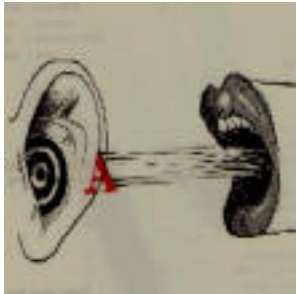


Lecture 12: Semantics II



Professor Robert C. Berwick
berwick@csail.mit.edu

The Menu Bar

- Administrivia: Projects
 - What knowledge do we need beyond syntax?
 - How should we represent this?
 - How can we learn it?
-
- From last time: representing ‘meaning’ as logical statements...we left off here...

Quantifier Order

- *Gilly swallowed a goldfish in a booth*
 - $\exists e \text{ past}(e), \text{ act}(e, \text{swallowing}), \text{ swallower}(e, \text{Gilly}), \text{ exists}(\text{goldfish}, \text{ swallowee}(e)), \text{ exists}(\text{booth}, \text{ location}(e)), \dots$
- *Gilly swallowed a goldfish in every booth*
 - $\exists e \text{ past}(e), \text{ act}(e, \text{swallowing}), \text{ swallower}(e, \text{Gilly}), \text{ exists}(\text{goldfish}, \text{ swallowee}(e)), \text{ all}(\text{booth}, \text{ location}(e)), \dots$

$\exists g \text{ goldfish}(g), \text{ swallowee}(e, g) \quad \forall b \text{ booth}(b) \Rightarrow \text{location}(e, b)$

- Does this mean what we'd expect??

says that there's only one event
with a single goldfish getting swallowed
that took place in a lot of booths ...

Quantifier Order

- Groucho Marx celebrates quantifier order ambiguity:
 - *In this country a woman gives birth every 15 minutes*
 - *Our job is to find that woman and stop her*
 - $\exists \text{woman } (\forall 15\text{min gives-birth-during}(\text{woman}, 15\text{min}))$
 - $\forall 15\text{min } (\exists \text{woman gives-birth-during}(15\text{min}, \text{woman}))$
 - Surprisingly, both are possible in natural language!
 - Which is the joke meaning (where it's always the same woman) and why?

Quantifier Order

- *Gilly swallowed a goldfish in a booth*
 - $\exists e$ past(e), act(e,swallowing), swallower(e,Gilly), exists(goldfish, swallowee(e)), exists(booth, location(e)), ...
- *Gilly swallowed a goldfish in every booth*
 - $\exists e$ past(e), act(e,swallowing), swallower(e,Gilly), exists(goldfish, swallowee(e)), all(booth, location(e)), ...

$\exists g$ goldfish(g), swallowee(e,g) $\forall b$ booth(b) \Rightarrow location(e,b)

- Does this mean what we'd expect??
 - It's $\exists e \forall b$ which means same event for every booth
 - Probably false unless Gilly can be in every booth during her swallowing of a single goldfish

Quantifier Order

- *Gilly swallowed a goldfish in a booth*
 - $\exists e \text{ past}(e), \text{ act}(e, \text{swallowing}), \text{ swallower}(e, \text{Gilly}), \text{ exists}(\text{goldfish}, \text{ swallowee}(e)), \text{ exists}(\text{booth}, \text{ location}(e)), \dots$
- *Gilly swallowed a goldfish in every booth*
 - $\exists e \text{ past}(e), \text{ act}(e, \text{swallowing}), \text{ swallower}(e, \text{Gilly}), \text{ exists}(\text{goldfish}, \text{ swallowee}(e)), \text{ all}(\text{booth}, \lambda b \text{ location}(e, b))$
- Other reading ($\forall b \exists e$) involves quantifier raising:
 - $\text{ all}(\text{booth}, \lambda b [\exists e \text{ past}(e), \text{ act}(e, \text{swallowing}), \text{ swallower}(e, \text{Gilly}), \text{ exists}(\text{goldfish}, \text{ swallowee}(e)), \text{ location}(e, b)])$
 - “for all booths b, there was such an event in b”

Intensional Arguments

- *Willy wants a unicorn*
 - $\exists e \text{ act}(e, \text{wanting}), \text{wanter}(e, \text{Willy}), \text{exists}(\text{unicorn}, \lambda u \text{ wantee}(e, u))$
 - “there is a unicorn u that Willy wants”
 - here the *wantee* is an individual entity
 - $\exists e \text{ act}(e, \text{wanting}), \text{wanter}(e, \text{Willy}), \text{wantee}(e, \lambda u \text{ unicorn}(u))$
 - “Willy wants any entity u that satisfies the unicorn predicate”
 - here the *wantee* is a type of entity
- *Willy wants Lilly to get married*
 - $\exists e \text{ present}(e), \text{act}(e, \text{wanting}), \text{wanter}(e, \text{Willy}), \text{wantee}(e, \lambda e' [\text{act}(e', \text{marriage}), \text{marrier}(e', \text{Lilly})])$
 - “Willy wants any event e' in which Lilly gets married”
 - Here the *wantee* is a type of event
 - Sentence doesn't claim that such an event exists
- Intensional verbs besides *want*: *hope, doubt, believe, ...*

Intensional Arguments

- *Willy wants a unicorn*
 - $\exists e \text{ act}(e, \text{wanting}), \text{wanter}(e, \text{Willy}), \text{wantee}(e, \lambda g \text{ unicorn}(g))$
 - “Willy wants anything that satisfies the unicorn predicate”
 - here the wantee is a type of entity
- Problem (a fine point I’ll gloss over):
 - $\lambda g \text{ unicorn}(g)$ is defined by the actual set of unicorns (“extension”)
 - But this set is empty: $\lambda g \text{ unicorn}(g) = \lambda g \text{ FALSE} = \lambda g \text{ dodo}(g)$
 - Then *wants a unicorn* = *wants a dodo*. Oops!
 - So really the wantee should be criteria for unicornness (“intension”)
- Traditional solution involves “possible-world semantics”
 - Can imagine **other worlds** where set of unicorns \neq set of dodos

Control

- *Willy wants Lilly to get married*
 - $\exists e \text{ present}(e), \text{ act}(e, \text{wanting}), \text{ wanter}(e, \text{Willy}), \text{ wantee}(e, \lambda f [\text{act}(f, \text{marriage}), \text{ marrier}(f, \text{Lilly})])$
- *Willy wants to get married*
 - Same as *Willy wants Willy to get married*
 - Just as easy to represent as *Willy wants Lilly ...*
 - The only trick is to construct the representation from the syntax. The empty subject position of “to get married” is said to be controlled by the subject of “wants.”

Nouns and Their Modifiers

- *expert*
 - $\lambda g \text{ expert}(g)$
- *big fat expert* (cf, *big fat Greek wedding...*)
 - $\lambda g \text{ big}(g), \text{ fat}(g), \text{ expert}(g)$
 - But: *bogus expert*
 - Wrong: $\lambda g \text{ bogus}(g), \text{ expert}(g)$
 - Right: $\lambda g (\text{bogus}(\text{expert}))(g)$... *bogus maps to new concept*
- *Boston expert* (*white-collar expert, TV expert ...*)
 - $\lambda g \text{ Related}(\text{Boston}, g), \text{ expert}(g)$ – expert from Boston
 - Or with different intonation:
 - $\lambda g (\text{Modified-by}(\text{Boston}, \text{expert}))(g)$ – expert on Boston
 - Can't use *Related* for that case: *law expert and dog catcher*
= $\lambda g \text{ Related}(\text{law}, g), \text{ expert}(g), \text{ Related}(\text{dog}, g), \text{ catcher}(g)$
= *dog expert and law catcher*

Nouns and Their Modifiers

- *the goldfish that Gilly swallowed*
- *every goldfish that Gilly swallowed*
- *three goldfish that Gilly swallowed*

λg [goldfish(g), swallowed(Gilly, g)]

like an adjective!

- *three swallowed-by-Gilly goldfish*

Or for real: λg [goldfish(g), $\exists e$ [past(e), act(e,swallowing),
swallower(e,Gilly), swallowee(e,g)]]

Adverbs

- *Lili passionately wants Billy*
 - Wrong?: $\text{passionately}(\text{want}(\text{Lili}, \text{Billy})) = \text{passionately}(\text{true})$
 - Better: $(\text{passionately}(\text{want}))(\text{Lili}, \text{Billy})$
 - Best: $\exists e \text{ present}(e), \text{act}(e, \text{wanting}), \text{wanter}(e, \text{Lili}), \text{wantee}(e, \text{Billy}), \text{manner}(e, \text{passionate})$
- *Lili often stalks Billy*
 - $(\text{often}(\text{stalk}))(\text{Lili}, \text{Billy})$
 - $\text{many}(\text{day}, \lambda d \exists e \text{ present}(e), \text{act}(e, \text{stalking}), \text{stalker}(e, \text{Lili}), \text{stalkee}(e, \text{Billy}), \text{during}(e, d))$
- *Lili obviously likes Billy*
 - $(\text{obviously}(\text{like}))(\text{Lili}, \text{Billy})$ – one reading
 - $\text{obvious}(\text{likes}(\text{Lili}, \text{Billy}))$ – another reading

Speech Acts

- What is the meaning of a full sentence?
 - Depends on the punctuation mark at the end. 😊
 - Billy likes Lili. → **assert**(like(B,L))
 - Billy likes Lili? → **ask**(like(B,L))
 - or more formally, “Does Billy like Lili?”
 - Billy, like Lili! → **command**(like(B,L))
- Let’s try to do this a little more precisely, using event variables etc.

Speech Acts

- *What did Gilly swallow?*
 - **ask**($\lambda x \exists e \text{ past}(e), \text{act}(e, \text{swallowing}),$
 $\text{swallower}(e, \text{Gilly}), \text{swallowee}(e, x)$)
 - Argument is identical to the modifier “that Gilly swallowed”
 - Is there any common syntax?
- *Eat your fish!*
 - **command**($\lambda f \text{ act}(f, \text{eating}), \text{eater}(f, \text{Hearer}), \text{eatee}(\dots)$)
- *I ate my fish.*
 - **assert**($\exists e \text{ past}(e), \text{act}(e, \text{eating}), \text{eater}(f, \text{Speaker}),$
 $\text{eatee}(\dots)$)
- How do we map from syntax to this kind of semantics???

Standard answer to the last question, at least

- Syntax directed translation
- Output: instructions to do something in terms of some machine model
- Note the principles involved:
 - For each syntactic rule, there's a corresponding local semantic translation rule
 - Compositionality: meaning of whole is *entirely* a function of the meaning of its parts

Hard case

(But the Accord was redesigned for the 2003 model year.)

The roomier, faster, and sleeker sedan's sales stabilized last year, falling by just 1,230 units -- a strong showing in a market that saw combined total passenger car sales fall by 471,000 units.

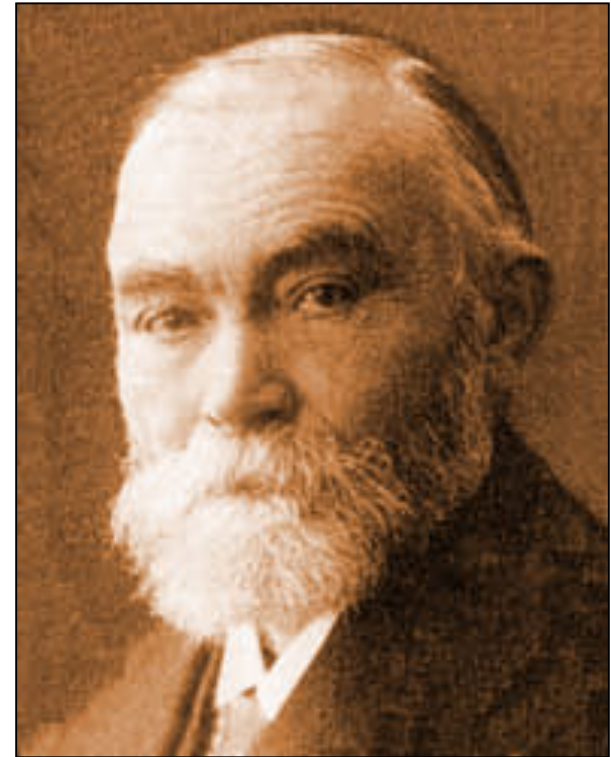
The Principles

- [Rule-to-rule hypothesis](#) (Frege):
- semantic interpretation guided
- by syntactic structure;

For each syntactic rule, there is a corresponding rule of semantic interpretation

- [Compositionality](#)

We assume that the meaning of a complex expression is determined by the meaning of its parts

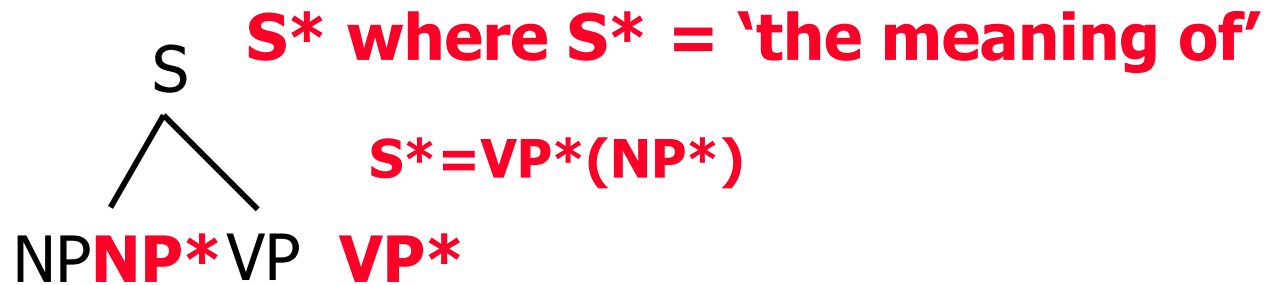


Context-free semantics: associate interpretation rule with each context-free rule

<u>Item or rule</u>	<u>Semantic translation</u>
$S \rightarrow NP VP$	S^*: apply $VP^*(NP^*)$
$VP \rightarrow \textit{sleeps}$	VP^*: $\lambda x \text{ SLEEPS}(x)$
$NP \rightarrow \text{Name}$	NP^*: $\lambda x x$
$\text{Name} \rightarrow \textit{rocky}$	Name^*: ‘rocky’ (ie, a constant)

The master principles

- Compositionality
- In a structure like this:



- The meaning of the S is computed as the function application of the meaning of the VP to the meaning of the NP:
- Intuitively: the concept expressed by the VP is asserted of the object to which the NP refers

6.001 to the rescue

- The function f can be given by the λ -expression

$\lambda x \text{ SLEEPS}(x)$

- When this lambda abstraction - a function - is applied to the argument 'Poirot', as usual this binds the variable x :

