Of Lambdas and Linguists

Montague Semantics

Haskell Meetup
Singapore (7 August 2019)

Linguistics in a Haskell Meetup?

Natural languages

- \+ weird! wonderful! intuitive
- \- messy and hard to work with

typed
lambda
calculus?

Logical languages

- \+ easy to reason with
- \- not how people talk

Disclaimer 1

I’m a coder, not a linguist!

linguists, computational linguists, logicians
I love you all

Disclaimer 2

Siri? Alexa?
Google Translate?

🤷🏻‍♂️

sorry, no idea!

Logical languages 🤖

Pieces of a logic

1. **Vocabulary**
2. Model
3. Language
4. Satisfaction

```
data Vocab =
   V0 Thing
 | V1 Rel1
 | V2 Rel2


data VocabThing =
   Bart
 | Lisa
 | SantaLH -- Santa's Little Helper


data VocabRel1 =
   Student
 | Greyhound
 | Run

data VocabRel2 =
   Sibling
 | Torment
```

Pieces of a logic

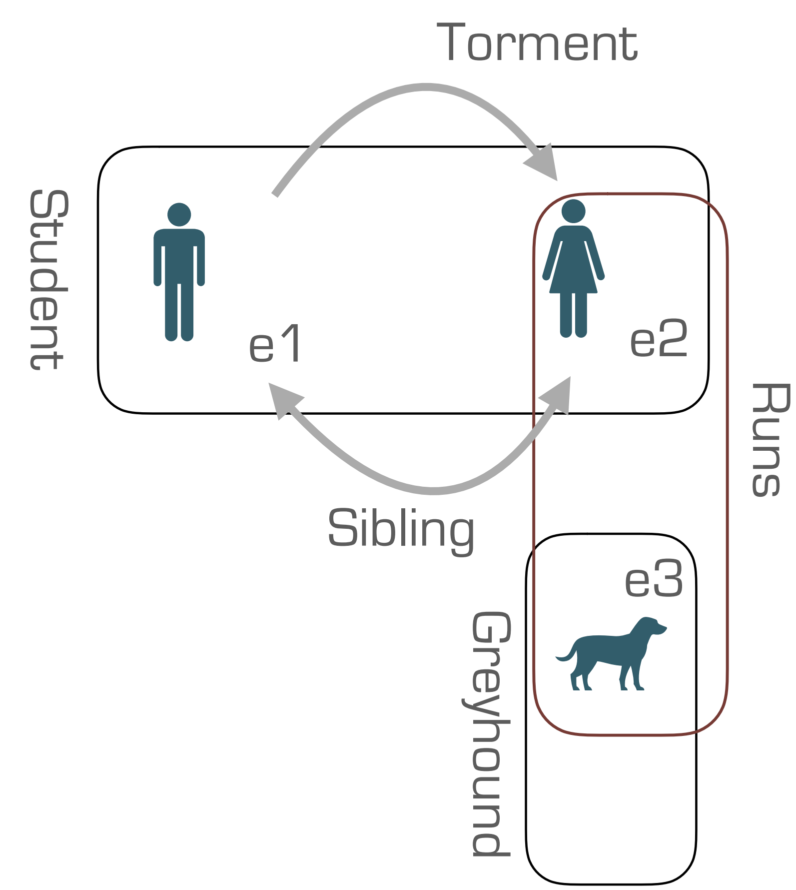
Model ▸ Domain

```
data Entity = E1 | E2 | E3 | E4..
  deriving (Bounded)
```

Model ▸ Interpretation Function

```
model9871 = (model9871_t, model9871_u, model9871_b)

model9871_t :: VocabThing → {Entity}
model9871_t Bart          = {E1}
model9871_t Lisa          = {E2}
model9871_t SantaLH       = {E3}

model9871_u :: VocabRel1 → {Entity}
model9871_u Student       = {E1, E2}
model9871_u Greyhound     = {E3}
model9871_u Runs          = {E2, E3}

model9871_b :: VocabRel2 → {(Entity, Entity)}
model9871_b Sibling       = {(E1, E2), (E2, E1)}
model9871_b Torment       = {(E1, E2)}
```

pseudoHaskell



Pieces of a logic

1. Vocabulary
2. Model
3. **Language**
4. Satisfaction

```
newtype Variable = Variable { fromVariable :: String }

data Term =
    Const Thing                -- Bart
  | Var Variable               -- x


data Formula =
    UnaryRel Rel1 Term         -- Greyhound(x)
  | BinaryRel Rel2 Term Term   -- Torment(Bart)(Lisa)
  | Not Formula                -- ¬Torment(Lisa)(Bart)
  | And Formula Formula        -- Greyhound(x) ∧ Run(x)
  | Or Formula Formula         -- Greyhound(x) ∨ Student(y)
  | Implies Formula Formula    -- Student(x) → Torment(x)(Lisa)
  | Exists Variable Formula    -- ∃z.Sibling(z)(SantaLH)
  | ForAll Variable Formula    -- ∀z.¬Torment(Lisa)(z)
  | Possibly Formula           -- ◇ Sibling(Lisa)(Bart)
  | Necessarily Formula        -- □ Sibling(Bart)(Lisa)
```

will ignore these...

Pieces of a logic

1. Vocabulary
2. Model
3. Language
4. **Satisfaction**

```
type Assignment = Variable → Maybe Entity

assign3193 :: Assignment
assign3193 “x” = Just E1
assign3193 “y" = Just E2
assign3193 _ = Nothing


class Interp res v where
  interp :: Model → Assignment → v → {res}


instance Interp Entity Variable where
  interp mdl asg v = maybe emptySet singleton (asg v)

instance Interp Entity VocabThing where
  interp mdl asg t = (fst3 mdl) t

instance Interp (Entity, Entity) VocabRel2 where
  interp mdl asg t = (thd3 mdl) t

instance Interp Entity Term where
  interp mdl asg (Constant c) = interp mdl asg c
  interp mdl asg (Var v)      = interp mdl asg v
```

Pieces of a logic

1. Vocabulary
2. Model
3. Language
4. **Satisfaction**

```
satisfies :: Model → Assignment → Formula → Maybe Bool
satisfies model asg form =
  case interp model asg form of
    {} -> Nothing
    xs -> Just (or xs)

instance Interp Bool Formula where
 interp mdl asg f = case f of
   UnaryRel  r t      -> ⟨ int t ∈ int r ⟩
   BinaryRel r t1 t2 -> ⟨(int t1, int t2) ∈ int r⟩
   Not f        -> ⟨ not (sat f) ⟩
   And f1 f2    -> ⟨ int f1 && int f2 ⟩
   Or f1 f2     -> ⟨ int f1 || int f2 ⟩
   Implies p q -> ⟨ not (int f1) || int f2 ⟩
   Exists x f  -> ⟨ any (\a -> interp mdl a f) (variants asg x) ⟩
   ForAll x f  -> ⟨ all (\a -> interp mdl a f) (variants asg x) ⟩
  where
   int = interp mdl asg


variants :: Assignment → Variable → {Assignment}
variants asg v0 = Set.fromList $

  [ \v -> if v == v0 then Just e else asg v
  | e <- [minBound..maxBound]] -- for all entities
```

pseudoHaskell

Jobs for a 🤖

Is it true? (querying, model checking)

```
alwaysSatisfies model47 (canOpenThePodBayDoors Hal)
```

$\mathfrak{M} \models \mathrm{PodBayDoors}(d) \wedge \mathrm{CanOpen}(\mathrm{Hal}, d)$

Does it make sense? (consistency, satisfiability)

$(\forall x.\mathrm{Greyhound}(x) \rightarrow \mathrm{Runs}(x)) \wedge \mathrm{Greyhound}(\mathrm{SantasLH}) \wedge \neg \mathrm{Runs}(\mathrm{SantasLH})$

Is this new? (informativity)

Bart and Lisa are students

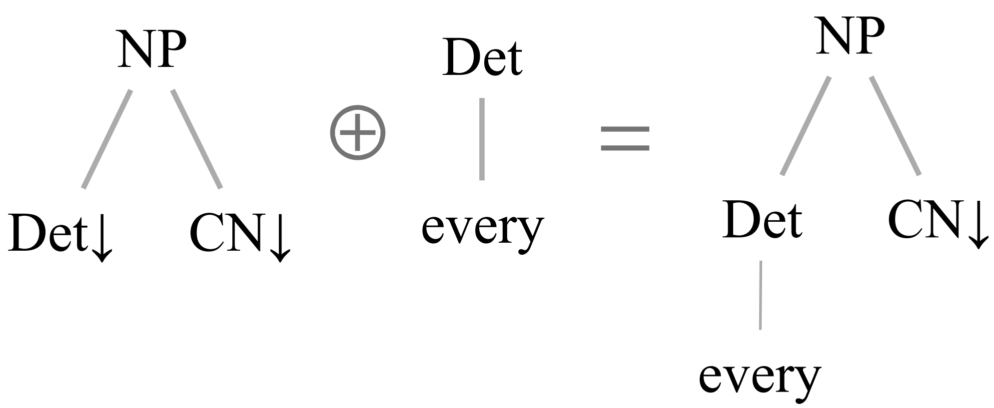
Bart is a student ???

👩🏽‍🎨

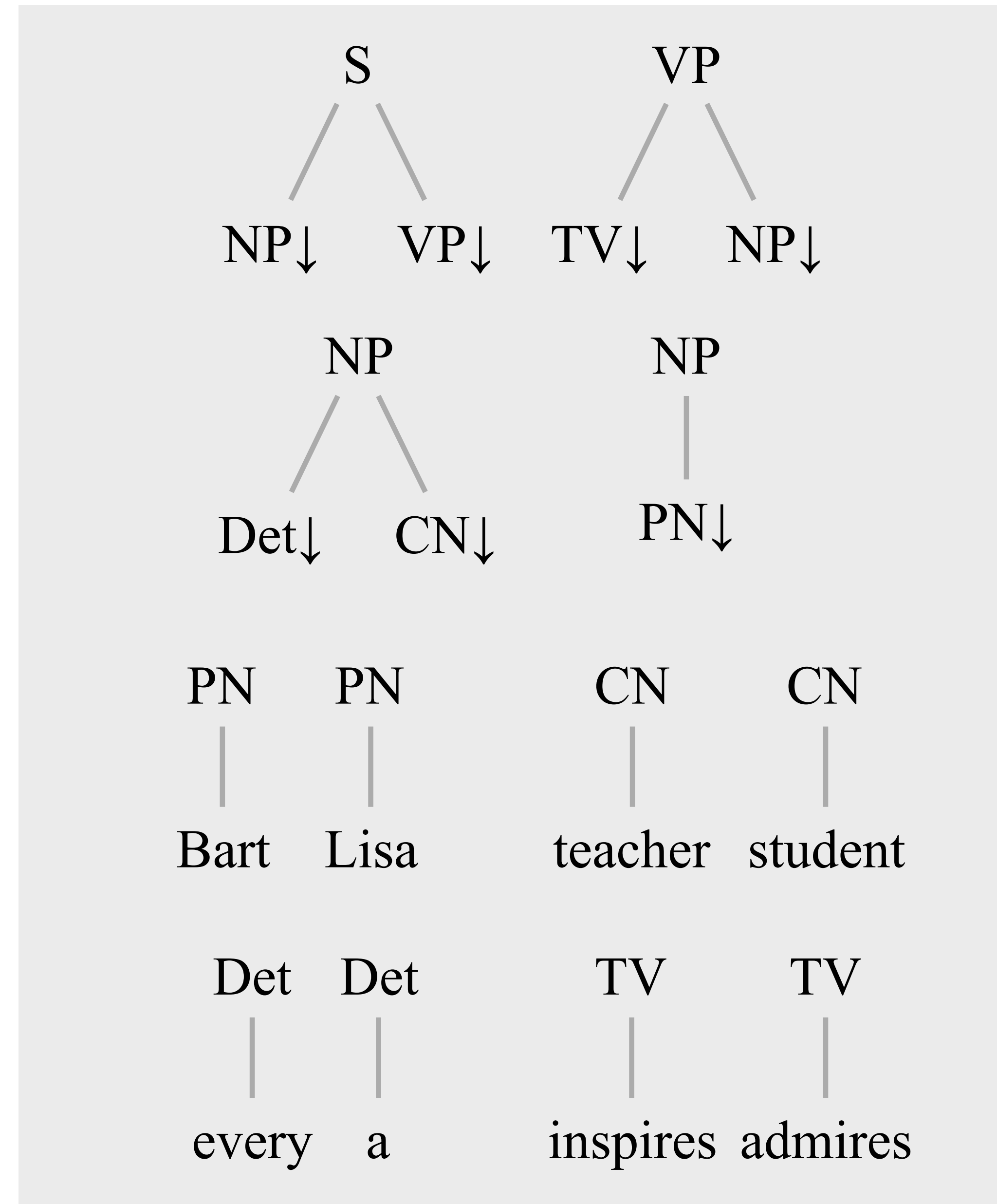
very nice, but
most people don't
speak in logic

