



CMPT 413: Computational Linguistics

SEM1: Natural Language Semantics

Anoop Sarkar

<http://www.cs.sfu.ca/~anoop>

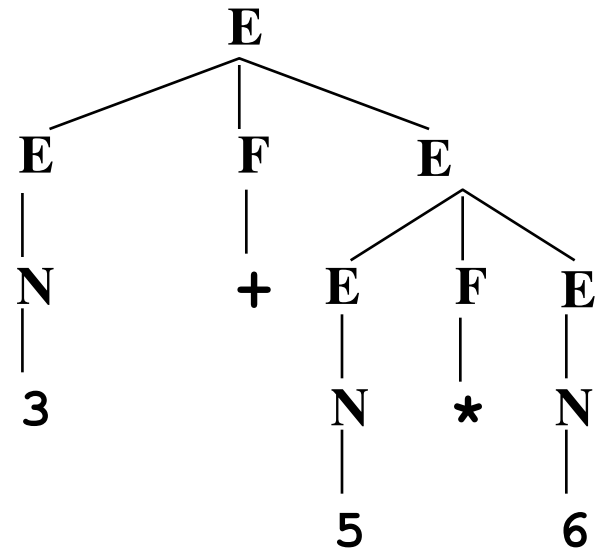
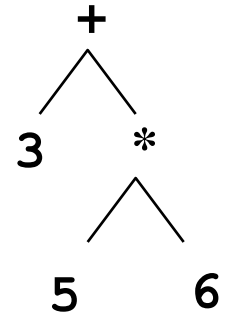
Semantics

From Syntax to Meaning!

Adapted from slides by Jason Eisner
used in: 600.465 - Intro to NLP - JHU

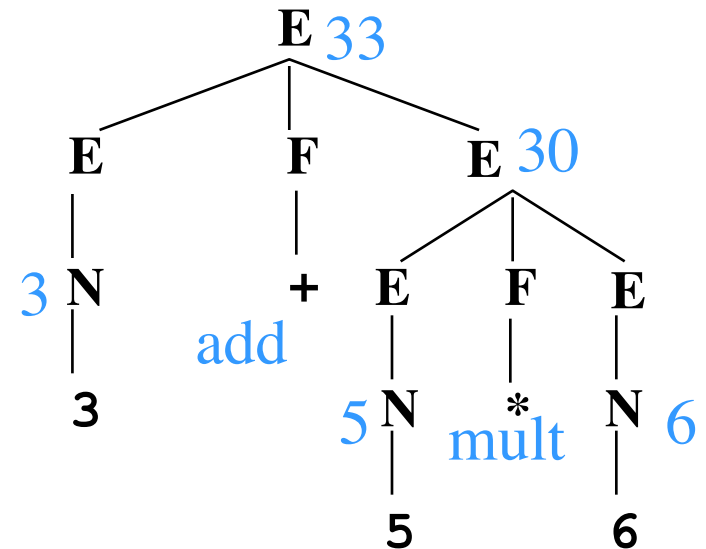
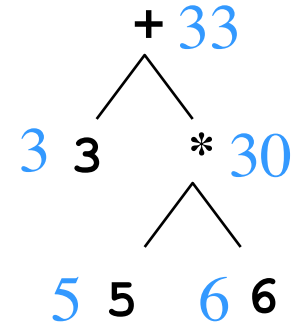
Programming Language Interpreter

- What is meaning of $3+5*6$?
- First parse it into $3+(5*6)$



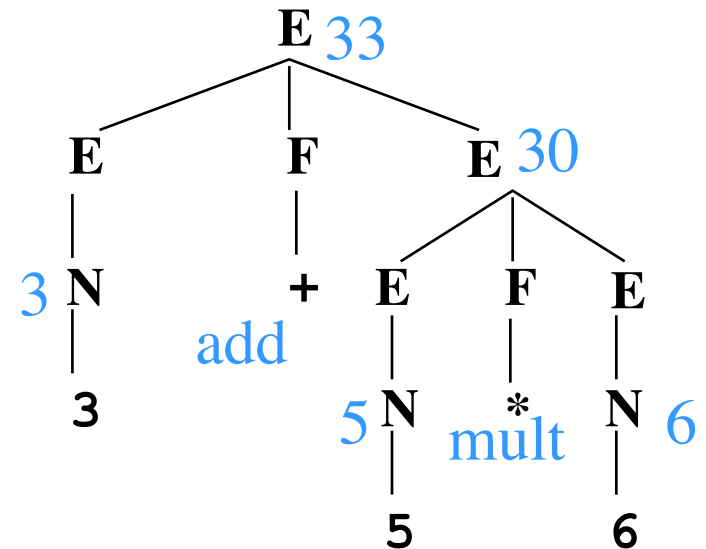
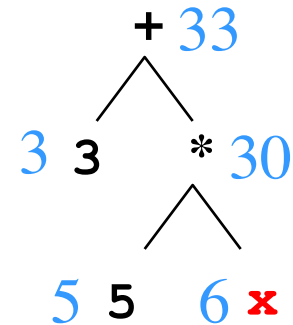
Programming Language Interpreter

- What is meaning of $3+5*6$?
- First parse it into $3+(5*6)$
- Now give a meaning to each node in the tree (bottom-up)



Interpreting in an Environment

- How about $3 + 5 * x$?
- Same thing: the meaning of x is found from the environment (it's 6)
- Analogies in language?



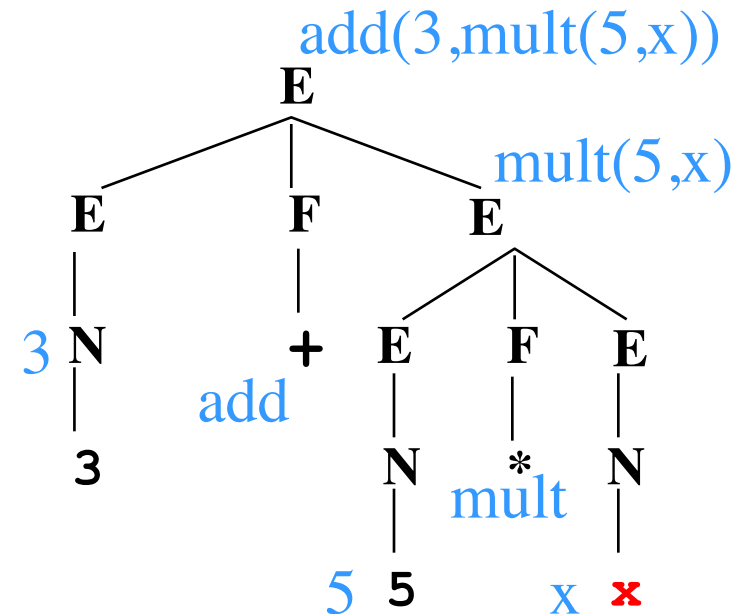
Compiling

- How about $3 + 5 * x$?
- Don't know x at compile time
- “Meaning” at a node is a piece of code, not a number

$5 * (x + 1) - 2$ is a different expression that produces *equivalent* code

(can be converted to the previous code by optimization)

Analogies in language?



What Counts as Understanding?

some notions

- We understand if we can respond appropriately
 - ok for commands, questions (these demand response)
 - “Computer, warp speed 5”
 - “throw axe at dwarf”
 - “put all of my blocks in the red box”
 - imperative programming languages
 - database queries and other questions
- We understand statement if we can determine its truth
 - Truth can be determined by checking a model
 - A model stores facts about the world, beliefs, etc.
 - There are well-known equivalences between a formula in a logic and model for that formula, so **we map NL into logic**

What Counts as Understanding?

some notions

- We understand a statement if we know *how* to determine its truth
 - A logic is an abstract language of statements such that:
 - Every statement has a model, and
 - A statement can be converted into another statement iff both statements are equivalent according to the same model
 - A statement is true iff it is **satisfiable** in a model
 - What are exact conditions under which it would be true? necessary + sufficient
 - Equivalently, derive all its consequences

Logic: Some Preliminaries

Three major kinds of objects

Booleans

- Roughly, the semantic values of sentences

Entities

- Values of NPs, e.g., objects like this slide
- Maybe also other types of entities, like times

Functions of various types

- A function returning a boolean is called a “predicate”
 - e.g., `frog(x)`, `green(x)`
- Functions might return other functions!
- Function might take other functions as arguments!

Logic: Lambda Terms

- Lambda terms:
 - A way of writing “anonymous functions”
 - No function header or function name
 - But defines the key thing: **behavior** of the function
 - Just as we can talk about 3 without naming it “x”
 - Let `square = λp p*p`
 - Equivalent to `int square(p) { return p*p; }`
 - But we can talk about `λp p*p` without naming it
 - Format of a lambda term: `λ variable expression`

Logic: Lambda Terms

- Lambda terms:
 - Let $\text{square} = \lambda p \, p * p$
 - Then $\text{square}(3) = (\lambda p \, p * p)(3) = 3 * 3$
 - **Note:** $\text{square}(x)$ isn't a function! It's just the value $x * x$.
 - But $\lambda x \, \text{square}(x) = \lambda x \, x * x = \lambda p \, p * p = \text{square}$
(proving that these functions are equal – and indeed they are,
as they act the same on all arguments: what is $(\lambda x \, \text{square}(x))(y)$?)