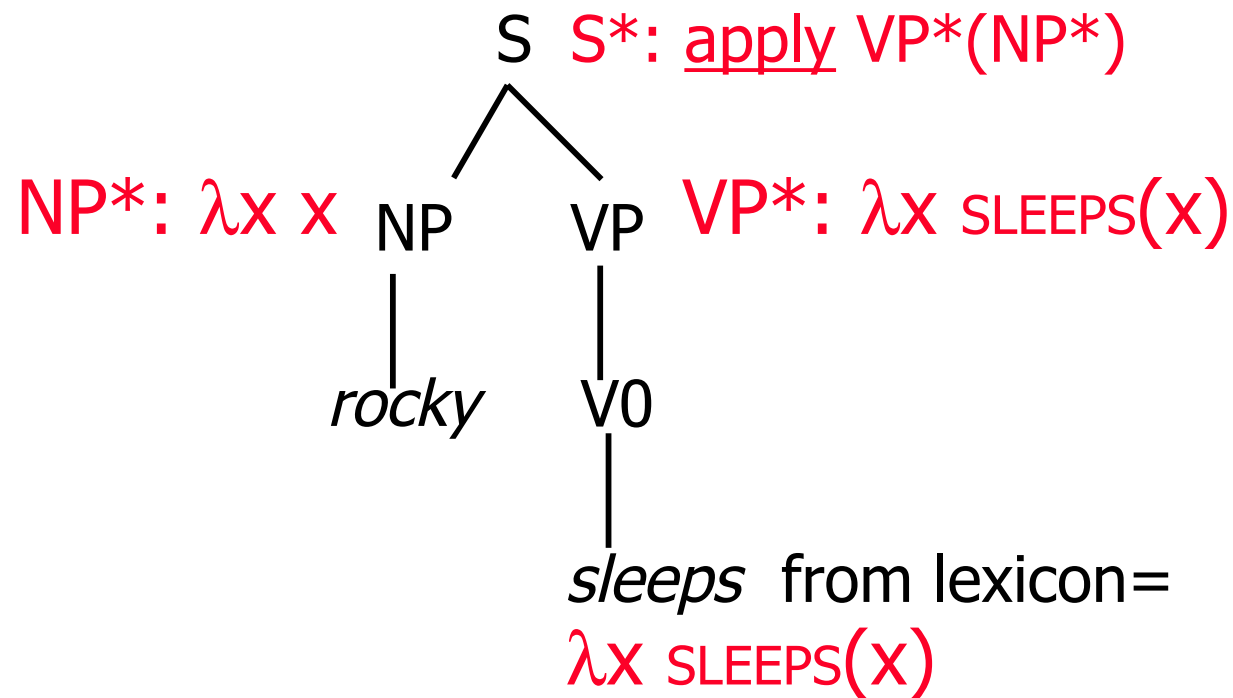
$\lambda x \text{ SLEEPS}(x).\text{rocky} \rightarrow \text{SLEEPS}(\text{rocky})$

- The lambda variable acts as a placeholder for missing info
- The “**rockyt**” denotes the substitution
- Alternatively, to make this more visible, we write:

$\lambda x \text{ SLEEPS}(x)@rocky \rightarrow \text{SLEEPS}(\text{rocky})$

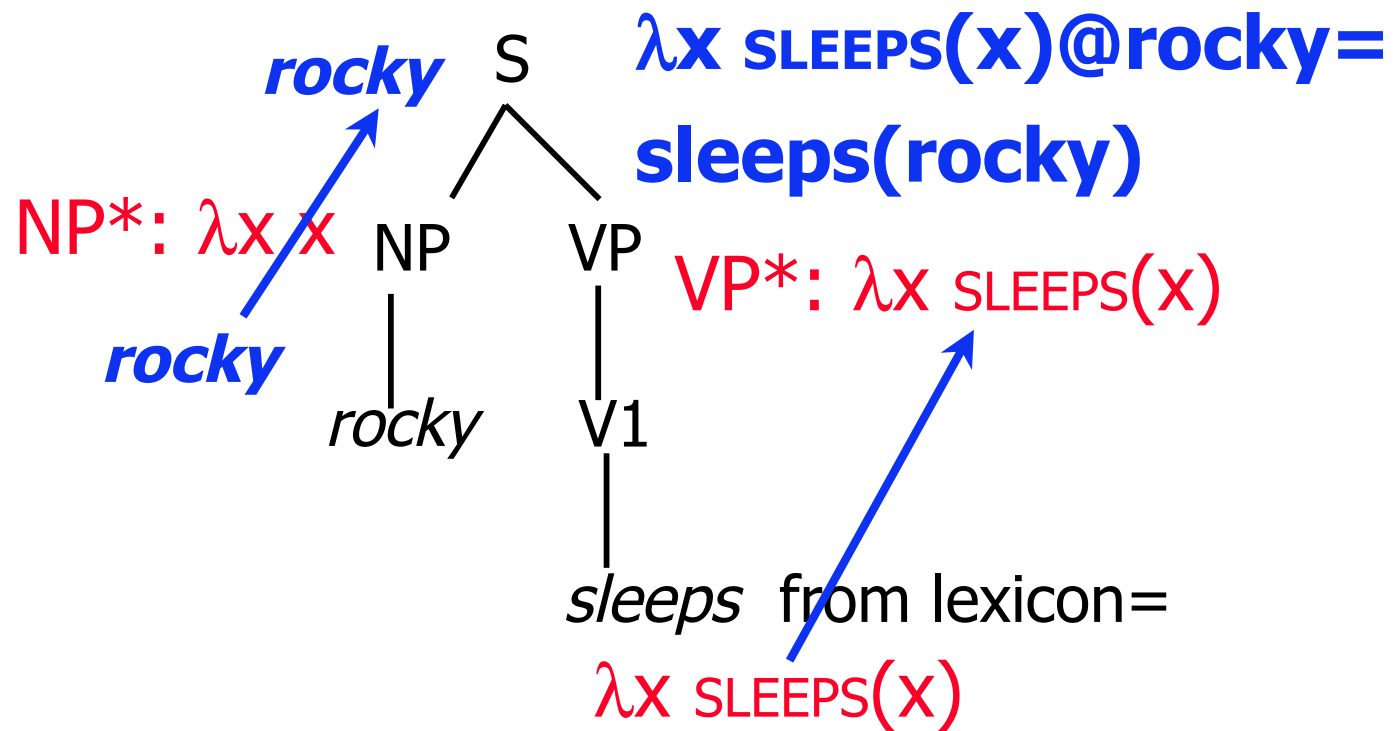
- Where the “**@**” denotes function application
- This process of substitution is also called beta reduction

Rule-to-rule principle: decorate syntax tree with



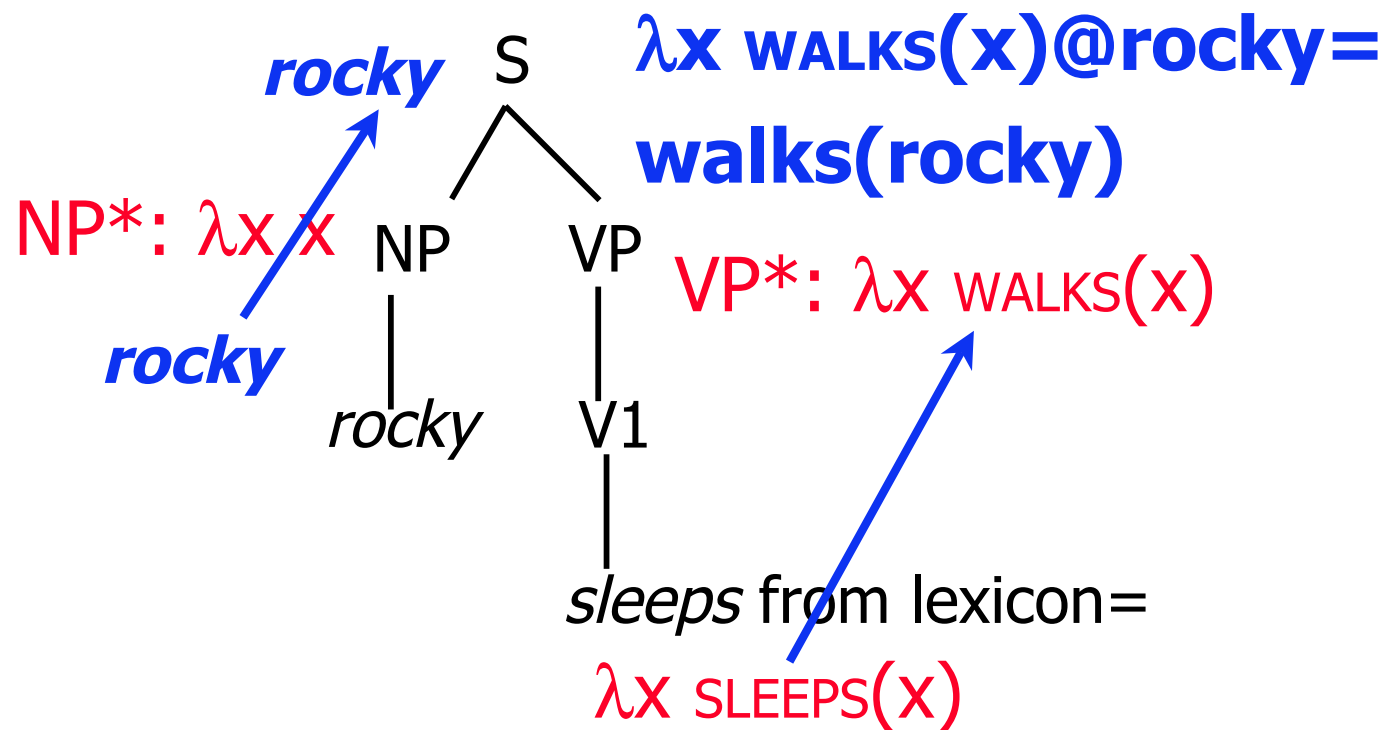
Doing the beta reductions...

S^* : apply $VP^*(NP^*) \rightarrow$



What about 'rocky walks'?

S^* : apply $VP^*(NP^*) \rightarrow$



Q1: in general, how can we abstract over all such intransitive verbs?? (answer in a bit)

The recipe

The lexical items (i.e. the words) in a sentence give us the basic ingredients for our representation

Syntactic structure tells us how the semantic contributions of the parts of a sentence are to be joined together

The output will be some sort of first-order predicate calculus, or its ‘equivalent’ in SQL terms: e.g.,
“Rocky likes Bullwinkle” → `LIKES (Rocky, Bullwinkle)`

DB – the ‘truth’ as SQL

Rocky likes Bullwinkle →

Insert into Like (liker, like) values (r, b)

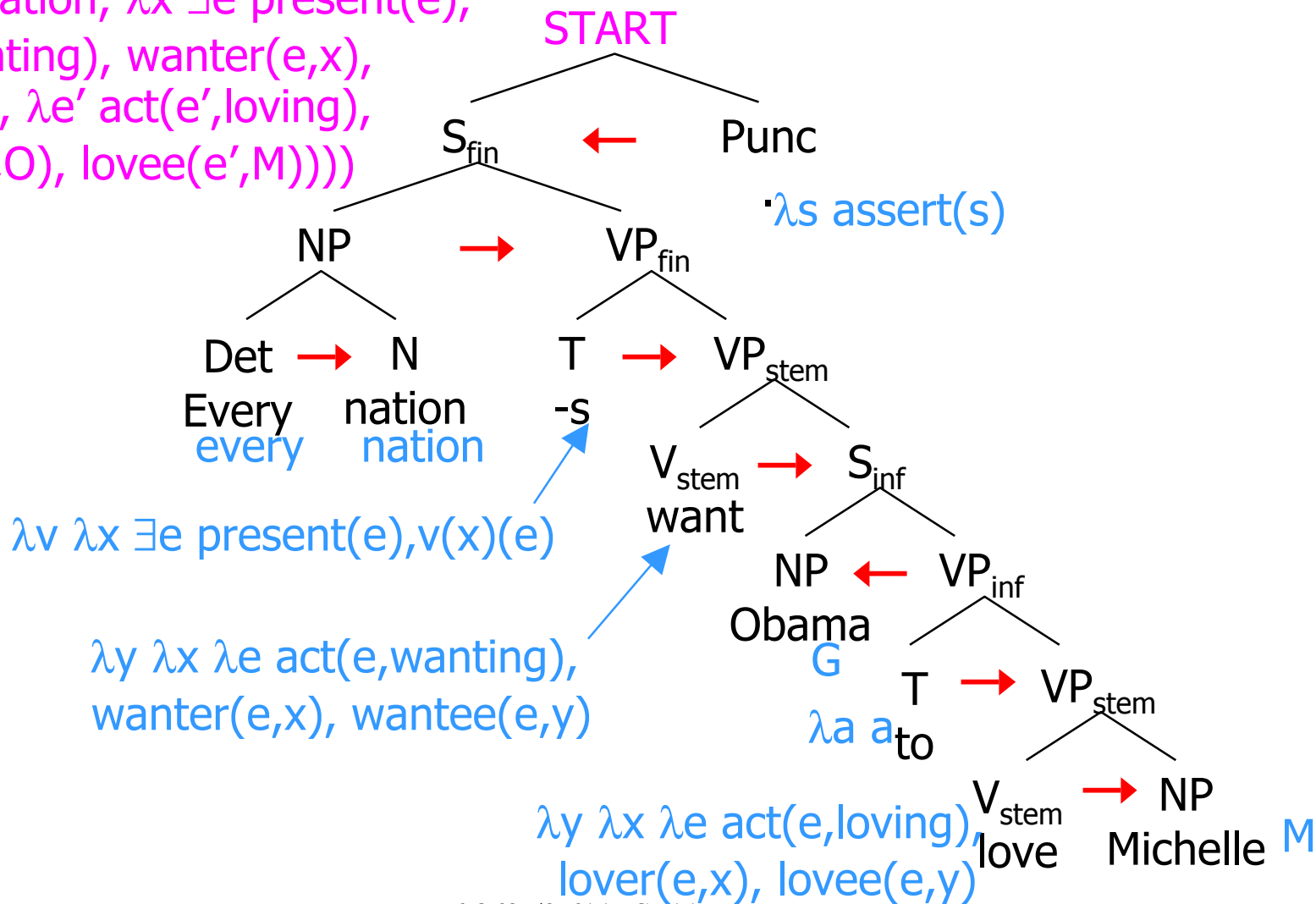
- Then we can answer a question, e.g.,

Does Rocky like Bullwinkle → **via the SQL:**

**select 'yes' from Like where Like.liker = r
and Like.likee = b**

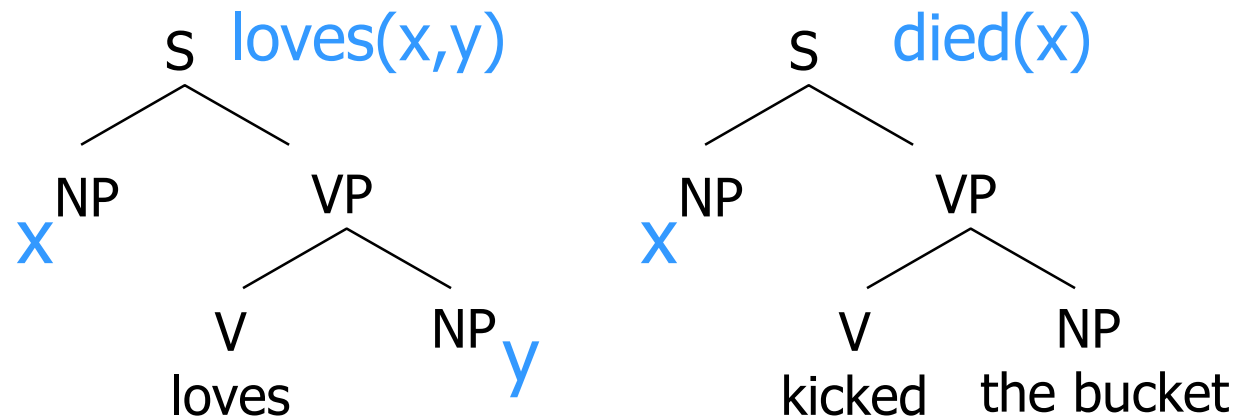
Compositional Semantics

assert(every(nation, $\lambda x \exists e$ present(e),
 act(e,wanting), wanter(e,x),
 wantee(e, $\lambda e'$ act(e',loving),
 lover(e',O), lovee(e',M))))