Natural languages 👩🏽‍🎨

1960s: formal study of syntax

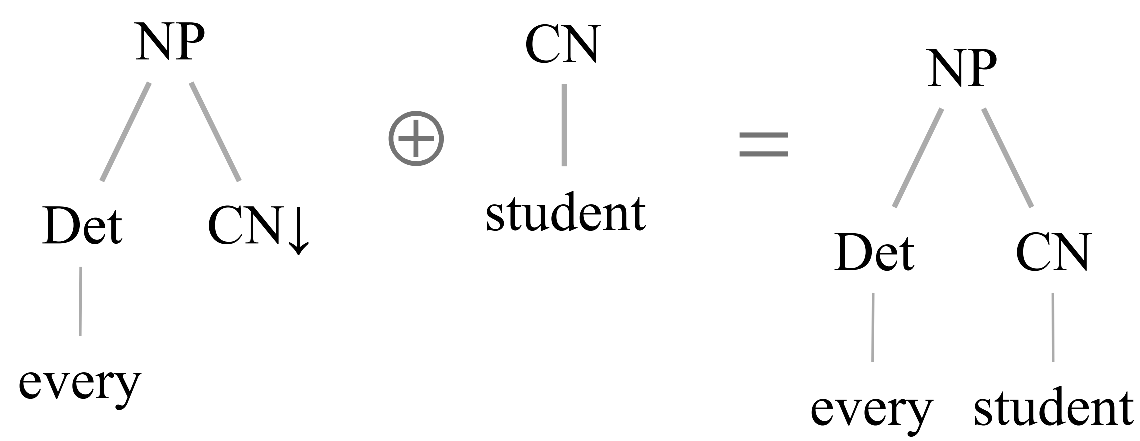


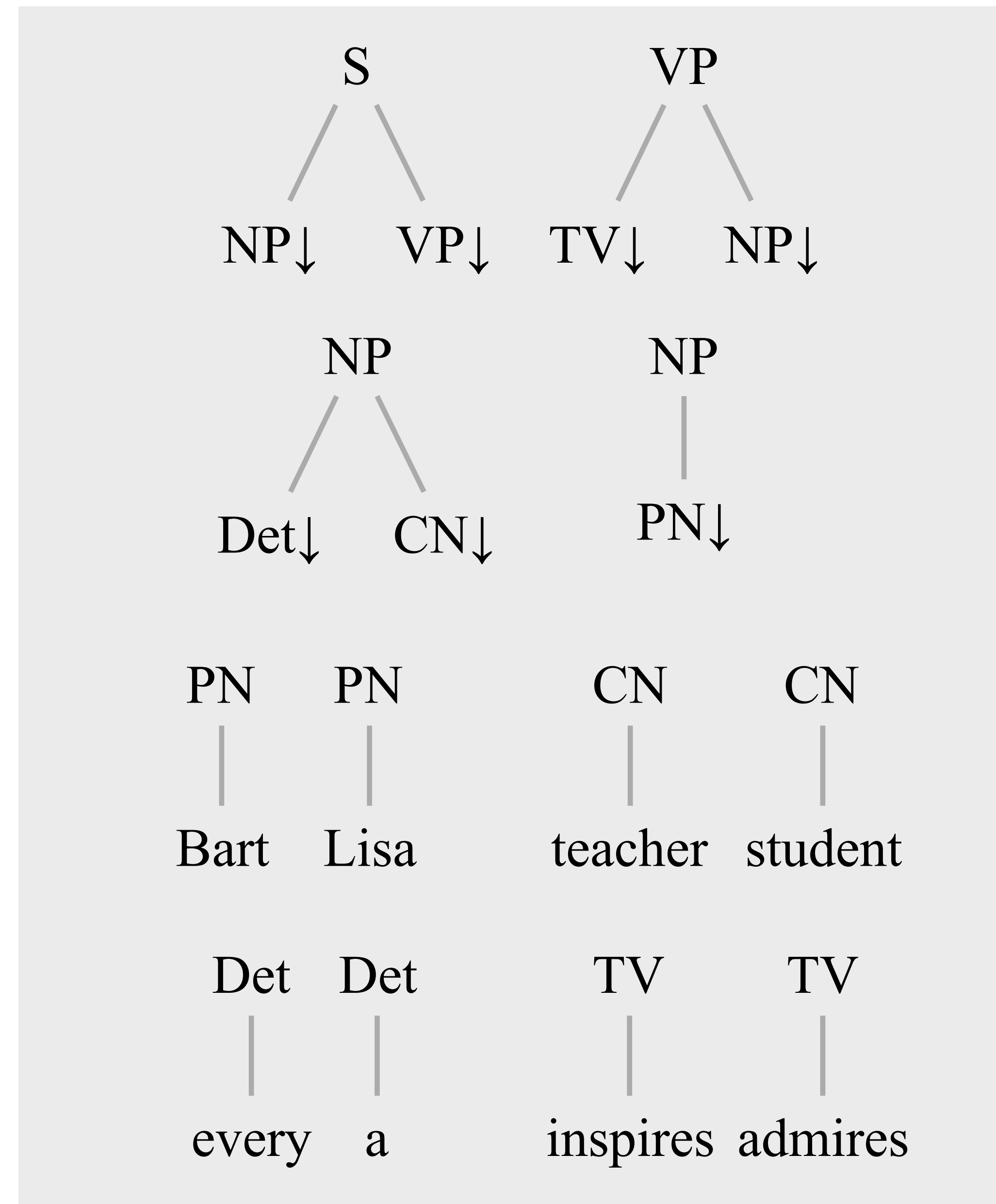
Every student admired Lisa



Context Free Grammar

1960s: formal study of syntax

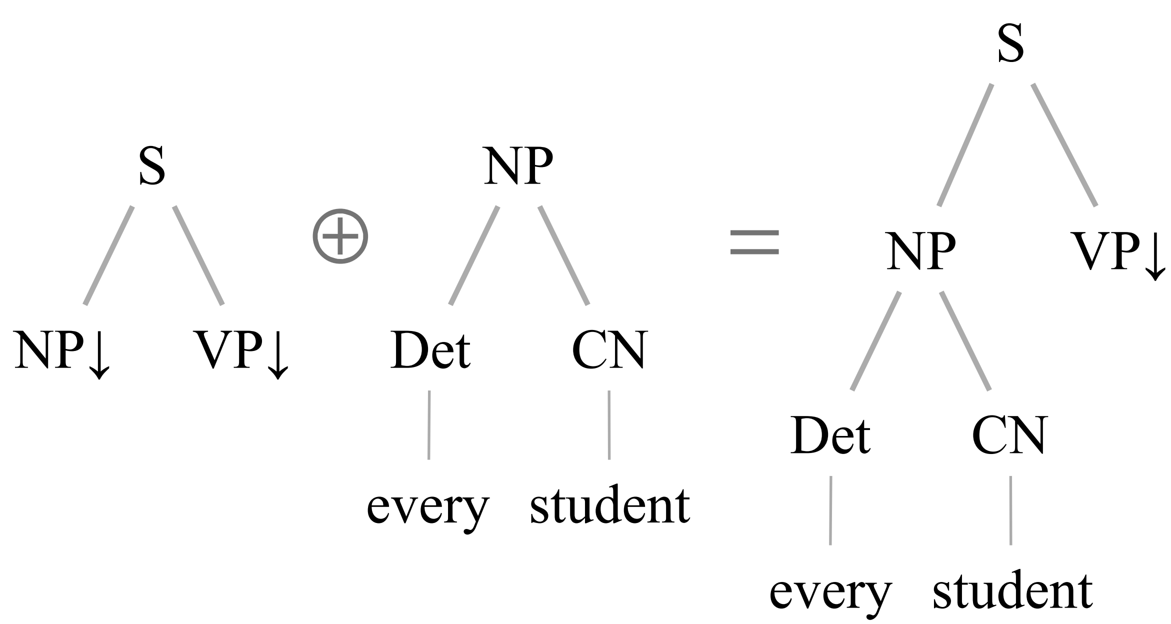


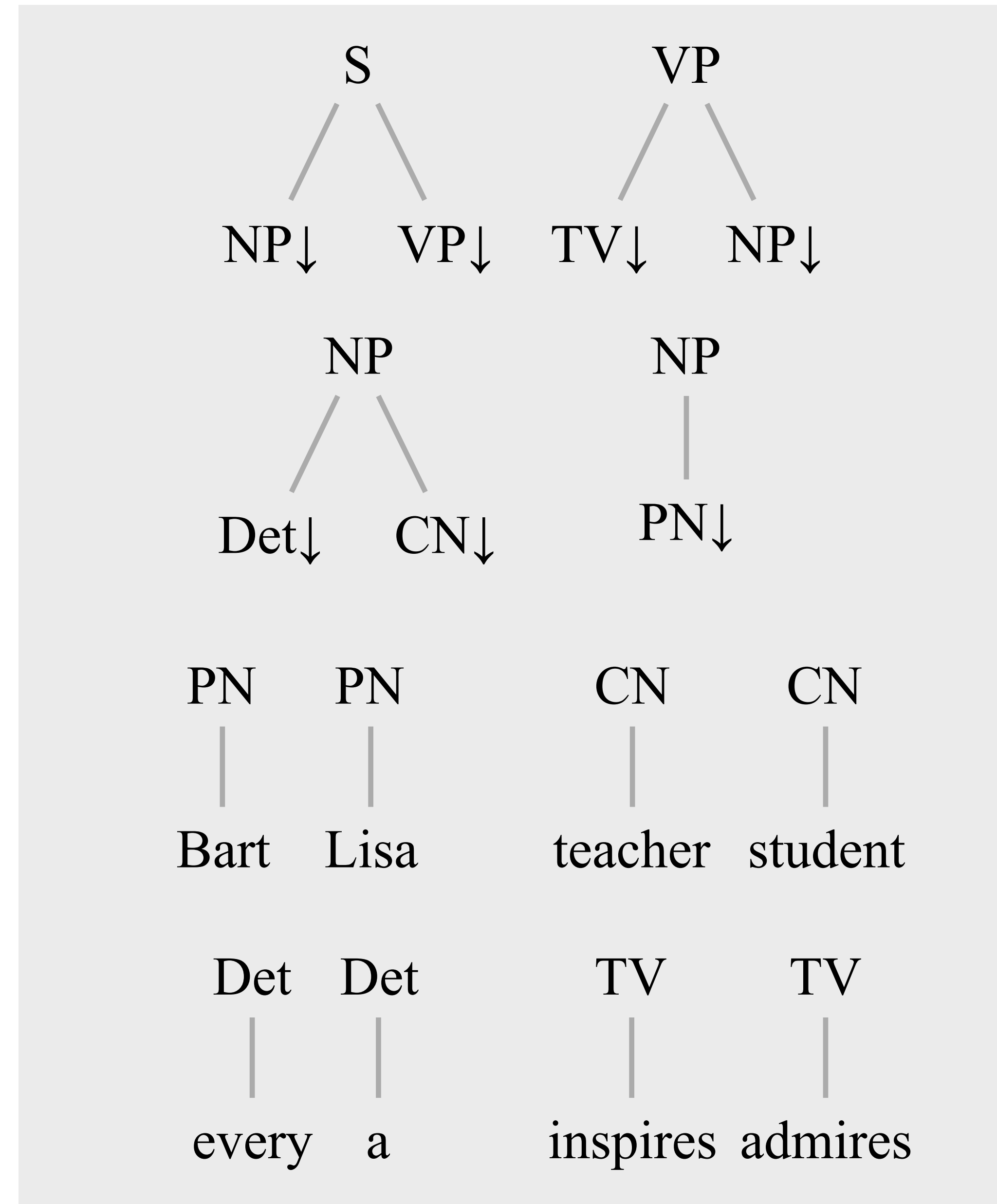


Context Free Grammar

Every student admired Lisa

1960s: formal study of syntax

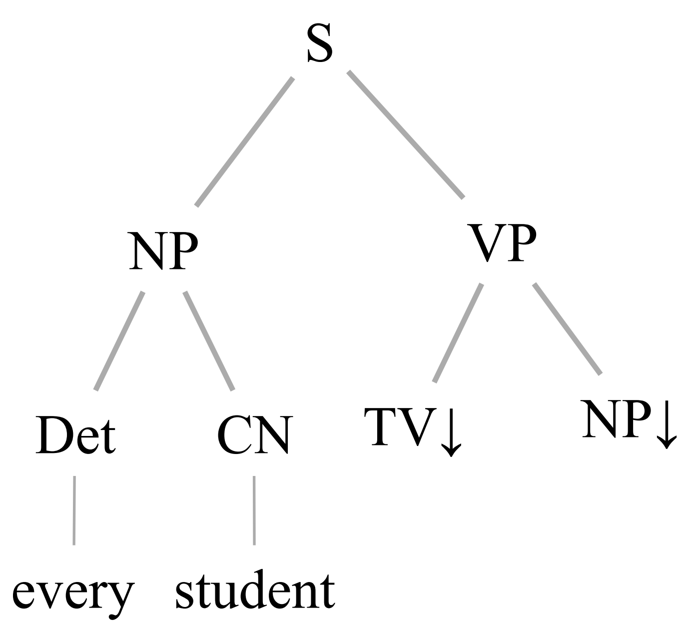




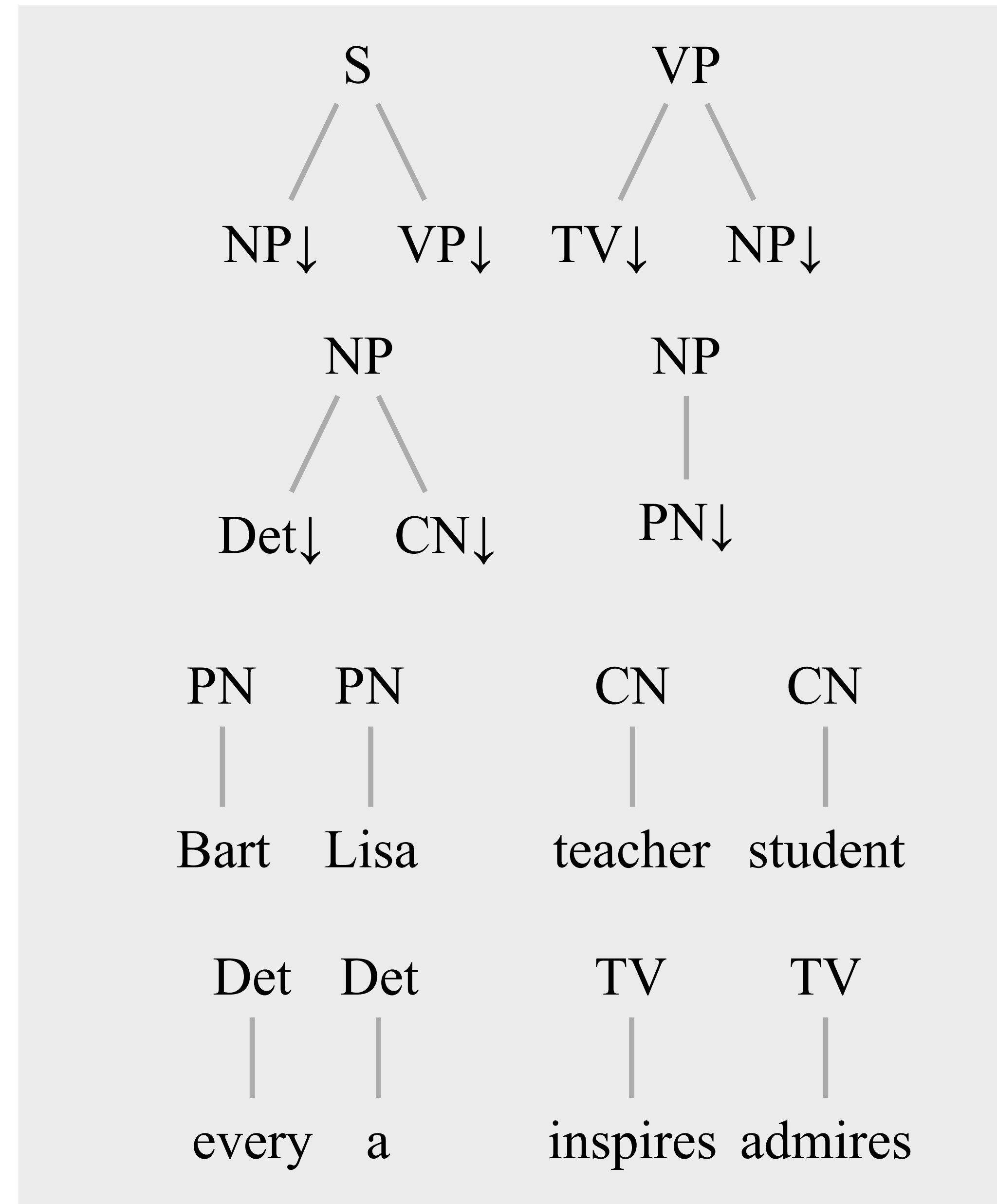
Context Free Grammar

Every student admired Lisa