- Let $\text{even} = \lambda p \, (p \bmod 2 == 0)$ a predicate: returns true/false
- $\text{even}(x)$ is true if x is even
- How about $\text{even}(\text{square}(x))$?
- $\lambda x \, \text{even}(\text{square}(x))$ is true of numbers with even squares
 - Just apply rules to get $\lambda x \, (\text{even}(x * x)) = \lambda x \, (x * x \bmod 2 == 0)$
 - This happens to denote the same predicate as even does

Logic: Multiple Arguments

- All lambda terms have one argument
- But we can fake multiple arguments ...
- Suppose we want to write $\text{times}(5,6)$
- Remember: square can be written as $\lambda x \text{ square}(x)$
- Similarly, times is equivalent to $\lambda x \lambda y \text{ times}(x,y)$
- **Claim that $\text{times}(5)(6)$ means same as $\text{times}(5,6)$**
 - $\text{times}(5) = (\lambda x \lambda y \text{ times}(x,y)) (5) = \lambda y \text{ times}(5,y)$
 - $\text{times}(5)(6) = (\lambda y \text{ times}(5,y))(6) = \text{times}(5,6)$

Logic: Multiple Arguments

- All lambda terms have one argument
- But we can fake multiple arguments ...
- Claim that $\text{times}(5)(6)$ means same as $\text{times}(5,6)$
 - $\text{times}(5) = (\lambda x \lambda y \text{ times}(x,y)) (5) = \lambda y \text{ times}(5,y)$
 - If this function weren't anonymous, what would we call it?
 - $\text{times}(5)(6) = (\lambda y \text{ times}(5,y))(6) = \text{times}(5,6)$
- So we can always get away with 1-arg functions ...
 - ... which might return a function to take the next argument.
 - We'll still allow $\text{times}(x,y)$ as syntactic sugar, though

Grounding out

- So what does **times** actually mean???
- How do we get from **times(5,6)** to **30** ?
 - Whether **times(5,6) = 30** depends on whether symbol **times** actually denotes the multiplication function!
- Well, maybe **times** was defined as another lambda term, so substitute to get **times(5,6) = (blah blah blah)(5)(6)**
- But we can't keep doing substitutions forever!
 - Eventually we have to ground out in a **primitive term**
 - Primitive terms are bound to object code
- Maybe **times(5,6)** just executes a multiplication function
- What is executed by **loves(john, mary)** ?

Logic: Interesting Constants

- Thus, have “constants” that name some of the entities and functions (e.g., **times**):
 - **Gilly** - an entity
 - **red** – a predicate on entities
 - holds of just the red entities: $\text{red}(x)$ is true if x is red!
 - **loves** – a predicate on 2 entities
 - **loves(Gilly, Lilly)**
 - *Question*: What does **loves(Lilly)** denote?
- Constants used to define meanings of words
- Meanings of phrases will be built from the constants

Logic: Interesting Constants

- **most** – a predicate on 2 predicates on entities
 - **most(pig, big)** = “most pigs are big”
 - Equivalently, **most(λx pig(x), λx big(x))**
 - returns true if most of the things satisfying the first predicate also satisfy the second predicate
- similarly for other quantifiers
 - **all(pig, big)** (equivalent to **$\forall x$ pig(x) \Rightarrow big(x)**)
 - **exists(pig, big)** (equivalent to **$\exists x$ pig(x) AND big(x)**)
 - can even build complex quantifiers from English phrases:
 - “between 12 and 75”; “a majority of”; “all but the smallest 2”

A reasonable representation?

- Gilly swallowed a goldfish
- First attempt: `swallowed(Gilly, goldfish)`
- Returns true or false. Analogous to
 - `prime(17)`
 - `equal(4, 2+2)`
 - `loves(Gilly, Lilly)`
 - `swallowed(Gilly, Jilly)`
- ... or is it analogous?

A reasonable representation?

- Gilly swallowed a goldfish
 - First attempt: `swallowed(Gilly, goldfish)`
- But we're not paying attention to `a`!
- `goldfish` isn't the name of a unique object the way `Gilly` is
- In particular, don't want
Gilly swallowed a goldfish and Milly
swallowed a goldfish
to translate as
`swallowed(Gilly, goldfish) AND swallowed(Milly, goldfish)`
since probably not the same goldfish ...

Use a Quantifier

- Gilly swallowed a goldfish
 - First attempt: `swallowed(Gilly, goldfish)`
- Better: $\exists g \text{ goldfish}(g) \text{ AND } \text{swallowed}(\text{Gilly}, g)$
- Or using one of our quantifier predicates:
 - `exists($\lambda g \text{ goldfish}(g)$, $\lambda g \text{ swallowed}(\text{Gilly}, g)$)`
 - Equivalently: `exists(goldfish, swallowed(Gilly))`
 - “In the set of goldfish there exists one swallowed by Gilly”
- Here `goldfish` is a predicate on entities
 - This is the same semantic type as `red`
 - But `goldfish` is noun and `red` is adjective .. `#@!?`

Tense

- Gilly swallowed a goldfish
 - Previous attempt: $\text{exists}(\text{goldfish}, \lambda g \text{ swallowed}(\text{Gilly}, g))$
- Improve to use tense:
 - Instead of the 2-arg predicate $\text{swallowed}(\text{Gilly}, g)$ try a 3-arg version $\text{swallow}(t, \text{Gilly}, g)$ where t is a time
 - Now we can write:
 $\exists t \text{ past}(t) \text{ AND } \text{exists}(\text{goldfish}, \lambda g \text{ swallow}(t, \text{Gilly}, g))$
 - “There was some time in the past such that a goldfish was among the objects swallowed by Gilly at that time”

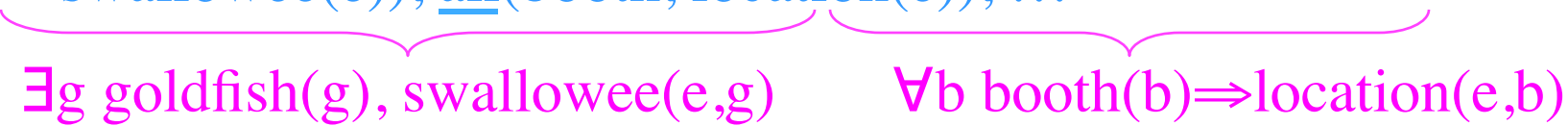
(Simplify Notation)

- Gilly swallowed a goldfish
 - Previous attempt: `exists(goldfish, swallowed(Gilly))`
- Improve to use tense:
 - Instead of the 2-arg predicate `swallowed(Gilly,g)` try a 3-arg version `swallow(t,Gilly,g)`
 - Now we can write:
`∃t past(t) AND exists(goldfish, swallow(t,Gilly))`
 - “There was some time in the past such that a goldfish was among the objects swallowed by Gilly at that time”

Event Properties

- Gilly swallowed a goldfish
 - Previous: $\exists t \text{ past}(t) \text{ AND exists(goldfish, swallow}(t, \text{Gilly}))$
- Why stop at time? An event has other properties:
 - [Gilly] swallowed [a goldfish] [on a dare] [in a telephone booth] [with 30 other freshmen] [after many bottles of vodka had been consumed].
 - Specifies who what why when ...
- Replace time variable t with an event variable e
 - $\exists e \text{ past}(e), \text{act}(e, \text{swallowing}), \text{swallower}(e, \text{Gilly}), \text{exists}(\text{goldfish}, \text{swallowee}(e)), \text{exists}(\text{booth}, \text{location}(e)), \dots$
 - As with probability notation, a comma represents AND
 - Could define past as $\lambda e \exists t \text{ before}(t, \text{now}), \text{ended-at}(e, t)$

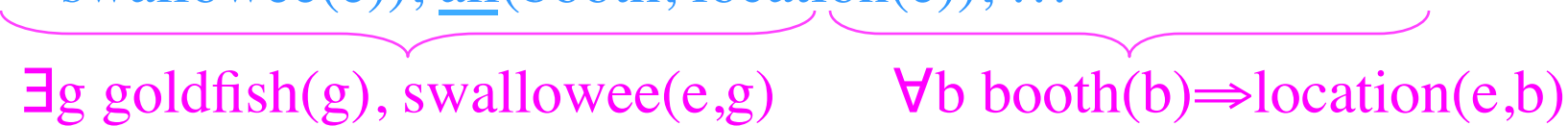
Quantifier Order

- 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 min. Our job is to find that woman and stop her.
 - $\exists \text{woman } (\forall 15\text{min } \text{gives-birth-during}(\text{woman}, 15\text{min}))$
 - $\forall 15\text{min } (\exists \text{woman } \text{gives-birth-during}(15\text{min}, \text{woman}))$
 - Surprisingly, both are possible in natural language!
 - Which is the joke meaning (where it's always the same woman) and why?
- What about:
 - Every prof admires, and every student detests, some course.

