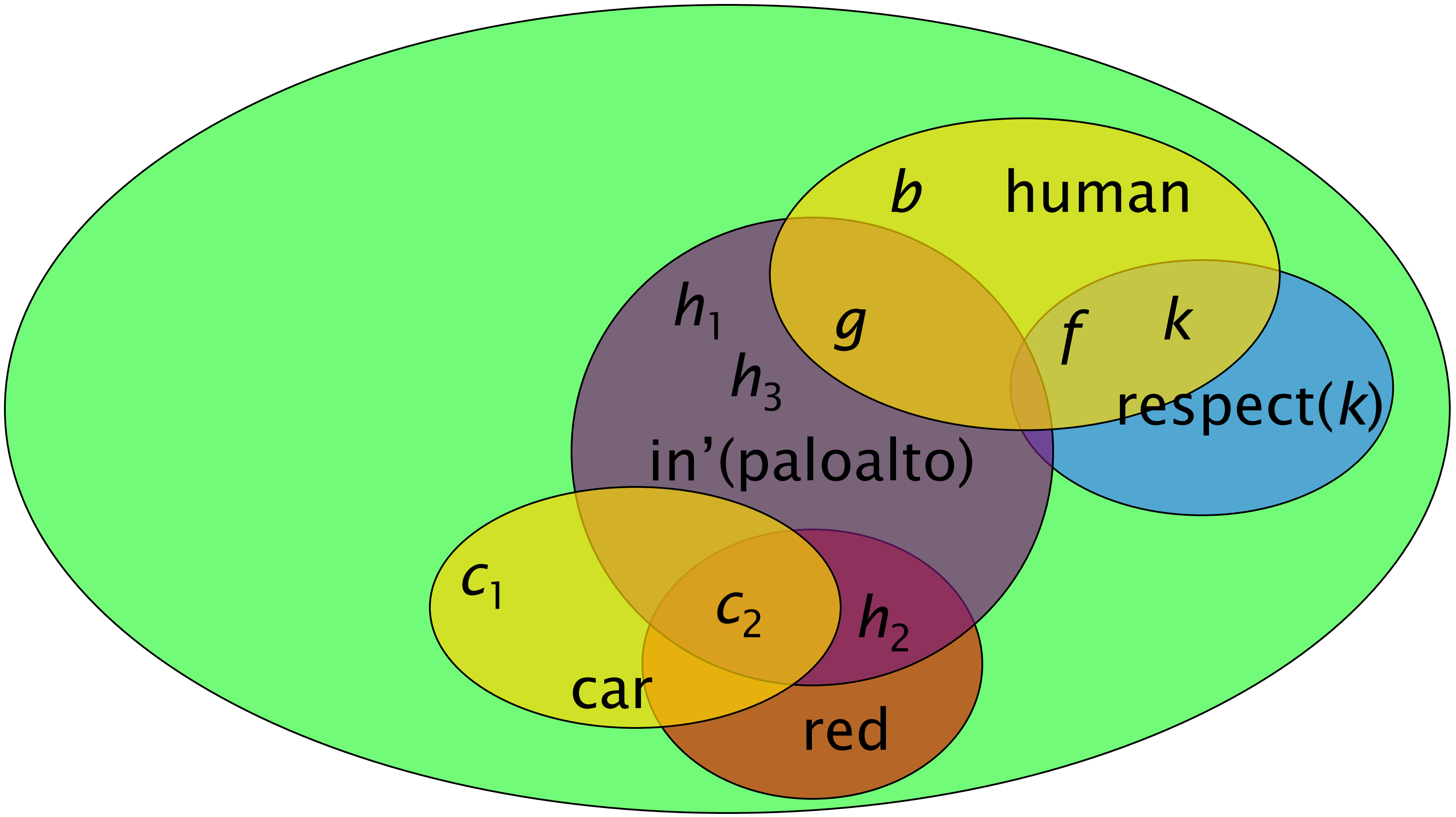
A grammar fragment

- *Kathy*, NP : $\mathbf{kathy}_{\mathbf{Ind}}$
  Fong, NP : $\mathbf{fong}_{\mathbf{Ind}}$
  Palo Alto, NP : $\mathbf{paloalto}_{\mathbf{Ind}}$
  car, N : $\mathbf{car}_{\mathbf{Ind} \to \mathbf{Bool}}$
  overpriced, Adj : $\mathbf{overpriced}_{(\mathbf{Ind} \to \mathbf{Bool}) \to (\mathbf{Ind} \to \mathbf{Bool})}$
  outside, PP : $\mathbf{outside}_{(\mathbf{Ind} \to \mathbf{Bool}) \to (\mathbf{Ind} \to \mathbf{Bool})}$
  red, Adj : $\lambda P.(\lambda x.P(x) \wedge \mathbf{red}'(x))$
  in, P : $\lambda y.\lambda P.\lambda x.(P(x) \wedge \mathbf{in}'(y)(x))$
  the, Det : ι
  a, Det : $\mathbf{some}^{2}{}_{(\mathbf{Ind} \to \mathbf{Bool}) \to (\mathbf{Ind} \to \mathbf{Bool}) \to \mathbf{Bool}}$
  runs, V : $\mathbf{run}_{\mathbf{Ind} \to \mathbf{Bool}}$
  respects, V : $\mathbf{respect}_{\mathbf{Ind} \to \mathbf{Ind} \to \mathbf{Bool}}$
  likes, V : $\mathbf{like}_{\mathbf{Ind} \to \mathbf{Ind} \to \mathbf{Bool}}$

A grammar fragment

- **in′** is **Ind** → **Ind** → **Bool**
- **in** $\stackrel{\text{def}}{=}$ $\lambda y.\lambda P.\lambda x.(P(x) \wedge$ **in′**$(y)(x))$ is **Ind** → (**Ind** → **Bool**) →(**Ind** → **Bool**)
- **red′** is **Ind** → **Bool**
- **red** $\stackrel{\text{def}}{=}$ $\lambda P.(\lambda x.(P(x) \wedge$ **red′**$(x))$ is (**Ind** → **Bool**) →(**Ind** → **Bool**)

Model theory – A formalization of a "database"