1960s: formal study of syntax

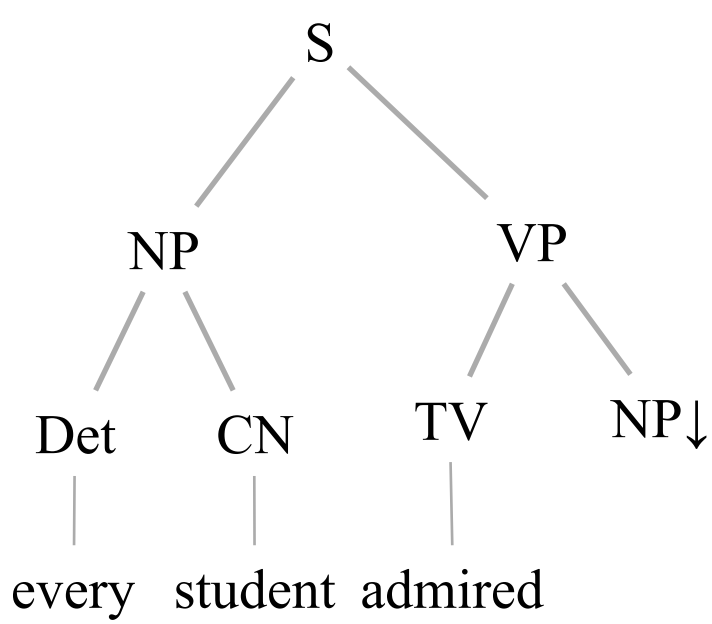


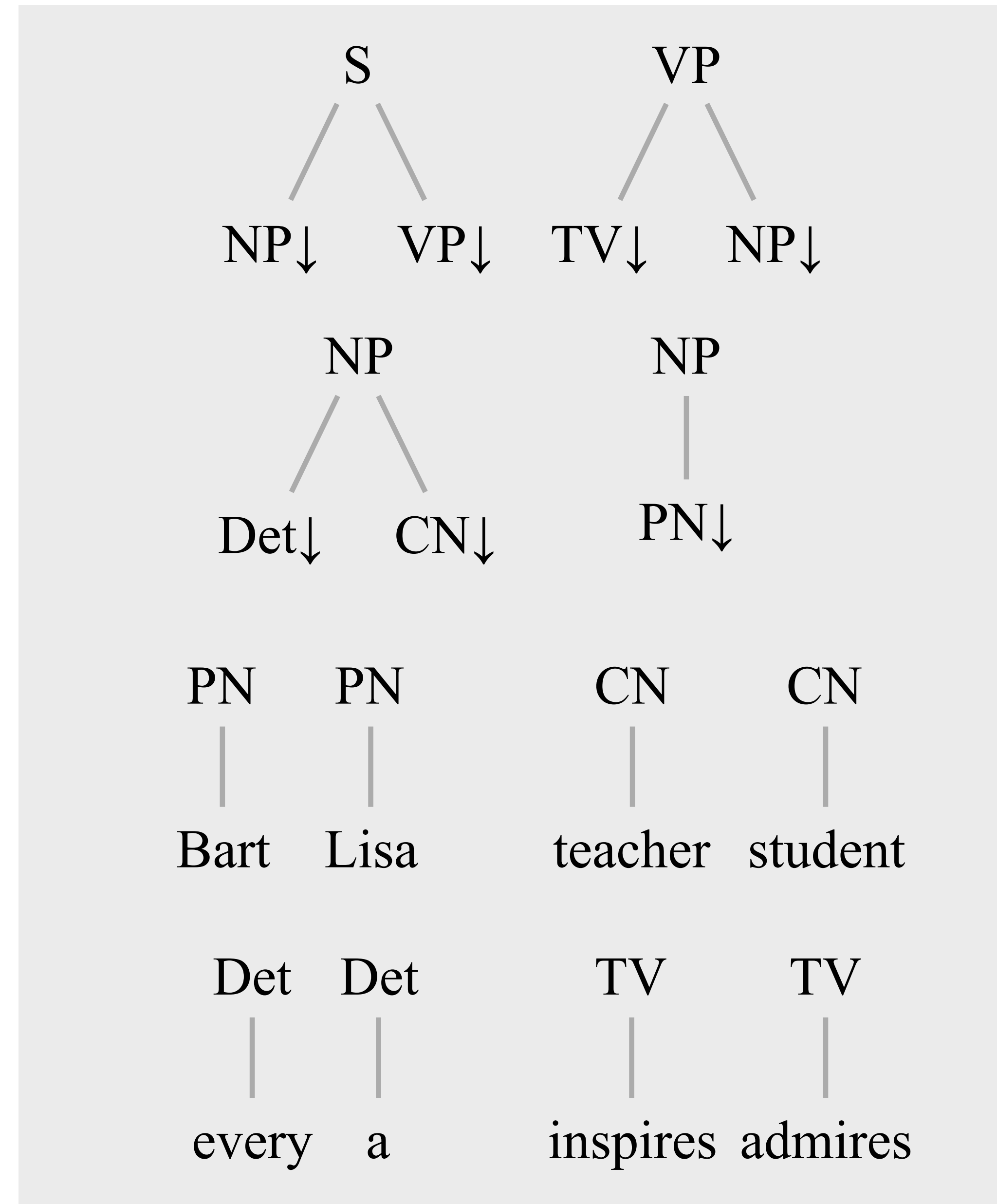
Every student admired Lisa



Context Free Grammar

1960s: formal study of syntax

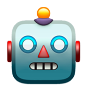




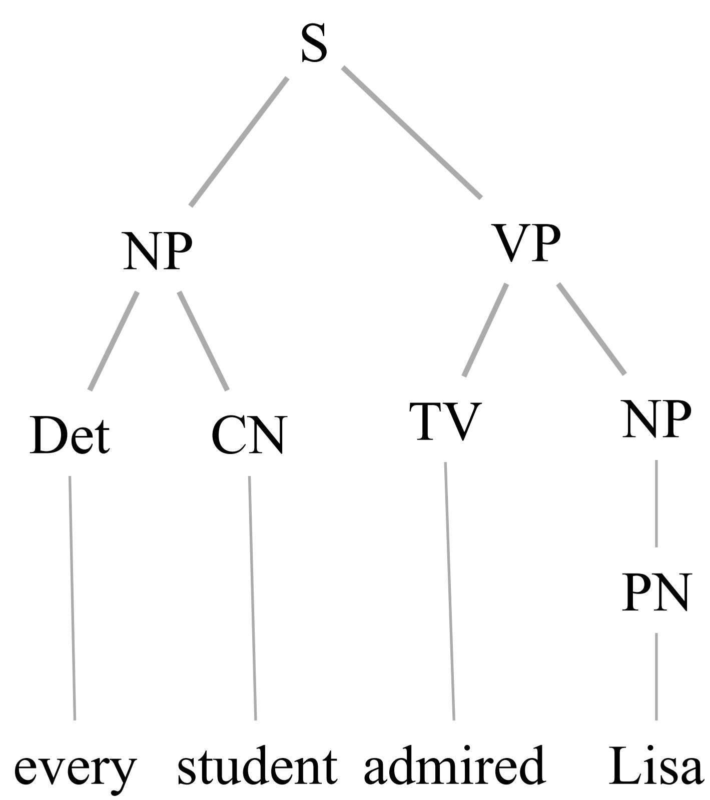
Context Free Grammar

Every student admired Lisa

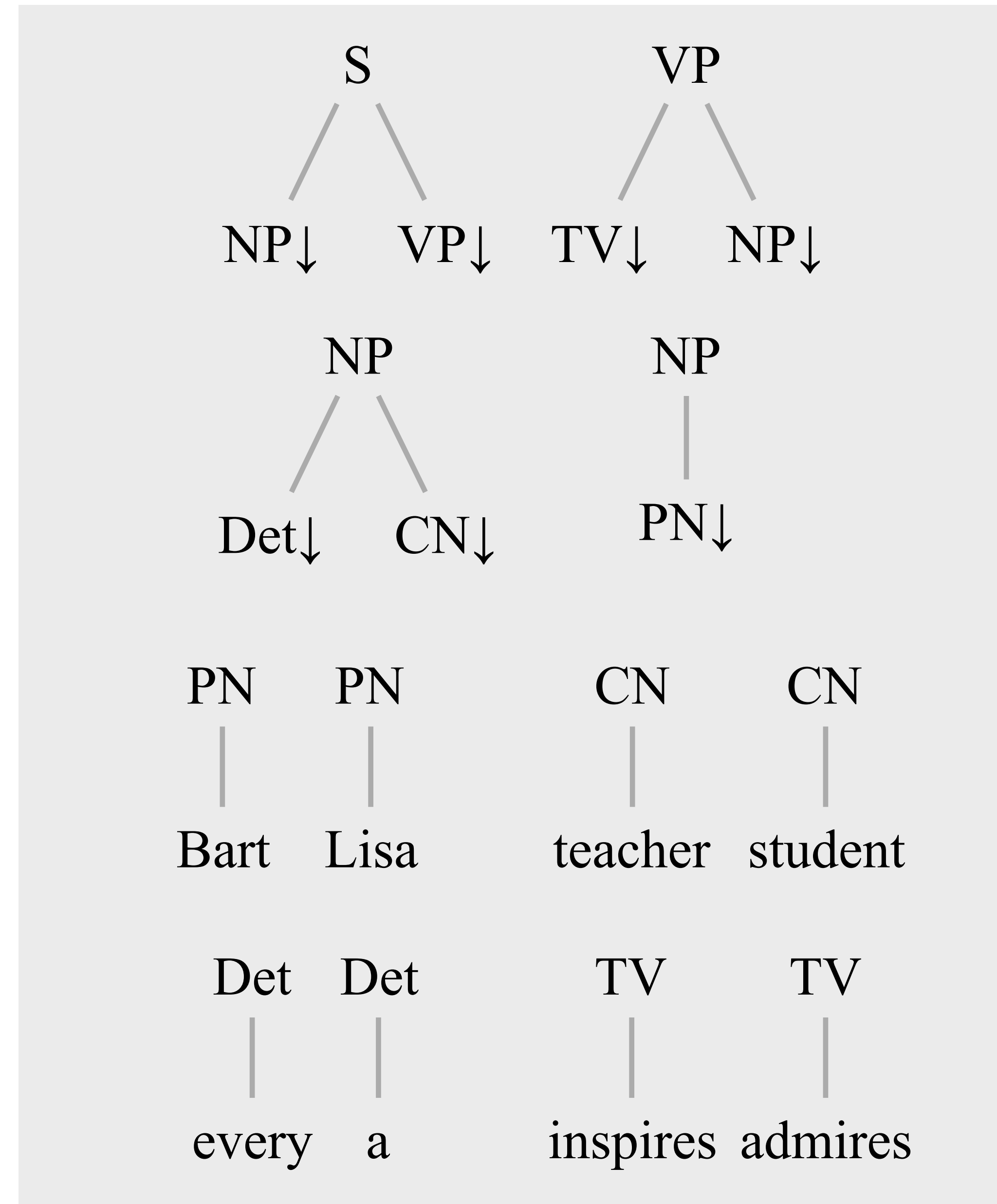
1960s: formal study of syntax



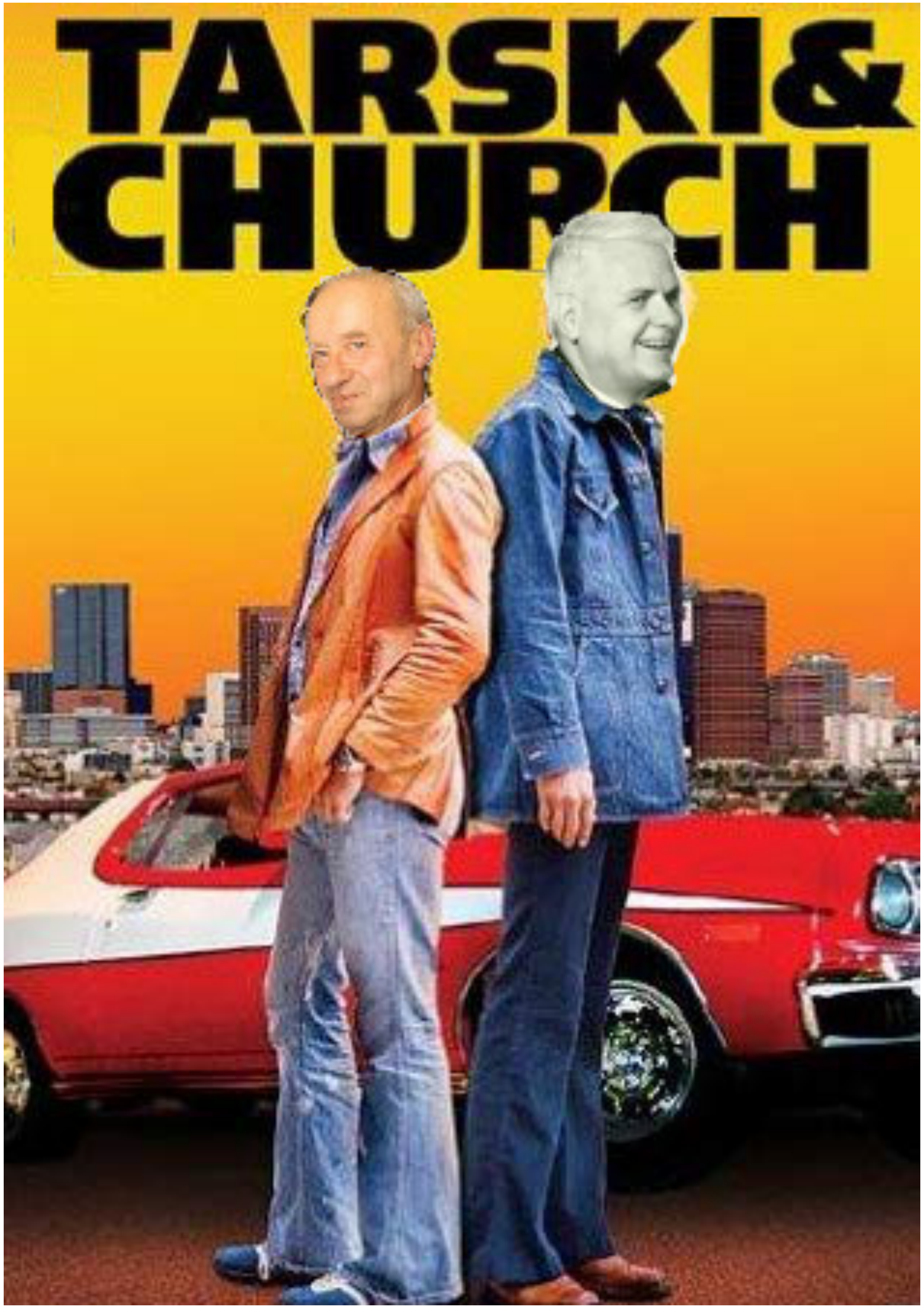
very nice, but what about meaning?



Every student admired Lisa