Quantifier Order

- Gilly swallowed a goldfish in a booth
 - $\exists e \text{ past}(e), \text{act}(e, \text{swallowing}), \text{swallower}(e, \text{Gilly}), \text{exists}(\text{goldfish}, \text{swallowee}(e)), \underline{\text{exists}}(\text{booth}, \text{location}(e)), \dots$
- Gilly swallowed a goldfish in every booth
 - $\exists e \text{ past}(e), \text{act}(e, \text{swallowing}), \text{swallower}(e, \text{Gilly}), \text{exists}(\text{goldfish}, \text{swallowee}(e)), \underline{\text{all}}(\text{booth}, \text{location}(e)), \dots$
 $\exists g \text{ goldfish}(g), \text{swallowee}(e, g) \quad \forall b \text{ booth}(b) \Rightarrow \text{location}(e, b)$
- Does this mean what we'd expect??
 - 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”

Nouns and Their Modifiers

- expert
 - $\lambda g \text{ expert}(g)$
- big fat expert
 - $\lambda g \text{ big}(g), \text{fat}(g), \text{expert}(g)$
 - 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
- Baltimore expert (white-collar expert, TV expert ...)
 - $\lambda g \text{ Related}(\text{Baltimore}, g), \text{expert}(g)$ – expert from Baltimore
 - Or $\lambda g (\text{Modified-by}(\text{Baltimore}, \text{expert}))(g)$ – expert on Baltimore
 - 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

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

Speech Acts

- What is the meaning of a full sentence?
 - Depends on the punctuation mark!?
 - Billy likes Lilly. → **assert**(like(B,L))
 - Billy likes Lilly? → **ask**(like(B,L))
 - or more formally, “Does Billy like Lilly?”
 - Billy, like Lilly! → **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**($\exists e \text{ past}(e), \text{act}(e, \text{eating}), \text{eater}(f, \text{Speaker}),$
 $\text{eatee}(\dots))$

Compositional Semantics

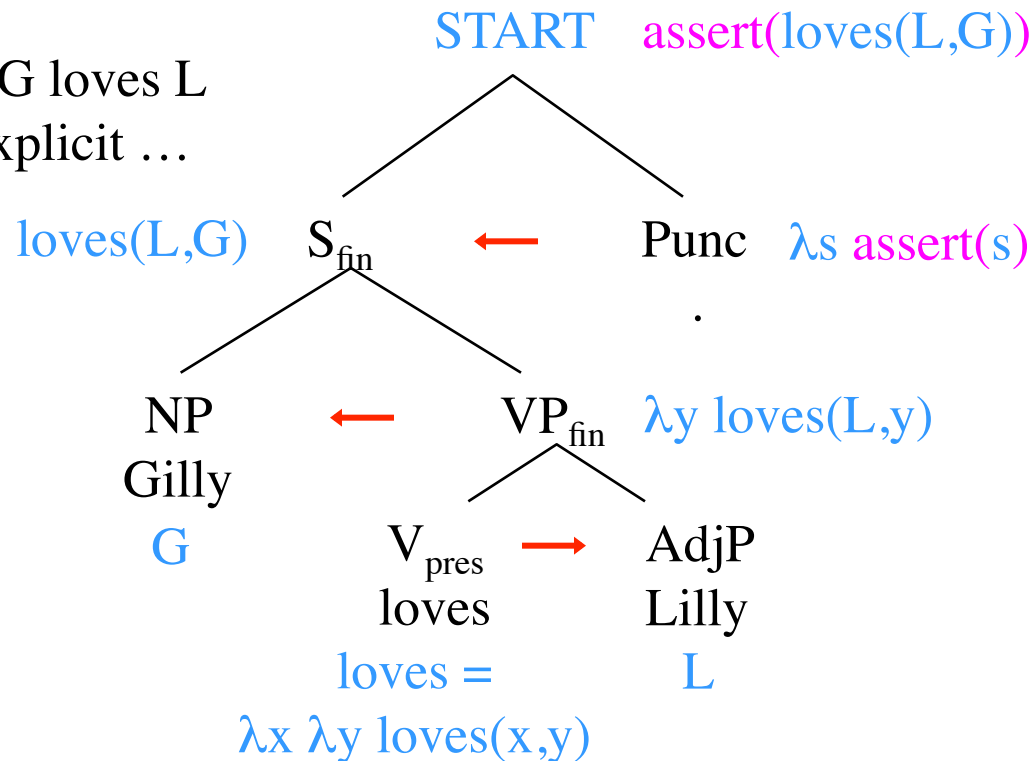
- We've discussed what semantic representations should look like.
- **But how do we get them from sentences???**
- **First** - parse to get a syntax tree.
- **Second** - look up the semantics for each word.
- **Third** - build the semantics for each constituent
 - Work from the bottom up
 - The syntax tree is a “recipe” for how to do it

Compositional Semantics

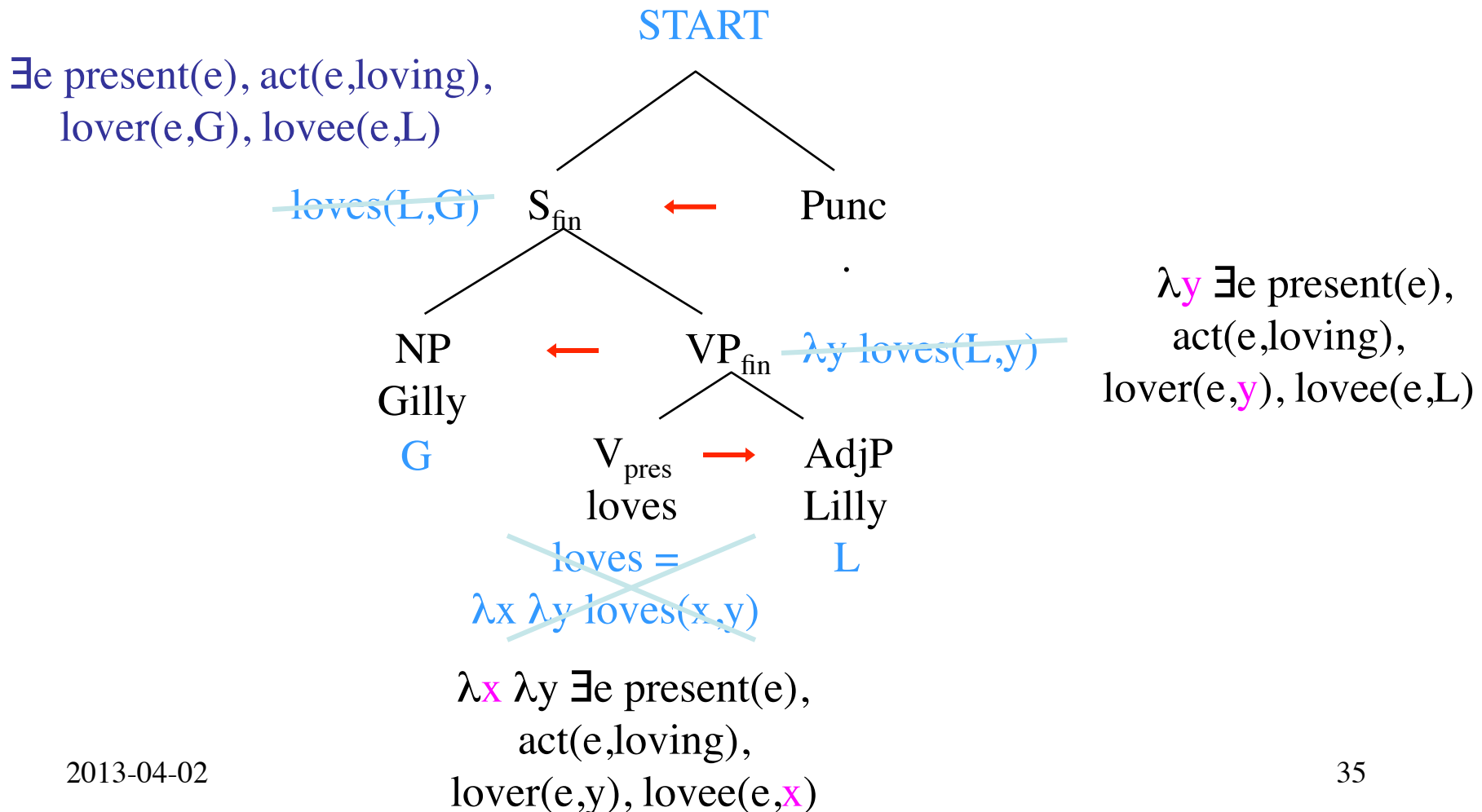
- Instead of $S \rightarrow NP \text{ loves } NP$
 - $S[\text{sem}=\text{loves}(x,y)] \rightarrow NP[\text{sem}=x] \text{ loves } NP[\text{sem}=y]$
- might want general rules like $S \rightarrow NP \text{ VP}$:
 - $V[\text{sem}=\text{loves}] \rightarrow \text{loves}$
 - $VP[\text{sem}=\text{v}(\text{obj})] \rightarrow V[\text{sem}=\text{v}] NP[\text{sem}=\text{obj}]$
 - $S[\text{sem}=\text{vp}(\text{subj})] \rightarrow NP[\text{sem}=\text{subj}] VP[\text{sem}=\text{vp}]$
- Now `Gilly loves Lilly` has $\text{sem}=\text{loves}(\text{Lilly})(\text{Gilly})$
- In this manner we'll sketch a version where
 - Still compute semantics bottom-up
 - Grammar is in Chomsky Normal Form
 - So each node has 2 children: 1 function & 1 argument
 - To get its semantics, apply function to argument!