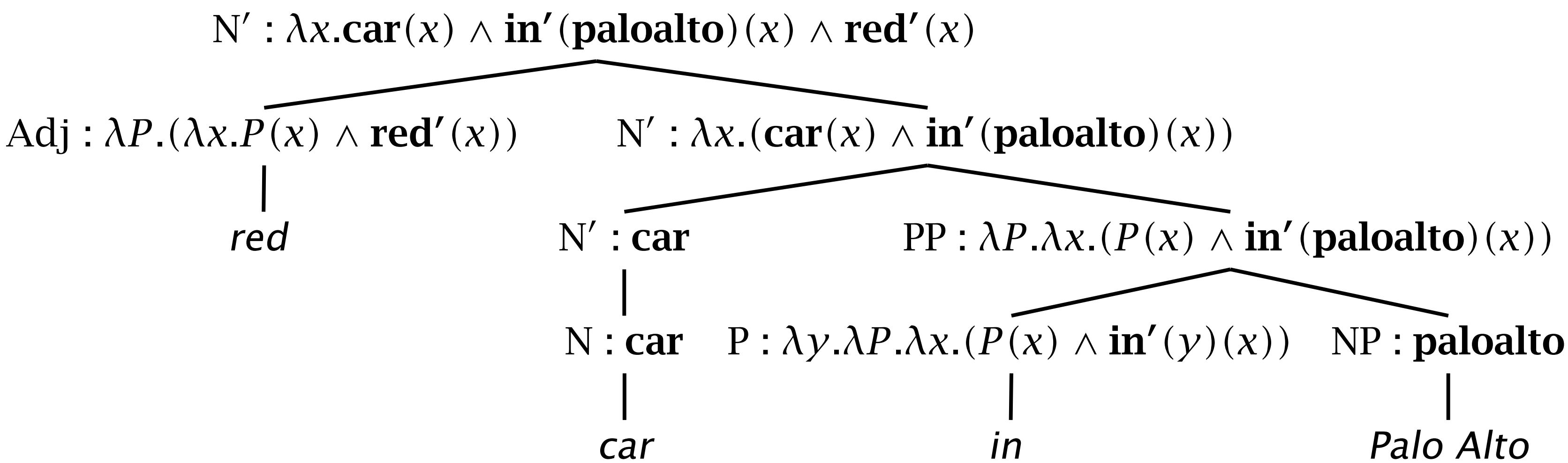
Properties

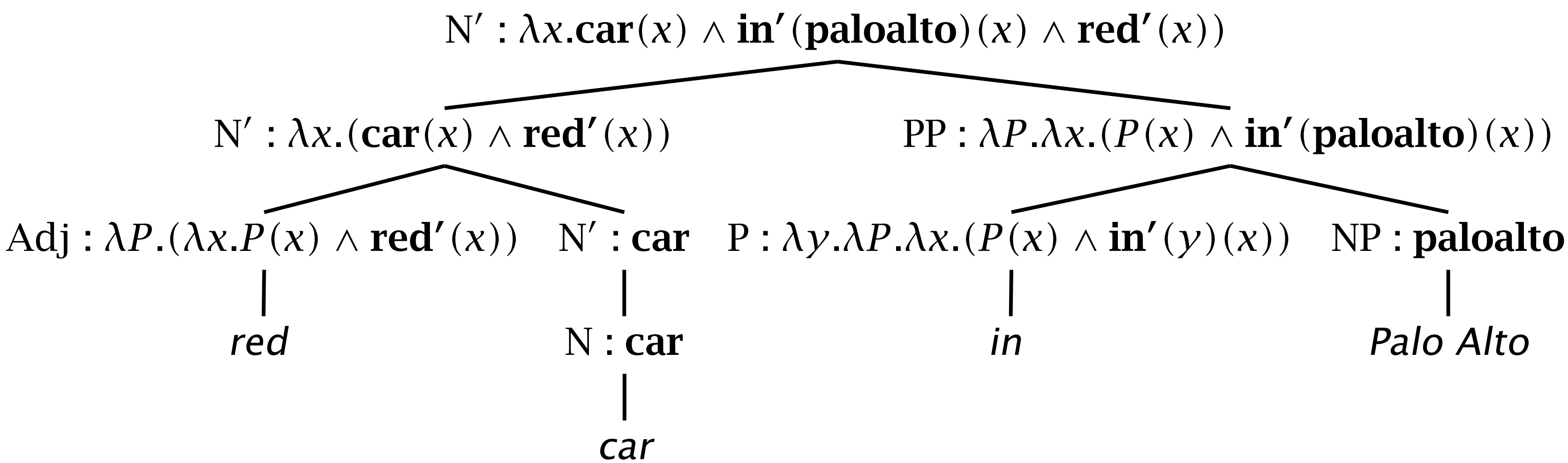
Curried multi-argument functions

$$[\![\mathbf{respect}]\!] = [\![\lambda y.\lambda x.\mathbf{respect}(x, y)]\!] = \begin{bmatrix} f \mapsto \begin{bmatrix} f \mapsto \mathbf{0} \\ k \mapsto \mathbf{1} \\ b \mapsto \mathbf{0} \end{bmatrix} \\ k \mapsto \begin{bmatrix} f \mapsto \mathbf{1} \\ k \mapsto \mathbf{1} \\ b \mapsto \mathbf{0} \end{bmatrix} \\ b \mapsto \begin{bmatrix} f \mapsto \mathbf{1} \\ k \mapsto \mathbf{0} \\ b \mapsto \mathbf{0} \end{bmatrix} \end{bmatrix}$$

$$[\![\lambda x.\lambda y.\mathbf{respect}(y)(x)(b)(f)]\!] = \mathbf{1}$$

Adjective and PP modification

- 

- 

Intersective adjectives

- Syntactic ambiguity is spurious: you get the same semantics either way
- Database evaluation is possible via a table join

Non-intersective adjectives

- For non-intersective adjectives get different semantics depending on what they modify
- **overpriced**(**in**(**paloalto**)(**house**))
- **in**(**paloalto**)(**overpriced**(**house**))
- But probably won't be able to evaluate it on database!

Why things get more complex

- When doing predicate logic did you wonder why:
  - *Kathy runs* is $\textbf{run}(\textbf{kathy})$
  - *no kid runs* is $\neg(\exists x)(\textbf{kid}(x) \wedge \textbf{run}(x))$
- Somehow the NP's meaning is wrapped around the predicate
- Or consider why this argument doesn't hold:
  - Nothing is better than a life of peace and prosperity.
    A cold egg salad sandwich is better than nothing.