Compositional Semantics

- Add a “sem” feature to each context-free rule
 - $S \rightarrow NP \text{ loves } NP$
 - $S[\text{sem}=\text{loves}(x,y)] \rightarrow NP[\text{sem}=x] \text{ loves } NP[\text{sem}=y]$
 - Meaning of S depends on meaning of NPs
- TAG version:



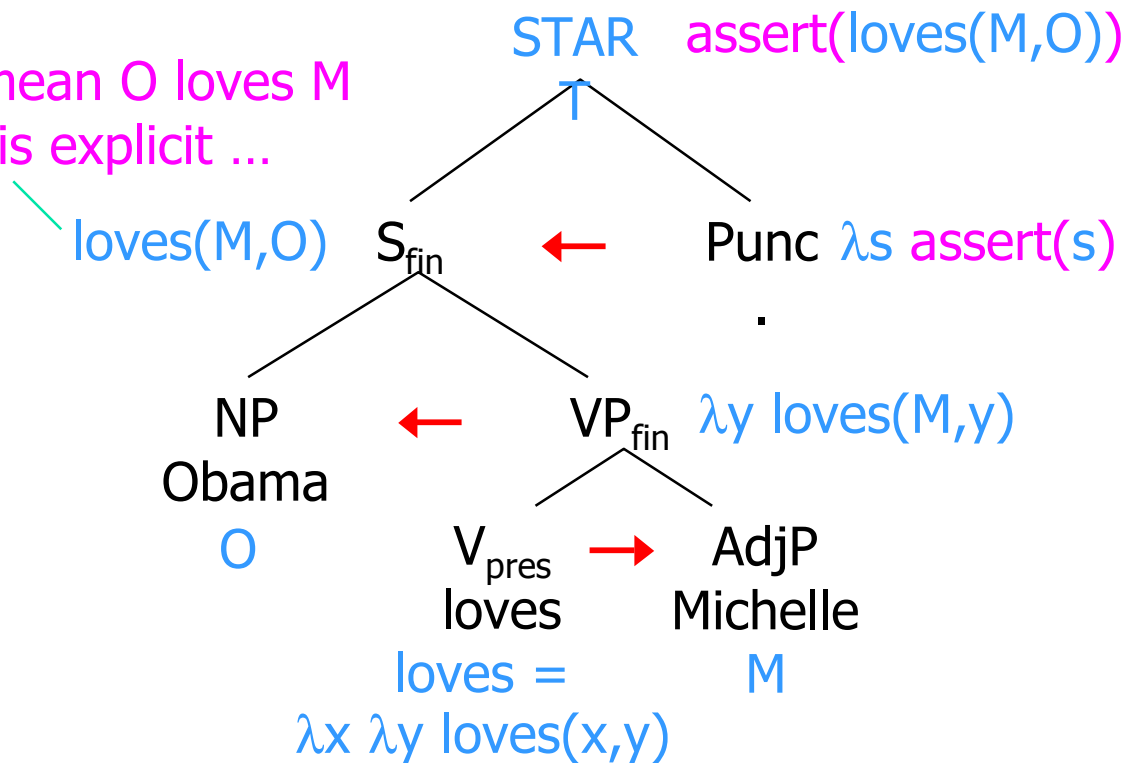
- Template filling: $S[\text{sem}=\text{showflights}(x,y)] \rightarrow$
I want a flight from NP[sem= x] to NP[sem= y]

Compositional Semantics

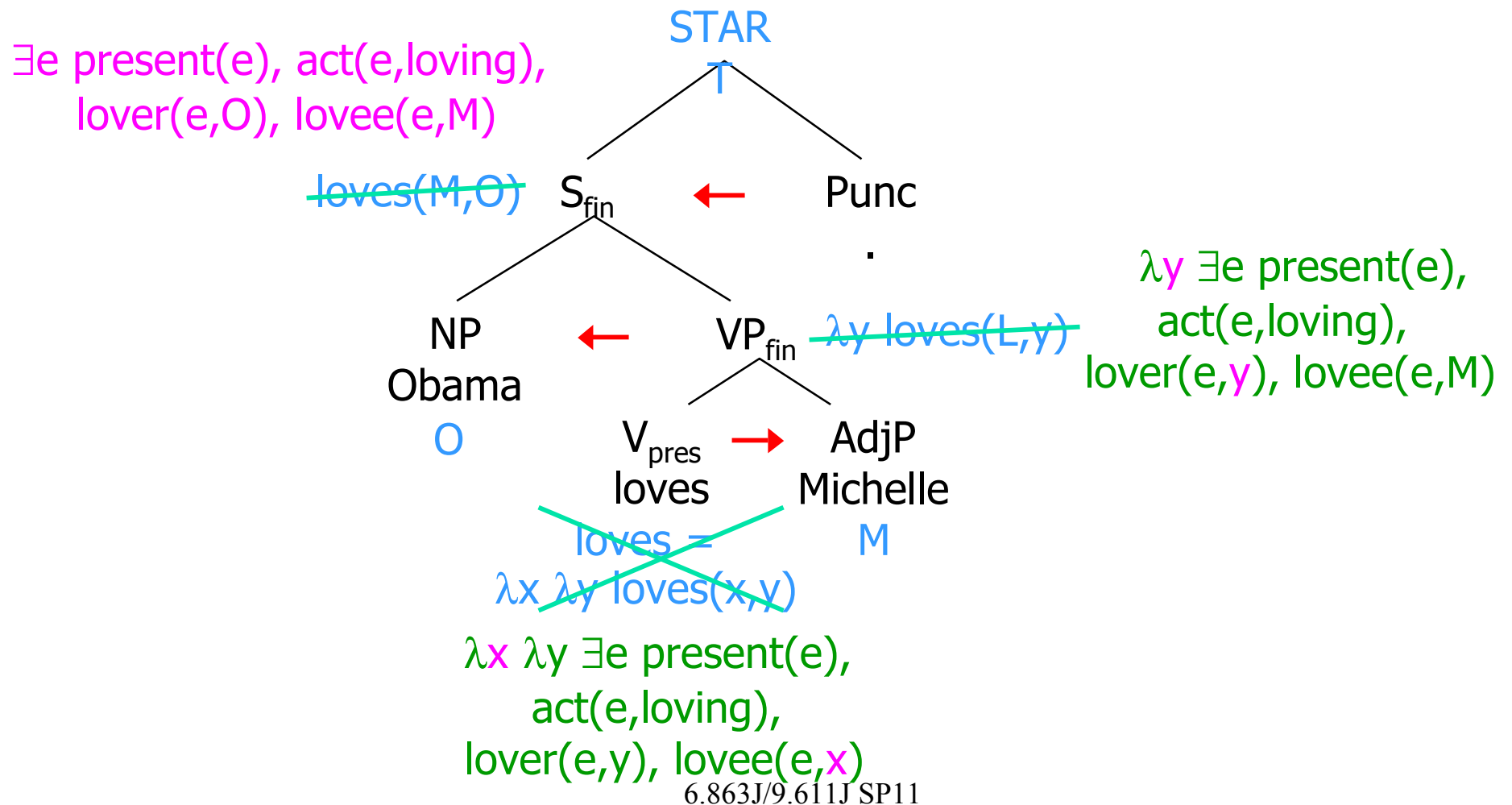
- Instead of $S \rightarrow NP \text{ loves } NP$
 - $S[\text{sem}=\text{loves}(x,y)] \rightarrow NP[\text{sem}=x] \text{ loves } NP[\text{sem}=y]$
- might want general rules like $S \rightarrow NP VP$:
 - $V[\text{sem}=\text{loves}] \rightarrow \text{loves}$
 - $VP[\text{sem}=\text{v}(\text{obj})] \rightarrow V[\text{sem}=\text{v}] NP[\text{sem}=\text{obj}]$
 - $S[\text{sem}=\text{vp}(\text{subj})] \rightarrow NP[\text{sem}=\text{subj}] VP[\text{sem}=\text{vp}]$
- Now *Obama loves Michelle* has
 $\text{sem}=\text{loves}(\text{Michelle})(\text{Obama})$
 - **To get its semantics, apply function to argument!**

Compositional Semantics

Intended to mean O loves M
Let's make this explicit ...



Compositional Semantics



Names for things: one more trick needed

- So far we have names (constants) like ‘rocky’
- Instead of “rocky” referring to an **Individual**, it will refer to the set of this individual’s properties (viz., the predicates it satisfies, e.g., sleeps, walks, squirrel,...)
- Each property is a set, so this ‘proper name’ denotes a set of sets
- We will convert this to a function, in the following way:
- A proper name like Rocky will have the form:

$(\lambda P P)@name$, where $(\lambda P P)$ denotes some predicate to be supplied later

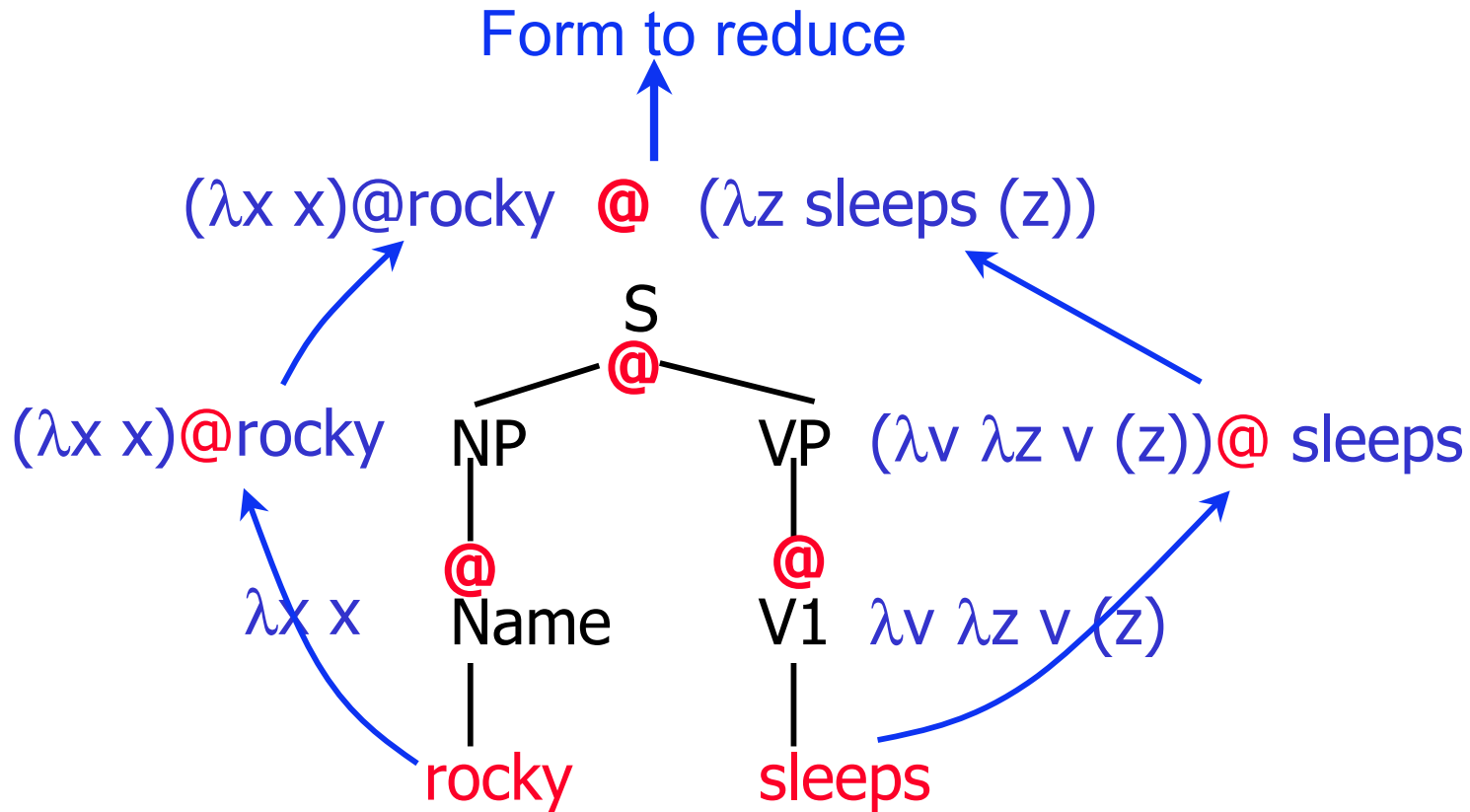
$\lambda P P@rocky =$ a *set* of sets

- This is called type raising because now, instead of ‘rocky’ being of a type “**Individual**” it is of a higher type (a function, that maps predicates P to truth values)

Names & intransitive verbs

- This trick lets us ‘flip’ functions and arguments:
instead of $(f a)$ we have: $(\lambda P (P a) f)$
- We need this because sometimes the syntax of the natural language sentence doesn’t have the structure (function argument) in the output semantic form, or vice-versa
- For intransitive verbs, in general, we can abstract out the predicate i.e. for an intransitive verb, v :
 $\lambda v \lambda z v (z)$
- Now, every RHS rule with a single ‘preterminal’ adds a function application $@$ plus the preterminal, while every RHS with two nonterminals simply ‘glues’ the two pieces from below together, this way:

We can how assemble full lambda form mechanically



Take this form and do beta-reductions
(function applications) mechanically...