Compositional Semantics

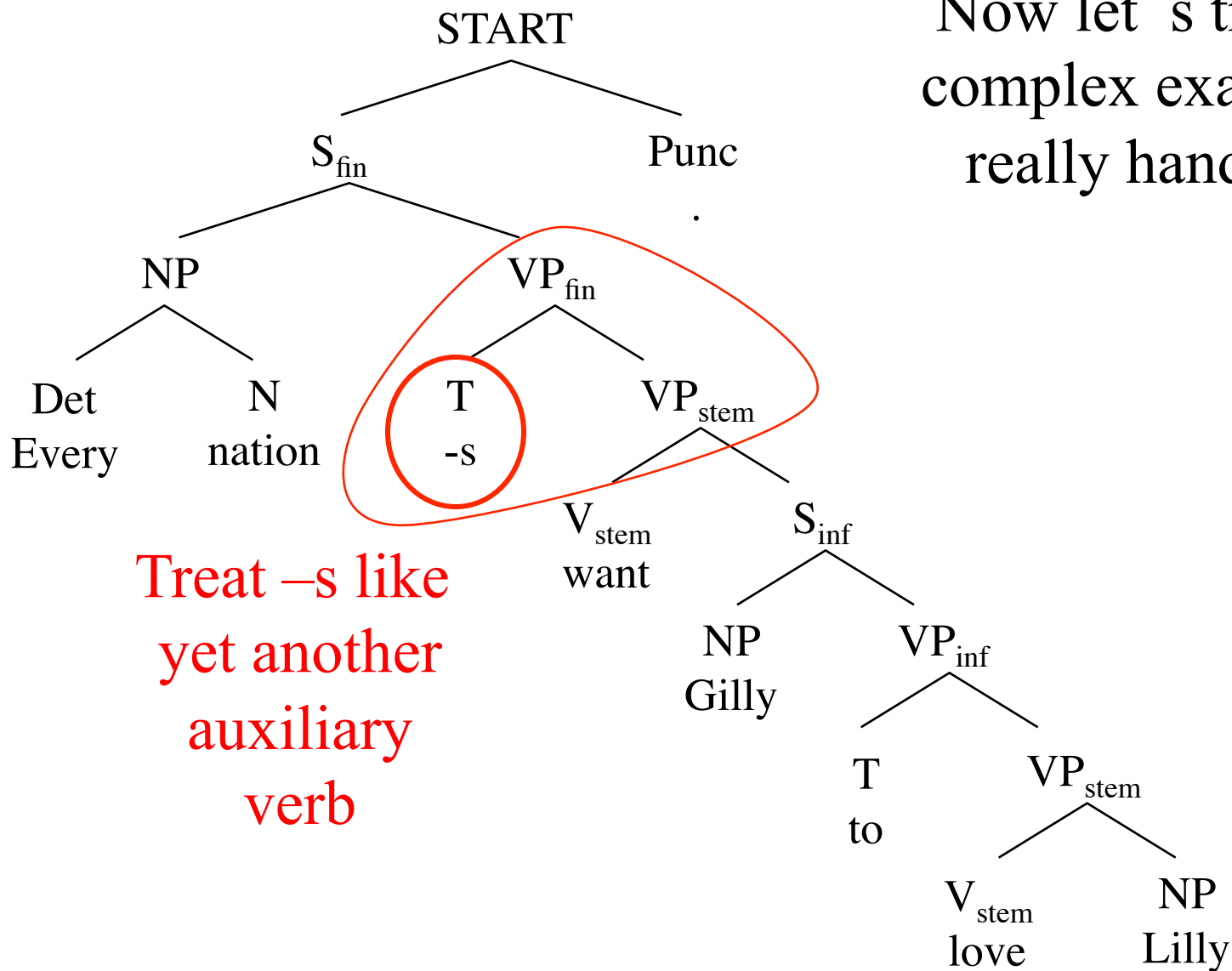
Intended to mean G loves L
Let's make this explicit ...

