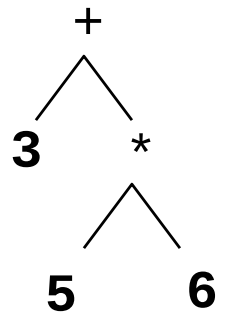
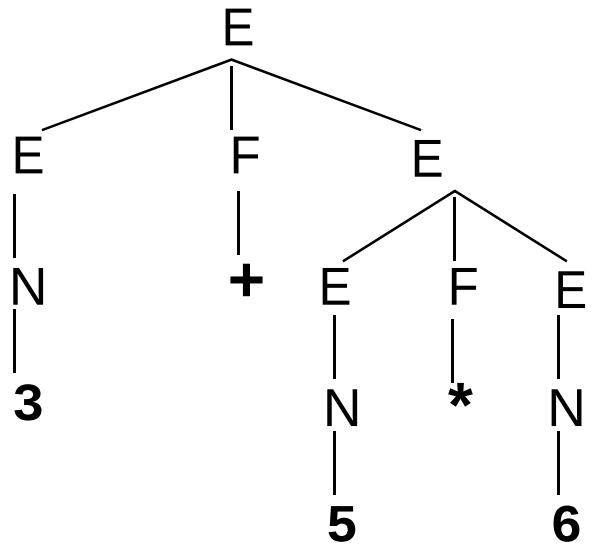
Semantics

From Syntax to Meaning!

Programming Language Interpreter

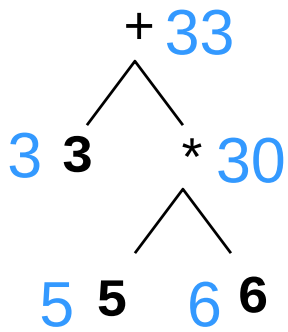
- What is meaning of 3+5*6?
- First parse it into 3+(5*6)

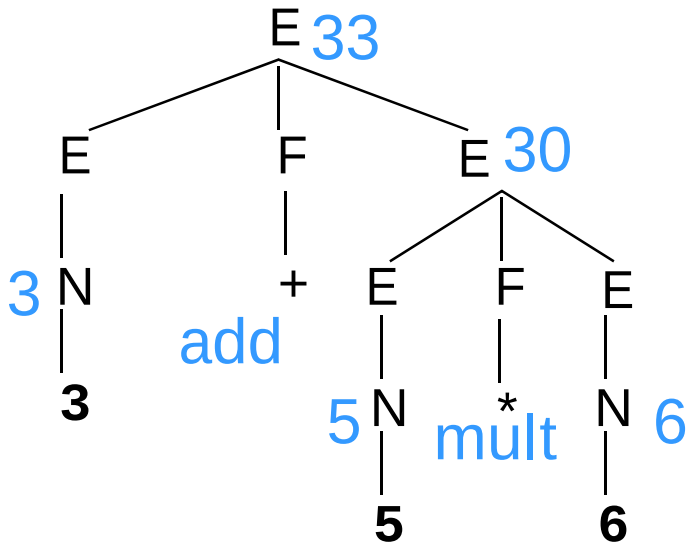




Programming Language Interpreter

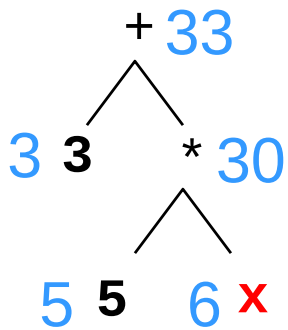
- What is meaning of 3+5*6?
- First parse it into 3+(5*6)
- Now give a meaning to each node in the tree (bottom-up)

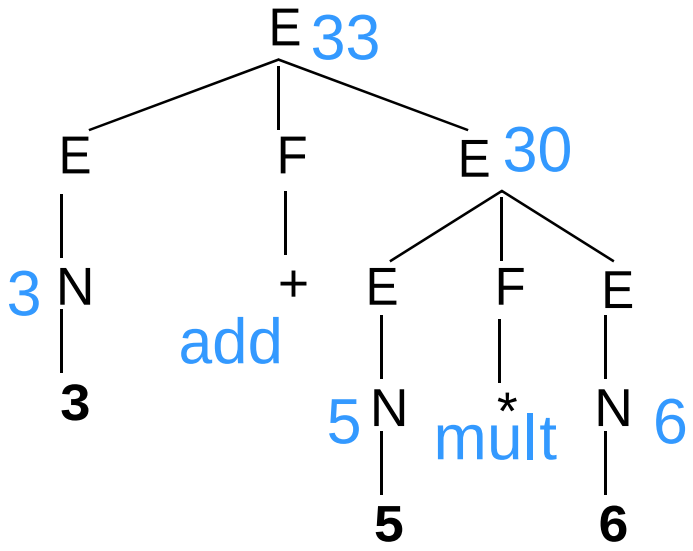




Interpreting in an Environment

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





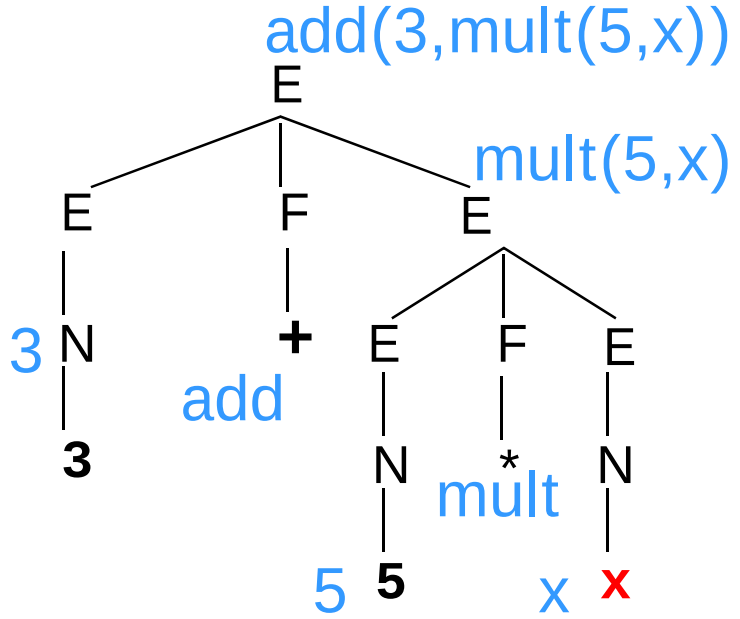
Compiling

- 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

- Be able to translate (a compiler is a translator ...)
  - Good definition? Depends on target language.
  - English to English? bah humbug!
  - English to French? reasonable
  - English to Chinese? requires deeper understanding
  - English to logic? deepest - the definition we'll use!
    - all humans are mortal = $\forall x\ [human(x) \Rightarrow mortal(x)]$

- Assume we have logic-manipulating rules that then tell us how to act, draw conclusions, answer questions ...

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 a statement if we can determine its truth
  - If you can easily determine whether it’s true, why did anyone bother telling it to you?
  - Comparable notion for understanding NP is to identify what it refers to. Useful, but what if it’s out of sight?

What Counts as Understanding? some notions

- We understand statement if we know how to determine its truth (in principle!)
  - Compile it into a procedure for checking truth against the world
    - "All owls in outer space are bachelors"

```
for every object
  if x is a owl
    if location(x) ∈ outerspace
      if x is not a bachelor
        return false
return true
```

meaning

    - What if you don't have an flying robot? (Write the code anyway)
    - How do you identify owls and bachelors? (Assume library calls)
    - What if space is infinite, so the procedure doesn't halt? Same problem for "All prime integers ..." (You won't actually run it)

What Counts as Understanding? some notions

- We understand statement if we know how one could (in principle) determine its truth
  - Compile it into a procedure that checks truth against the world
  - Better: Compile it into a mathematical formula
    - ∀x owl(x) ^ outerspace(x) ➔ bachelor(x)
    - Now you don't have to worry about running it
    - Either true or false in the world: a mathematical question!
      - Statement claims that the world is such that this statement is true.
      - Auden (1956): "A sentence uttered makes a world appear
Where all things happen as it says they do."
    - But does this help? Can you check math against the real world?
      - What are the x's that ∀x ranges over? Which ones make owl(x) true?
    - Model the world by an infinite collection of facts and entities
      - Wittgenstein (1921): "The world is all that is the case. The world is the totality of facts, not of things."

What Counts as Understanding? some notions

- We understand statement if we know how one could (in principle) determine its truth
  - Compile it into a procedure that checks truth against the world
  - Better: Compile it into a mathematical formula
    - ∀x owl(x) ^ outerspace(x) ➔ bachelor(x)
  - Equivalently, be able to derive all logical consequences
    - What else is true in every world where this statement is true?
      - Necessary conditions – let us draw other conclusions from sentence
    - And what is false in every world where this sentence is false
      - Sufficient conditions – let us conclude the sentence from other facts
    - "Recognizing textual entailment" is an NLP task (∃ competitions!)
      - John ate pizza. Can you conclude that John opened his mouth?
    - Knowing consequences lets you answer questions (in principle):
      - Easy: John ate pizza. What was eaten by John?
      - Hard: White's first move is P-Q4. Can Black checkmate?

Lecture Plan

- Today:
  - First, intro to λ-calculus and logical notation
  - Let's look at some sentences and phrases
    - What logical representations would be reasonable?
- Tomorrow:
  - How can we build those representations?
- Another course (AI):
  - How can we reason with those representations?

Logic: Some Preliminaries

Three major kinds of objects

1. Booleans
   - Roughly, the semantic values of sentences
2. Entities
   - Values of NPs, e.g., objects like this slide
   - Maybe also other types of entities, like times
3. 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

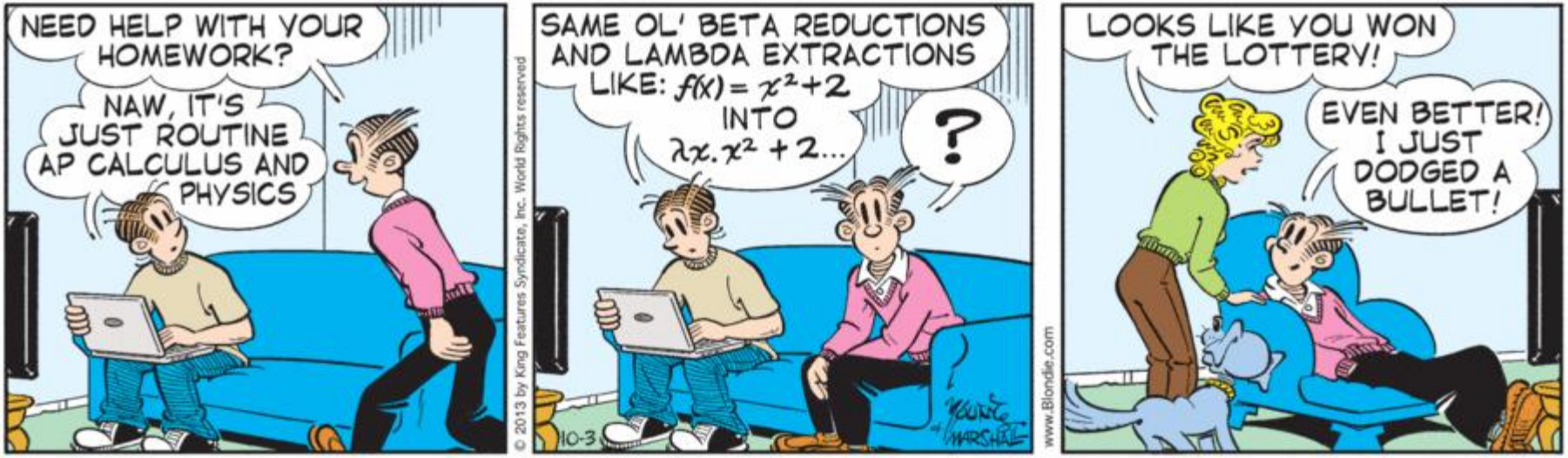
Logic: Lambda Terms

- Lambda terms:
  - Let square = λp p*p
  - Then square(3) = (λp p*p)(3) = 3*3
  - Note: square(x) isn't a function! It's just the value x*x.
  - But ᵃx square(x) = λx x*x = λp p*p = square

    (proving that these functions are equal – and indeed they are, as they act the same on all arguments: what is (λx square(x))(y)?)