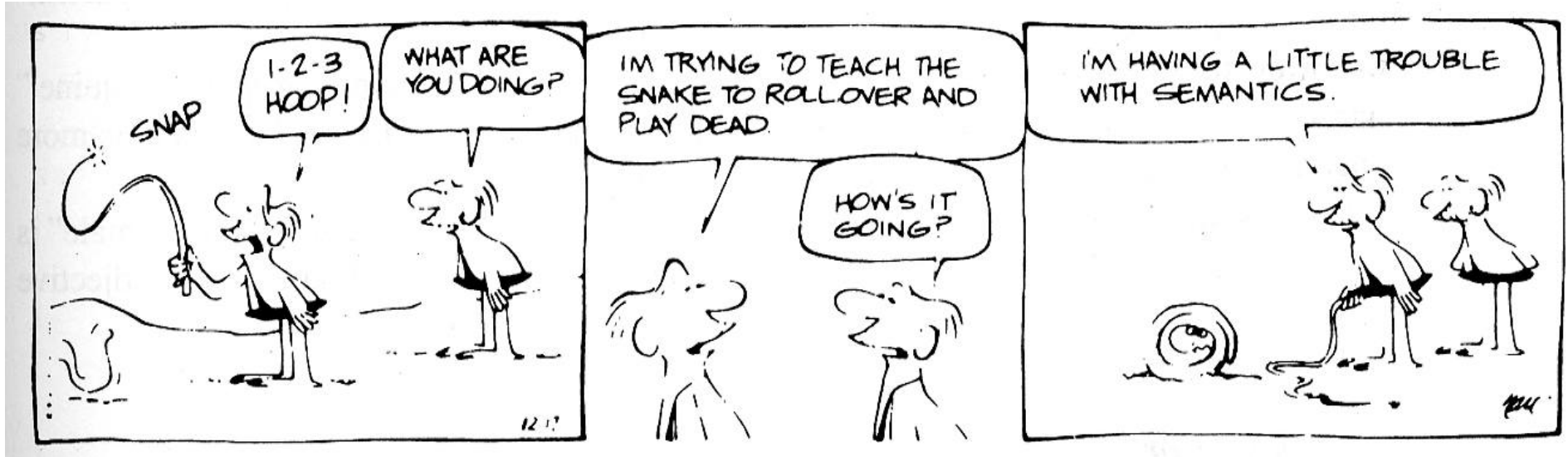
$(\lambda x x)@rocky@ (\lambda z \text{sleeps}(z))$