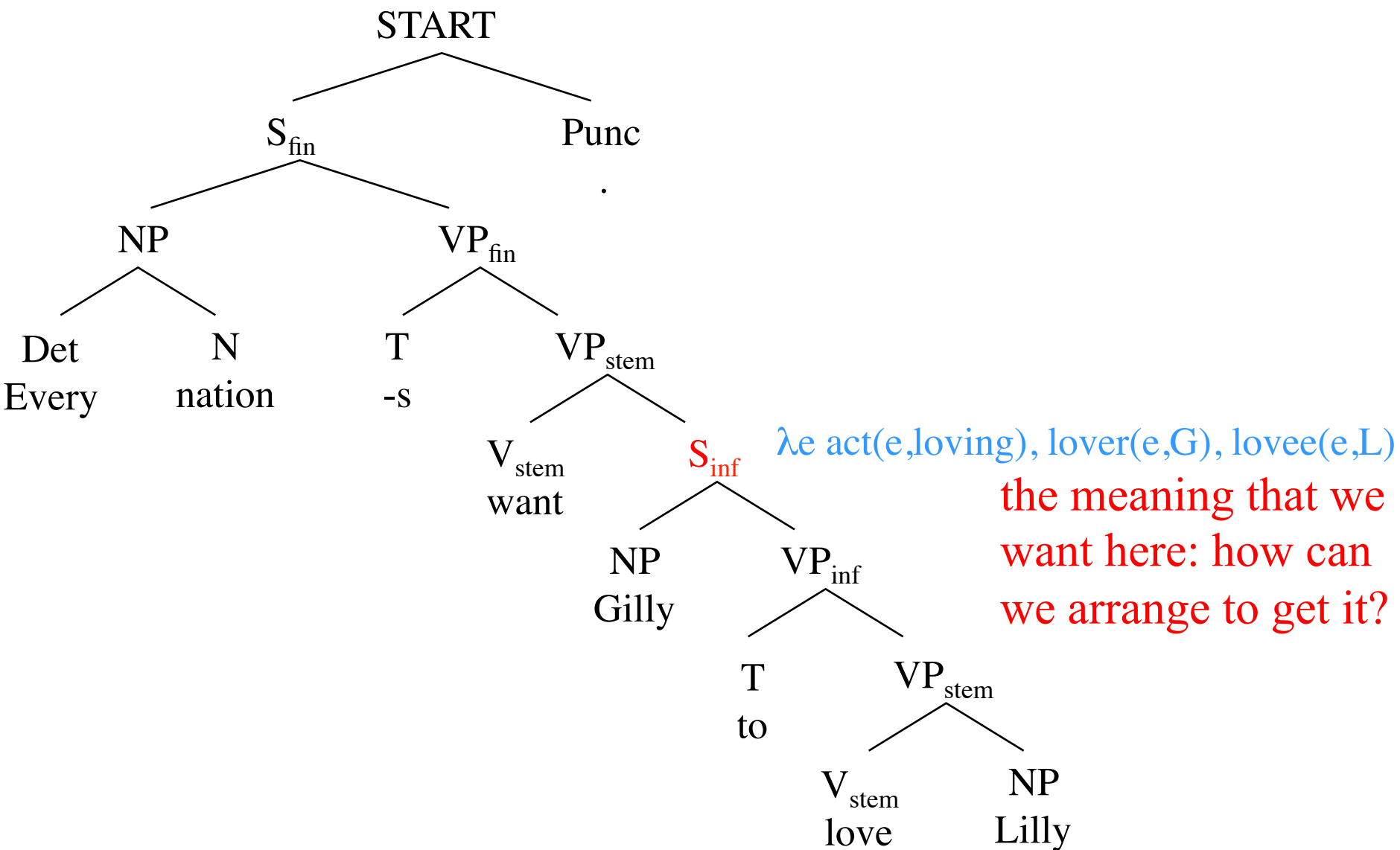


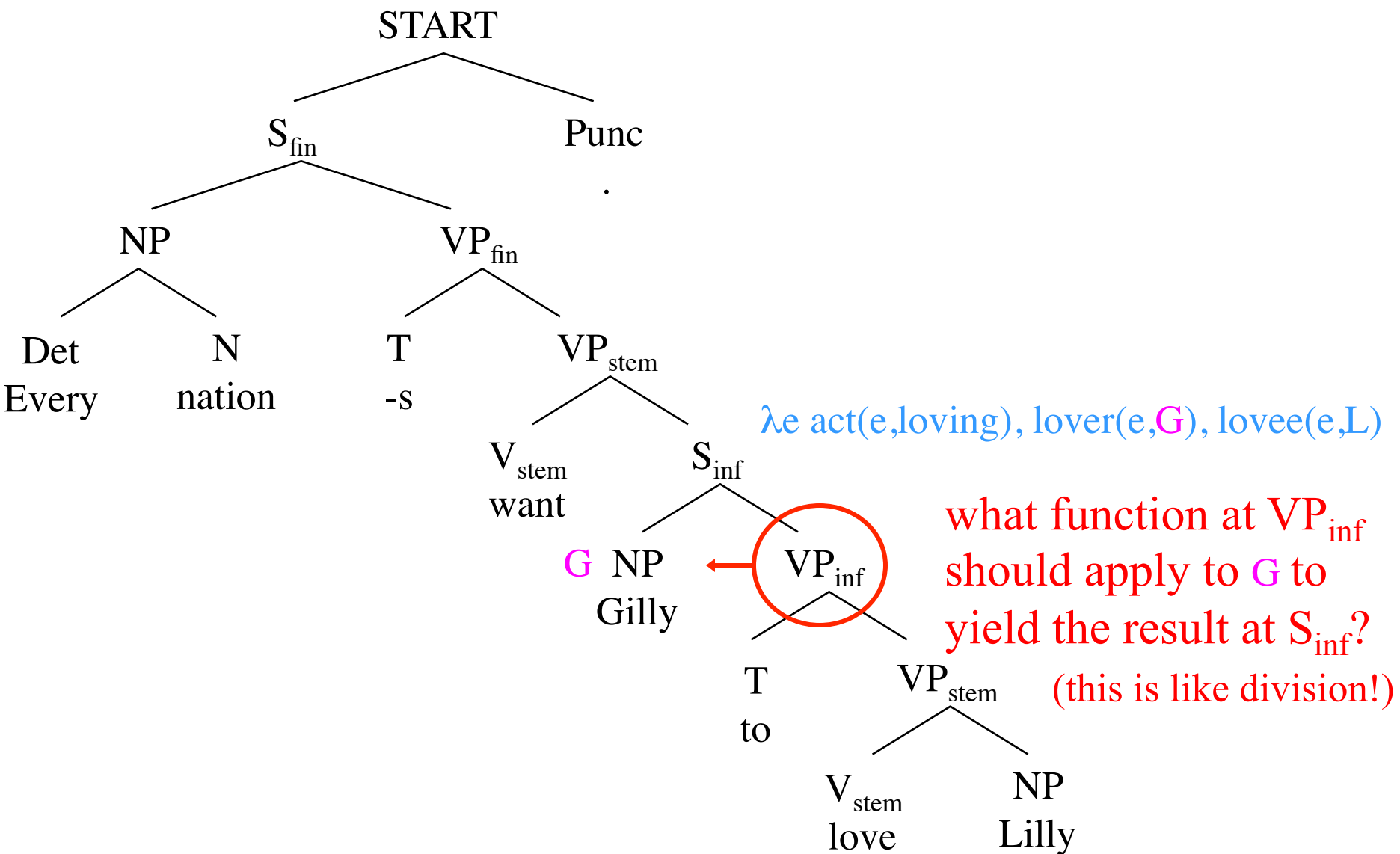
Compositional Semantics

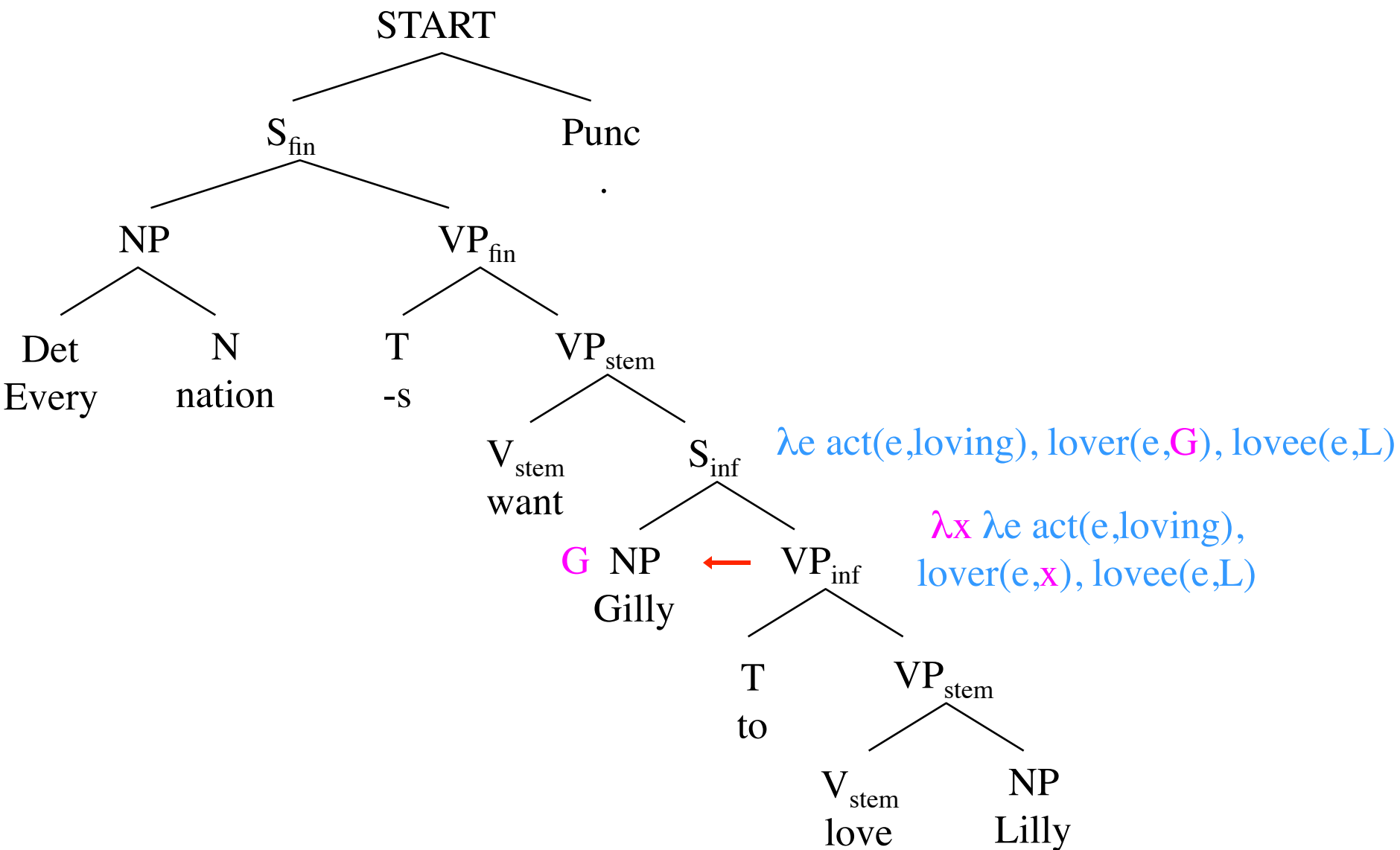


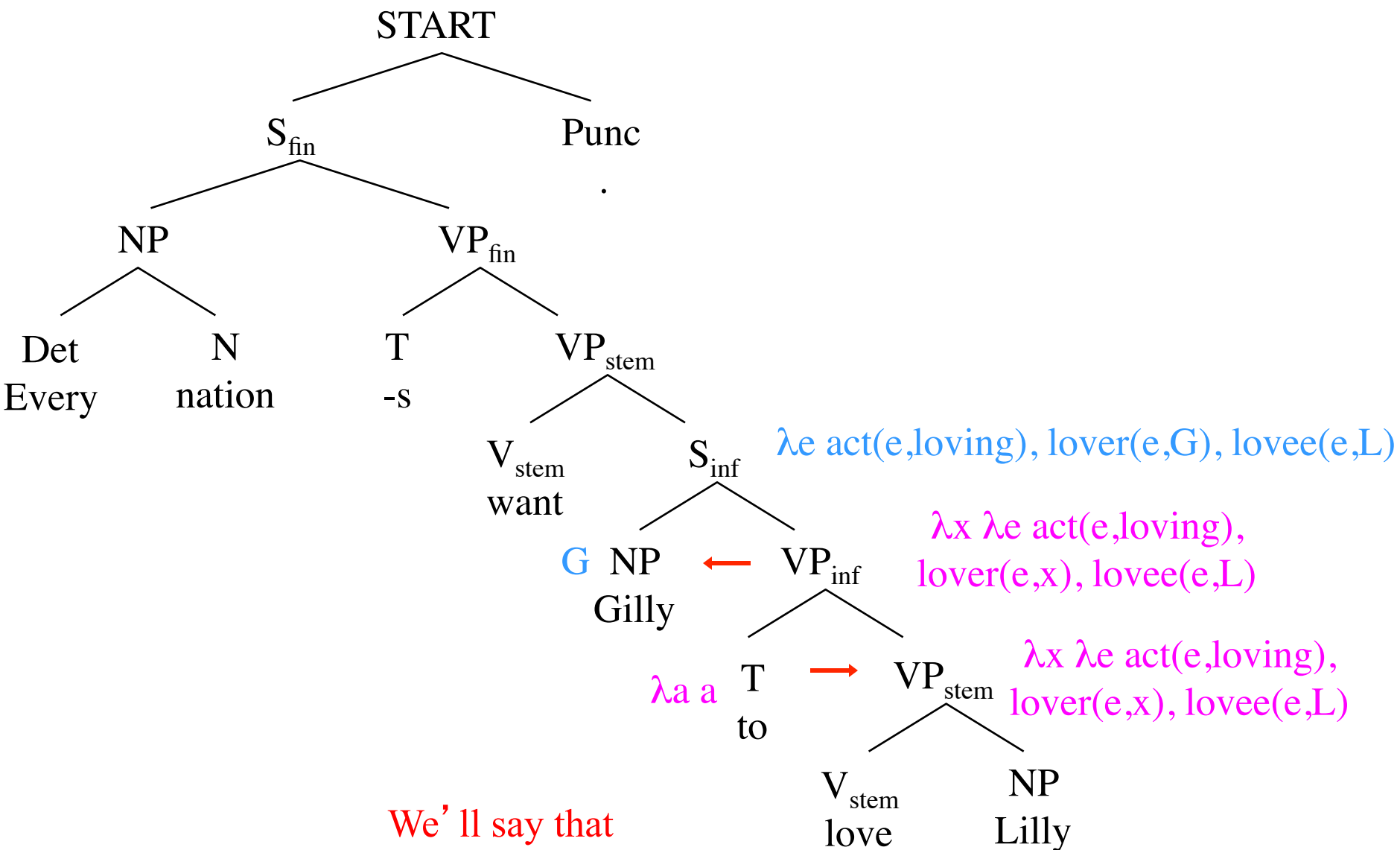
Now let's try a more complex example, and really handle tense.