  - Let even = λp (p mod 2 == 0) a predicate: returns true/false
  - even(x) is true if x is even
  - How about even(square(x))?
  - λx even(square(x)) is true of numbers with even squares
    - Just apply rules to get λx (even(x*x)) = λx (x*x mod 2 == 0)
    - This happens to denote the same predicate as even does

Lambda calculus vs. AP calculus



Blondie, Oct. 3, 2013

Logic: Multiple Arguments

- Lambda terms denote functions of 1 argument
- But how about functions like multiplication?
- We can fake multiple arguments ["currying"]

- Define times as λx λy (x*y)
- Claim that times(5)(6) is 30
  - times(5) = (λx λy x*y) (5) = λy 5*y
    - If this function weren't anonymous, what would we call it?
  - times(5)(6) = (λy 5*y)(6) = 5*6 = 30

Logic: Multiple Arguments

- All lambda terms have one argument
- But we can fake multiple arguments ...
- We'll write "times(5,6)" as syntactic sugar for times(5)(6) or perhaps times(6)(5)

  Notation varies; doesn't matter as long as you're consistent

  - times(5,6) = times(5)(6)
    = (λx λy x*y) (5)(6) = (λy 5*y)(6) = 5*6 = 30
- So we can always get away with 1-arg functions ...
  - ... which might return a function to take the next argument. Whoa.
- Remember: square can be written as λx square(x)
  - And now times can be written as λx λy times(x,y)

Grounding out

- So what does times actually mean???
  - times was defined in terms of * .
  - But does * mean multiplication?
  - If * was defined as another lambda term, then
    times(5,6) = *(5,6) = (blah blah blah)(5)(6)
    but where do we stop?
- Similarly, what does bachelor mean?
  - Maybe we defined
    bachelor = λx (male(x) and not married(x))
    but how is male defined?
- Same problem as in programming languages and dictionaries.

Grounding out

- As in programming languages: something has to be built in.
- Don't keep doing substitutions forever!
  - Eventually we have to "ground out" in a primitive term
  - Primitive terms are bound to object code
- Maybe *(5,6) is handled by the hardware
- Maybe male(John) is too [visual cortex]
- What code is executed by loves(John, Mary)?

Logic: Interesting Constants

- Thus, have "constants" that name some of the entities and functions (e.g., *):
  - BarackObama - an entity
  - red – a predicate on entities
    - holds of just the red entities: red(x) is true if x is red!
  - loves – a predicate on 2 entities
    - loves(BarackObama, MichelleObama)
    - Question: What does loves(MichelleObama) denote?
- Can define other named objects from the constants
- Can define a meaning for each English word from the named objects
- Meaning of each English word is defined in terms of the constants [maybe indirectly]

Logic: Connectives & Quantifiers

- p OR q (= p ∨ q) "p or q"
- p AND q (= p ∧ q = p,q) "p and q"
- NOT p (= ¬p = ~p) "not p"
- p ⇒ q "if p then q"
- ∀x "for all x"
- ∃x "there exists x"
- "all pigs are big"
  - ?x pig(x) ⇒ big(x) "for all x, if pig(x), then big(x)"
- "some pig is big"
  - Ax pig(x) AND big(x)

    there exists some x such that pig(x) AND big(x)
- "most pigs are big" ??

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 ∀x pig(x) ⇒ big(x))
  - exists(pig,big) (equivalent to ∃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"

Model Theory

- Equivalent notions:
  - A “world” (semantics)
  - A “outcome” (probability)
  - A “model” (math)
- All of these specify everything

Random Variables: What is "variable" in "p(variable=value)"?

Answer: variable is really a function of Outcome

- p(x_1=h) * p(x_2=o | x_1=h) * …
  - Outcome is a sequence of letters
  - x_2 is the second letter in the sequence
- p(number of heads=2) or just p(H=2) or p(2)
  - Outcome is a sequence of 3 coin flips
  - H is the number of heads
- p(weather's clear=true) or just p(weather's clear)
  - Outcome is a race
  - weather's clear is true or false

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(BarackObama, MichelleObama)
  - 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: ∃g goldfish(g) AND swallowed(Gilly, g)
- Or using one of our quantifier predicates:
  - exists(λg goldfish(g), λg swallowed(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: exists(goldfish, λg swallowed(Gilly,g))
- Improve to use tense:
  - Instead of the 2-arg predicate swallowed(Gilly,g)
    try a 3-arg version swallow(t,Gilly,g) where t is a time
  - Now we can write:
    ∃t past(t) AND exists(goldfish, λg swallow(t,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: ∃t past(t) AND exists(goldfish, swallow(t,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
  - ∃e past(e), act(e,swallowing), swallower(e,Gilly), exists(goldfish, swallowee(e)), exists(booth, location(e)), ...
    - As with probability notation, a comma represents AND
    - Could define past as λe ∃t before(t,now), ended-at(e,t)

"Davidsonian event variable" (after Donald Davidson, 1980)

Quantifier Order

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

∃g goldfish(g), swallowee(e,g) ∀b booth(b)⇒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.**
  - ∃woman (∀15min gives-birth-during(15min, woman))
  - ∀15min (∃woman gives-birth-during(15min, 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
  - ∃e past(e), act(e,swallowing), swallower(e,Gilly), exists(goldfish, swallowee(e)), exists(booth, location(e)), ...
- Gilly swallowed a goldfish in every booth
  - ∃e past(e), act(e,swallowing), swallower(e,Gilly), exists(goldfish, swallowee(e)), all(booth, location(e)), ...

∃g goldfish(g), swallowee(e,g) ∀b booth(b)⇒location(e,b)

- Does this mean what we'd expect??
  - It's ∃e ∀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`
  - ∃e past(e), act(e,swallowing), swallower(e,Gilly), exists(goldfish, swallowee(e)), exists(booth, location(e)), ...
- `Gilly swallowed a goldfish in every booth`
  - ∃e past(e), act(e,swallowing), swallower(e,Gilly), exists(goldfish, swallowee(e)), all(booth, λb location(e,b))

- Other reading (∀b ∃e) involves quantifier raising:
  - all(booth, λb [∃e past(e), act(e,swallowing), swallower (e,Gilly), exists(goldfish, swallowee(e)), location(e,b)])
  - "for all booths b, there was such an event in b"

Intensional Arguments

- `Willy wants a unicorn`
  - ∃e act(e,wanting), wanter(e,Willy), exists(unicorn, λu wantee(e,u))
    - "there is a particular unicorn u that Willy wants"
    - In this reading, the wantee is an individual entity
  - ∃e act(e,wanting), wanter(e,Willy), wantee(e, λu unicorn(u))
    - "Willy wants any entity u that satisfies the unicorn predicate"
    - In this reading, the wantee is a type of entity
    - Sentence doesn't claim that such an entity exists
- `Willy wants Lilly to get married`
  - ∃e present(e), act(e,wanting), wanter(e,Willy), wantee(e, λe' [act(e',marriage), marrier(e',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`
  - ∃e act(e,wanting), wanter(e,Willy), wantee(e, λu unicorn(u))
    - "Willy wants anything that satisfies the unicorn predicate"
    - here the wantee is a type of entity
- Problem:
  - λg unicorn(g) is defined by the actual set of unicorns ("extension")
  - But this set is empty: λg unicorn(g) = λg FALSE = λg pegasus(g)
  - Then `wants a unicorn` = `wants a pegasus`. 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 ≠ set of pegasi

Possible Worlds

- Traditional solution involves "possible-world semantics"
  - Wittgenstein (1921): "The world is all that is the case. The world is the totality of facts, not of things."
  - Can imagine other worlds where set of unicorns ≠ set of pegasi
  - Most facts can vary according to which world w you're in:
    - loves(Barack, Michelle) → loves(w, Barack, Michelle)
    - most(λx pig(x), λx big(x)) → most(w, λx pig(w, x), λx big(w, x))

      most(pig , big) → most(w, pig(w) , big(w))
    - wants(Willy, unicorn) → wants(w, Willy, unicorn)

      wants(Willy, λu unicorn(u)) → wants(w, Willy, λw' λu unicorn(w',u))

"intension" of unicorn, not tied to current world w

Function checks in any world w' whether something is a unicorn

These criteria are the same in every world:

unicorn ≡ λw' λu (has_horn(w',u), horselike(w',u), magical(w',u), ...)

Possible Worlds: More uses

- Modals (woulda coulda shoulda)

deontic ∀ modal
- You must pay the rent
  - In all possible worlds that are "like" this world, and in which you fulfill your obligations: you do pay the rent

deontic ∃ modal
- You may pay the rent
  - In some possible world that is "like" this world, and in which you fulfill your obligations: you do pay the rent

epistemic ∀ modal *(how would you express epistemic A in English?)*
- You must have paid the rent
  - In all possible worlds that are "like" this world, and which are consistent with my observations: you paid the rent

bouletic ∃ modal
- You can pay the rent
  - In some possible world that is "like" this world, and in which you have no additional powers: you do pay the rent

… and more …
(varies by language, but always quantifies over some set of "accessible" worlds)

Possible Worlds: More uses

- Modals (woulda coulda shoulda)

deontic ∀ modal

- `You must pay the rent`
  - In ~~all possible~~ worlds that are "like" this world, and in which you fulfill your obligations: you pay the rent
- Counterfactuals
  - `If you hadn't, you'd be homeless`
    - In all possible worlds that are "like" this world, except that you didn't pay the rent: you are now homeless
- What are the "worlds that are 'like' this world"? ("accessible" worlds)
  - You don't pay rent, but otherwise change "as little as possible." (Same apartment, same eviction laws, no miracles to save you from the gutter, …)
  - But rather slippery how to figure out what those "minimum changes" are!
  - Lets's watch instant replays on the Subjunc-TV (Hofstadter, 1979):
    - "Here's what would've happened … if Palindromi hadn't stepped out of bounds"
    - "… if only it hadn't been raining" "… if only they'd been playing against Chicago"
    - "… if only they'd been playing baseball" "… if only 13 weren't prime"

Possible Worlds: More uses

- Modals (woulda coulda shoulda)

deontic ∀ modal

- You must pay the rent
  - In ~~all~~ possible worlds that are "like" this world, and in which you fulfill your obligations, you pay the rent

- Counterfactuals
  - If you hadn't, you'd probably be homeless
    - In ~~all~~ most possible worlds that are "like" this world, except that you didn't pay the rent, you are now homeless

p(homeless | didn't pay rent) > 0.5 But is this 0/0?

Traditional view is that some worlds are "accessible" and others aren't.
But reasoning about what would tend to happen if you didn't pay the rent seems to require probabilistic reasoning.
So maybe you have something like a probability distribution over worlds?

Estimate distribution from observing the world's facts and rules, but smoothed somehow? So my distribution will allocate a little probability to worlds where you didn't pay the rent and became homeless, or didn't pay the rent but moved in with your parents, etc. … even though I'm sure none of these worlds actually happened.

Control

- `Willy wants Lilly to get married`
  - ∃e present(e), act(e,wanting), wanter(e,Willy), wantee(e, λf [act(f,marriage), marrier(f,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

- Nouns and adjectives both restrict an entity's properties:
  - `expert:` λg expert(g)
  - `big fat expert:` λg big(g), fat(g), expert(g)
  - `Baltimore expert` (i.e., expert from Baltimore):
    λg Related(Baltimore, g), expert(g)

- But they sometimes first combine into compound concepts:
  - Adj+N: `bogus expert` (i.e., someone who has bogus_expertise):
    λg (bogus(expert))(g) [not λg bogus(g), expert(g) since they're not an expert!]
  - N+N: `Baltimore expert` (i.e., expert on Baltimore – different stress):
    λg (Modified-by(Baltimore, expert))(g)
  - (N+V)+ending: `dog catcher:`
    λg ∃e act(e,catching),catcher(e,g),exists(dog,catchee(e))
  - `garbage collection:`
    λe (act(e, collecting), exists(garbage,collectee(e)))

  - If we didn't make a compound concept first, things would go awry
    `law expert and dog catcher`
    `=` λg Related(law,g), expert(g), Related(dog, g), catcher(g) `**wrong**`
    `= dog expert and law catcher`

Nouns and Their Modifiers

We can argue about the details of the compound representations, e.g., how much of the semantics is explicit in the lambda-term, how much is in the semantics of individual words like bogus, and how much is shoved under the carpet into primitives like Modified-by, which are assumed to piece together a reasonable meaning using world knowledge and context.

- λg (bogus(expert))(g) ... bogus can construct a new concept
  or λg (Modified-by(bogus,expert))(g)?

- λg (Modified-by(Baltimore, expert))(g)

  or λg (Baltimore(expert))(g)?

  or λg (expert(Baltimore))(g)?

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), ∃e [past(e), act(e,swallowing), swallower(e,Gilly), swallowee(e,g)]]

Adverbs

- Lili passionately wants Billy
  - Wrong?: passionately(want(Lili,Billy)) = passionately(true)
  - Better: (passionately(want))(Lili,Billy)
  - Best: ∃e present(e), act(e,wanting), wanter(e,Lili), wantee(e, Billy), manner(e, passionate)
- Lili often stalks Billy
  - (often(stalk))(Lili,Billy)
  - many(day, λd ∃e present(e), act(e,stalking), stalker(e,Lili), stalkee(e, Billy), during(e,d))
- Lili obviously likes Billy
  - (obviously(like))(Lili,Billy) – one reading
  - obvious(like(Lili, 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))
    - or more accurately, "Let Billy like Lili!"
- Let's try to do this a little more precisely, using event variables.
- (We'll leave out the world variables.)