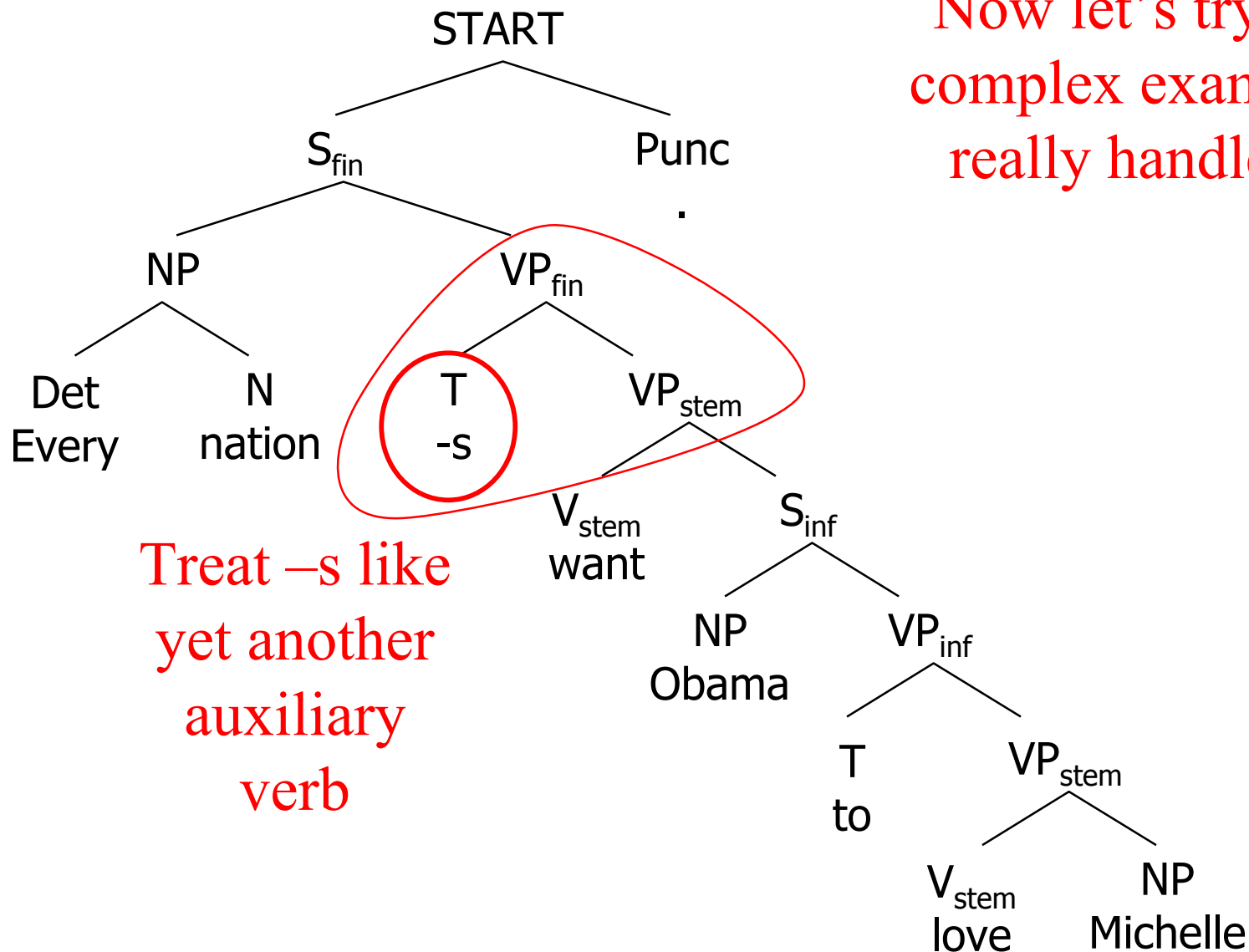
$(\lambda z \text{sleeps}(z))@rocky$

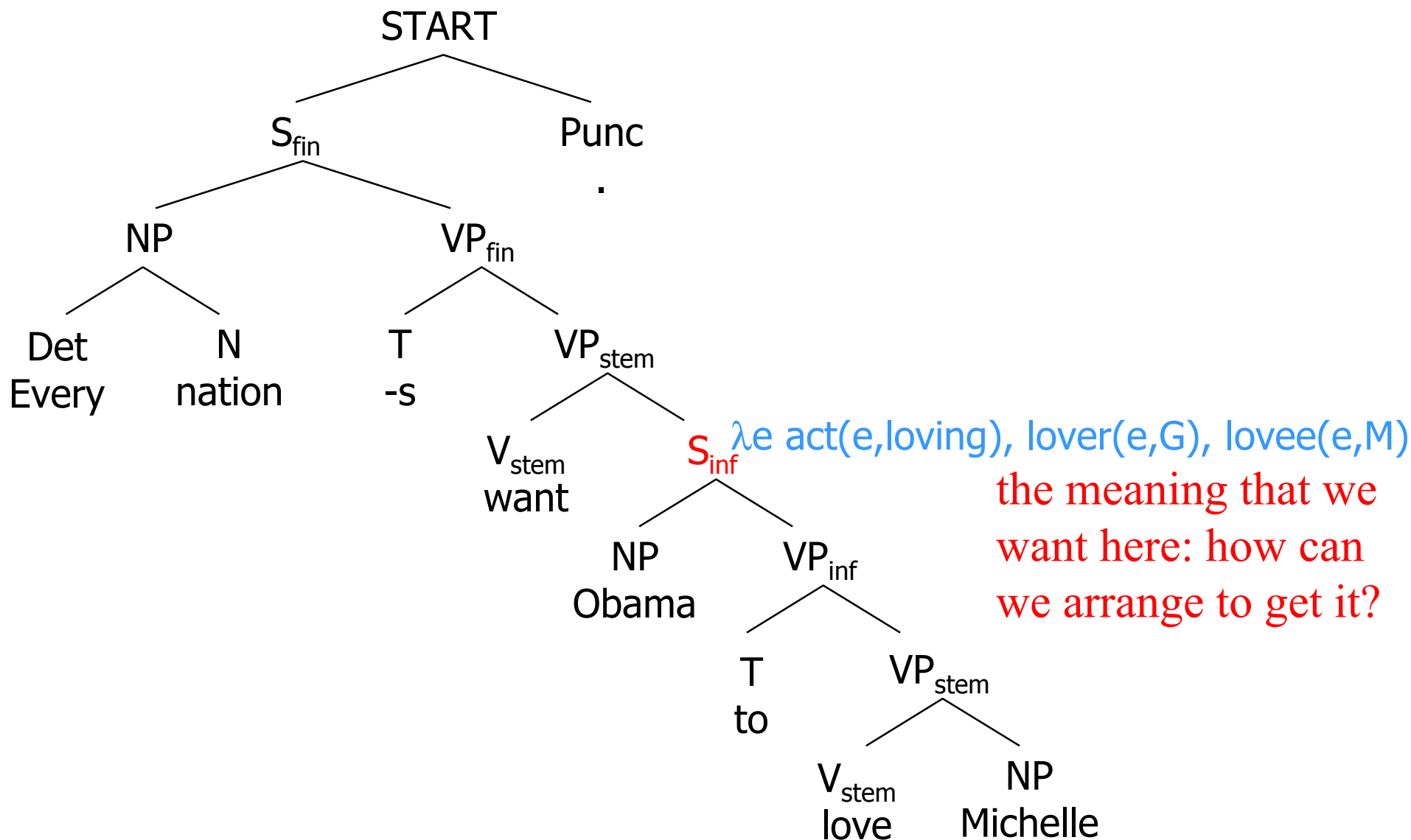


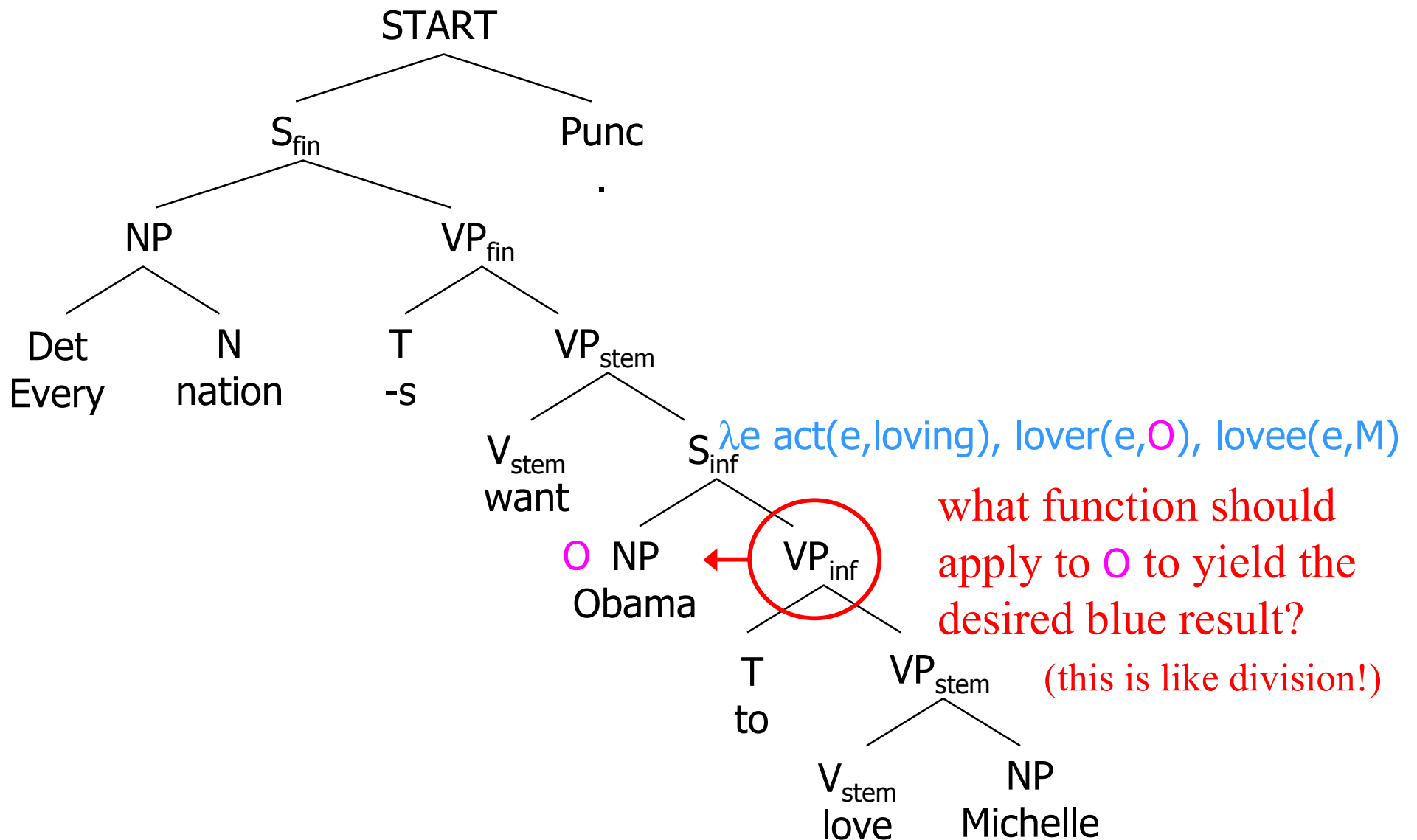
$\text{sleeps}(rocky)$

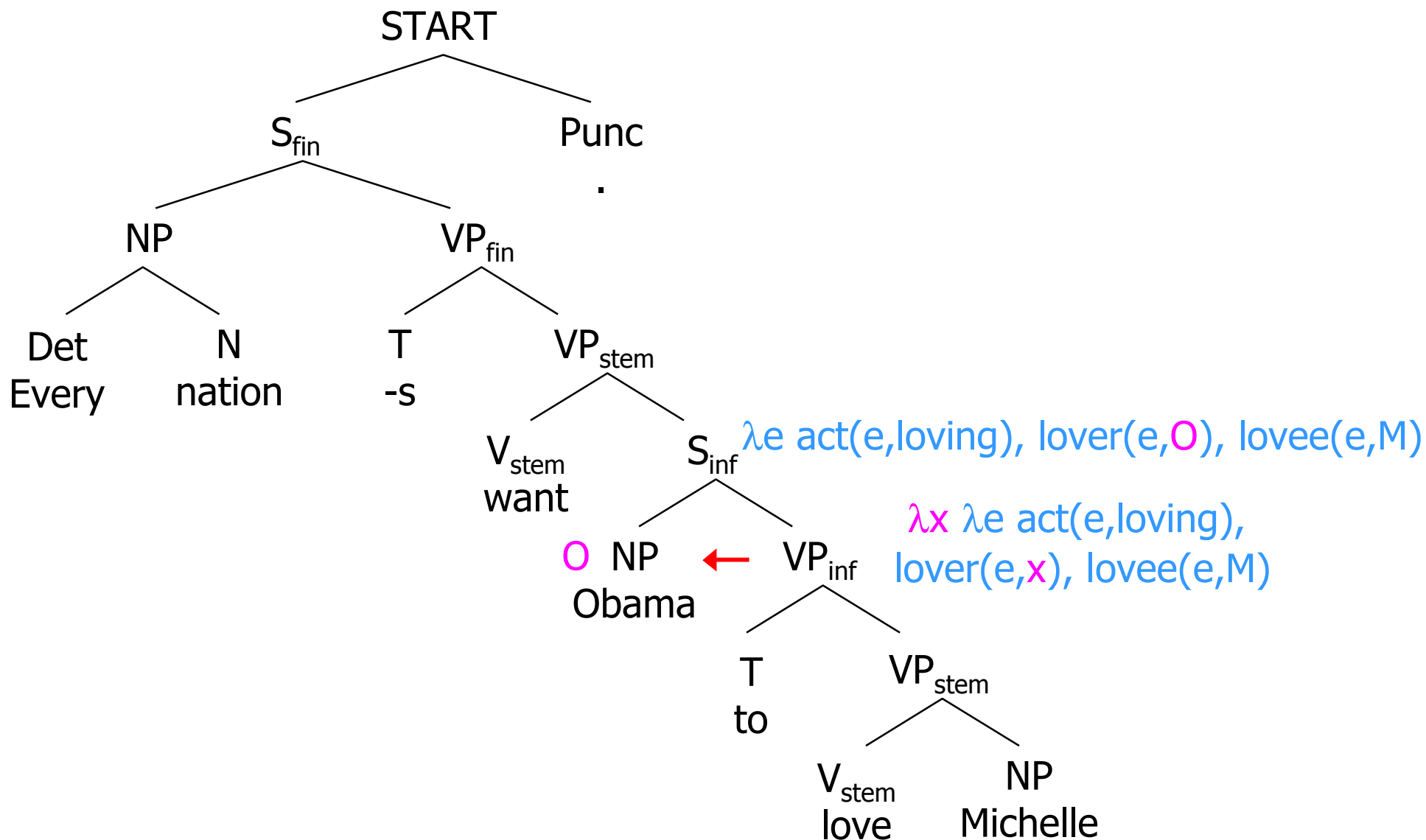


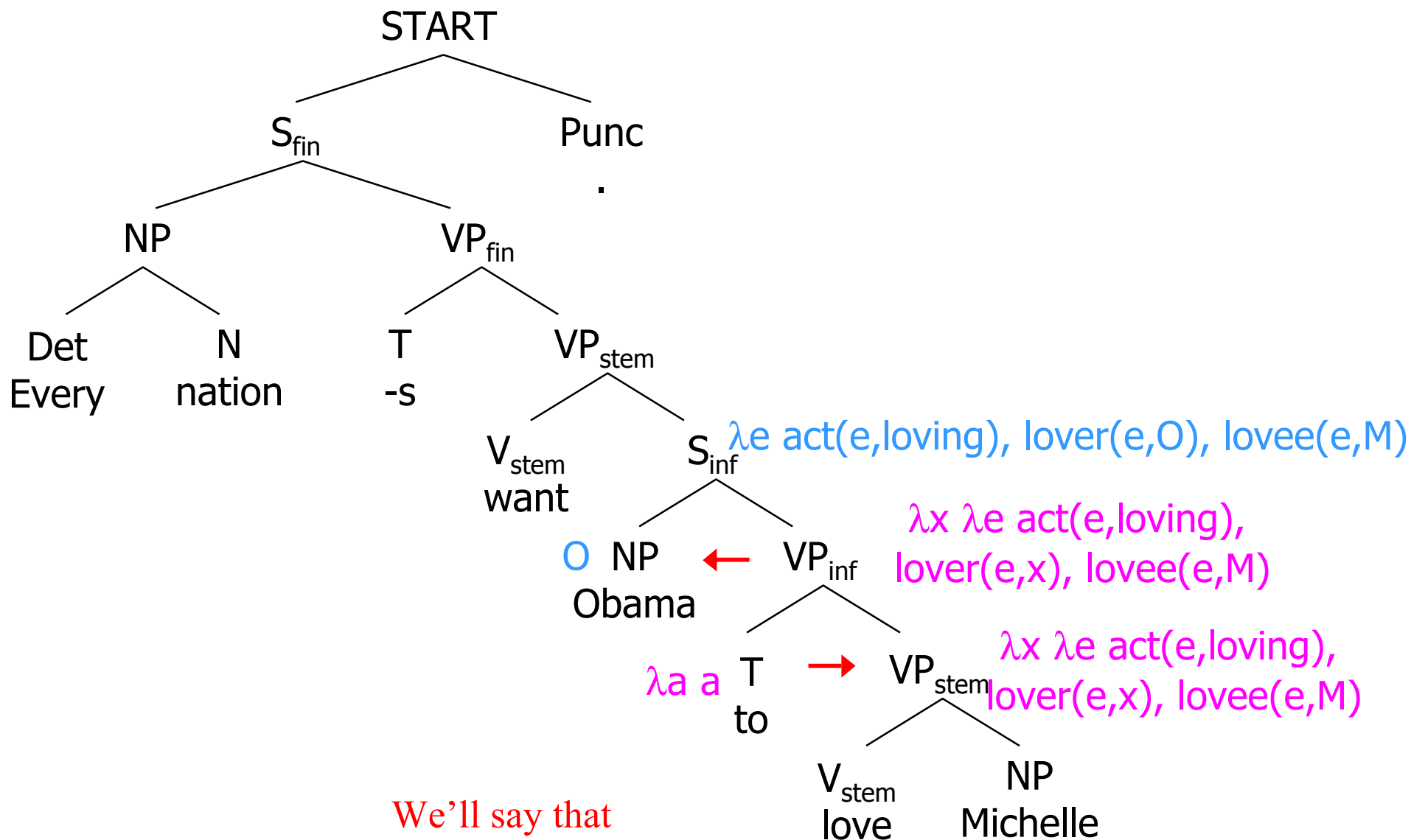
Now let's try a more complex example, and really handle tense.