Context Free Grammar


TARSKI&
CHURCH

The Agenda

Universal Grammar - Montague (1970)

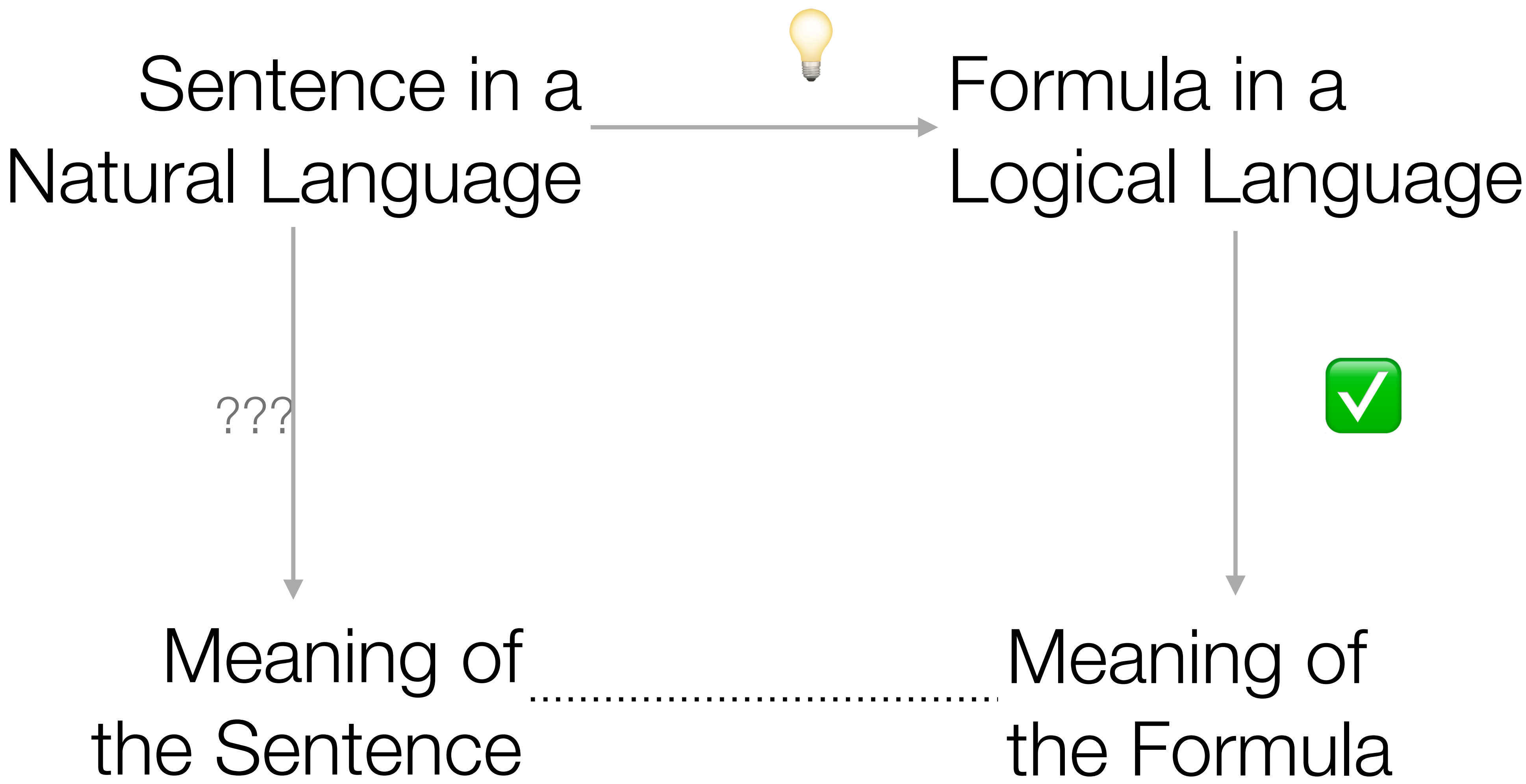
"There is in my opinion no important theoretical difference between natural languages and the artificial languages of logicians; indeed, I consider it possible to comprehend the syntax and semantics of both kinds of languages within a single natural and mathematically precise theory. On this point I differ from a number of philosophers, but agree, I believe, with Chomsky and his associates."