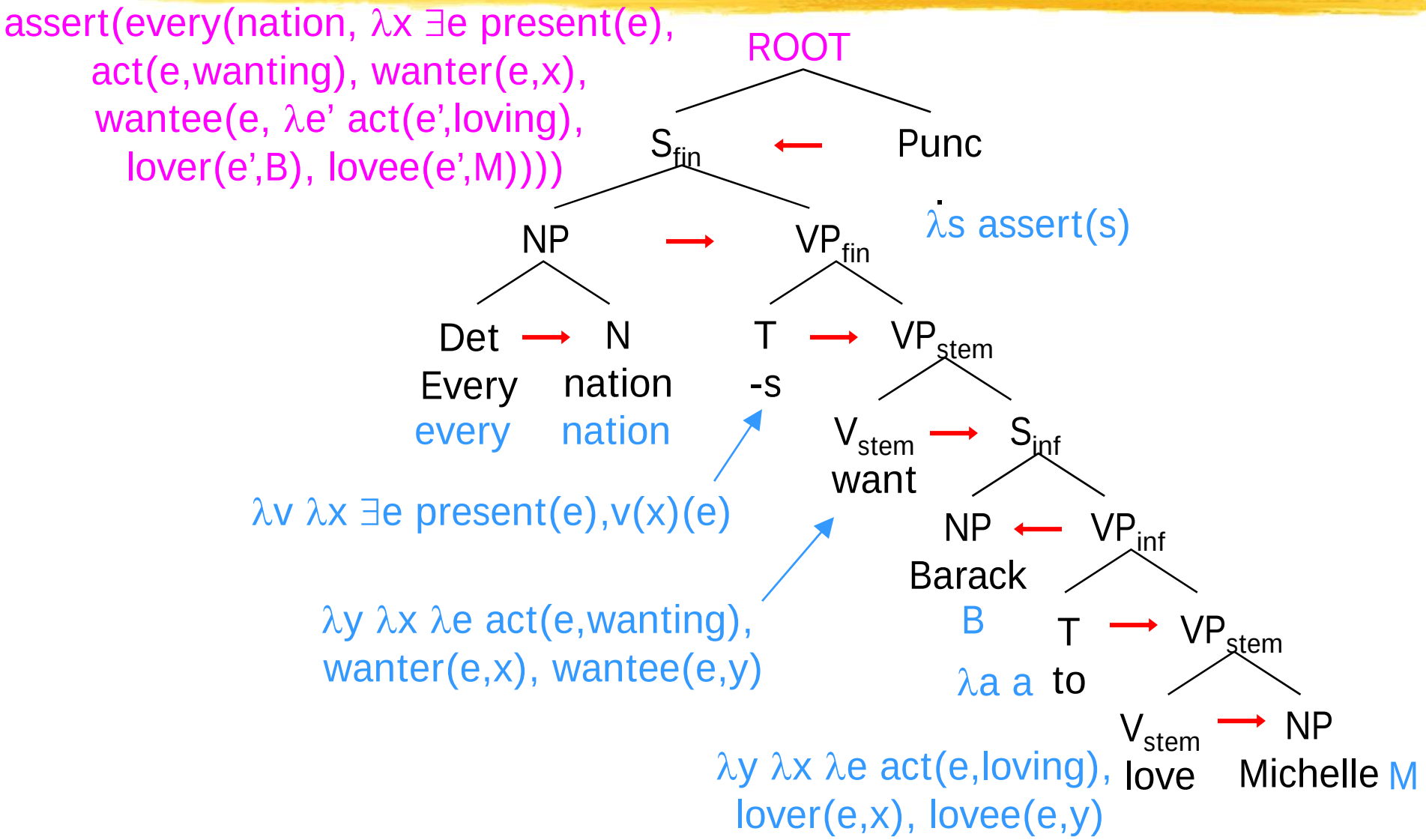
Speech Acts

- What did Gilly swallow?
  - ask(λx ∃e past(e), act(e,swallowing), swallower(e,Gilly), swallowee(e,x))
  - Argument is identical to the modifier "that Gilly swallowed"
  - Is there any common syntax?
- Eat your fish!
  - command(λf act(f,eating), eater(f,Hearer), eatee(...))
- I ate my fish.
  - assert(∃e past(e), act(e,eating), eater(f,Speaker), eatee(...))

Compositional Semantics

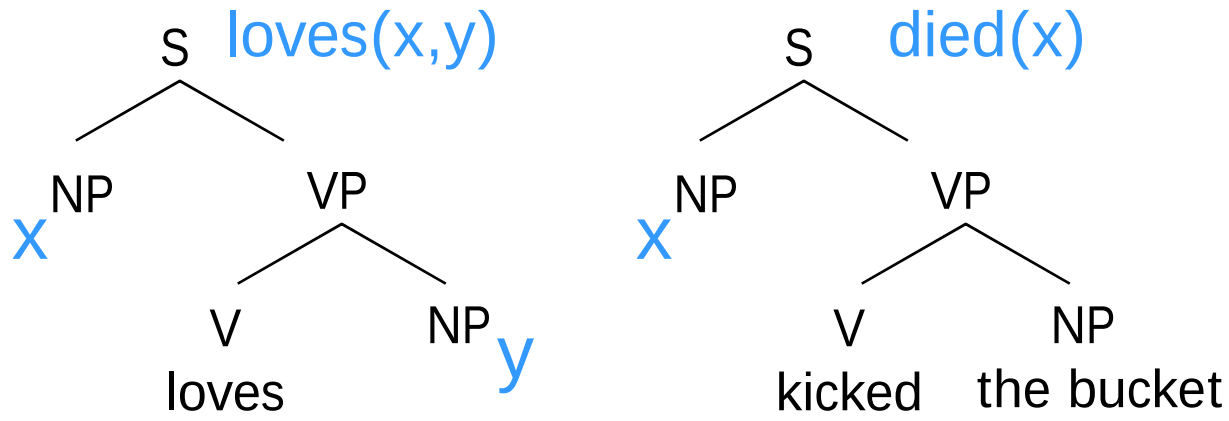
- We've discussed what semantic representations should look like.
- But how do we get them from sentences???
- First - parse to get a syntax tree.
- Second - look up the semantics for each word.
- Third - build the semantics for each constituent
  - Work from the bottom up
  - The syntax tree is a "recipe"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

Compositional Semantics



Compositional Semantics

- Add a "sem" attribute to each context-free rule
  - S → NP loves NP
  - S[sem=loves(x,y)] → NP[sem=x] loves NP[sem=y]
  - Meaning of S depends on meaning of NPs
- TAG version:

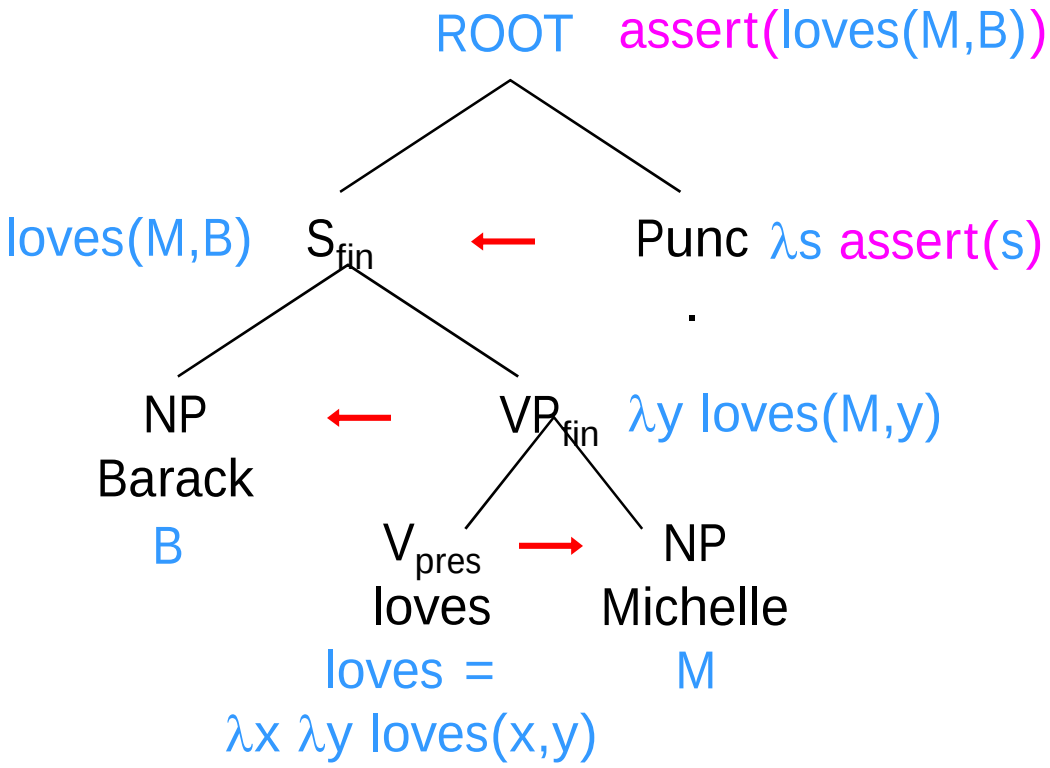


- Template filling: S[sem=showflights(x,y)] →
  I want a flight from NP[sem=x] to NP[sem=y]

Compositional Semantics

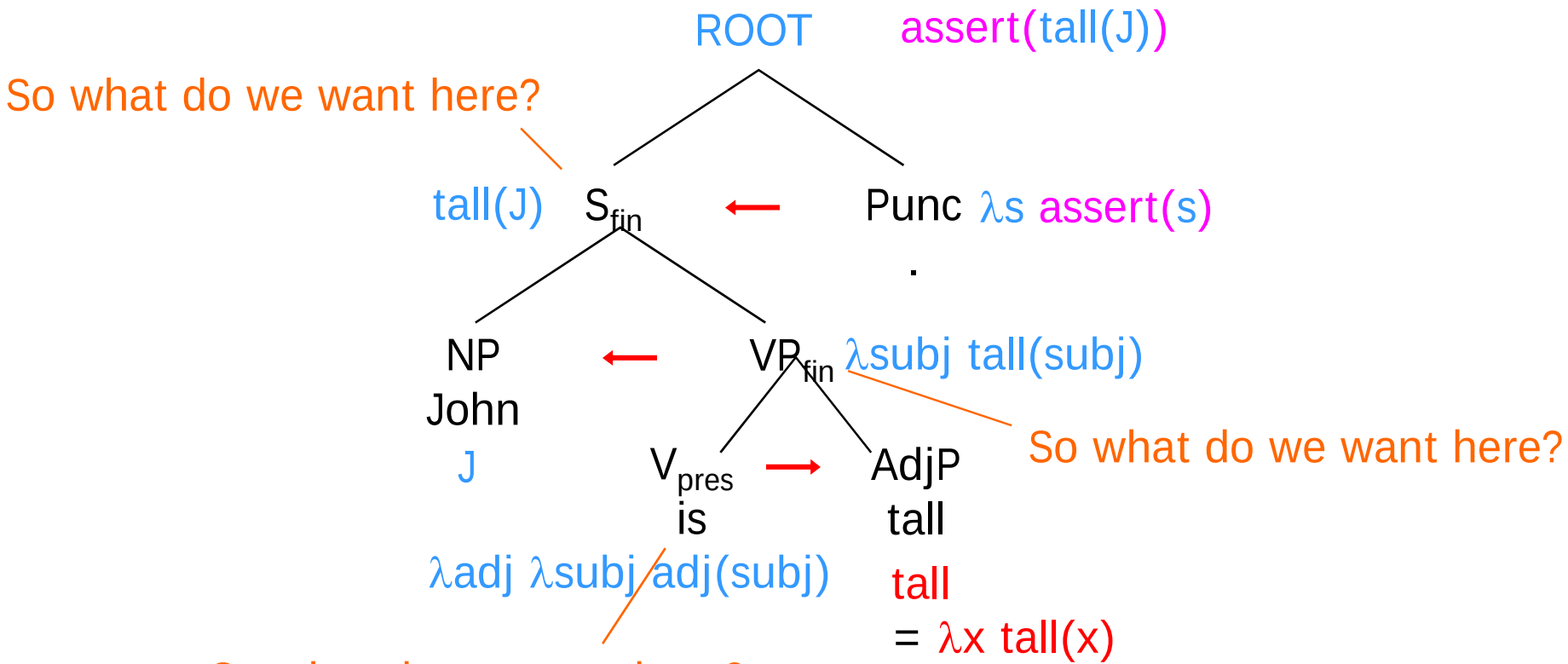
- Instead of S → NP loves NP
  - S[sem=loves(x,y)] → NP[sem=x] loves NP[sem=y]
- might want general rules like S → NP VP:
  - V[sem=loves] → loves
  - VP[sem=v(obj)] → V[sem=v] NP[sem=obj]
  - S[sem=vp(subj)] → NP[sem=subj] VP[sem=vp]
- Now `Barack loves Michelle` has sem=loves(Michelle)(Barack)
- In this style 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, just apply function to argument!
  - (version on homework will be a little less pure)