    A cold egg salad sandwich is better than a life of peace and prosperity.
- The problem is that *nothing* is a quantifier

Generalized Quantifiers

- We have a reasonable semantics for *red car in Palo Alto* as a property from **Ind** → **Bool**
- How do we represent noun phrases like *the red car in Palo Alto* or *every red car in Palo Alto*?
- $[\![\iota]\!](P) = a$ if ($P(b) = \mathbf{1}$ iff $b = a$)
  undefined, otherwise
- The semantics for *the* following Bertrand Russell, for whom *the* x meant the unique item satisfying a certain description

Generalized Quantifiers

- *red car in Palo Alto*

  select Cars.obj from Cars, Locations, Red where

  Cars.obj = Locations.obj AND

  Locations.place = 'paloalto' AND Cars.obj = Red.obj

  (here we assume the unary relations have one field, obj).

Generalized Quantifiers

- *the red car in Palo Alto*

- NP : $\iota(\lambda x.\mathbf{car}(x) \wedge \mathbf{in'}(\mathbf{paloalto})(x) \wedge \mathbf{red'}(x))$

  Det : ι N' : $\lambda x.\mathbf{car}(x) \wedge \mathbf{in'}(\mathbf{paloalto})(x) \wedge \mathbf{red'}(x)$

  the *red car in Palo Alto*

- *the red car in Palo Alto*

  select Cars.obj from Cars, Locations, Red where

  Cars.obj = Locations.obj AND

  Locations.place = ‘paloalto’ AND Cars.obj = Red.obj

  having count(*) = 1

Generalized Quantifiers

- What then of *every red car in Palo Alto*?
- A generalized determiner is a relation between two properties, one contributed by the restriction from the N′, and one contributed by the predicate quantified over:

$$(\textbf{Ind} \rightarrow \textbf{Bool}) \rightarrow (\textbf{Ind} \rightarrow \textbf{Bool}) \rightarrow \textbf{Bool}$$

- Here are some determiners

$$\textbf{some}^2(\textbf{kid})(\textbf{run}) \equiv \textbf{some}(\lambda x.\textbf{kid}(x) \wedge \textbf{run}(x))$$

$$\textbf{every}^2(\textbf{kid})(\textbf{run}) \equiv \textbf{every}(\lambda x.\textbf{kid}(x) \rightarrow \textbf{run}(x))$$

Generalized Quantifiers

- Generalized determiners are implemented via the quantifiers:

  every$(P) = \mathbf{1}$ iff $(\forall x)P(x) = \mathbf{1}$;

  i.e., if $P = \mathbf{Dom_{Ind}}$

  some$(P) = \mathbf{1}$ iff $(\exists x)P(x) = \mathbf{1}$; i.e., if $P \neq \emptyset$

Generalized Quantifiers

- Every student likes the red car
- $\mathrm{S} : \mathbf{every^2}(\mathbf{student})(\mathbf{like}(\iota(\lambda x.\mathbf{car}(x) \wedge \wedge \mathbf{red'}(x))))$
  - $\mathrm{NP} : \mathbf{every^2}(\mathbf{student})$
    - $\mathrm{Det} : \mathbf{every^2}$
      - *every*
    - $\mathrm{N'} : \mathbf{student}$
      - *student*
  - $\mathrm{VP} : \mathbf{like}(\iota(\lambda x.\mathbf{car}(x) \wedge \mathbf{red'}(x)))$
    - $\mathrm{V} : \mathbf{like}$
      - *likes*
    - $\mathrm{NP} : \iota(\lambda x.\mathbf{car}(x) \wedge \mathbf{red'}(x))$
      - $\mathrm{Det} : \iota$
        - *the*
      - $\mathrm{N'} : \lambda x.(\mathbf{car}(x) \wedge \mathbf{red'}(x))$
        - $\mathrm{Adj} : \lambda P.(\lambda x.P(x) \wedge \mathbf{red'}(x))$
          - *red*
        - $\mathrm{N'} : \mathbf{car}$
          - $\mathrm{N} : \mathbf{car}$
            - *car*

Questions with answers!