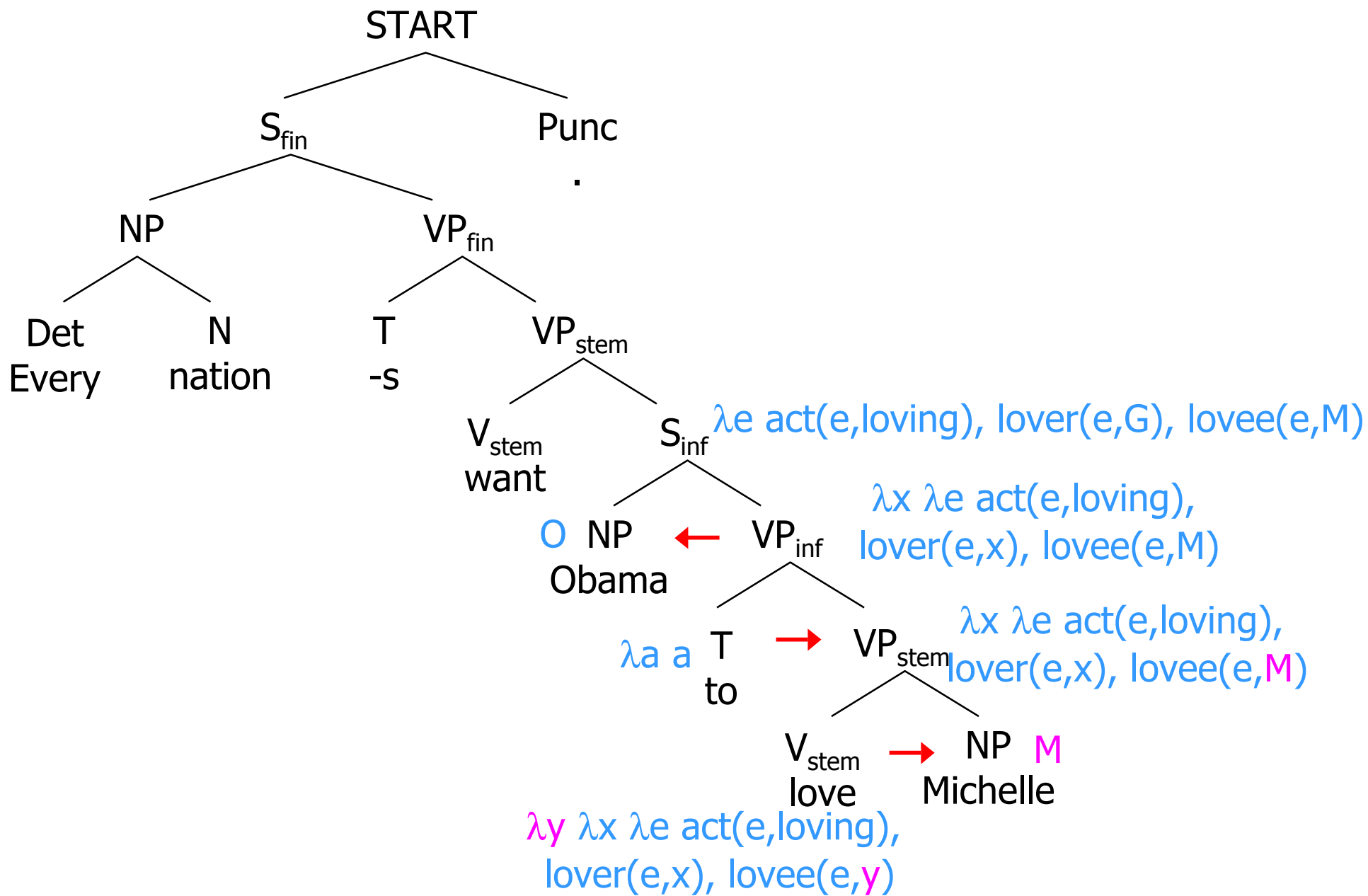


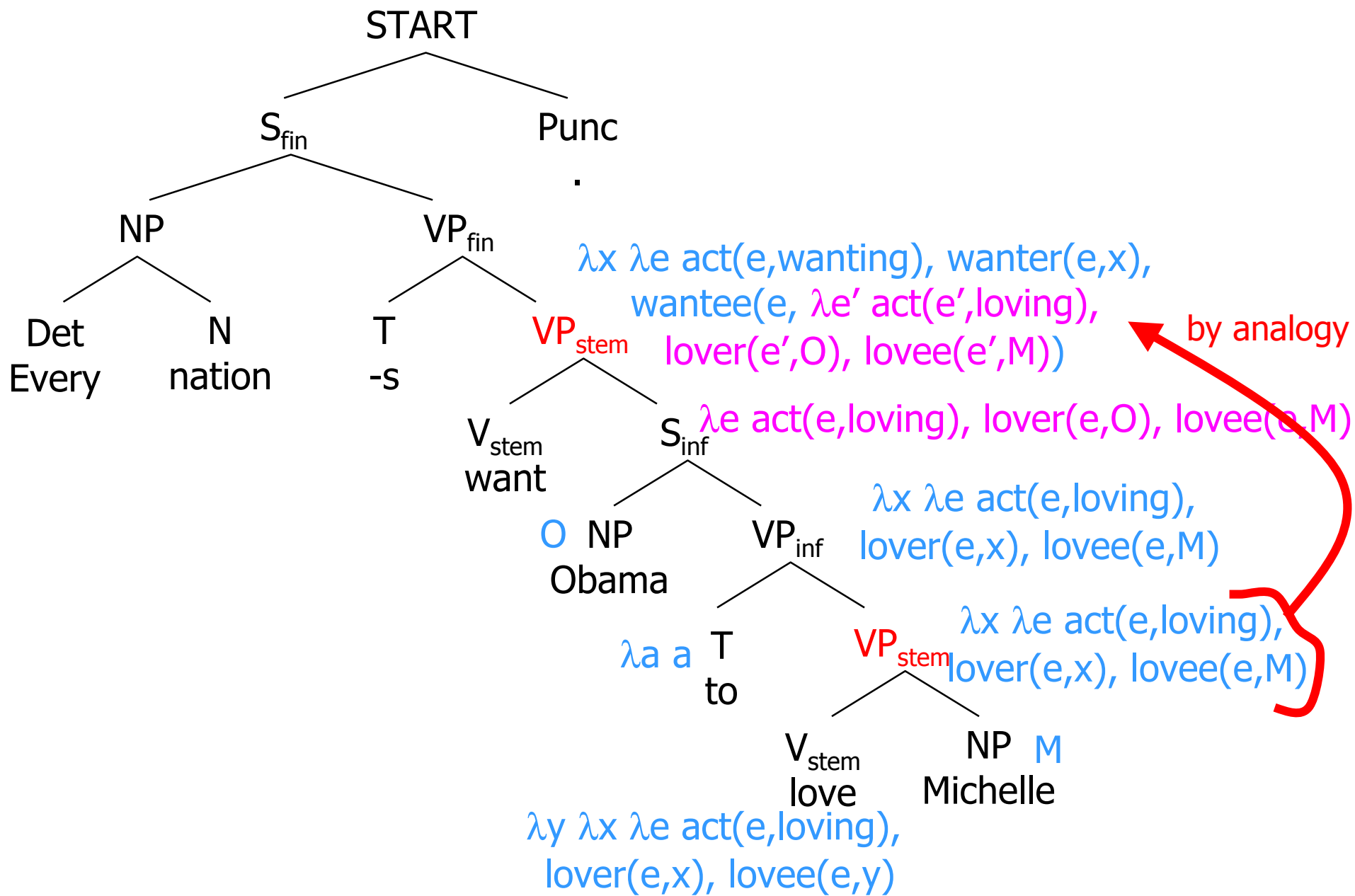


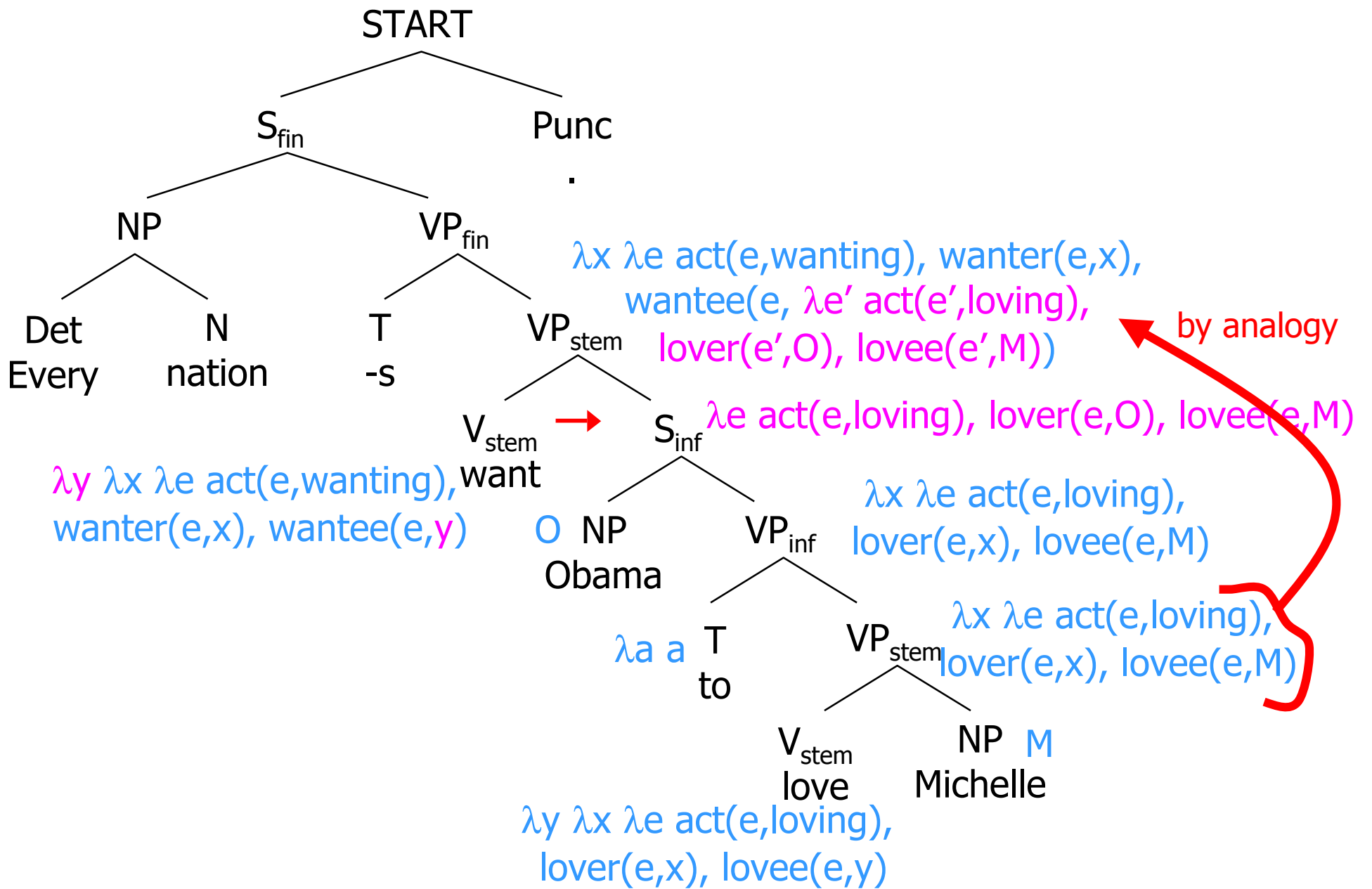


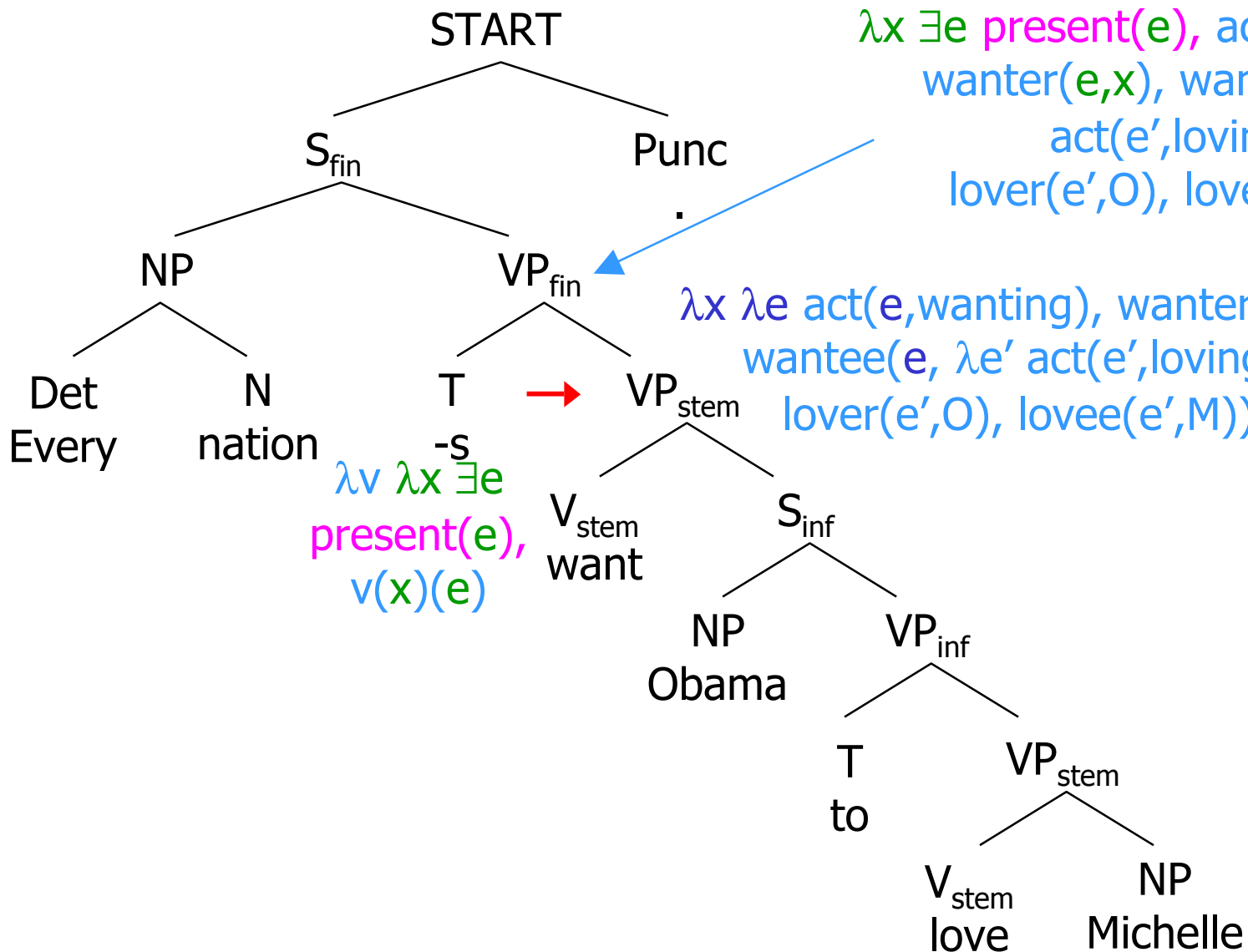


We'll say that "to" is just a bit of syntax that changes a VP_{stem} to a VP_{inf} with the same meaning.







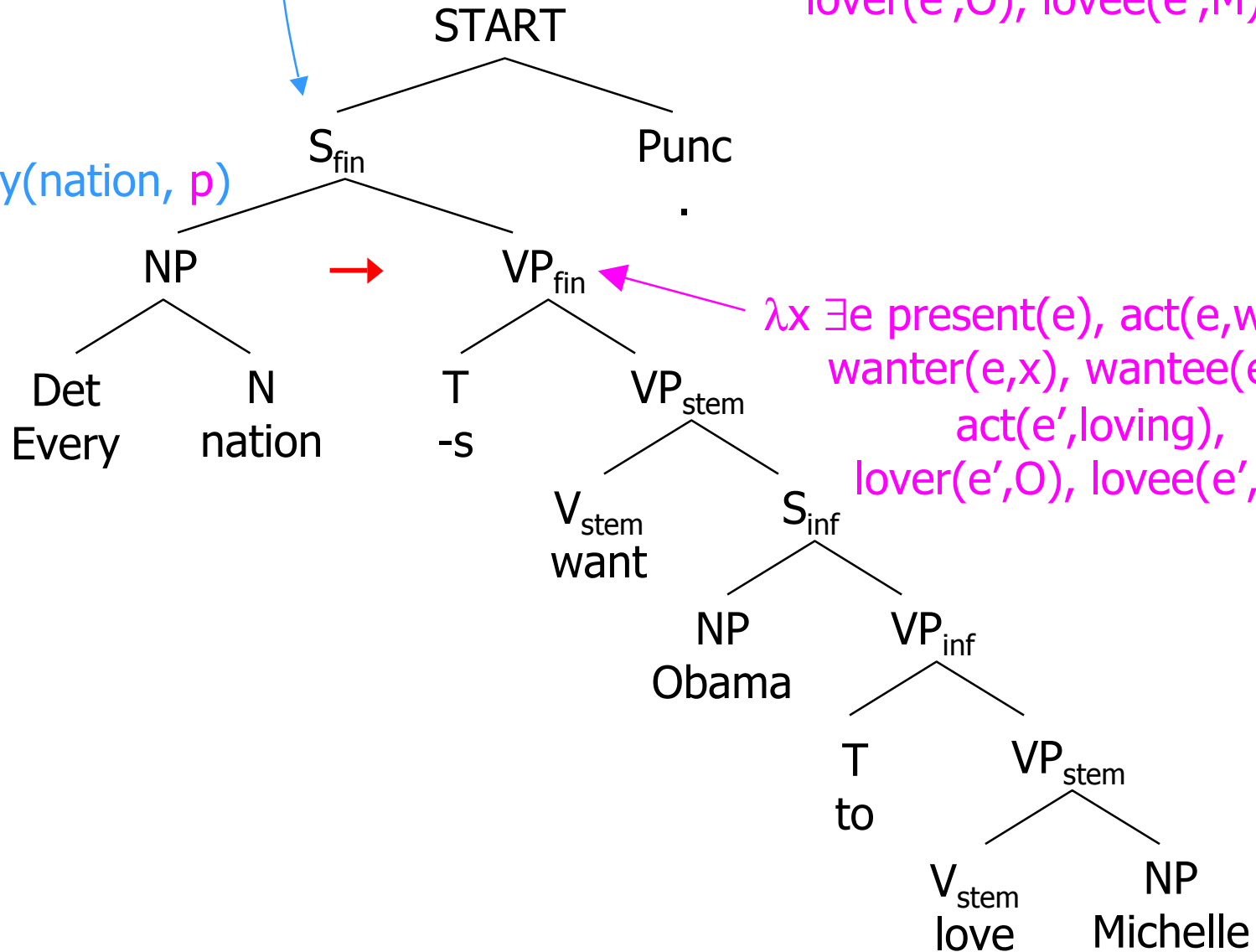


$\lambda x \exists e$ present(e), act(e,wanting),
 want(e,x), wantee(e, $\lambda e'$
 act(e',loving),
 lover(e',O), lovee(e',M))

$\lambda x \lambda e$ act(e,wanting), want(e,x),
 wantee(e, $\lambda e'$ act(e',loving),
 lover(e',O), lovee(e',M))

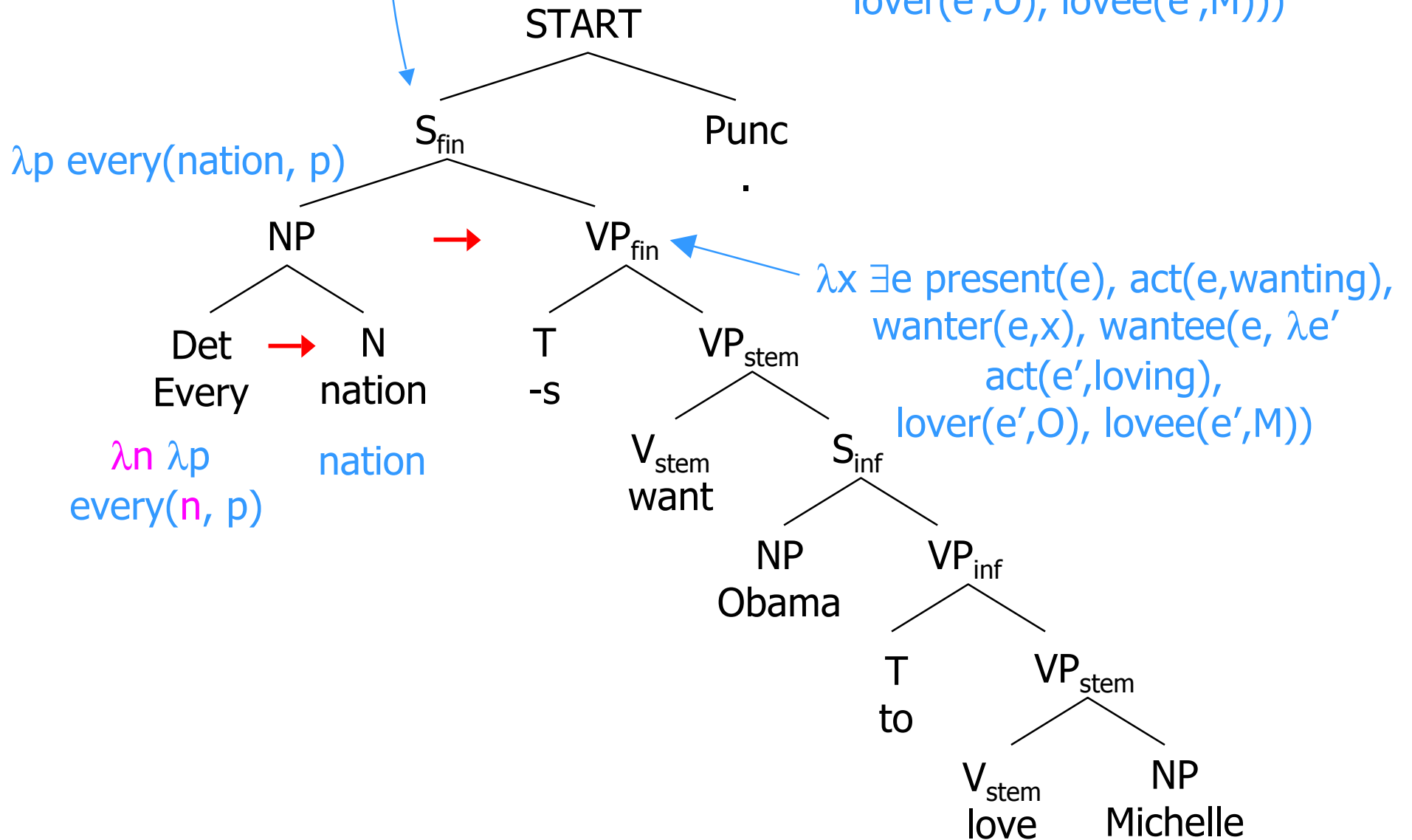
every(nation, $\lambda x \exists e$ present(e),
 act(e,wanting), wanter(e,x),
 wantee(e, $\lambda e'$ act(e',loving),
 lover(e',O), lovee(e',M)))

λp every(nation, p)