Compositional Semantics

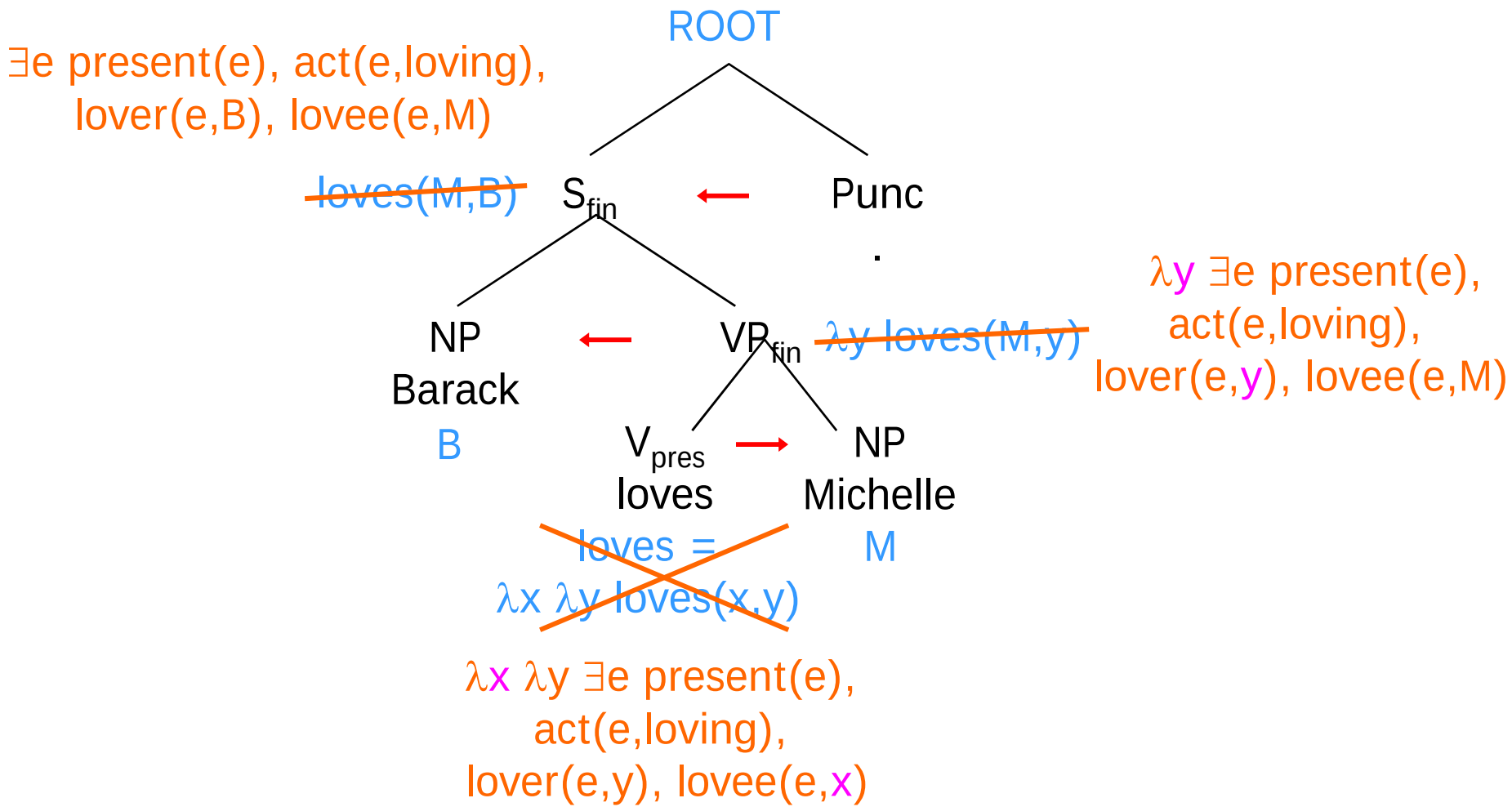


Question: Really the root meaning should be assert(loves(w_{curr},M,B))
Then what is the meaning of loves? λx λy λw loves(w,x,y)
And what is the meaning of period? λs assert(s(w_{curr}))

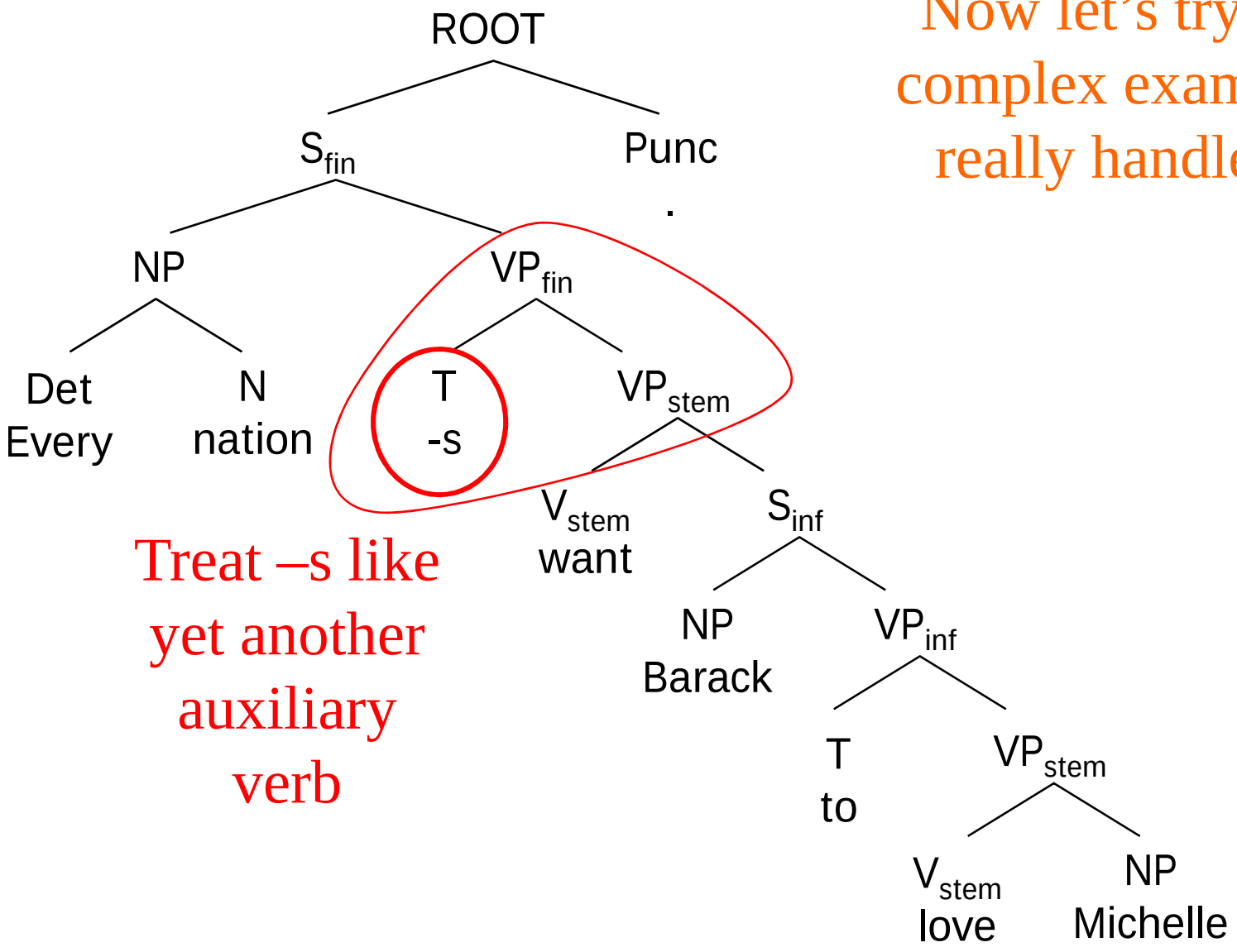
Compositional Semantics



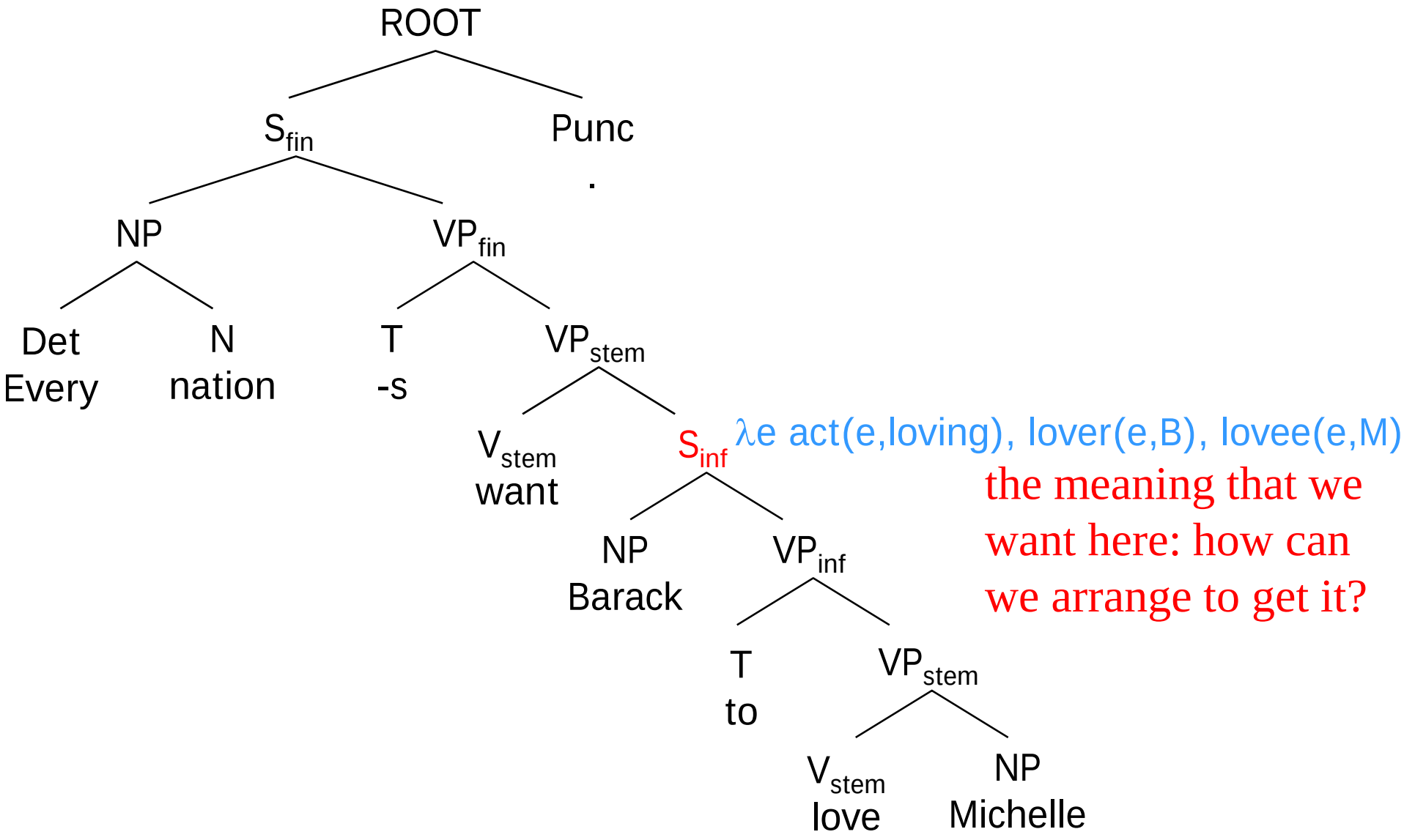
Compositional Semantics



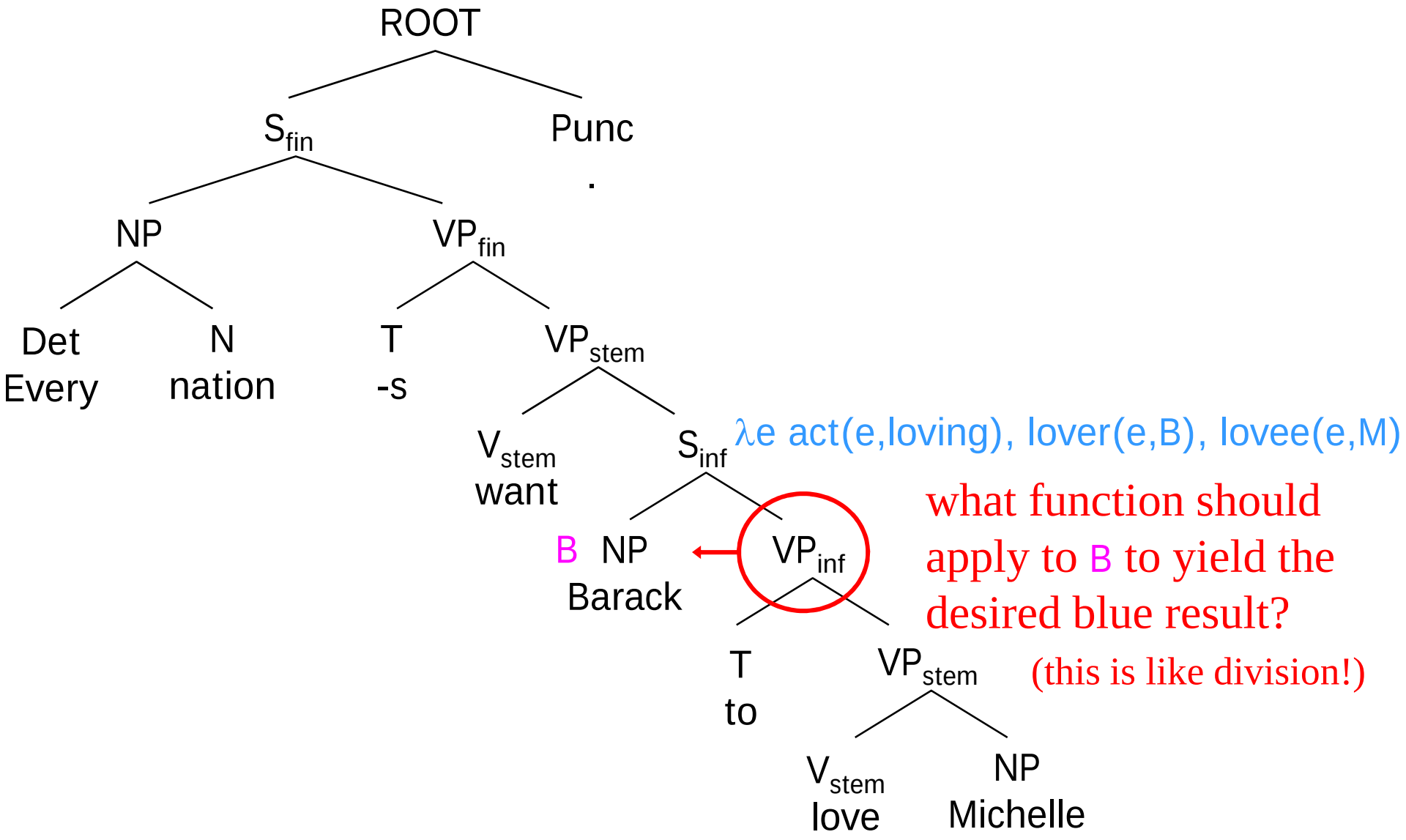
Now let's try a more complex example, and really handle tense.