The Agenda

Universal Grammar - Montague (1970)

"It is clear, however, that no adequate and comprehensive semantical theory has yet been constructed, and arguable that no comprehensive and semantically significant syntactical theory yet exists."

The rough idea

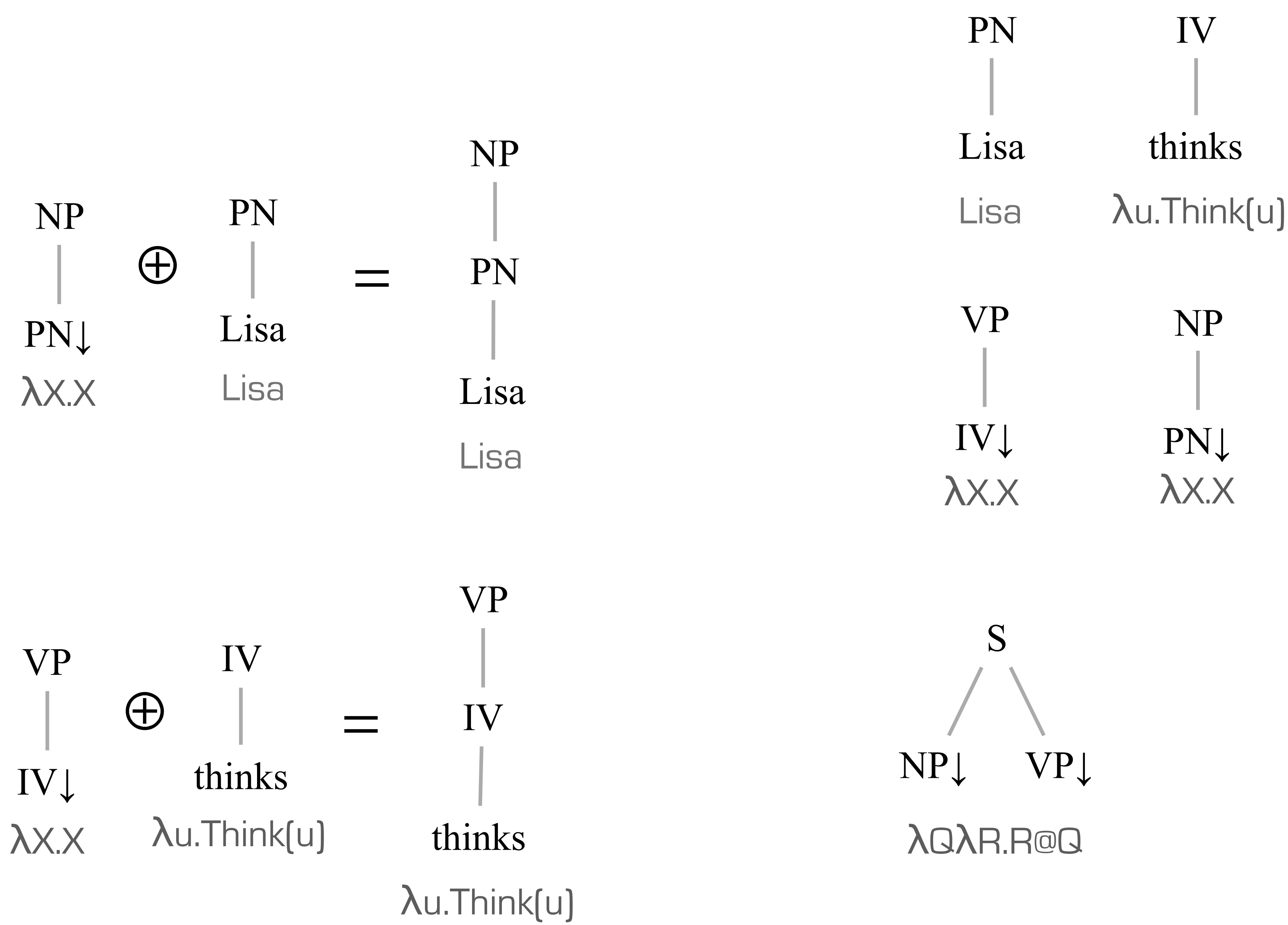


The secret weapon?
Lambda Calculus (1930)

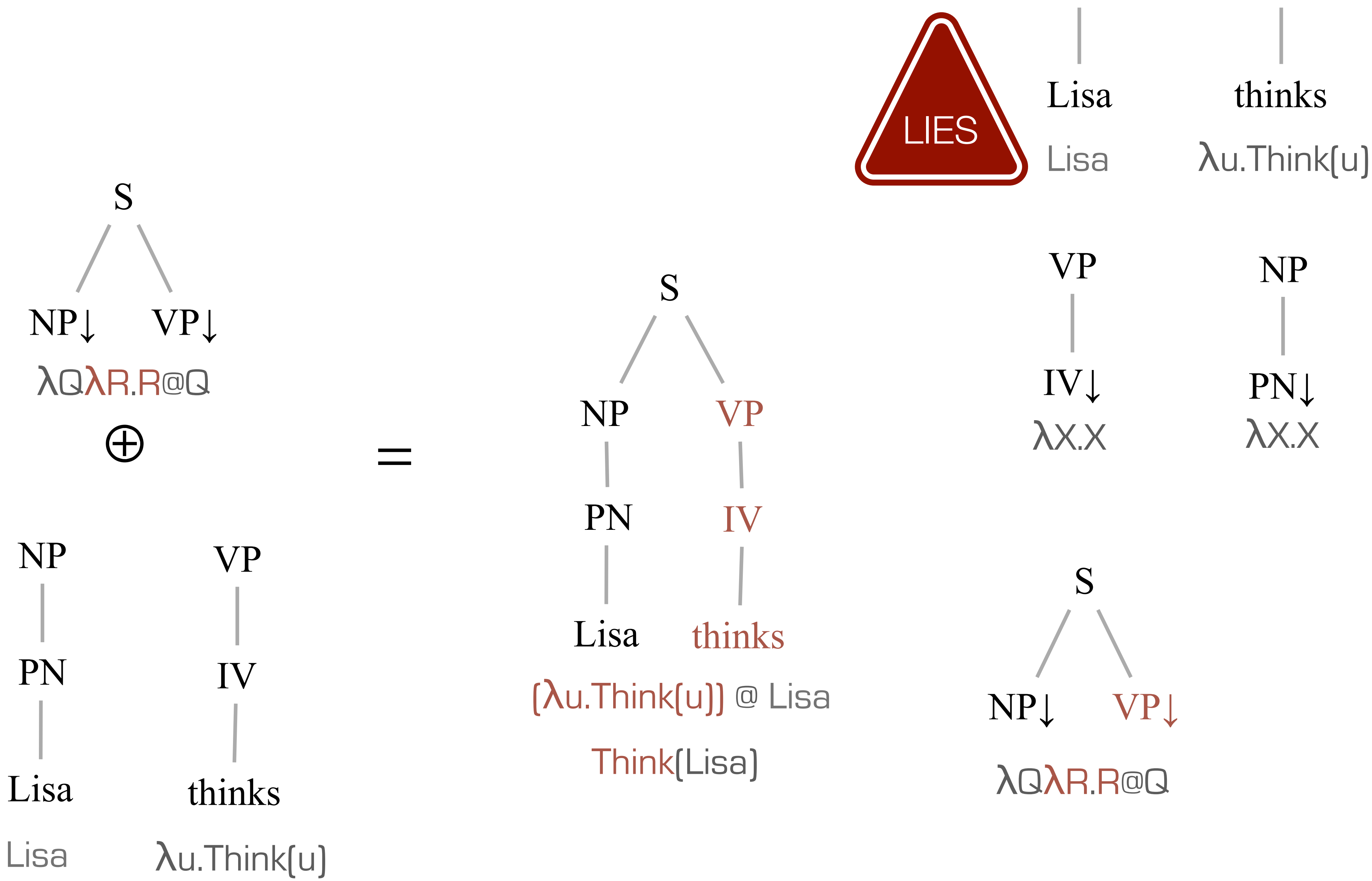
1. Lambda-fy your logic

```
data Term2 =
   Const  Thing
 | NVar   Variable
 | LVar  LVariable

data Formula2
 | Lambda LVariable NFormula          -- λx.Torment(x)(Lisa)
 | Apply  LVariable NFormula Formula2 -- λx.Torment(x)(Lisa) @ Bart
 | NForm  NFormula


data NFormula =
   UnaryRel  Rel1 Term2         -- Greyhound(x)
 | BinaryRel Rel2 Term2 Term2   -- Torment(Bart){Lisa)
 | Not NFormula                 -- ¬Torment(Lisa)(Bart)
 | And NFormula NFormula        -- Greyhound(x) ∧ Run(x)
 | Or  NFormula NFormula        -- Greyhound(x) ∨ Student(y)
 | Implies NFormula NFormula    -- Student(x) → Torment(x)(Lisa)
 | Exists Variable  NFormula    -- ∃z.Sibling(z,SantaLH)
 | ForAll Variable  NFormula    -- ∀z.¬Torment(Lisa)(z)
```

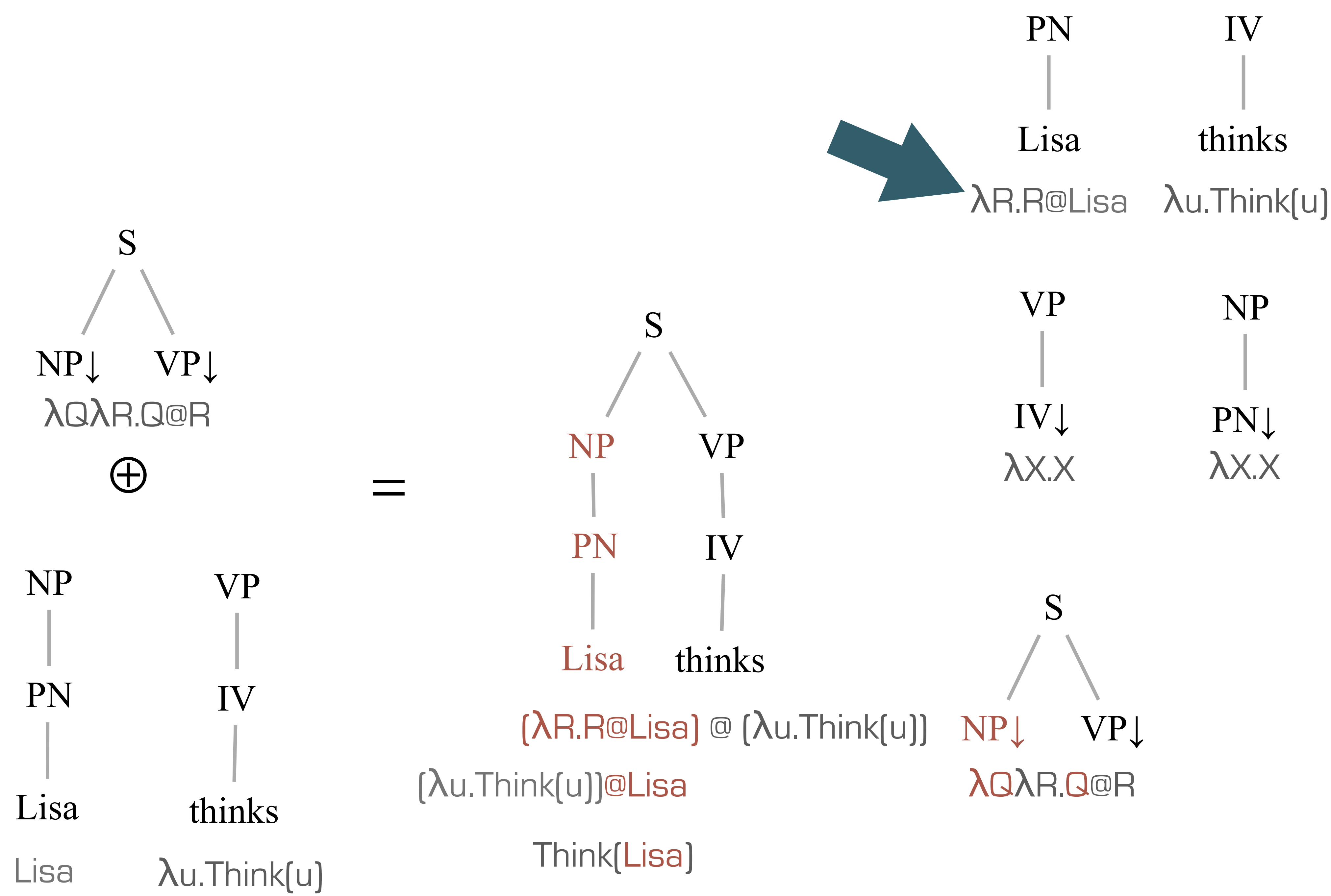
2. Pair syntactic rules with semantic fragments