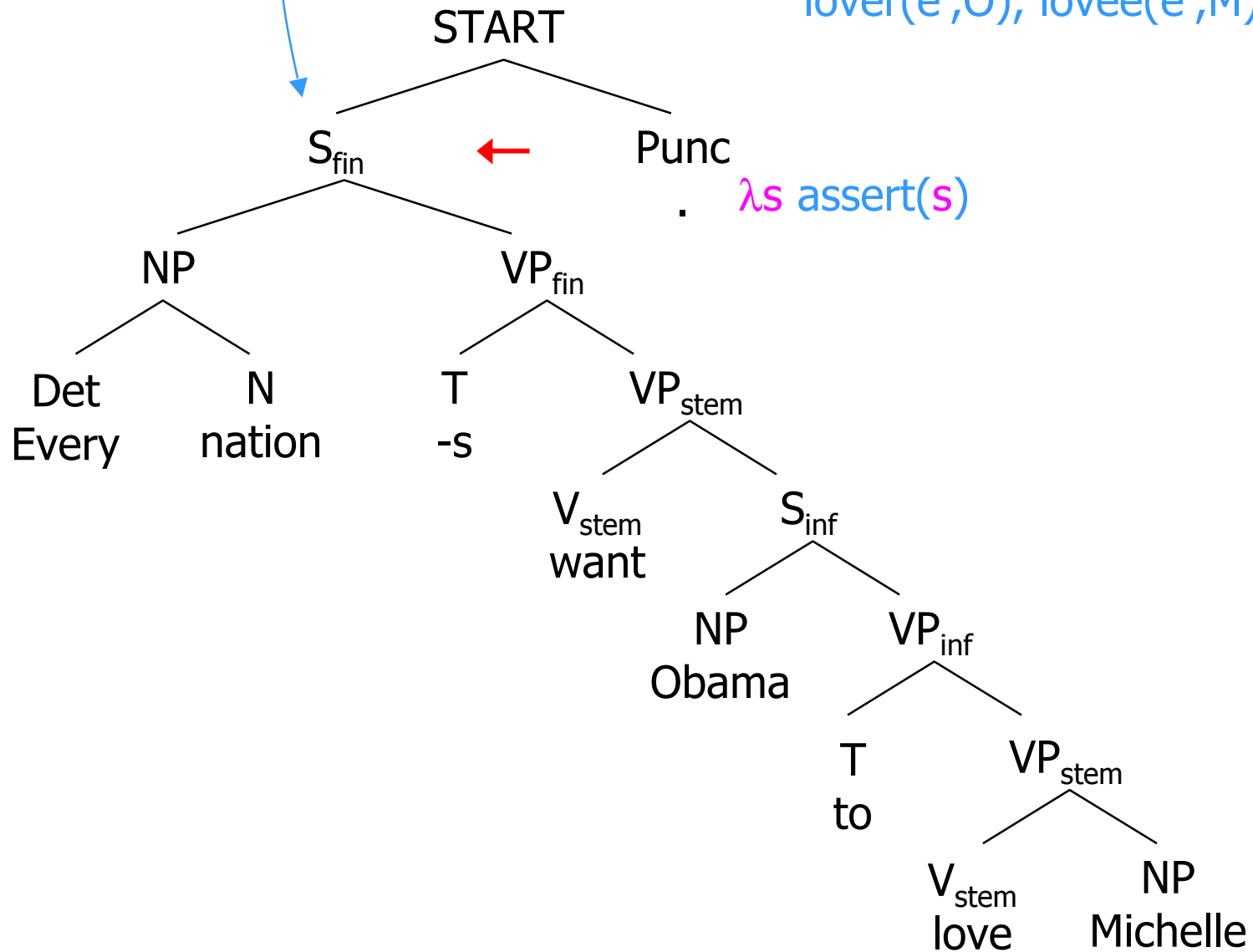


$\lambda x \exists e$ present(e), act(e,wanting),
 wanter(e,x), wantee(e, $\lambda e'$
 act(e',loving),
 lover(e',O), lovee(e',M))

every(nation, $\lambda x \exists e$ present(e),
 act(e,wanting), wanter(e,x),
 wantee(e, $\lambda e'$ act(e',loving),
 lover(e',O), lovee(e',M)))

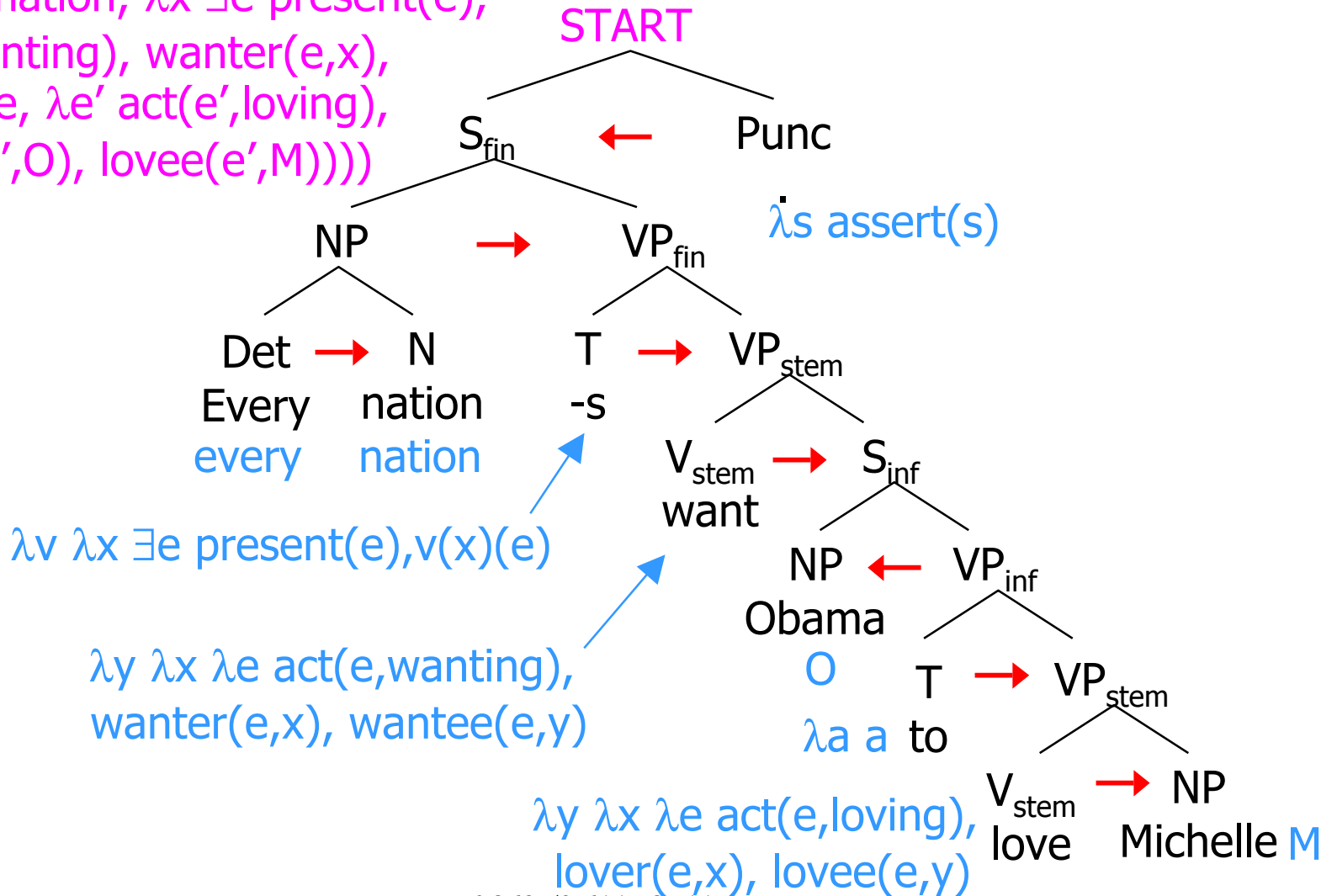


every(nation, $\lambda x \exists e$ present(e),
 act(e,wanting), wanter(e,x),
 wantee(e, $\lambda e'$ act(e',loving),
 lover(e',O), lovee(e',M)))



In Summary: From the Words

assert(every(nation, $\lambda x \exists e$ present(e),
 act(e,wanting), wanter(e,x),
 wantee(e, $\lambda e'$ act(e',loving),
 lover(e',O), lovee(e',M))))



In Summary II: paired syntactic-semantic rules

Lexicon

Kathy, NP : **kathy**

Fong, NP : **fong**

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

runs, V : $\lambda x.\mathbf{run}(x)$

Grammar

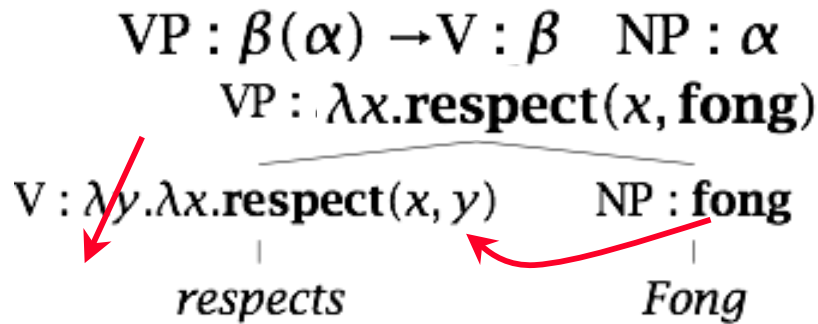
S : $\beta(\alpha) \rightarrow$ NP : α VP : β

VP : $\beta(\alpha) \rightarrow$ V : β NP : α

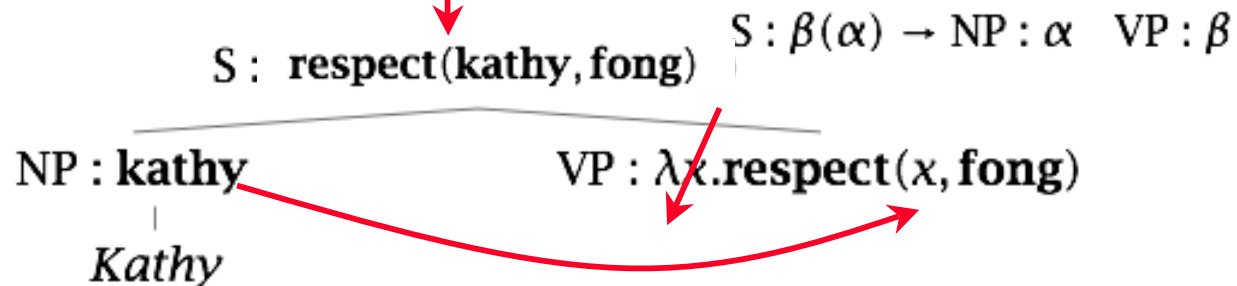
VP : $\beta \rightarrow$ V : β

- Nonterminal : semantic translation translation rules say how to associate semantic representations with syntactic representations
- In general, head-nonhead syntactic composition corresponds to function application

Sequence of function applications (beta reductions) tells us how to compose the semantic representation



$$[\text{VP respects Fong}] : [\lambda y. \lambda x. \text{respect}(x, y)](\text{fong}) = \lambda x. \text{respect}(x, \text{fong}) \quad [\beta \text{ red.}]$$



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

insert into **Respects**(respector, respected) values (*k*, *f*)

abbreviate some notation...

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

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

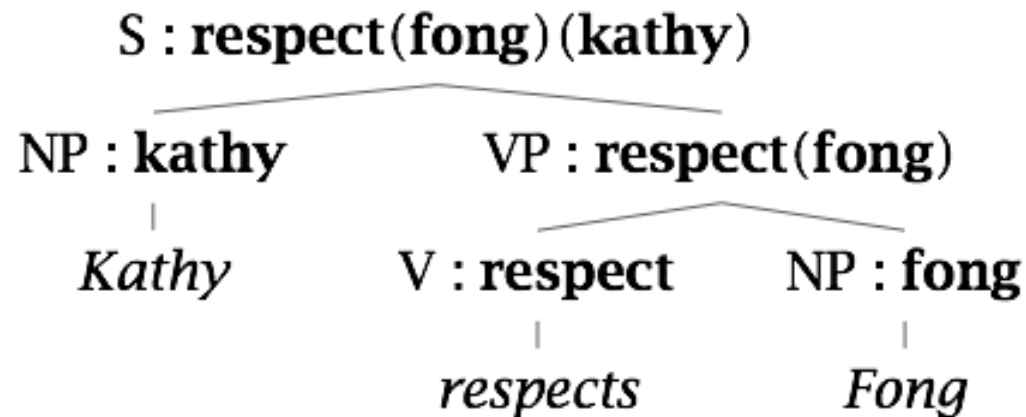

respect

respect is $\text{Ind} \rightarrow \text{Ind} \rightarrow \text{Bool}$

β, η long form: $\lambda x.\mathbf{run}(x), \lambda y.\mathbf{yesterday}(\lambda x.\mathbf{run}(x))(y)$

β, η normal form: **run**, **yesterday(run)**

So we can abbreviate the whole sentence semantics this way...



Which leads to an immediate SQL treatment:

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

Ok, what's next???

What about prepositions, adjectives, adverbs,...

New, improved grammar, with semantic augmentation

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

New lexicon

Kathy, NP : **kathy**_{Ind}

Fong, NP : **fong**_{Ind}

Palo Alto, NP : **paloalto**_{Ind}

car, N : **car**_{Ind → Bool}

overpriced, Adj : **overpriced**_{(Ind → Bool) → (Ind → Bool)}

outside, PP : **outside**_{(Ind → Bool) → (Ind → 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 : **some**²_{(Ind → Bool) → (Ind → Bool) → Bool}

runs, V : **run**_{Ind → Bool}

respects, V : **respect**_{Ind → Ind → Bool}

likes, V : **like**_{Ind → Ind → Bool}

sees, V : **see**_{Ind → Ind → Bool}

in' is **Ind → Ind → Bool**

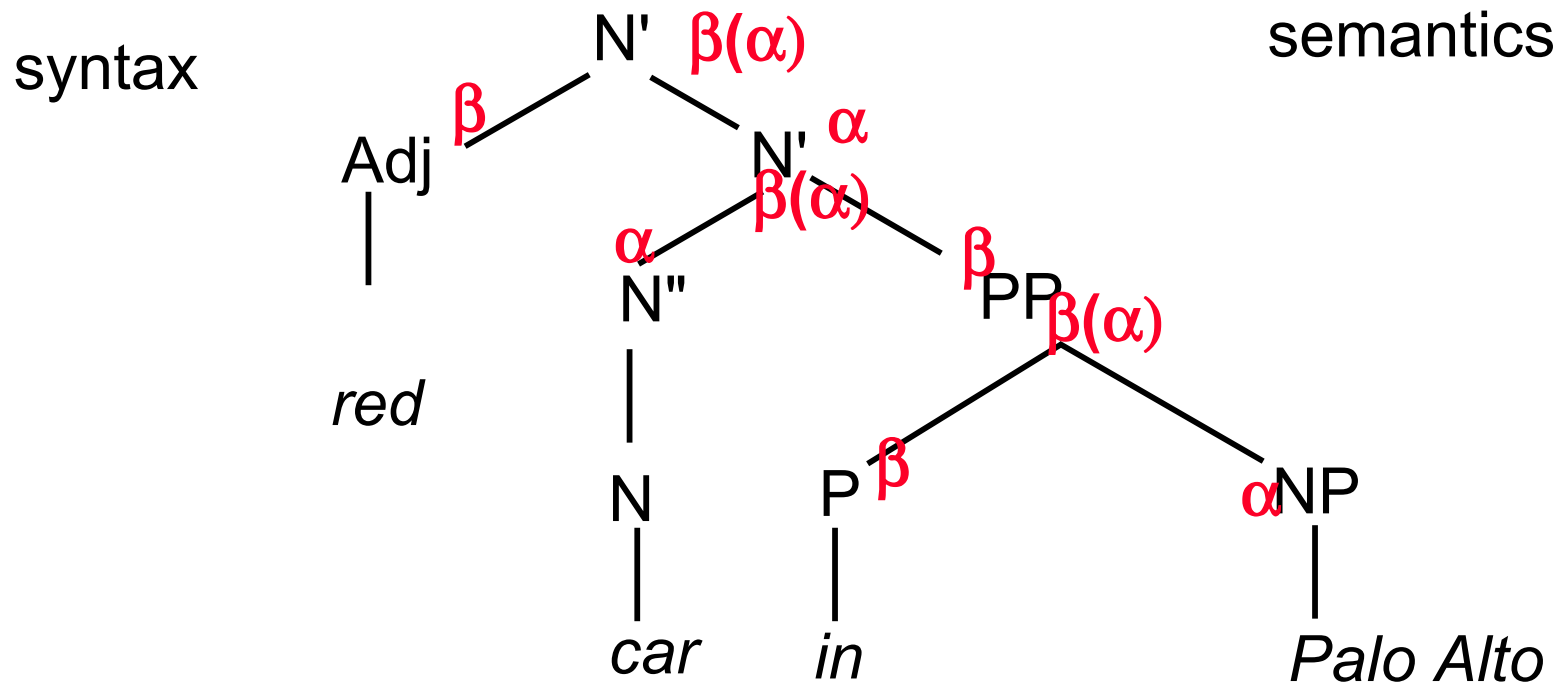
in $\cong \lambda y. \lambda P. \lambda x. (P(x) \wedge \mathbf{in}'(y)(x))$ is **Ind → (Ind → Bool) → (Ind → Bool)**

red' is **Ind → Bool**

red $\cong \lambda P. (\lambda x. (P(x) \wedge \mathbf{red}'(x)))$ is **(Ind → Bool) → (Ind → Bool)**