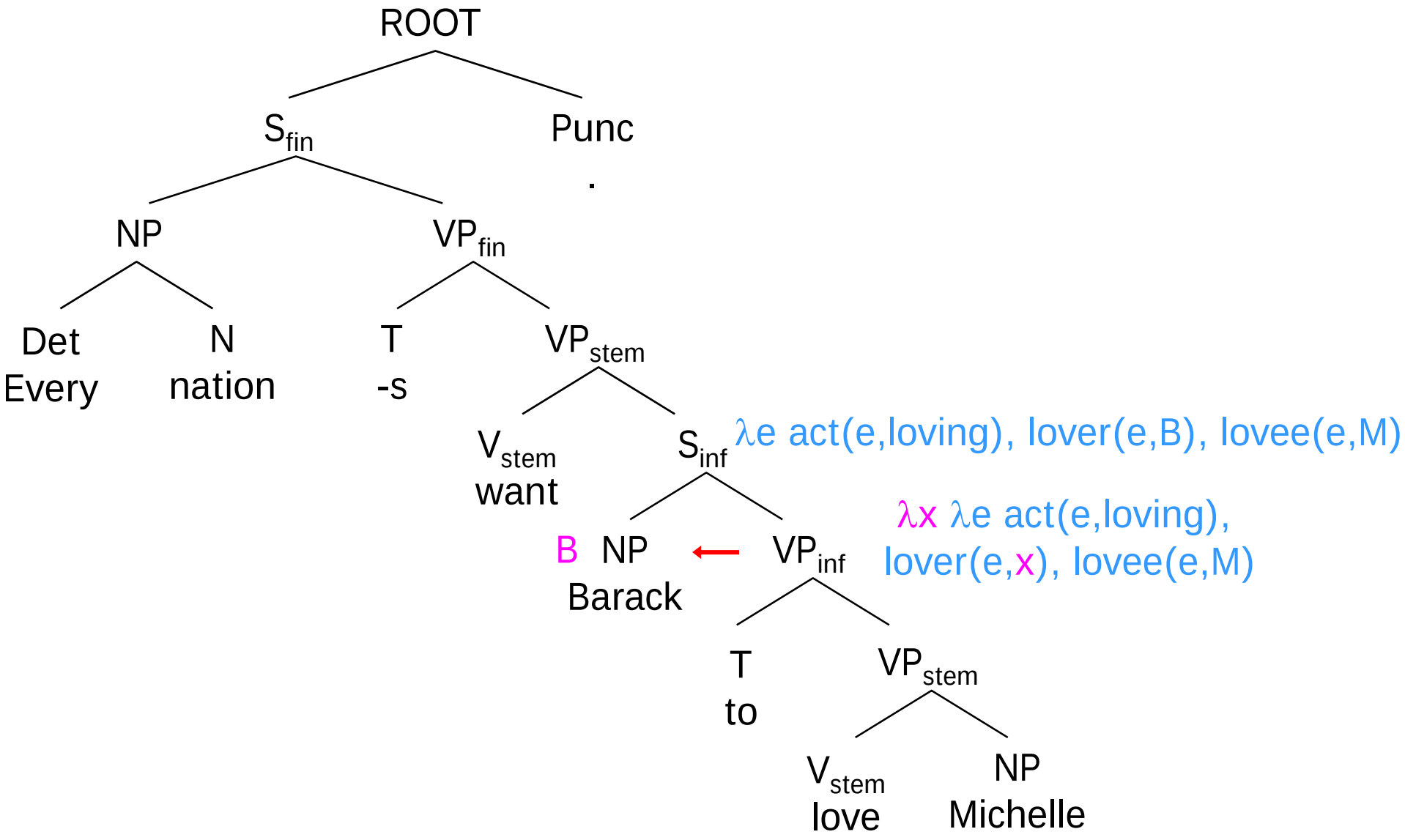


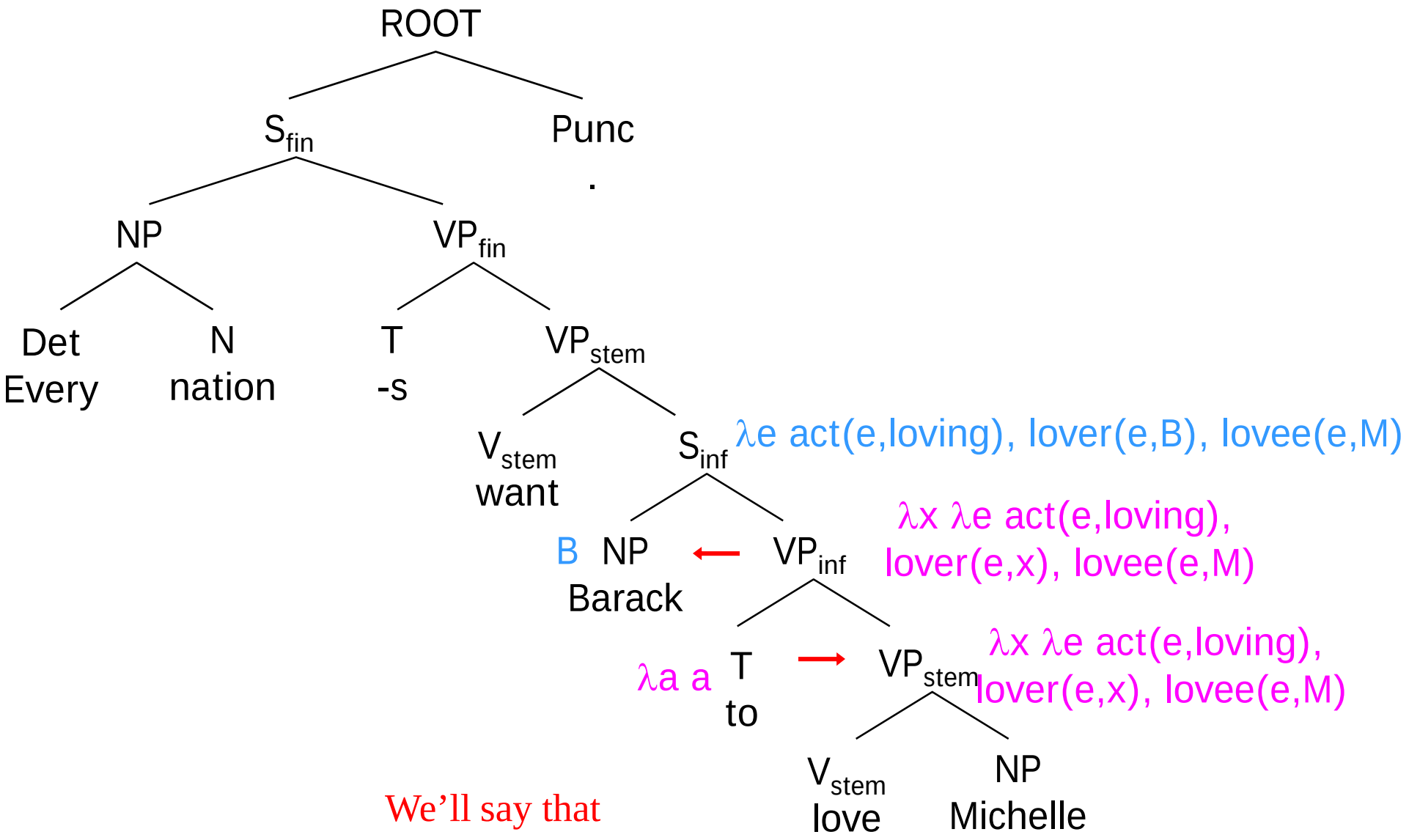
Treat –s like yet another auxiliary verb



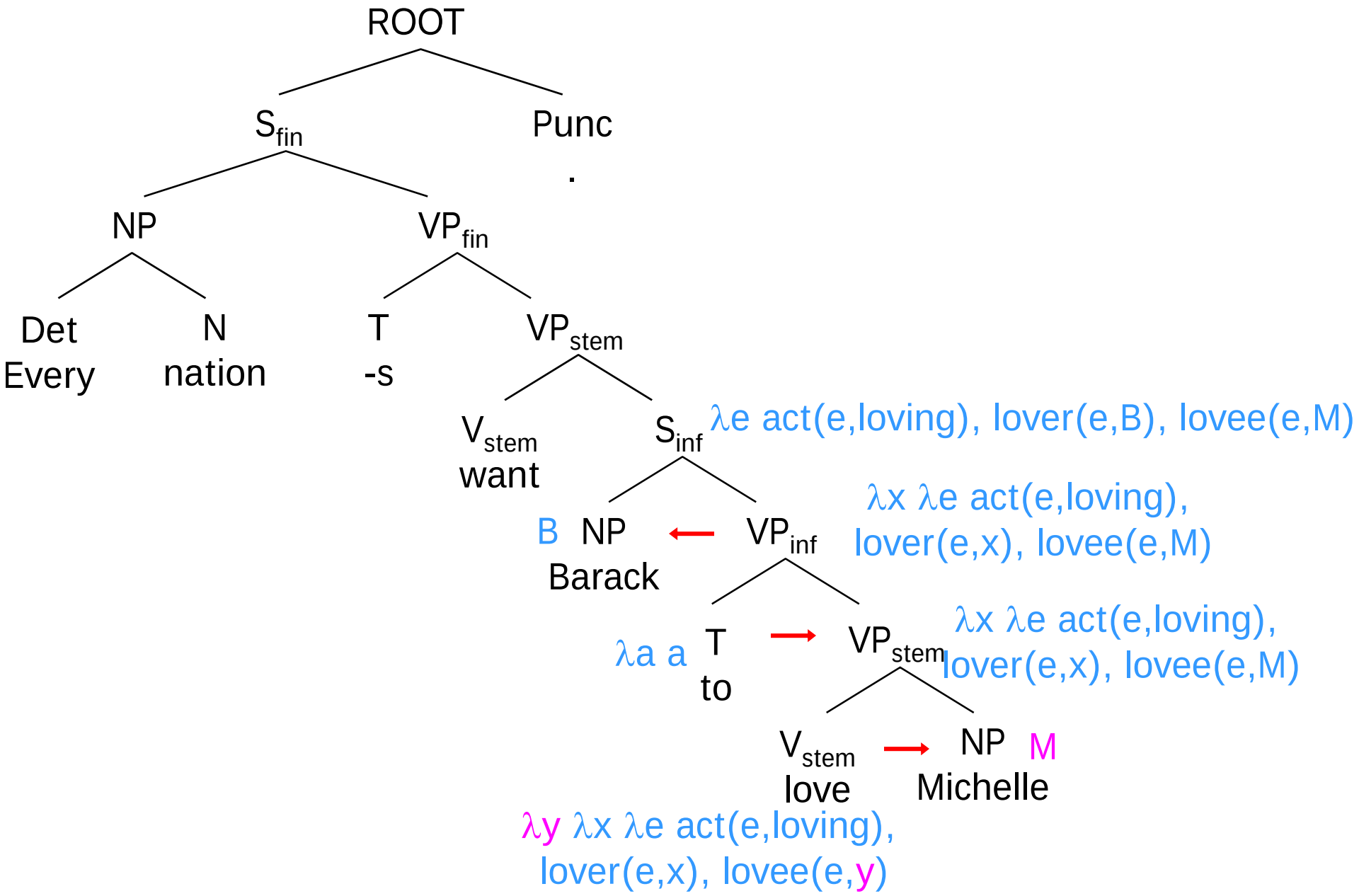
the meaning that we want here: how can we arrange to get it?