2. Pair syntactic rules with semantic fragments

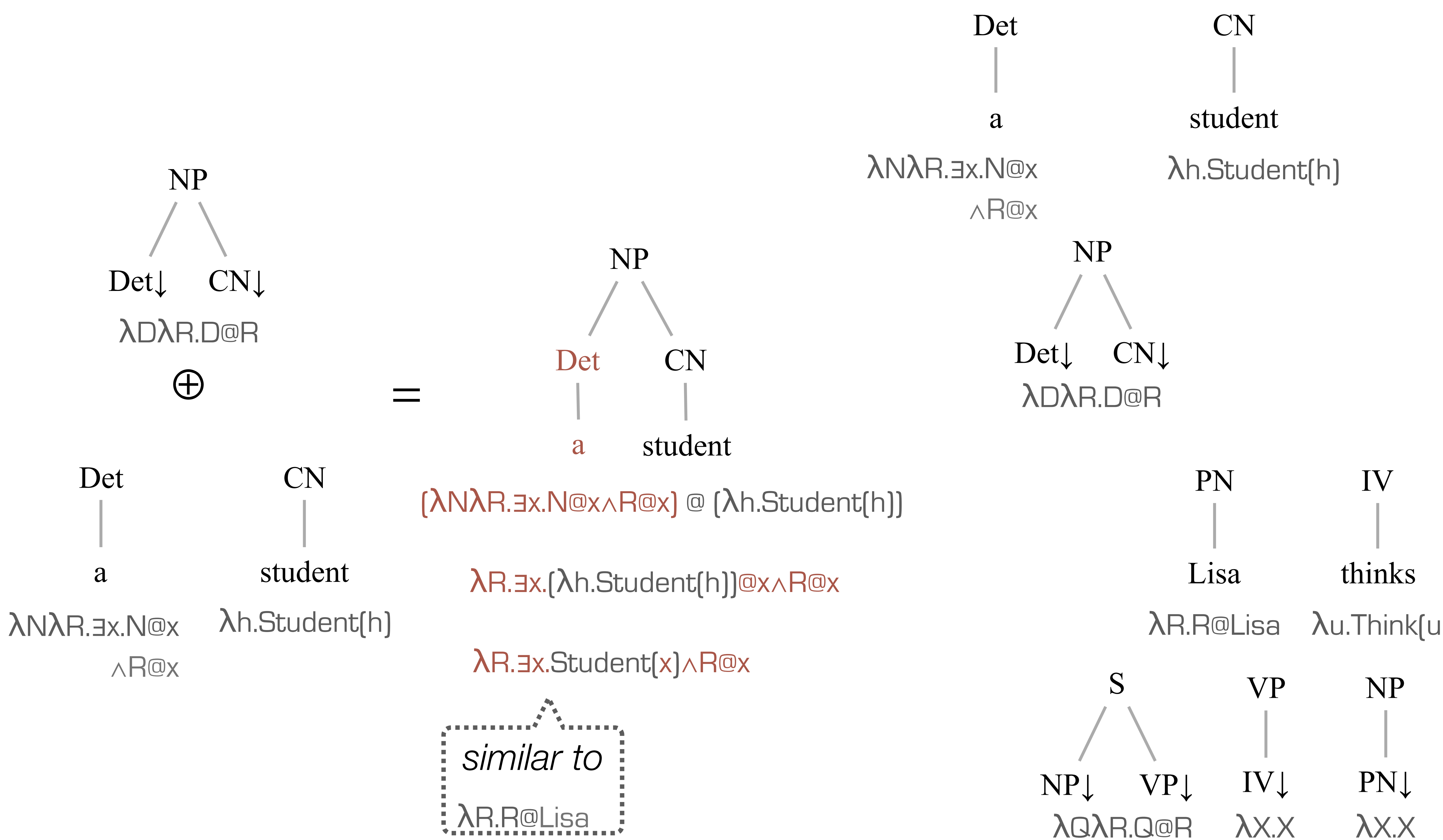


2. Pair syntactic rules with semantic fragments



3. Think real hard

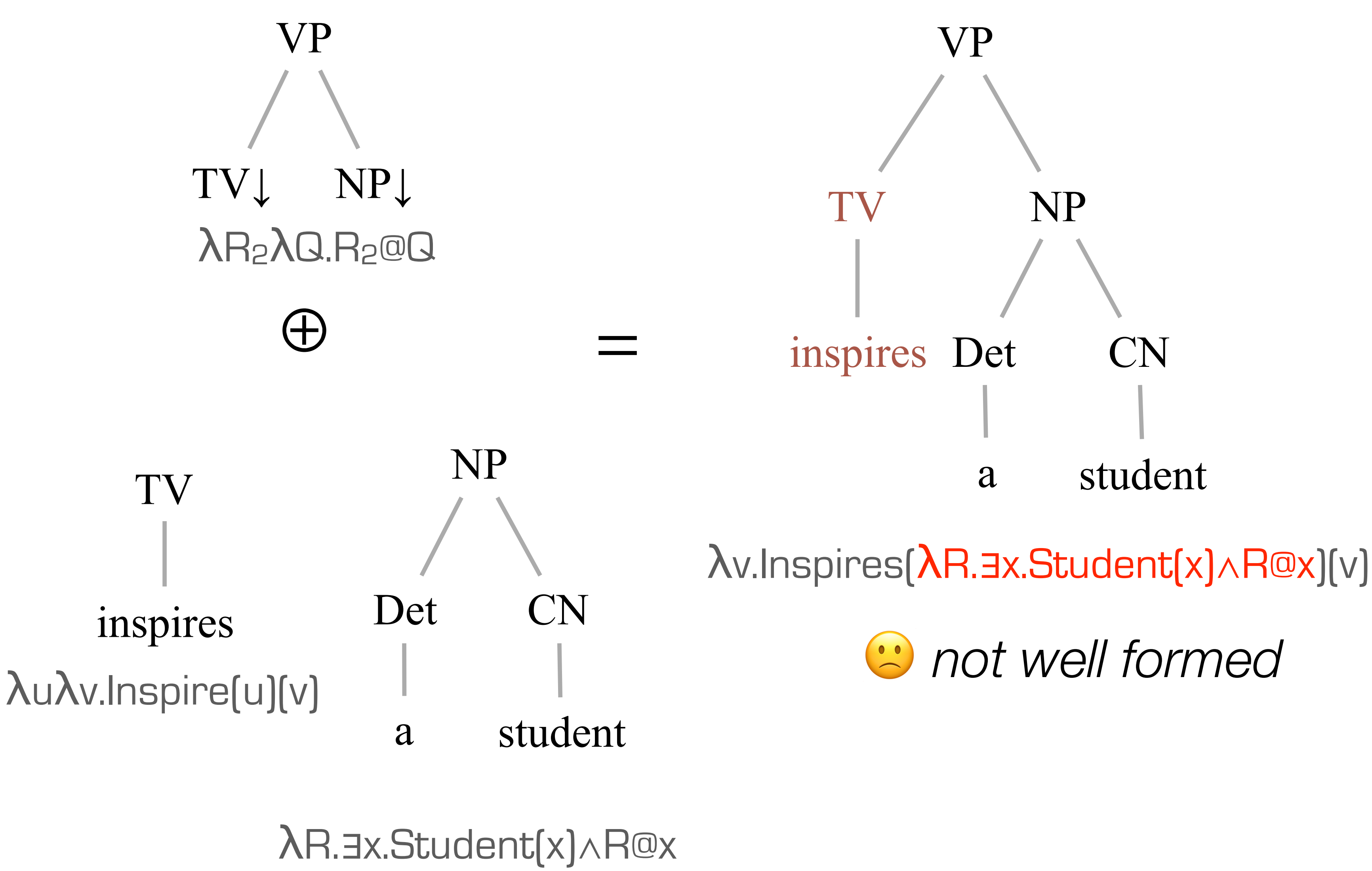
Determiners

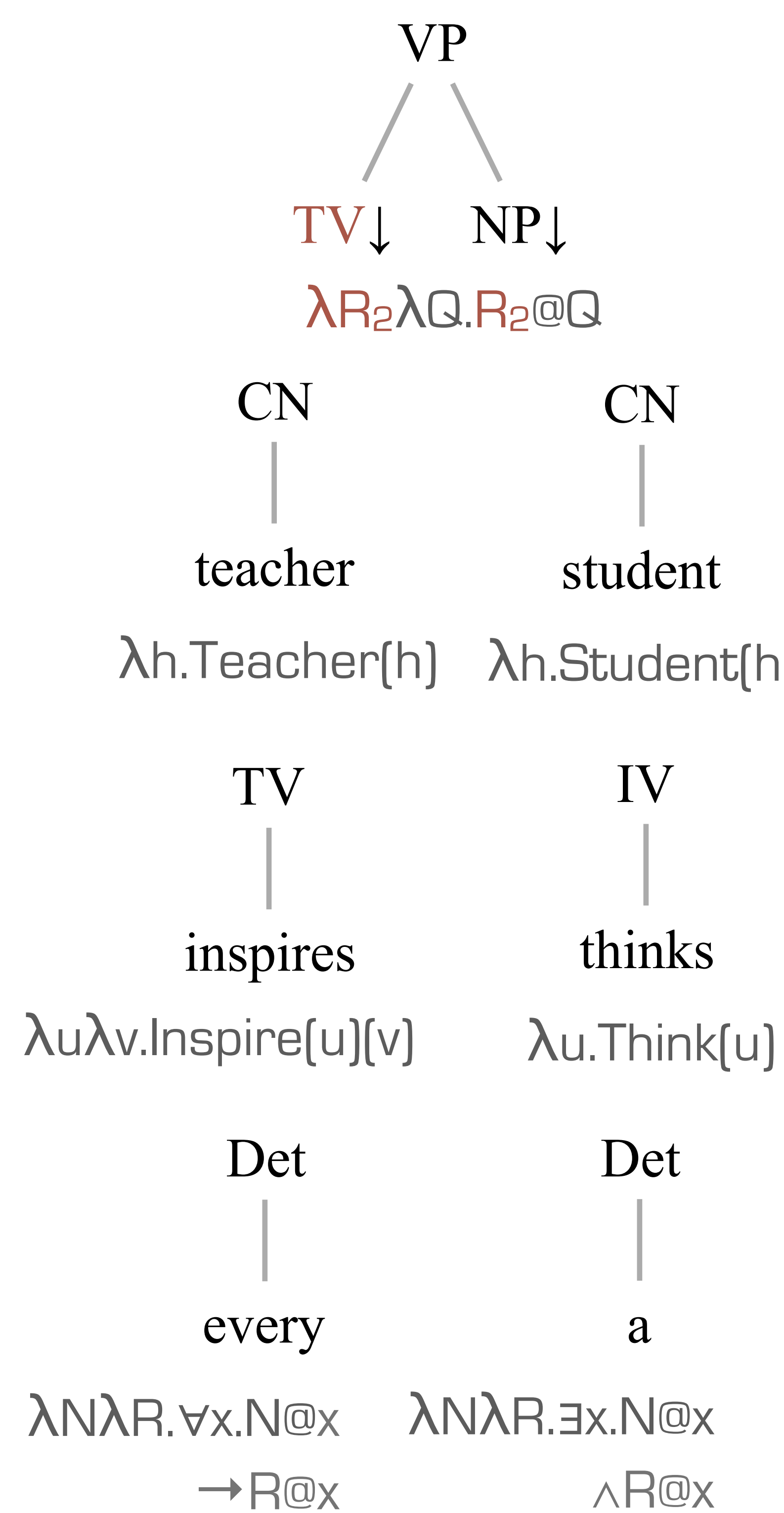


3. Think real hard

Transitive verbs

Every teacher inspires a student

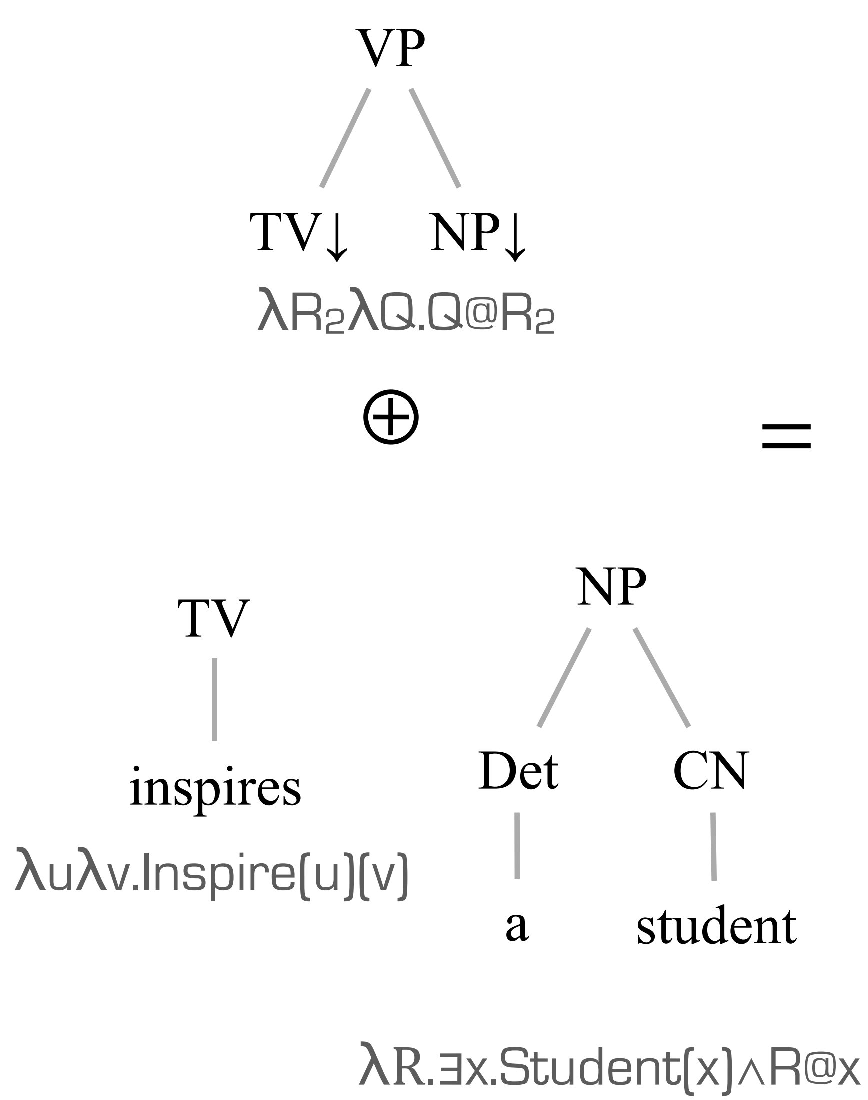


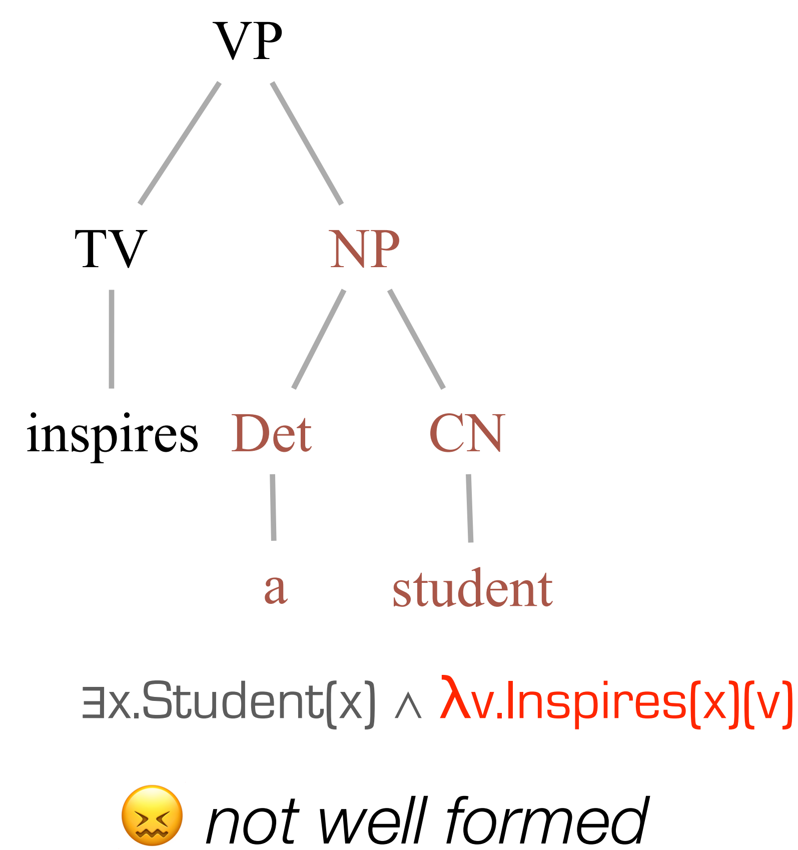


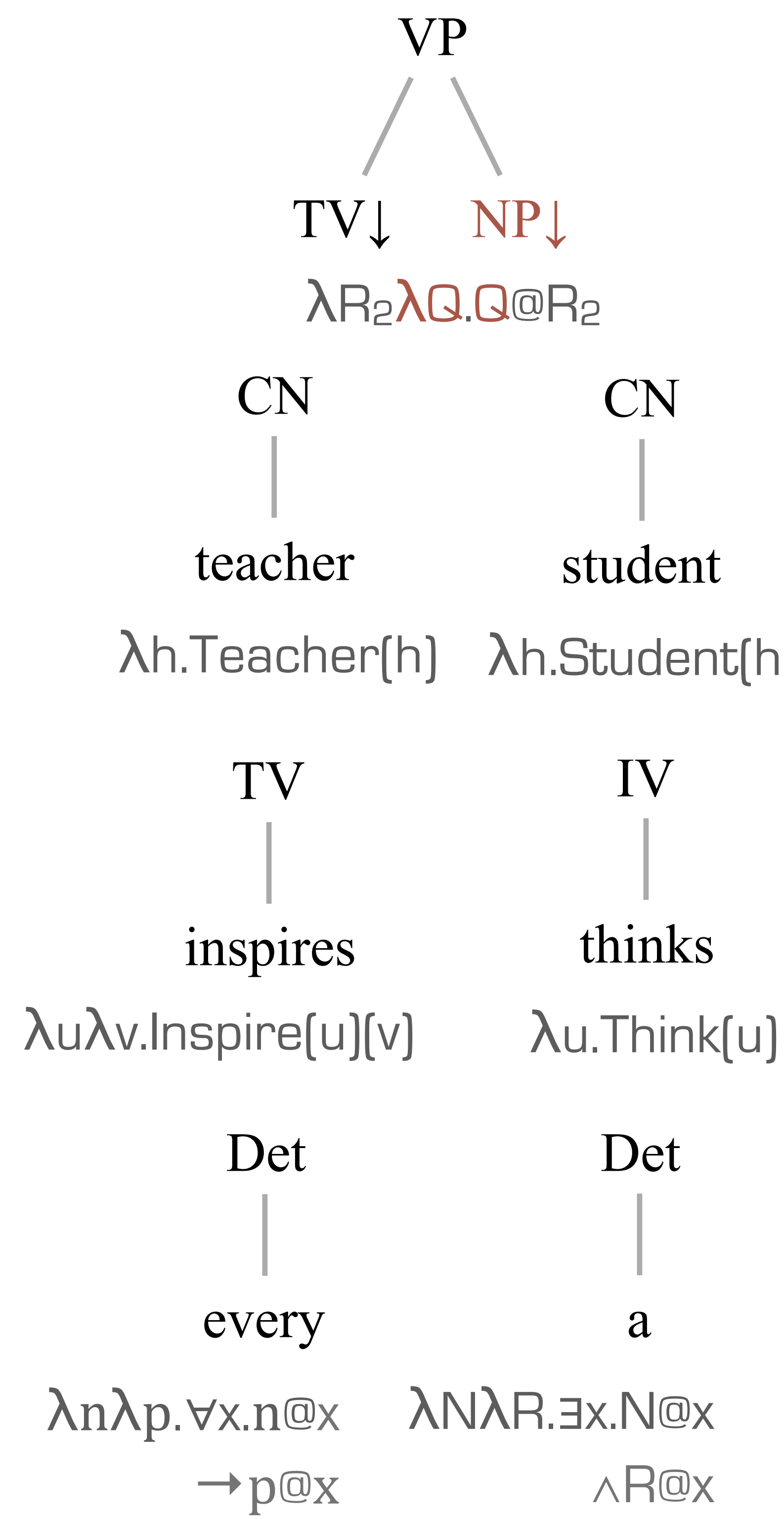
3. Think real hard

Transitive verbs

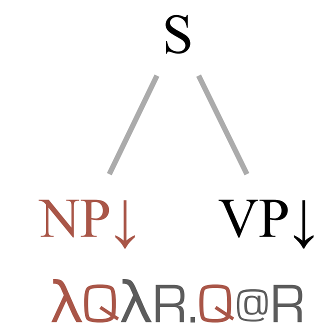
Every teacher inspires a student







This is way easier with types



type Rel1 = Entity → Bool