- A yes/no question (*Is Kathy running?*) will be something of type **Bool**, checked on database
- A content question (*Who likes Kathy?*) will be an *open proposition*, that is something semantically of the type *property* (**Ind** → **Bool**), and operationally we will consult the database to see what individuals will make the statement true.
- We use a grammar with a simple form of gap-threading for question words

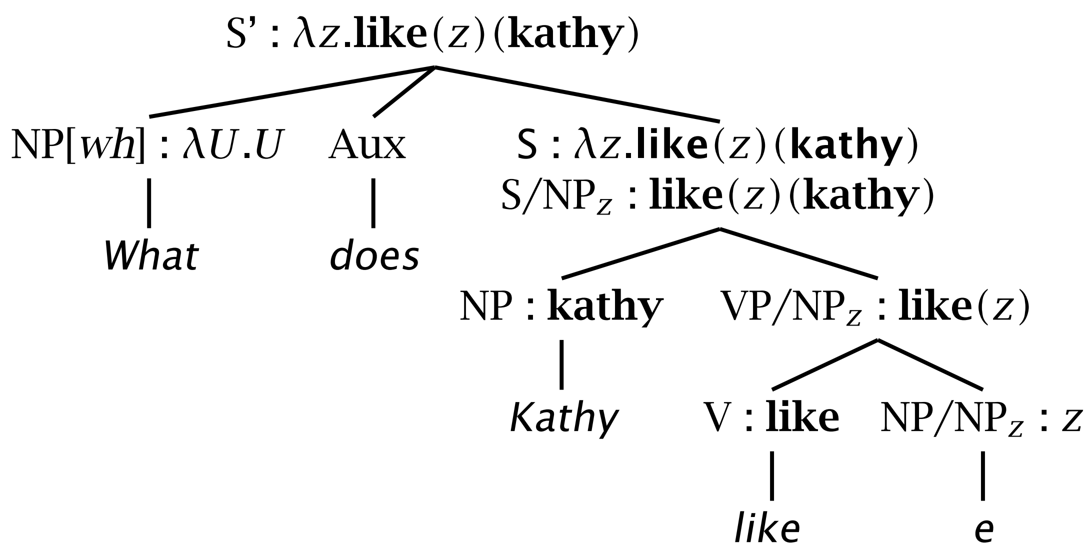
Syntax/semantics for questions

- $\mathsf{S}' : \beta(\alpha) \rightarrow \mathsf{NP}[wh] : \beta \quad \mathsf{Aux} \quad \mathsf{S} : \alpha$
  $\mathsf{S}' : \alpha \rightarrow \mathsf{Aux} \quad \mathsf{S} : \alpha$
  $\mathsf{NP/NP}_Z : z \rightarrow \mathsf{e}$
  $\mathsf{S} : \lambda z.F(\ldots z \ldots) \rightarrow \mathsf{S/NP}_Z : F(\ldots z \ldots)$