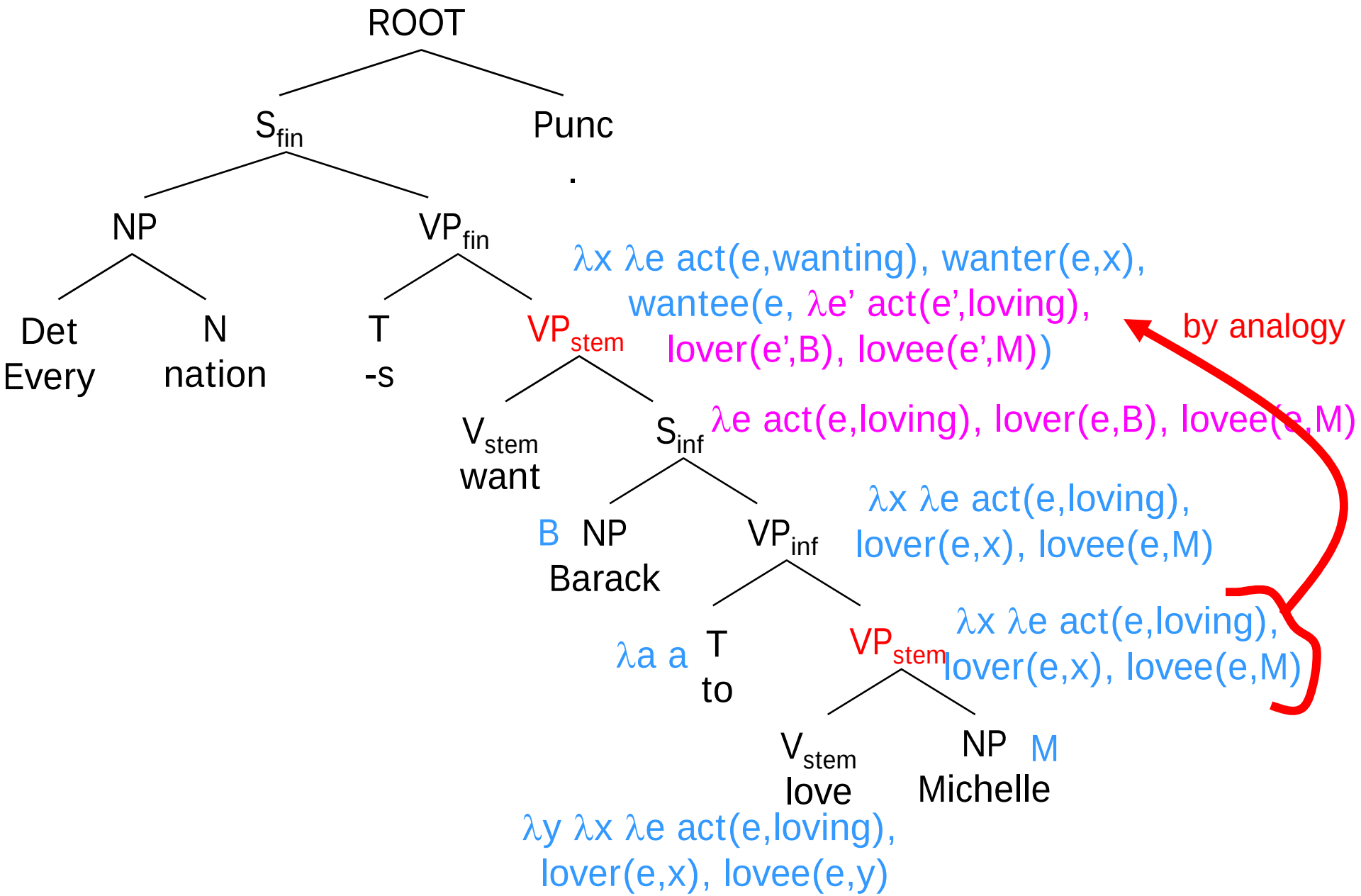

ROOT
S_{fin}
Punc
.
NP
VP_{fin}
Det
Every
N
nation
T
-s
VP_{stem}
V_{stem}
want
S_{inf}
λe act(e,loving), lover(e,B), lovee(e,M)
B NP
Barack
VP_{inf}
what function should apply to B to yield the desired blue result?
T
to
VP_{stem}
(this is like division!)
V_{stem}
love
NP
Michelle

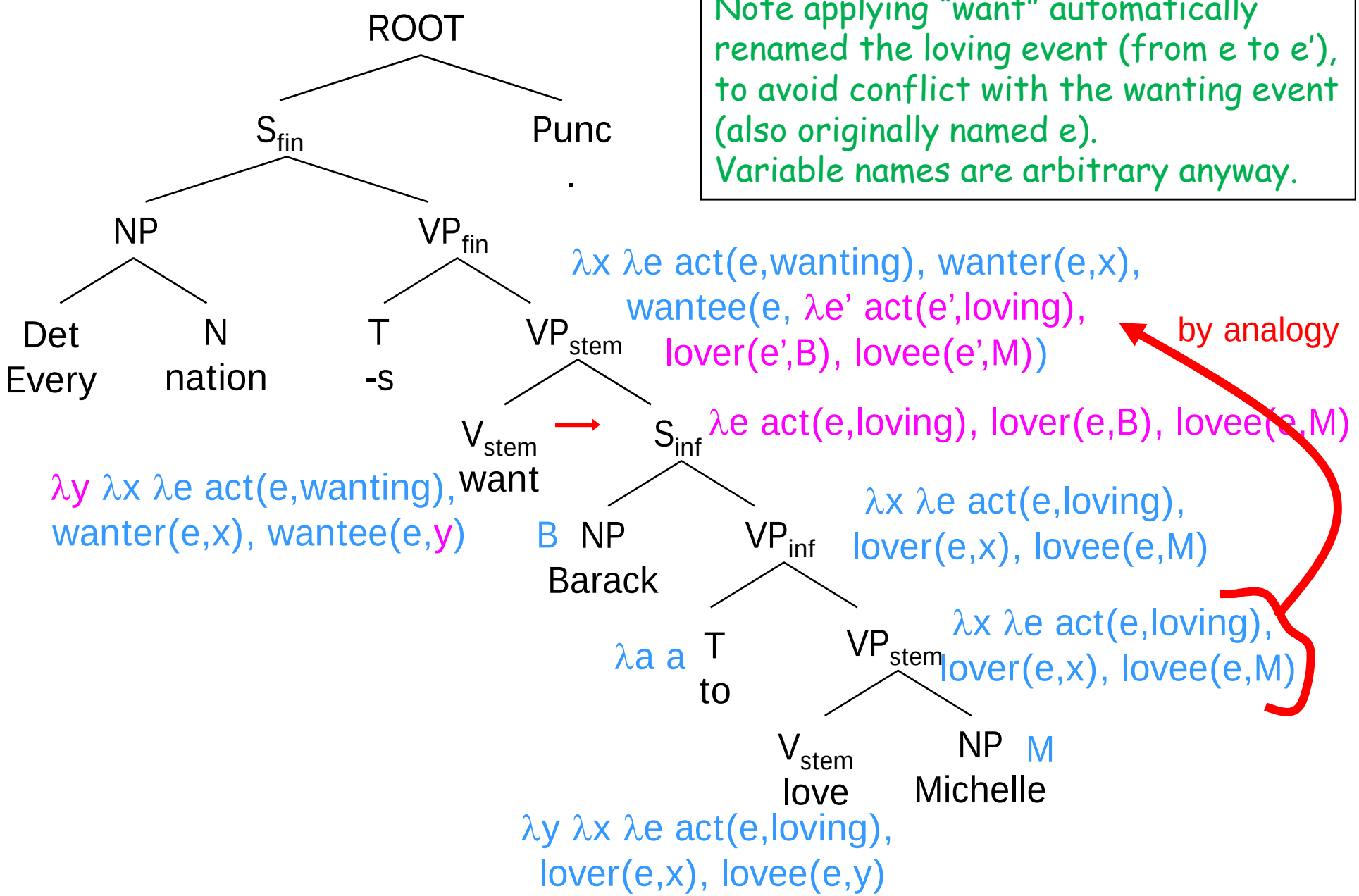

ROOT
S_{fin}
Punc
.
NP
VP_{fin}
Det
Every
N
nation
T
-s
VP_{stem}
V_{stem}
want
S_{inf}
λe act(e,loving), lover(e,B), lovee(e,M)
B NP
Barack
VP_{inf}
λx λe act(e,loving), lover(e,x), lovee(e,M)
T
to
VP_{stem}
V_{stem}
love
NP
Michelle