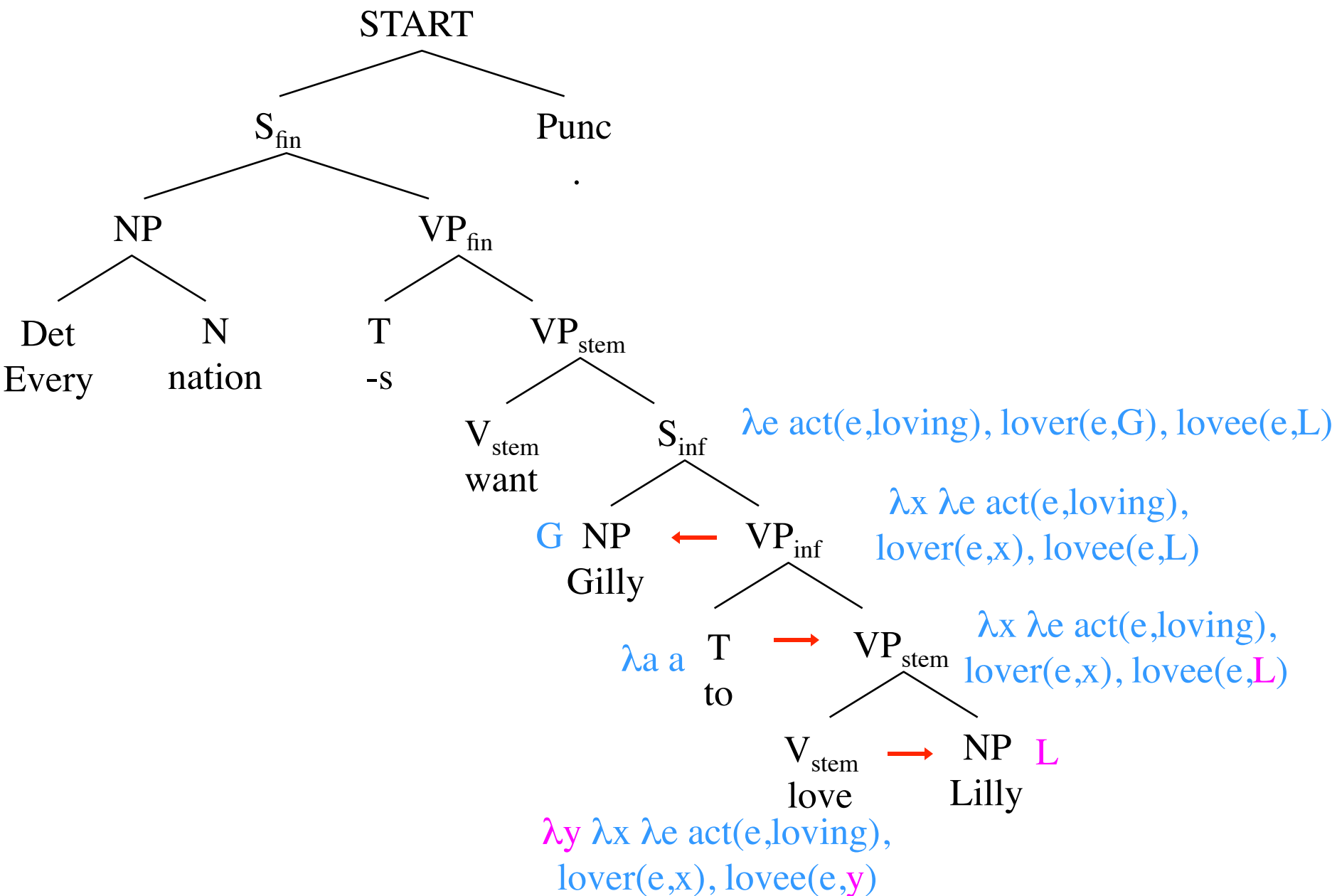


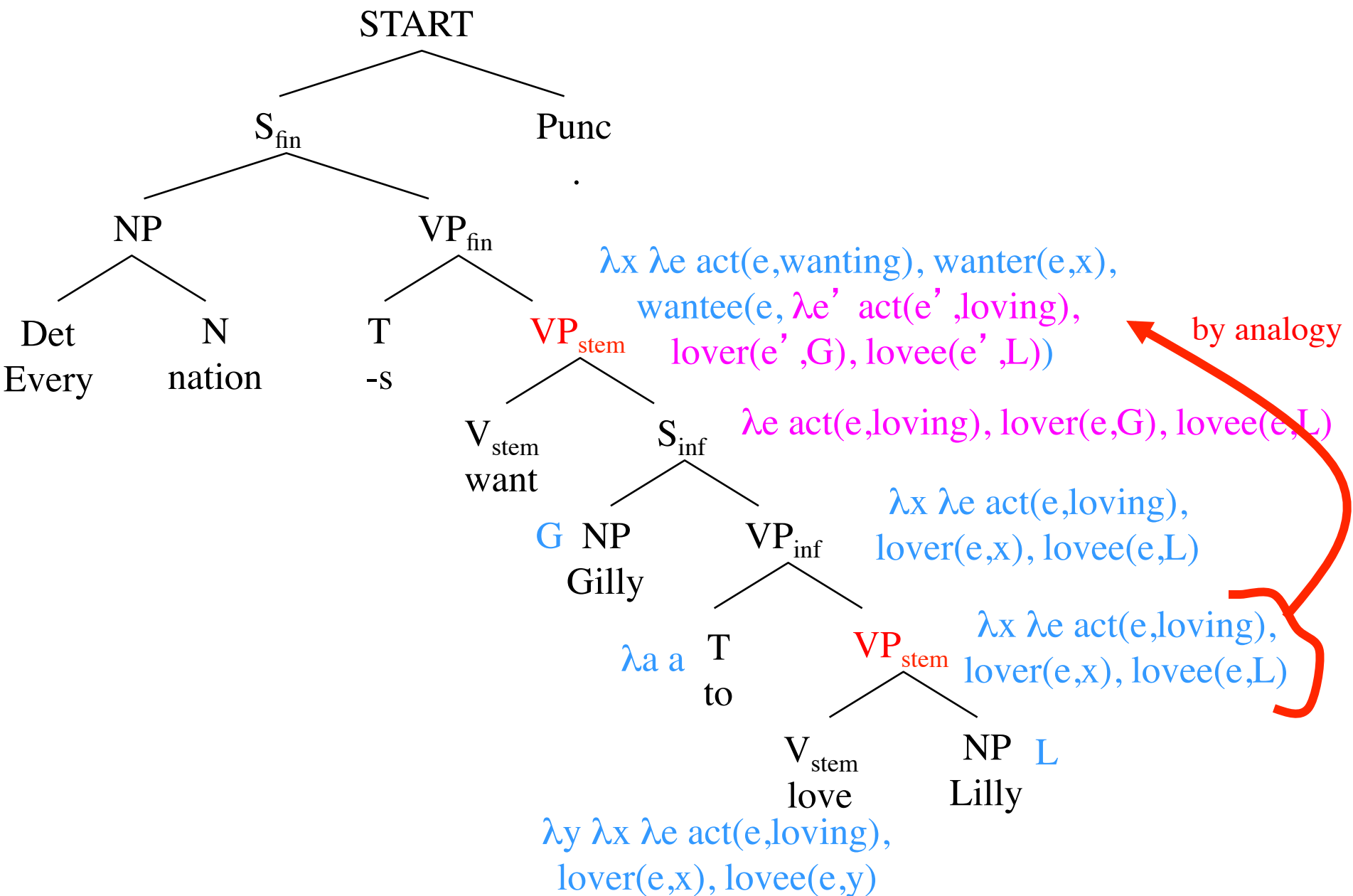


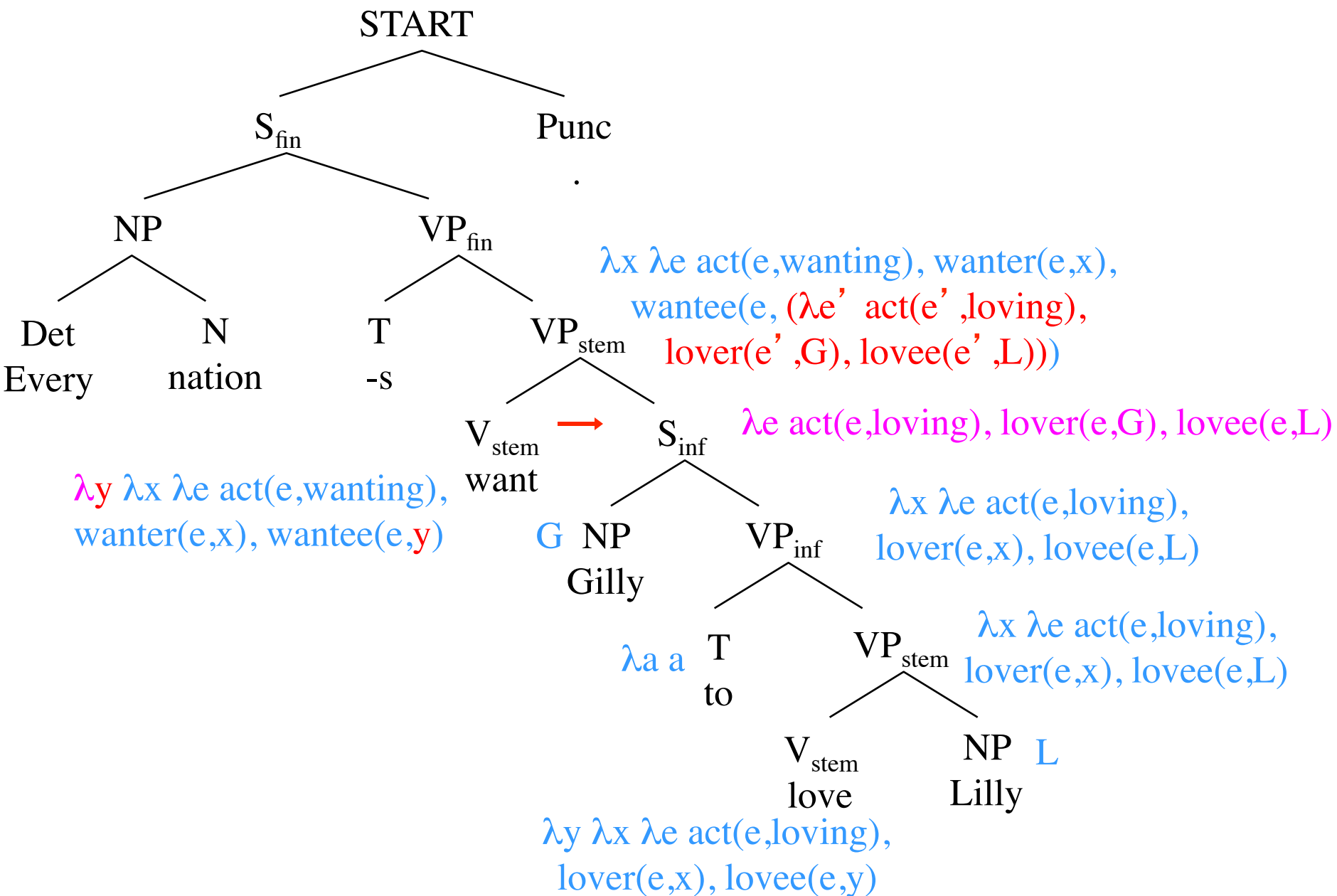


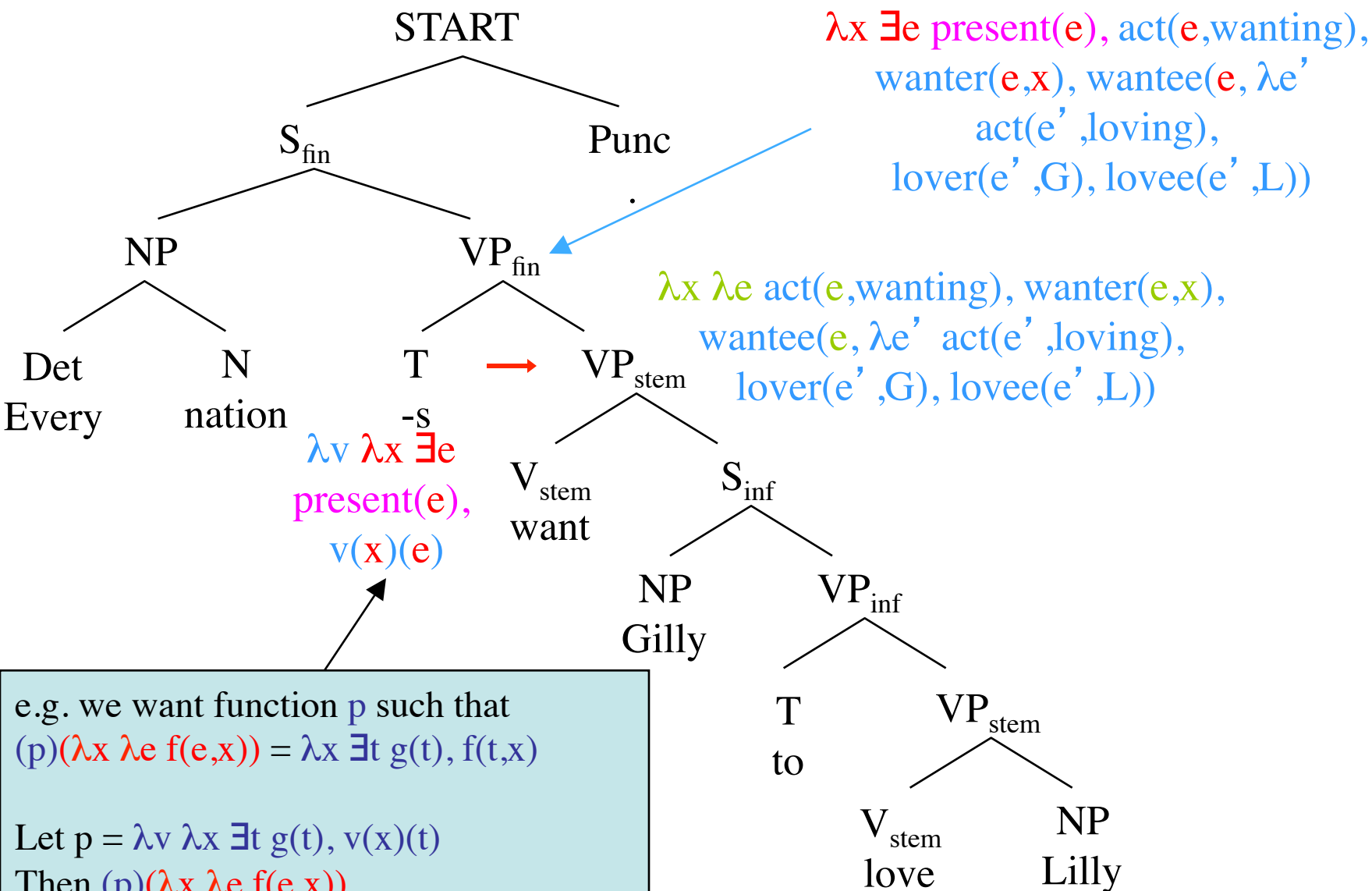


We'll say that
 "to" is just a bit of syntax that
 changes a VP_{stem} to a VP_{inf}
 with the same meaning.





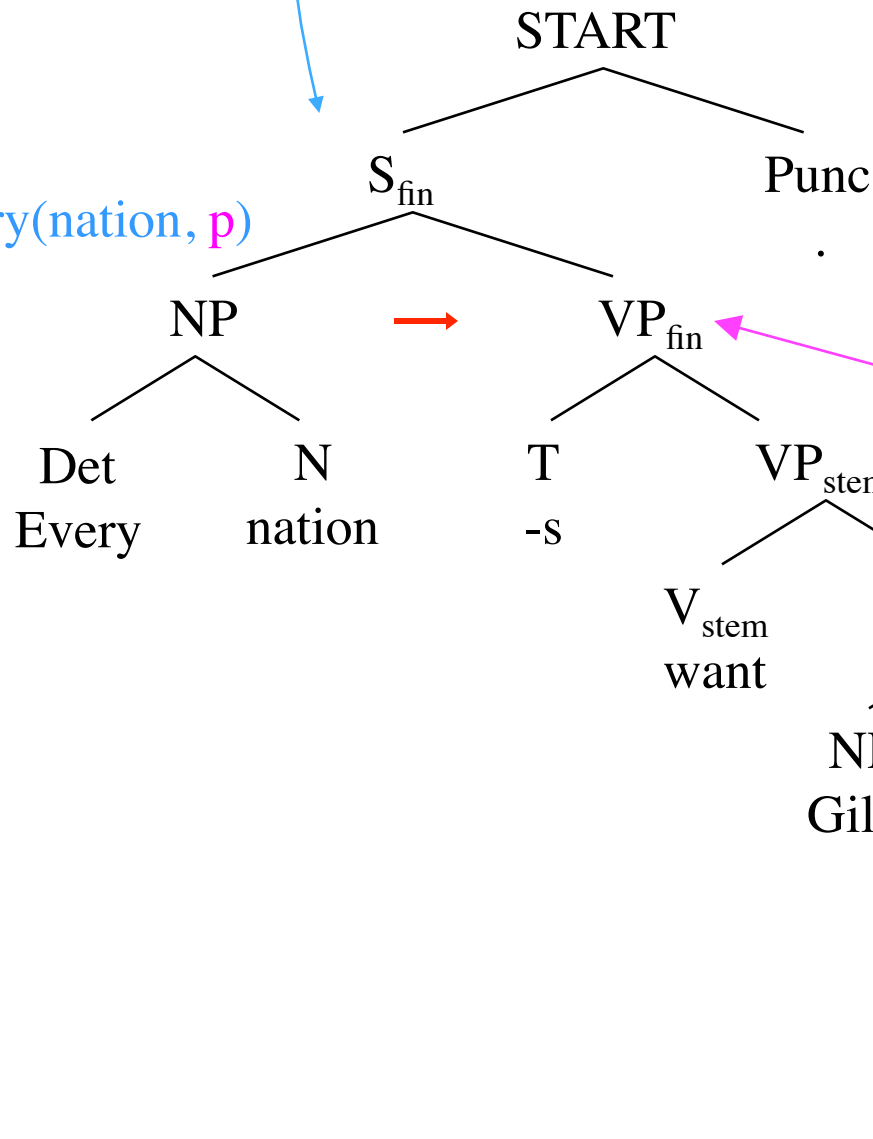




e.g. we want function p such that
 $(p)(\lambda x \lambda e f(e,x)) = \lambda x \exists t g(t), f(t,x)$