sentence :: (Rel1 → Bool) → Rel1 → Bool
sentence = λQλR.λQ@R

⟦lisa⟧ :: Rel1 → Bool
⟦lisa⟧ = λR.R@Lisa

⟦thinks⟧ :: Rel1
⟦thinks⟧ = λu.Think(u)

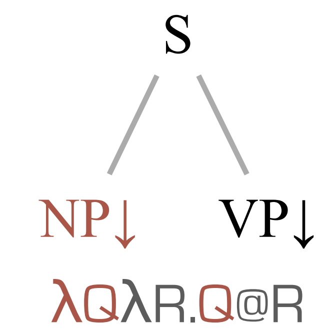
⟦Lisa thinks⟧ :: Bool
⟦Lisa⟧ :: Rel1 → Bool
⟦thinks⟧ :: Rel1

⟦a⟧ :: Rel1 → Rel1 → Bool
⟦a⟧ = λNλR.∃x.N@x∧R@x

⟦teacher⟧ :: Rel1
⟦teacher⟧ = λh.Teacher(h)

⟦a teacher thinks⟧ :: Bool
⟦a teacher⟧ :: Rel1 → Bool
⟦thinks⟧ :: Rel1

This is way easier with types



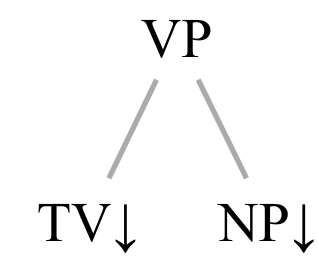

```
type Rel1 = Entity → Bool
⟦lisa⟧ :: Rel1 → Bool
⟦lisa⟧ = λR.R@Lisa
⟦thinks⟧ :: Rel1
⟦thinks⟧ = λu.Think(u)
```

```
sentence :: (Rel1 → Bool) → Rel1 → Bool
sentence = λQλR.λQ@R
```