Syntax/semantics for questions

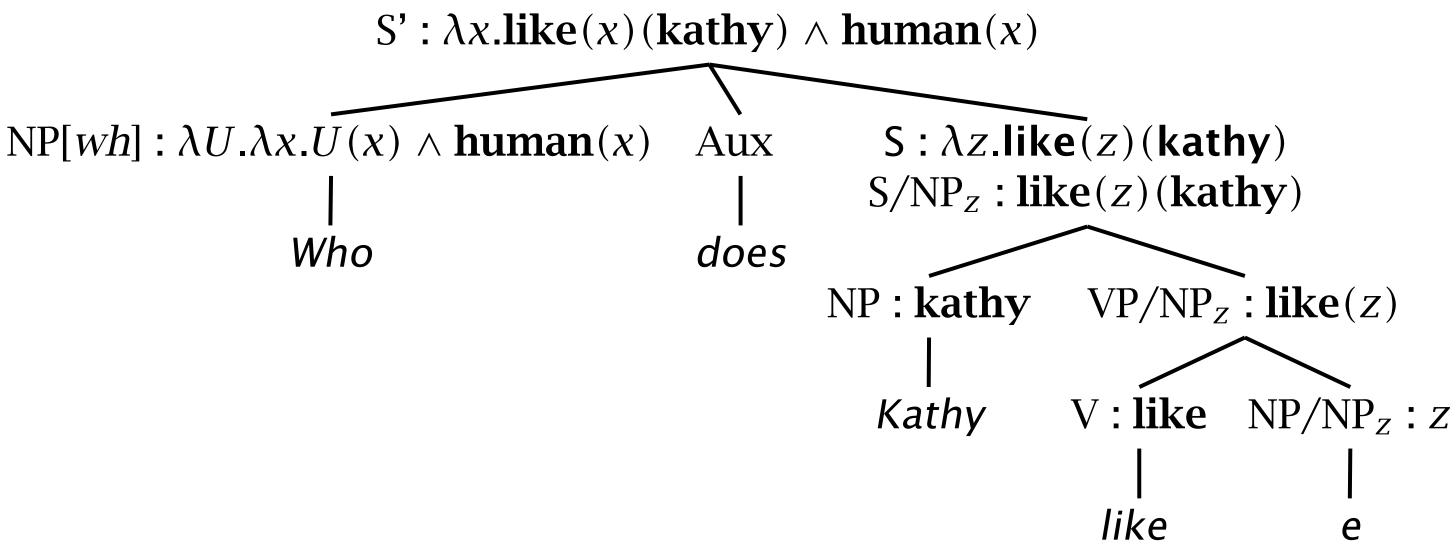
- *who*, NP[*wh*] : $\lambda U.\lambda x.U(x) \wedge \textbf{human}(x)$
  what, NP[*wh*] : $\lambda U.U$
  which, Det[*wh*] : $\lambda P.\lambda V.\lambda x.P(x) \wedge V(x)$
  how_many, Det[*wh*] : $\lambda P.\lambda V.|\lambda x.P(x) \wedge V(x)|$
- Where $|\cdot|$ is the operation that returns the cardinality of a set (count).

Question examples

- 

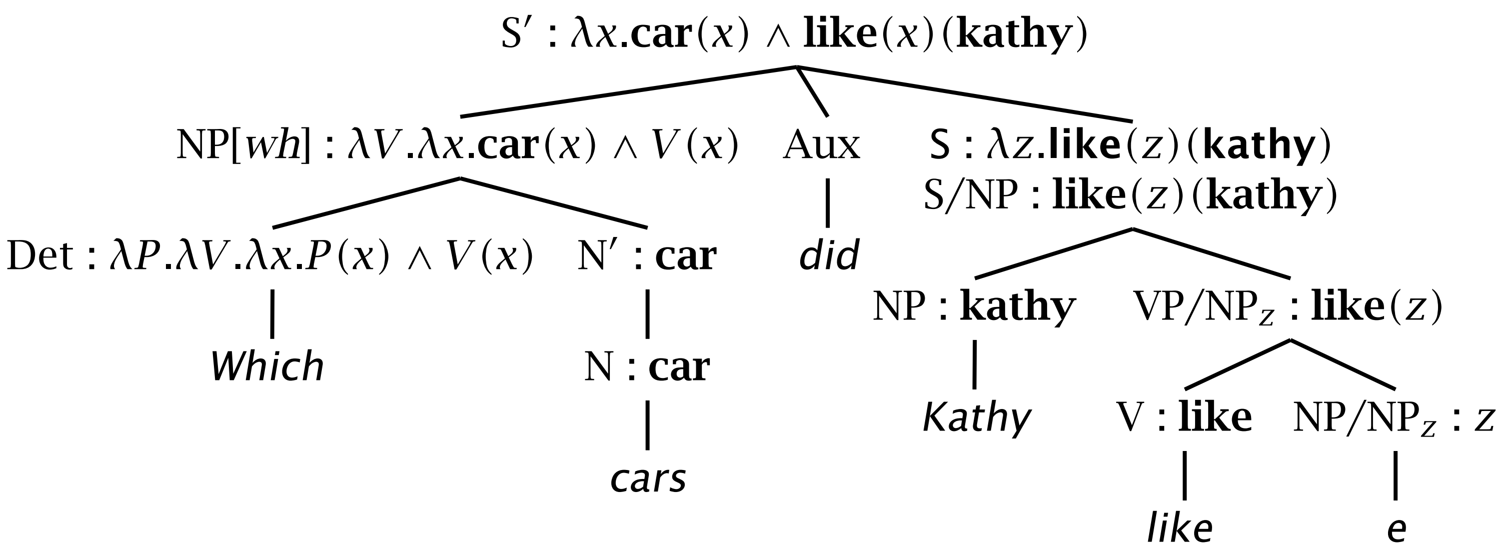
- select liked from Likes where Likes.liker='Kathy'