Let $p = \lambda v \lambda x \exists t g(t), v(x)(t)$
 Then $(p)(\lambda x \lambda e f(e,x))$
 $= (\lambda v \lambda x \exists t g(t), v(x)(t)) (\lambda x \lambda e f(e,x))$
 $= (\lambda x \exists t g(t), (\lambda x \lambda e f(e,x))(x)(t))$
 $= \lambda x \exists t g(t), f(t,x)$

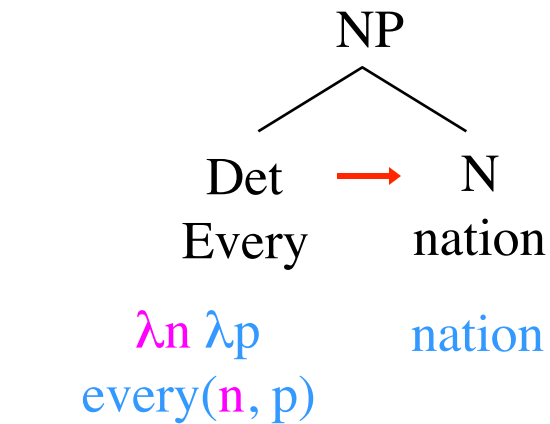
λp every(nation, p)



every(nation, ($\lambda x \exists e$ present(e),
act(e,wanting), want(e,x),
wantee(e, $\lambda e'$ act(e',loving),
lover(e',G), lovee(e',L))))