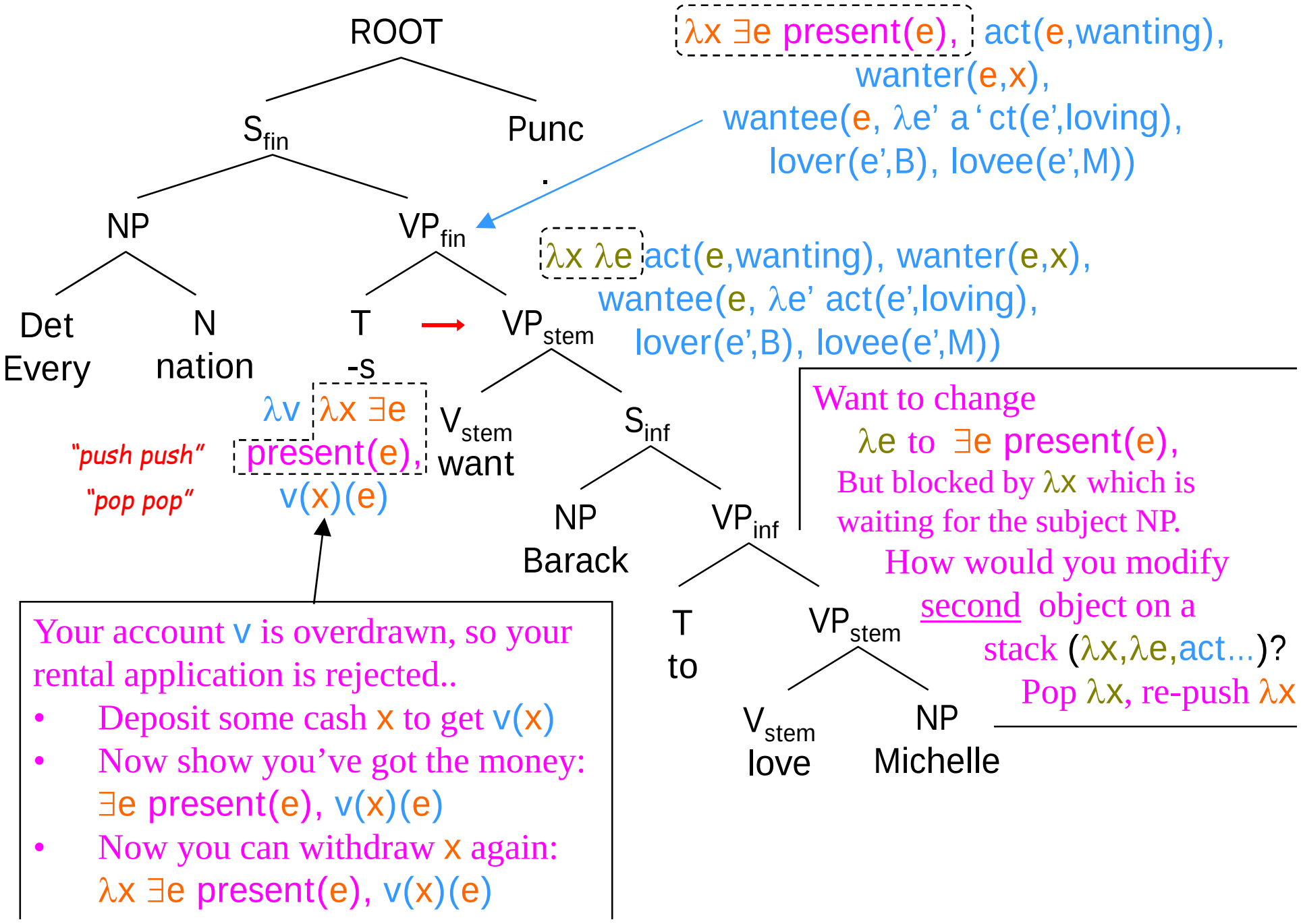


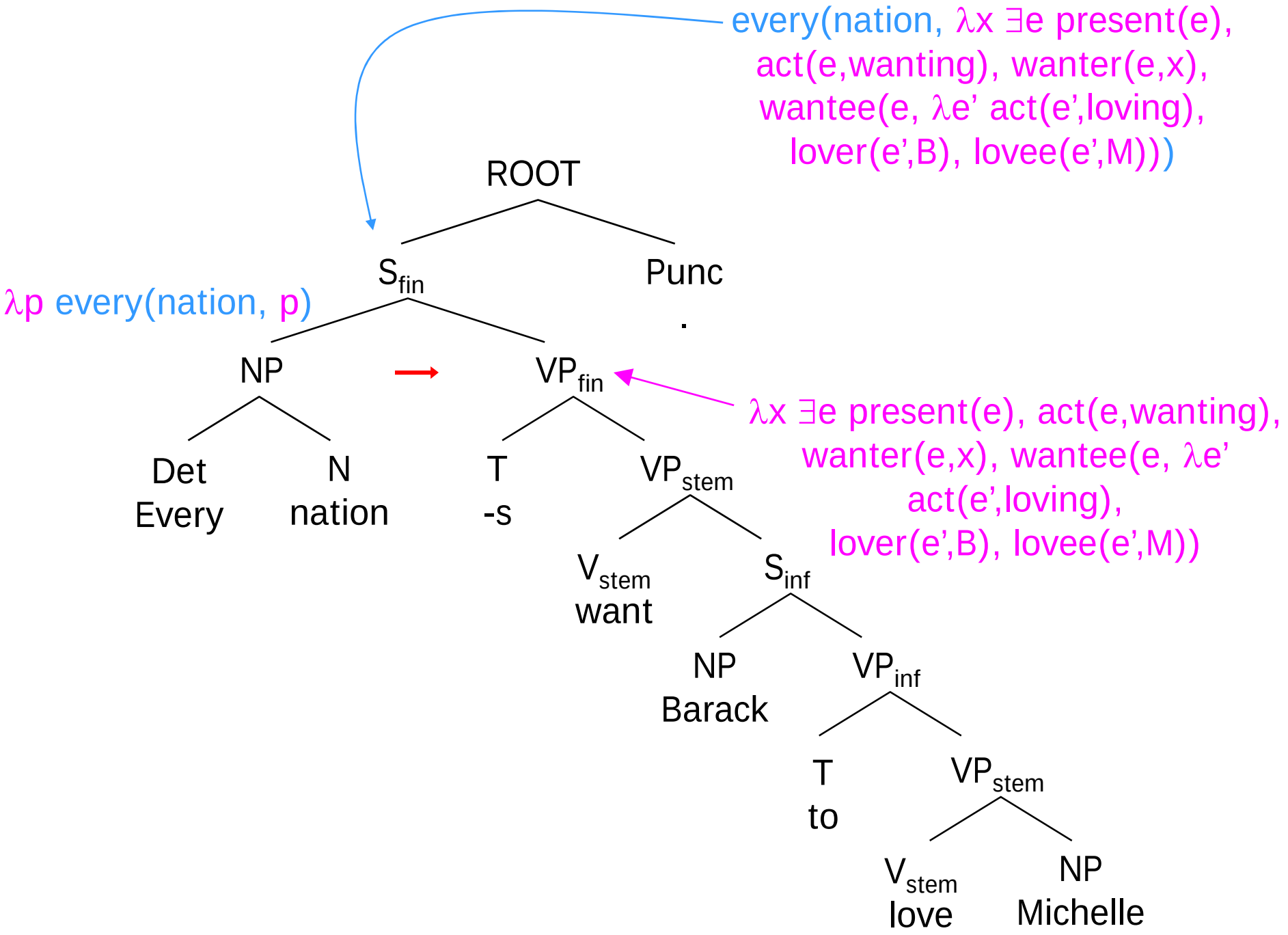
We'll say that
"to" is just a bit of syntax that
changes a VP_{stem} to a VP_{inf}
with the same meaning.

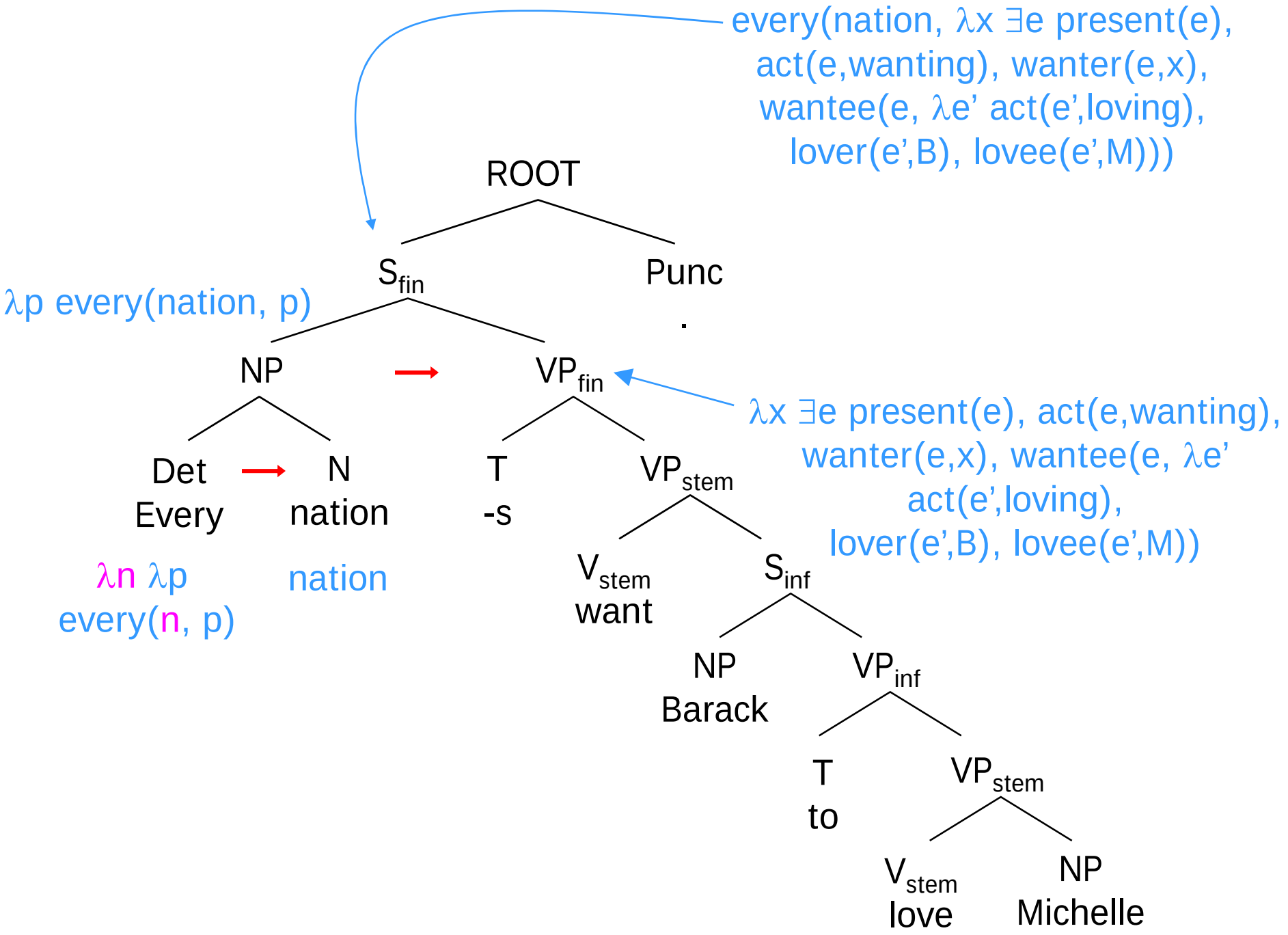

ROOT
S_{fin}
Punc
.
NP
VP_{fin}
Det
Every
N
nation
T
-s
VP_{stem}
V_{stem}
want
S_{inf}
λe act(e,loving), lover(e,B), lovee(e,M)
B NP
Barack
VP_{inf}
λx λe act(e,loving), lover(e,x), lovee(e,M)
λa a T
to
VP_{stem}
λx λe act(e,loving), lover(e,x), lovee(e,M)
V_{stem}
love
NP M
Michelle
λy λx λe act(e,loving), lover(e,x), lovee(e,y)

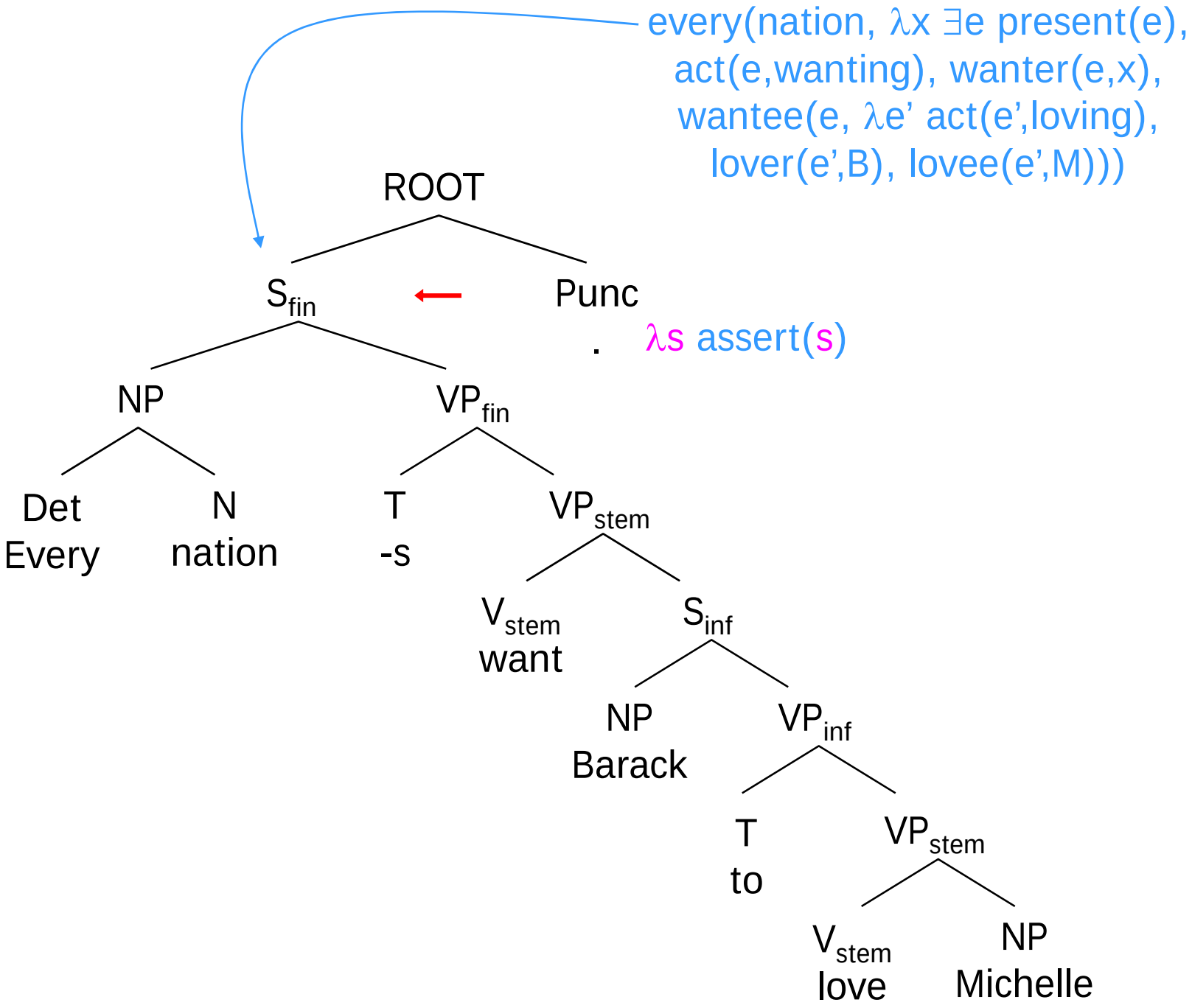

ROOT
S_{fin}
Punc
.
NP
VP_{fin}
Det
Every
N
nation
T
-s
VP_{stem}
V_{stem}
want
S_{inf}
B NP
Barack
VP_{inf}
λa a T
to
VP_{stem}
V_{stem}
love
NP M
Michelle
λy λx λe act(e,loving), lover(e,x), lovee(e,y)
λx λe act(e,loving), lover(e,x), lovee(e,M)
λx λe act(e,loving), lover(e,x), lovee(e,M)
λx λe act(e,wanting), wanter(e,x), wantee(e, λe' act(e',loving), lover(e',B), lovee(e',M))
λe act(e,loving), lover(e,B), lovee(e,M)
by analogy


ROOT
S_{fin}
Punc
.
NP
VP_{fin}
Det
Every
N
nation
T
-s
VP_{stem}
V_{stem}
want
S_{inf}
B
NP
Barack
VP_{inf}
λa a
T
to
VP_{stem}
V_{stem}
love
NP
Michelle
M
Note applying "want" automatically renamed the loving event (from e to e'), to avoid conflict with the wanting event (also originally named e).
Variable names are arbitrary anyway.
λx λe act(e,wanting), wanter(e,x), wantee(e, λe' act(e',loving), lover(e',B), lovee(e',M))
by analogy
λe act(e,loving), lover(e,B), lovee(e,M)
λy λx λe act(e,wanting), wanter(e,x), wantee(e,y)
λx λe act(e,loving), lover(e,x), lovee(e,M)
λx λe act(e,loving), lover(e,x), lovee(e,M)
λy λx λe act(e,loving), lover(e,x), lovee(e,y)