Question examples

- 

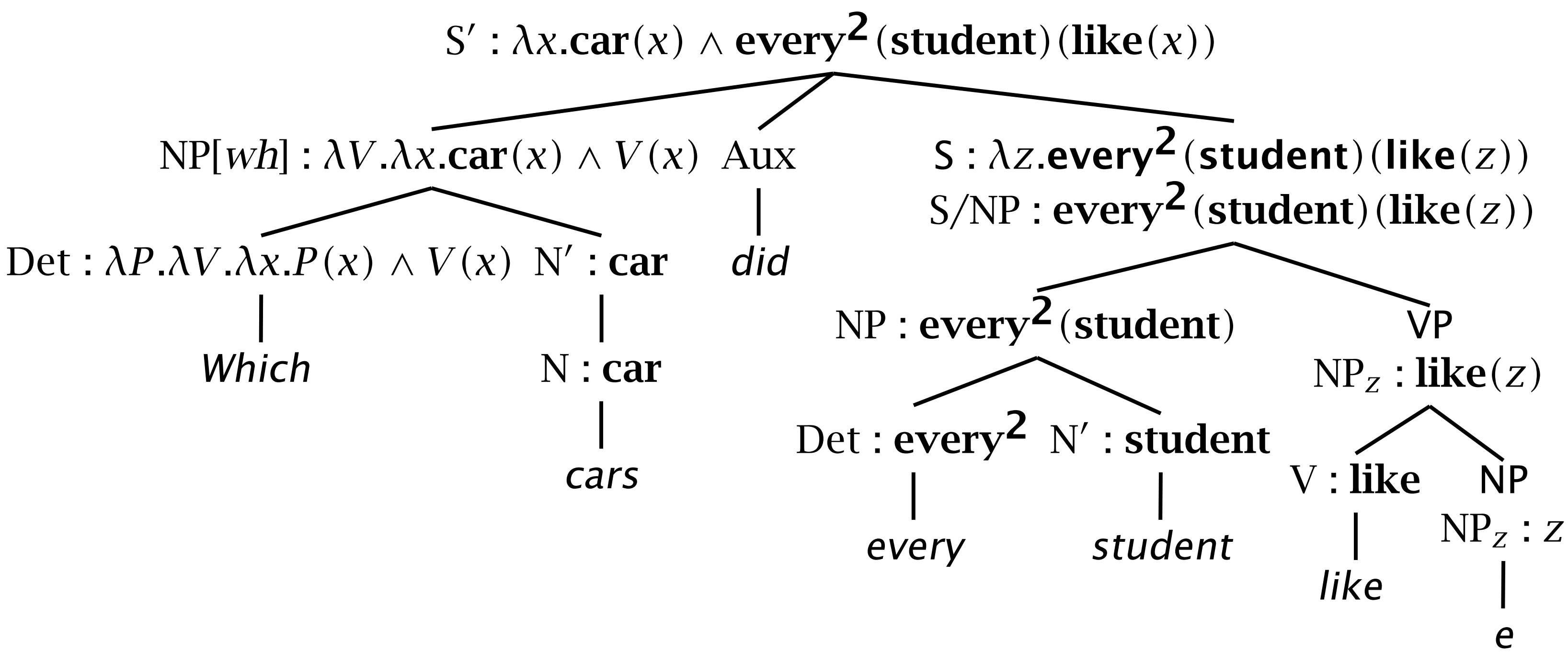
- select liked from Likes,Humans where Likes.liker='Kathy' AND Humans.obj = Likes.liked

Question examples

- 

- select liked from Cars,Likes where Cars.obj=Likes.liked AND Likes.liker='Kathy'