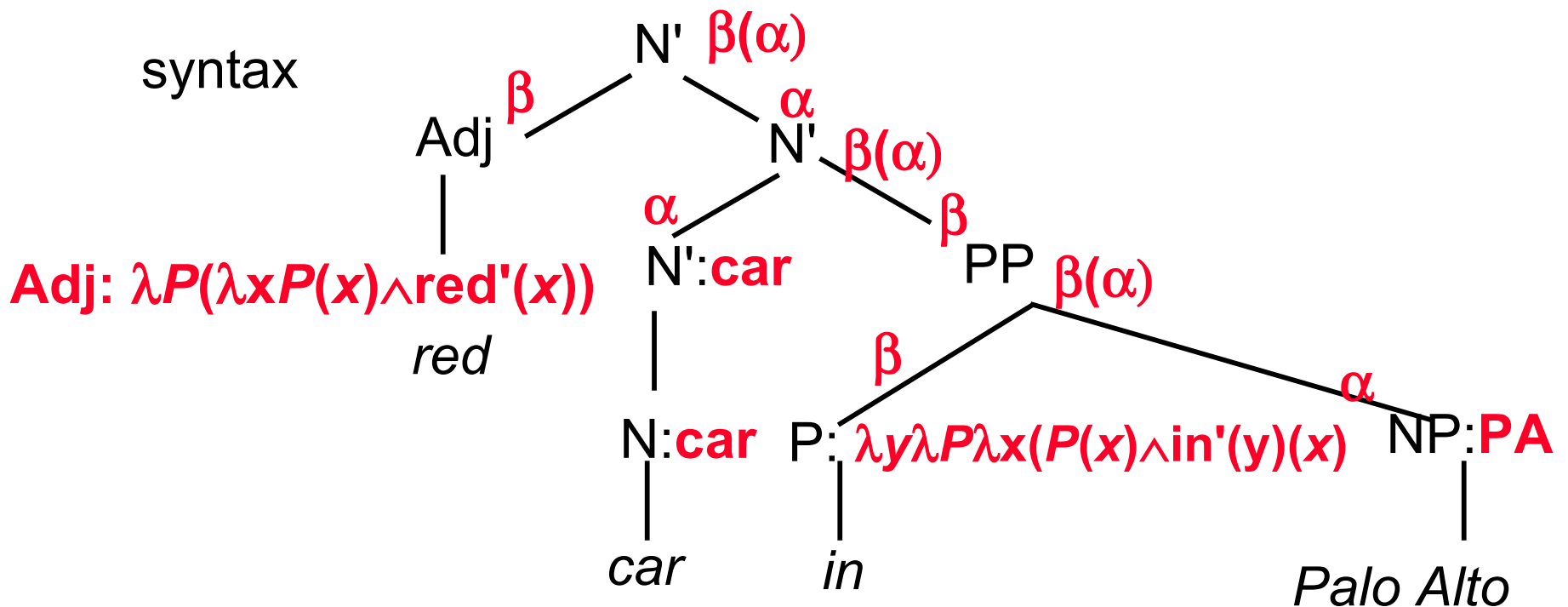
Let's try it out

Desired Goal $\lambda x. \text{car}(x) \wedge \text{in}'(\text{paloalto})(x) \wedge \text{red}'(x)$
red car in Palo Alto



Now add lexical entries and start doing
function application...

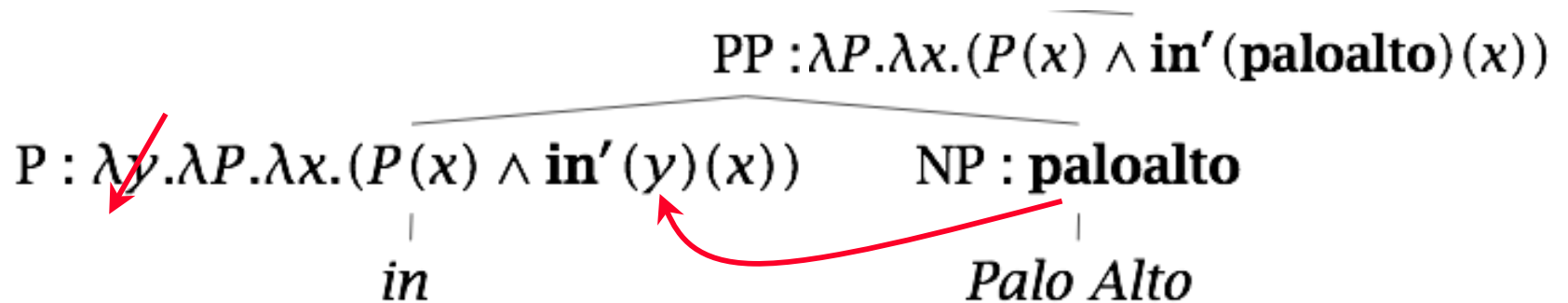
Adding lexical entries



Now do function applications (beta reductions)

Beta reduction 1: PP ‘*in Palo Alto*’

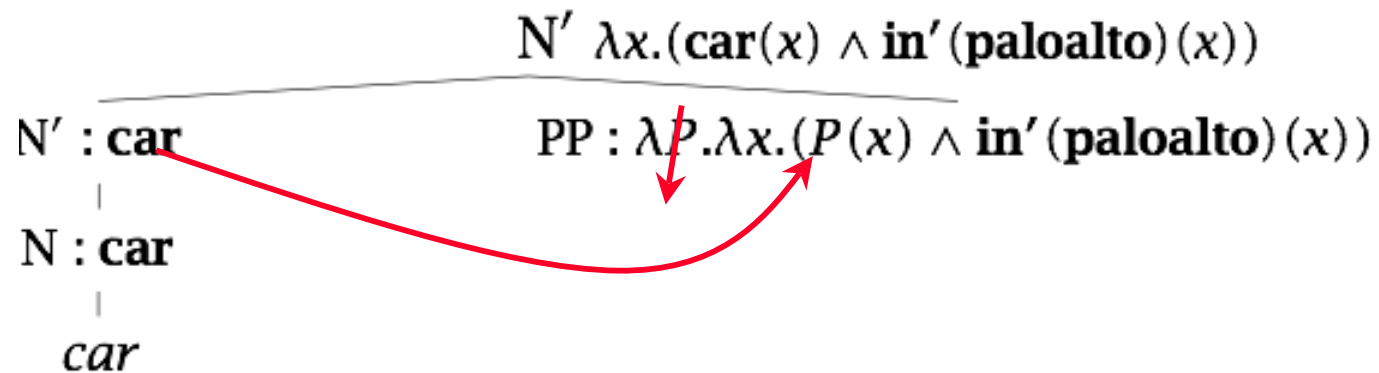
PP : $\beta(\alpha) \rightarrow P : \beta$ NP : α



NB: Church-Rosser theorem assures us we could do these reductions *in any order* and get same result at the end – good for computation

β -reduction 2: ‘*car* [in Palo Alto]’

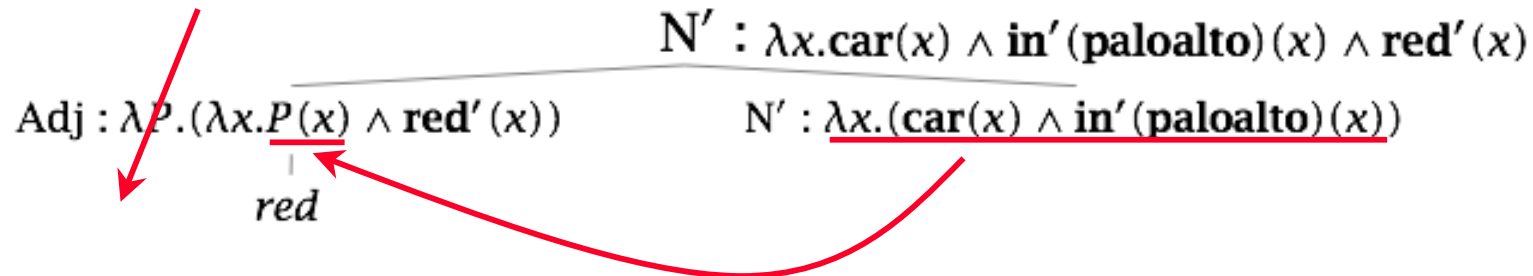
$$N' : \beta(\alpha) \rightarrow N' : \alpha \quad PP : \beta$$



$$N' : \beta \rightarrow N : \beta$$

β -reduction 3: ‘*red* [car in Palo Alto]’

$$N' : \beta(\alpha) \rightarrow \text{Adj} : \beta \quad N' : \alpha$$

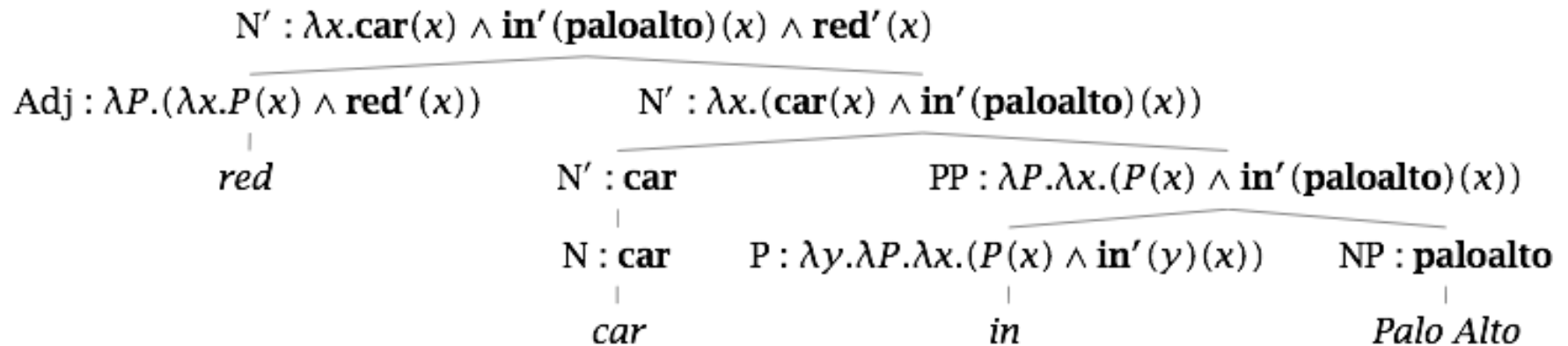


$\text{red}, \text{Adj} : \lambda P. (\lambda x. \overline{P(x)} \wedge \overline{\text{red}'(x)})$

lexicon

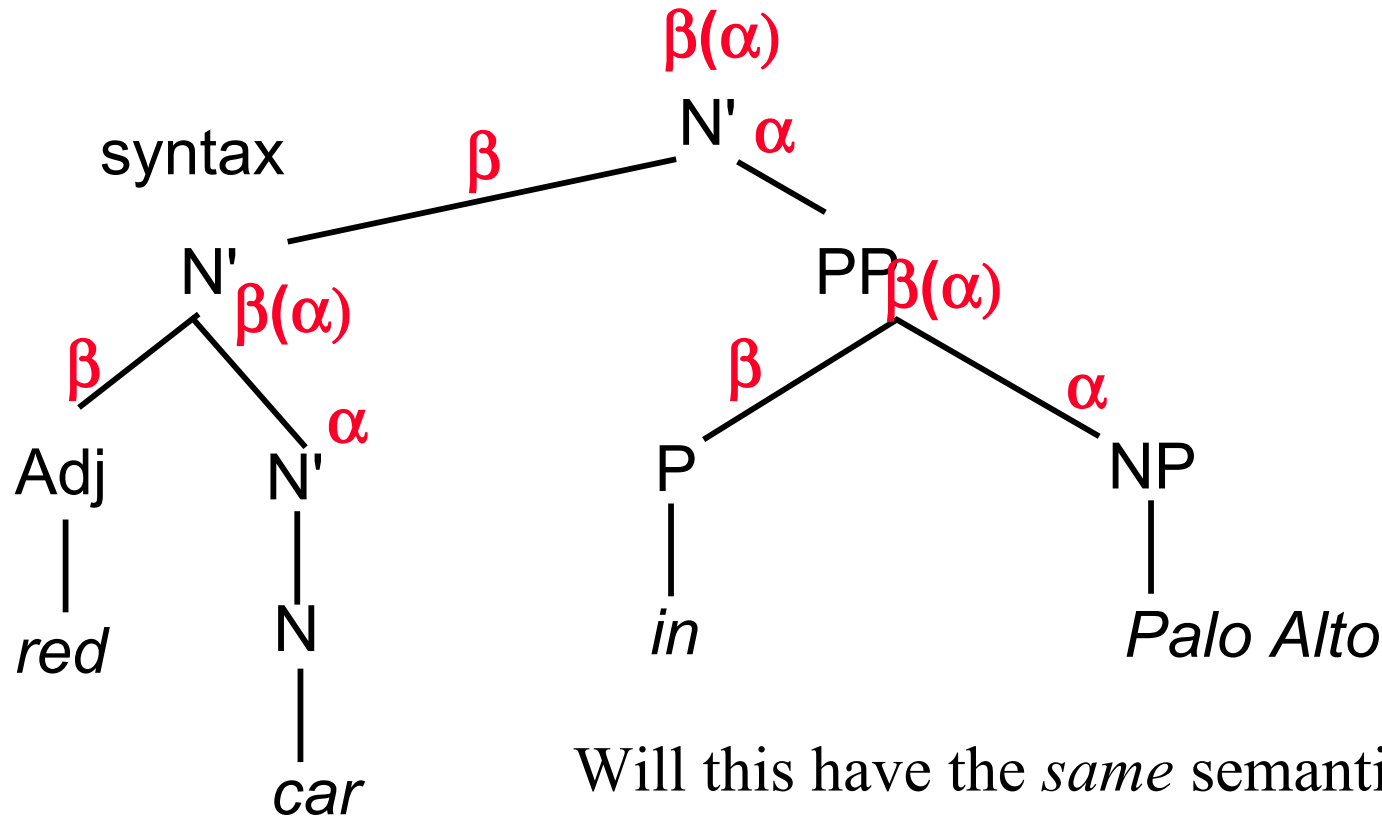
Hah, it works!!

We're done!!! Well, almost...



But...isn't there another parse...?

Parse #2 - what are *its* semantics?



Will this have the *same* semantics?

What does β -reduction construct?