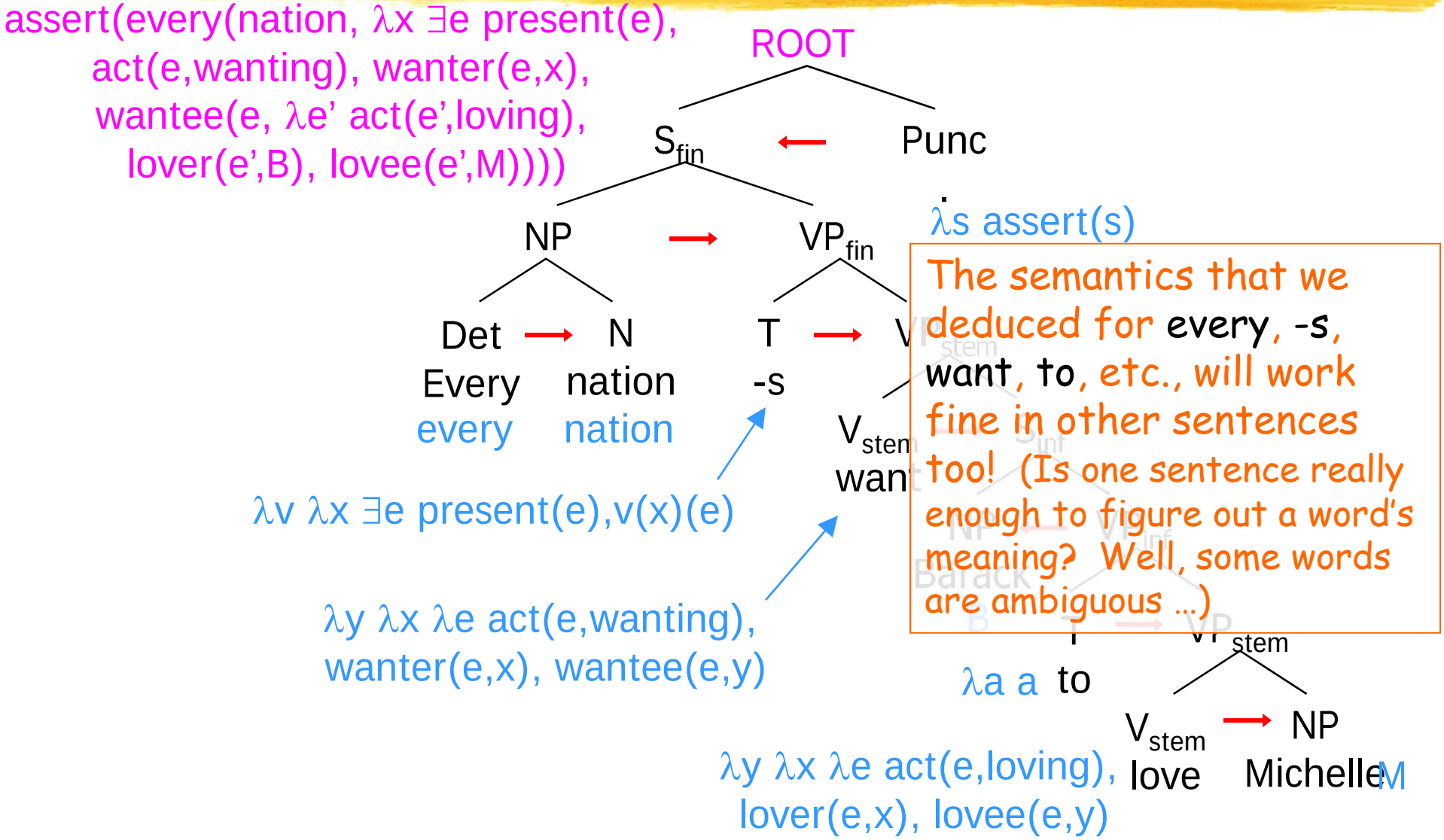

ROOT
S_{fin}
Punc
.
NP
VP_{fin}
Det
Every
N
nation
T
-s
VP_{stem}
λv λx ∃e present(e), v(x)(e)
"push push"
"pop pop"
V_{stem}
want
S_{inf}
NP
Barack
VP_{inf}
T
to
VP_{stem}
V_{stem}
love
NP
Michelle
λx ∃e present(e), act(e,wanting), wanter(e,x), wantee(e, λe' a'ct(e',loving), lover(e',B), lovee(e',M))
λx λe act(e,wanting), wanter(e,x), wantee(e, λe' act(e',loving), lover(e',B), lovee(e',M))
Want to change λe to ∃e present(e),
But blocked by λx which is waiting for the subject NP.
How would you modify second object on a stack (λx,λe,act...)?
Pop λx, re-push λx
Your account v is overdrawn, so your rental application is rejected..
• Deposit some cash x to get v(x)
• Now show you've got the money: ∃e present(e), v(x)(e)
• Now you can withdraw x again: λx ∃e present(e), v(x)(e)


every(nation, λx ∃e present(e), act(e,wanting), wanter(e,x), wantee(e, λe' act(e',loving), lover(e',B), lovee(e',M)))
ROOT
S fin
Punc
.
λp every(nation, p)
NP
VP fin
λx ∃e present(e), act(e,wanting), wanter(e,x), wantee(e, λe' act(e',loving), lover(e',B), lovee(e',M))
Det
Every
N
nation
T
-s
VP stem
V stem
want
S inf
NP
Barack
VP inf
T
to
VP stem
V stem
love
NP
Michelle


every(nation, λx ∃e present(e), act(e,wanting), wanter(e,x), wantee(e, λe' act(e',loving), lover(e',B), lovee(e',M)))
ROOT
S fin
Punc
.
λp every(nation, p)
NP
VP fin
λx ∃e present(e), act(e,wanting), wanter(e,x), wantee(e, λe' act(e',loving), lover(e',B), lovee(e',M))
Det
Every
N
nation
T
-s
VP stem
λn λp every(n, p)
nation
V stem
want
S inf
NP
Barack
VP inf
T
to
VP stem
V stem
love
NP
Michelle


every(nation, λx ∃e present(e), act(e,wanting), wanter(e,x), wantee(e, λe' act(e',loving), lover(e',B), lovee(e',M)))
ROOT
S fin
Punc
.
λs assert(s)
NP
VP fin
Det
Every
N
nation
T
-s
VP stem
V stem
want
S inf
NP
Barack
VP inf
T
to
VP stem
V stem
love
NP
Michelle

In Summary: From the Words



Other Fun Semantic Stuff: A Few Much-Studied Miscellany

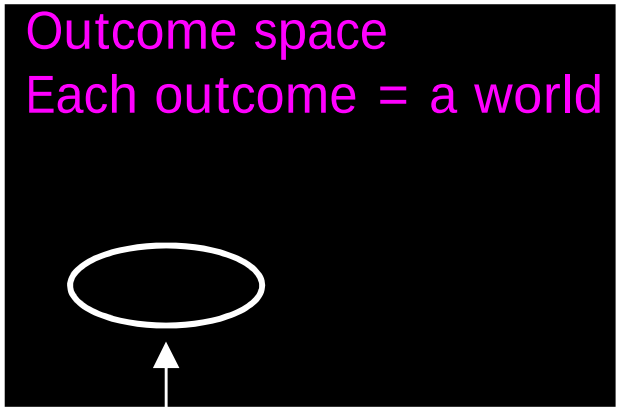
- Temporal logic
  - Gilly had swallowed eight goldfish before Milly reached the bowl
  - Billy said Jilly was pregnant
  - Billy said, "Jilly is pregnant."
- Generics
  - Typhoons arise in the Pacific
  - Children must be carried
- Presuppositions
  - The king of France is bald.
  - Have you stopped beating your wife?
- 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.

Pragmatics

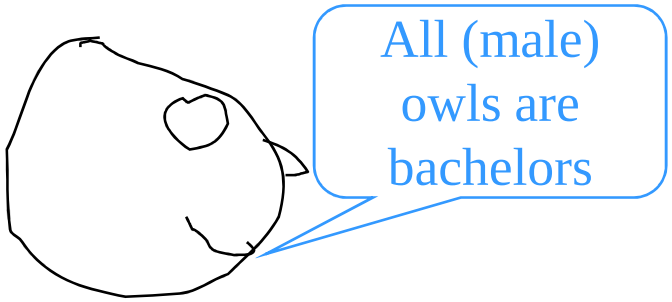
- I saw this sign in Seattle.
- I'd been in violation of it for approximately my entire adult life.

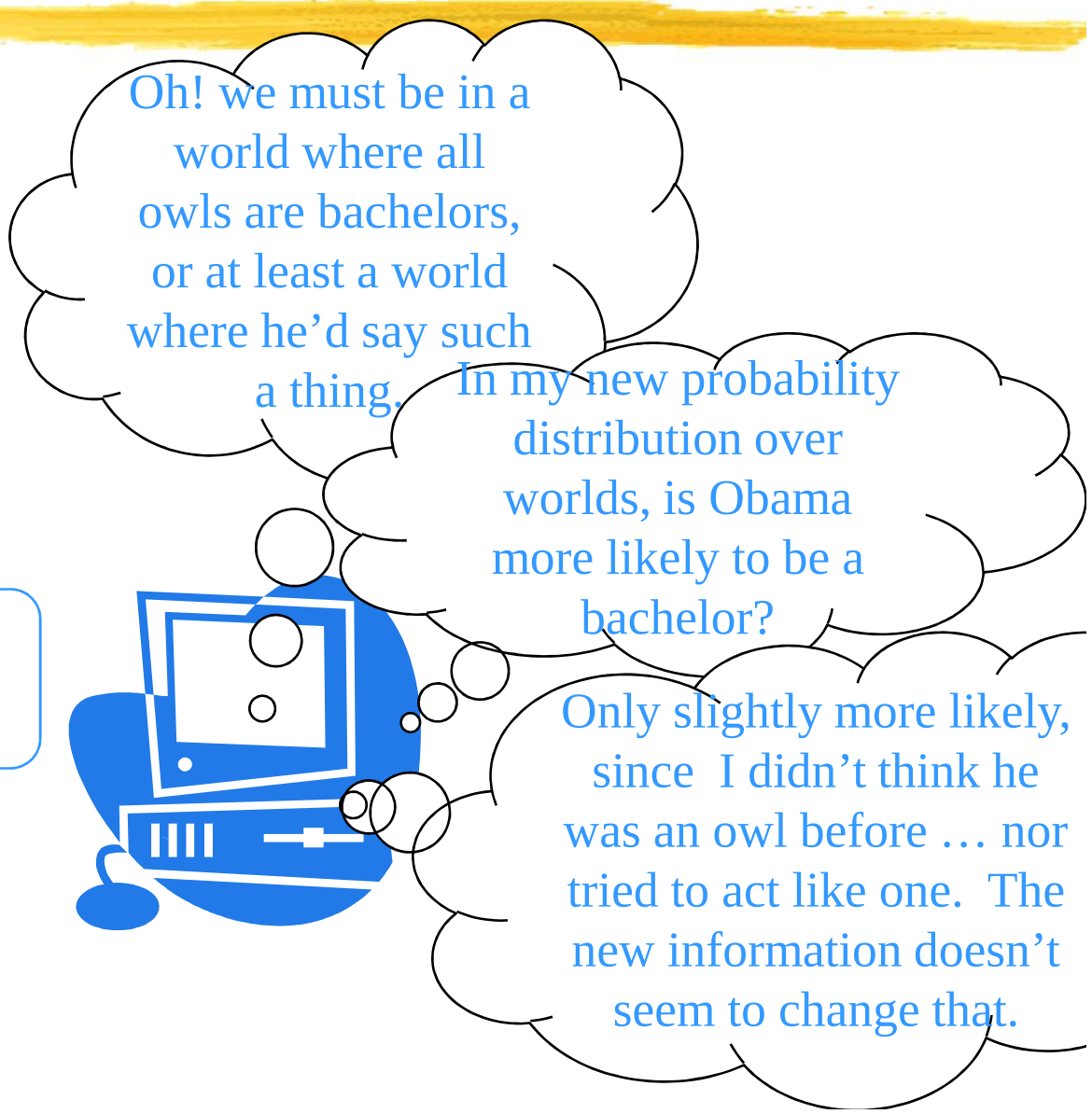
- But only technically.
  - Pragmatics is the study how we look past the literal meaning.
  - What conclusions should I actually draw from the fact that you said something?
    - Should I use Bayes' Theorem?
  - What conclusions were you trying to get me to draw?

Uncertainty about the World

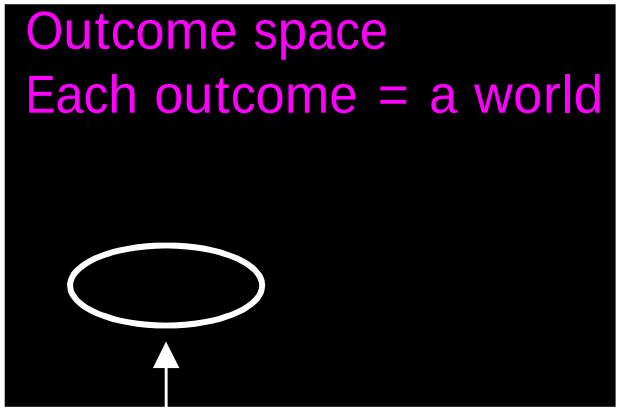


Low-prob set of worlds in which owl(BarackObama)=true





Uncertainty about the World



Low-prob set of worlds in which owl(BarackObama)=true

Oh! we must be in a world where all owls are bachelors …

… **and** where there's a First Lady.

Given everything else I believe about the world, this means that **almost certainly** it is a world where Obama is *not a bachelor* …

… and therefore not an owl.