Question examples

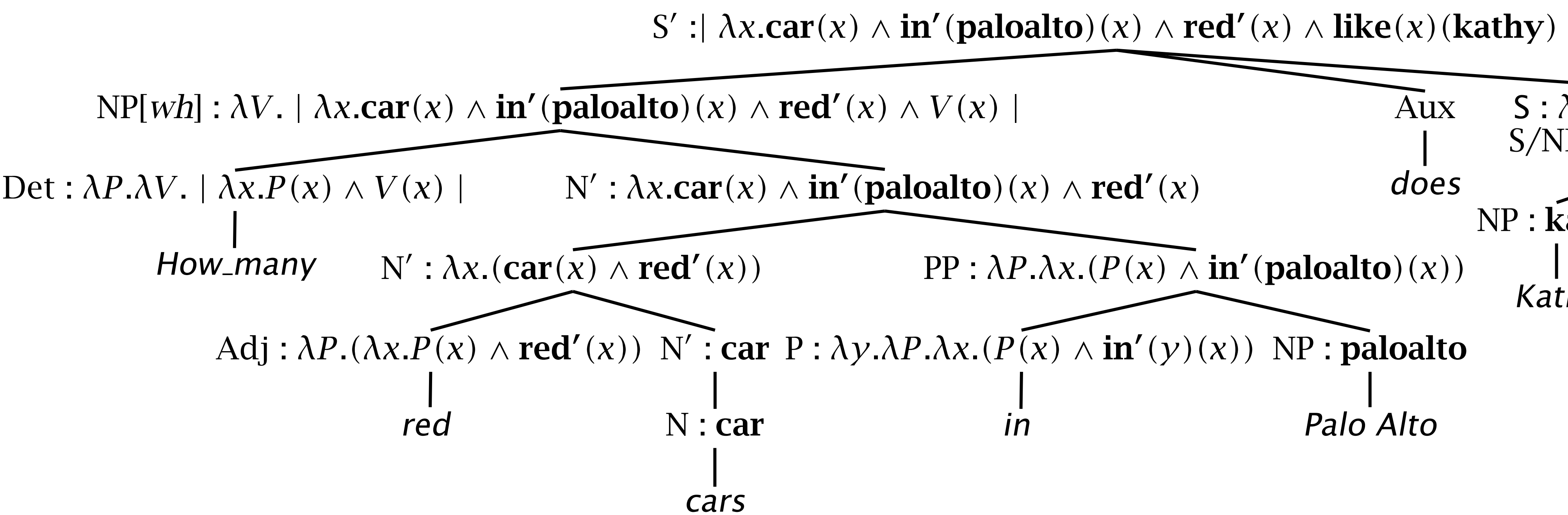
- 

- ???

Question examples

- *How many red cars in Palo Alto does Kathy like?*
- select count(*) from Likes,Cars,Locations,Reds where Cars.obj = Likes.liked AND Likes.liker = 'Kathy' AND Red.obj = Likes.liked AND Locations.place = 'Palo Alto' AND Locations.obj = Likes.liked
- *Did Kathy see the red car in Palo Alto?*
- select 'yes' where Seeings.seer = k AND Seeings.seen = (select Cars.obj from Cars, Locations, Red where Cars.obj = Locations.obj AND Locations.place = 'paloalto' AND Cars.obj = Red.obj having count(*) = 1)

How many red cars in Palo Alto does Kathy like?



Did Kathy see the red car in Palo Alto?

S' : $\mathbf{see}(\iota(\lambda x.\mathbf{car}(x) \wedge \mathbf{in'}(\mathbf{paloalto})(x) \wedge \mathbf{red'}(x)))(\mathbf{kathy})$

Aux   S : $\mathbf{see}(\iota(\lambda x.\mathbf{car}(x) \wedge \mathbf{in'}(\mathbf{paloalto})(x) \wedge \mathbf{red'}(x)))(\mathbf{kathy})$

Did   NP : $\mathbf{kathy}$ VP : $\mathbf{see}(\iota(\lambda x.\mathbf{car}(x) \wedge \mathbf{in'}(\mathbf{paloalto})(x) \wedge \mathbf{red'}(x)))$

Kathy   V : $\mathbf{see}$ NP : $\iota(\lambda x.\mathbf{car}(x) \wedge \mathbf{in'}(\mathbf{paloalto})(x) \wedge \mathbf{red'}(x))$

see   Det : ι N′ : $\lambda x.\mathbf{car}(x) \wedge \mathbf{in'}(\mathbf{paloalto})(x$

the   Adj : $\lambda P.(\lambda x.P(x) \wedge \mathbf{red'}(x))$ N′ : $\lambda x.(\mathbf{ca}$

red

N : $\mathbf{car}$

car   P : λ