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

$\lambda p \text{ every}(\text{nation}, p)$



START

S_{fin}

Punc

$\text{every}(\text{nation}, (\lambda x \exists e \text{ present}(e),$
 $\text{act}(e, \text{wanting}), \text{wanter}(e, x),$
 $\text{wantee}(e, \lambda e' \text{ act}(e', \text{loving}),$
 $\text{lover}(e', G), \text{lovee}(e', L))))$

VP_{fin}



T

-s

VP_{stem}

V_{stem}

want

S_{inf}

NP

Gilly

VP_{inf}

T

to

VP_{stem}

V_{stem}

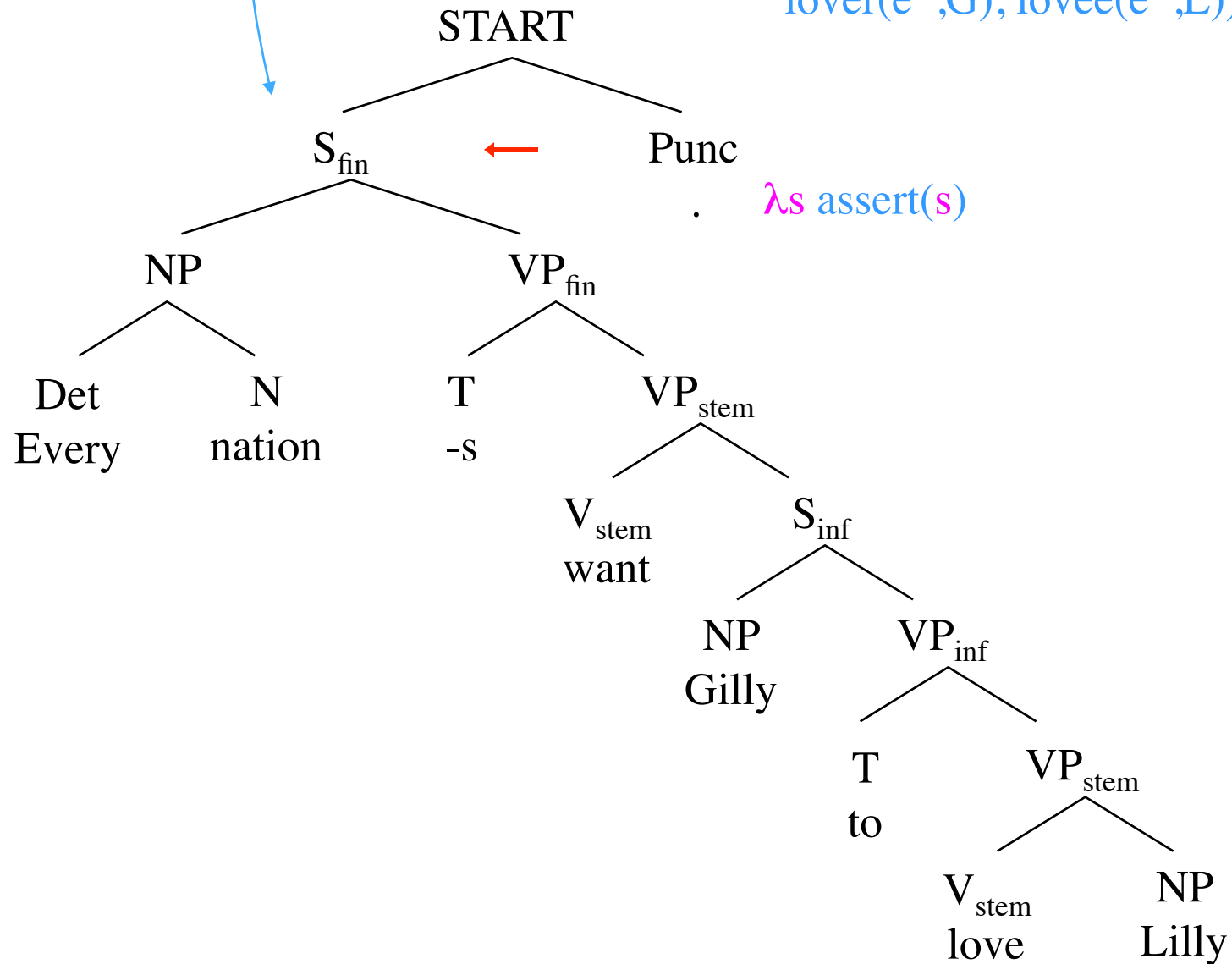
love

NP

Lilly

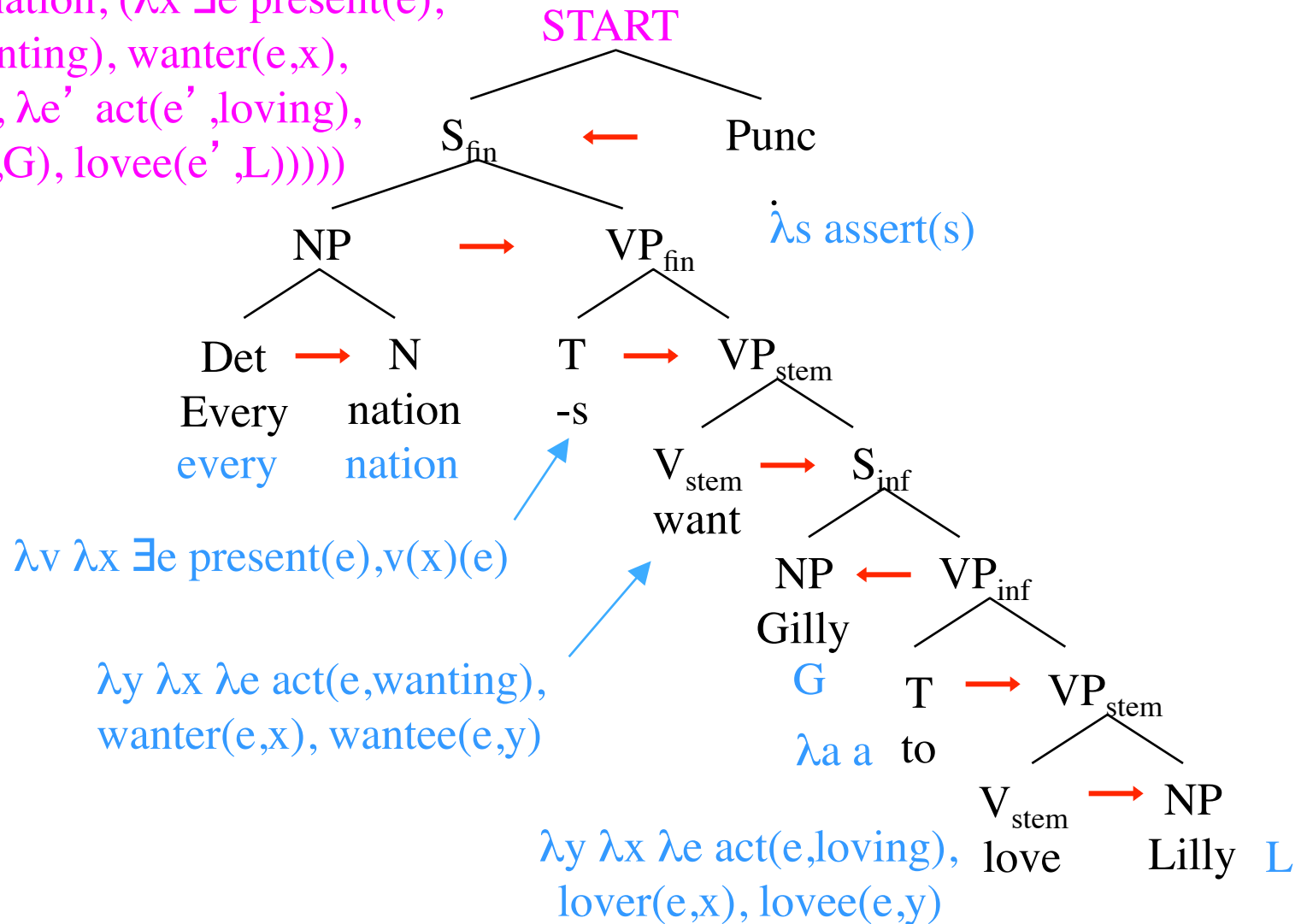
$\lambda x \exists e \text{ present}(e), \text{act}(e, \text{wanting}),$
 $\text{wanter}(e, x), \text{wantee}(e, \lambda e' \text{ act}(e', \text{loving}),$
 $\text{lover}(e', G), \text{lovee}(e', L))$

every(nation, ($\lambda x \exists e$ present(e),
 act(e,wanting), want(e,x),
 wantee(e, $\lambda e'$ act(e',loving),
 lover(e',G), lovee(e',L))))



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',G), lovee(e',L))))))



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 unicorn \neq set of dodos
 - Other worlds also useful for:

You must pay the rent

If you hadn’t,

You can pay the rent

you’d be homeless

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.”

Other Fun Semantic Stuff

- Temporal logic
 - Gilly had swallowed eight goldfish before Milly reached the bowl
 - Billy said Jilly was pregnant (sequence of tense)
 - Billy said, “Jilly is pregnant.”
- Generics (not quite the same as plurals)
 - Typhoons arise in the Pacific
 - Children must be carried
- Presuppositions
 - The king of France is bald. (if there is no such king is this true?)
 - Have you stopped beating your wife? (what is presupposed?)
- Pronoun-Quantifier Interaction (“bound anaphora”)
 - Every farmer who owns a donkey beats it.
 - If you have a dime, put it in the meter.
 - The woman who every Englishman loves is his mother.
 - I love my mother and so does Billy.