⟦Lisa thinks⟧ :: Bool
⟦Lisa⟧ :: Rel1 → Bool
⟦thinks⟧ :: Rel1

⟦Lisa inspires a student⟧ :: Bool
⟦Lisa⟧ :: Rel1 → Bool
⟦inspires a student⟧ :: Rel1

This is way easier with types



`type Rel1 = Entity → Bool`

⟦Lisa inspires a student⟧ `:: Bool`

⟦Lisa⟧ `:: Rel1 → Bool`

⟦inspires a student⟧ `:: Rel1`

`type Rel2 = Entity → Rel1`

⟦inspires a student⟧ `:: Rel1`

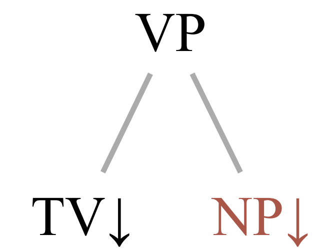
⟦a student⟧ `:: Rel1 → Bool`

⟦inspires⟧ `:: Entity → Entity → Bool`

⟦inspires⟧ `:: Entity → Rel1`

```
vpTrans :: (Entity → Rel1) → (Rel1 → Bool) → Rel1
vpTrans = ???
```

This is way easier with types




```
type Rel1 = Entity → Bool
```

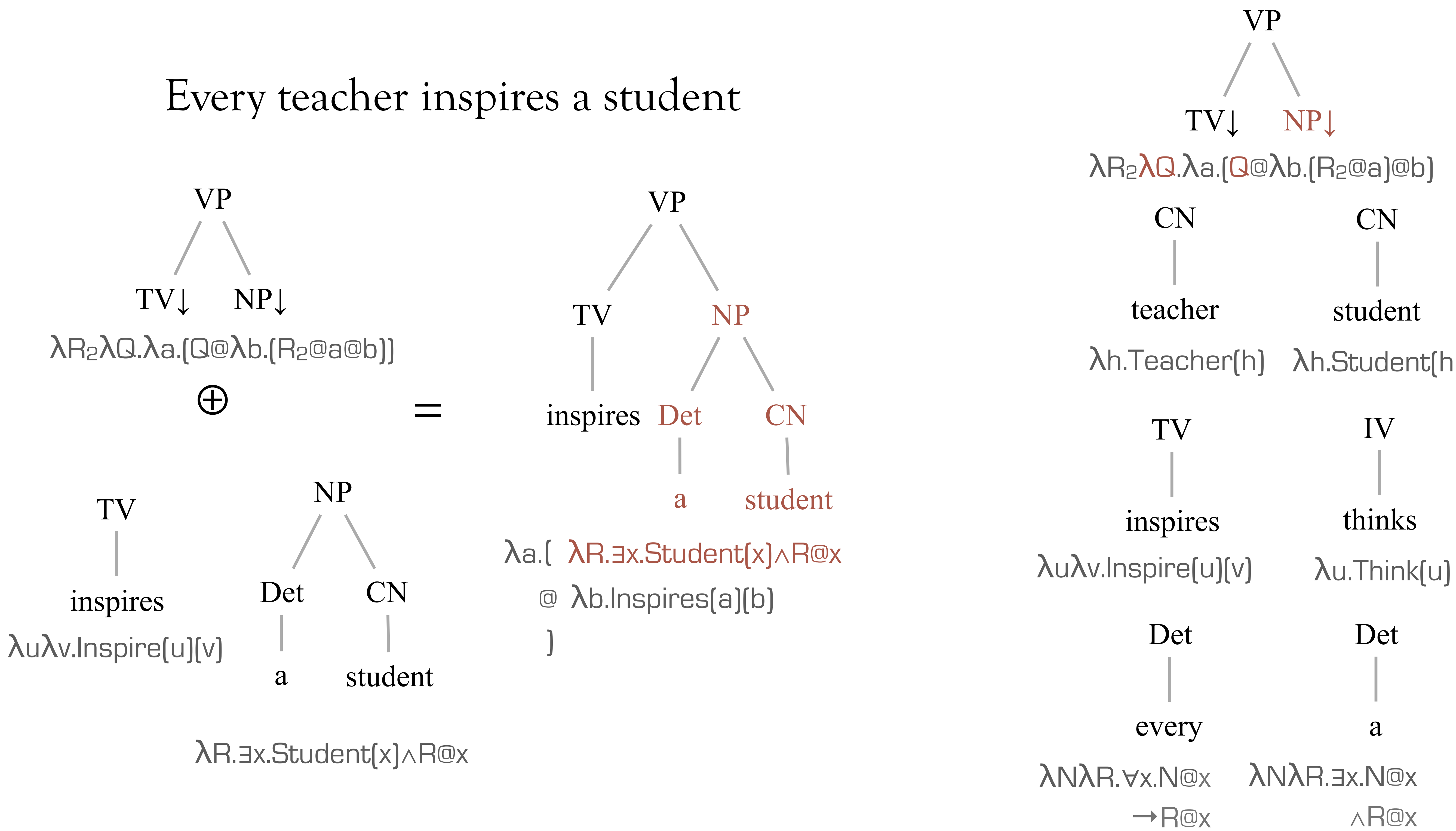
⟦inspires a student⟧ :: Rel1
⟦a student⟧ :: Rel1 → Bool
⟦inspires⟧ :: Entity → Entity → Bool
⟦inspires⟧ :: Entity → Rel1

```
vpTrans2 :: (Entity → Rel1) → (Rel1 → Bool) → Rel1
vpTrans2 r2 q = q r2 -- 😖 want Rel1 got (Entity → Rel1)
```

```
vpTrans3 :: (Entity → Rel1) → (Rel1 → Bool) → Rel1
vpTrans3 r2 q = λa → q (r2 a)
vpTrans3 r2 q = λa → q (λb → r2 a b) -- eta expanded
```

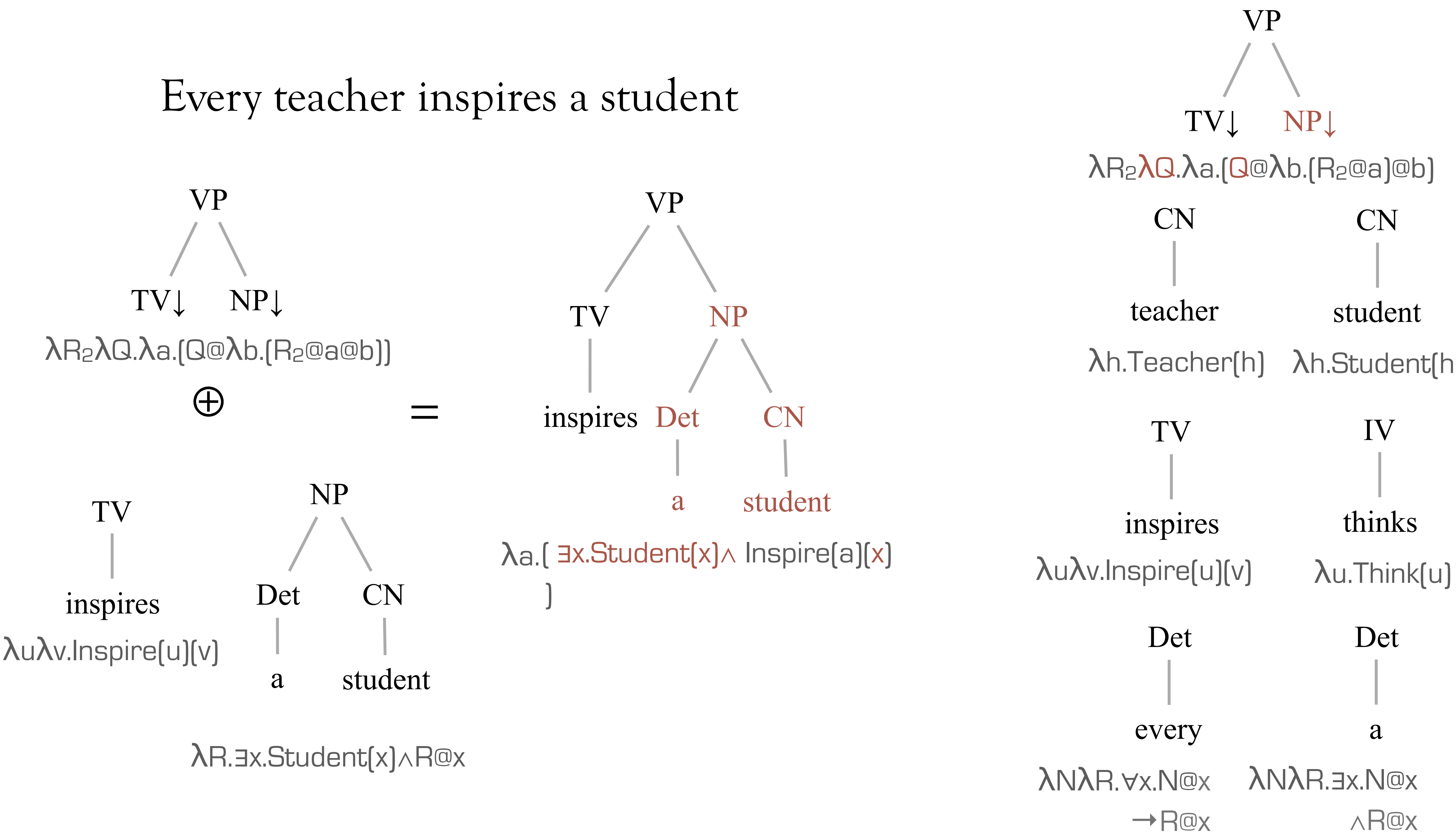
3. Think real hard

Transitive verbs



3. Think real hard

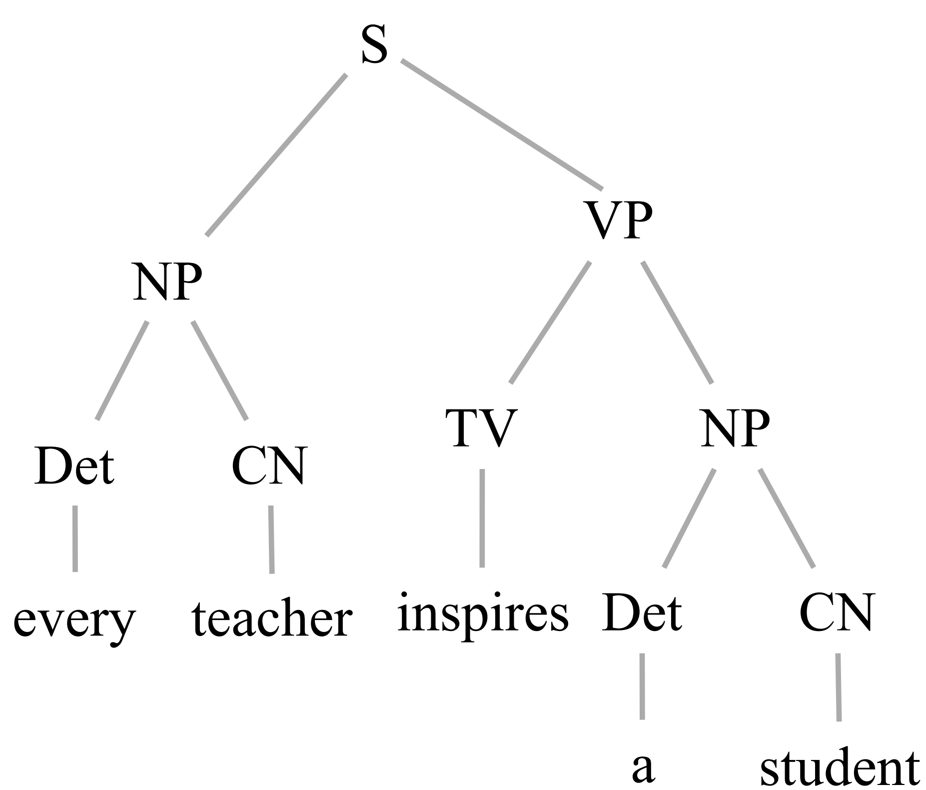
Transitive verbs



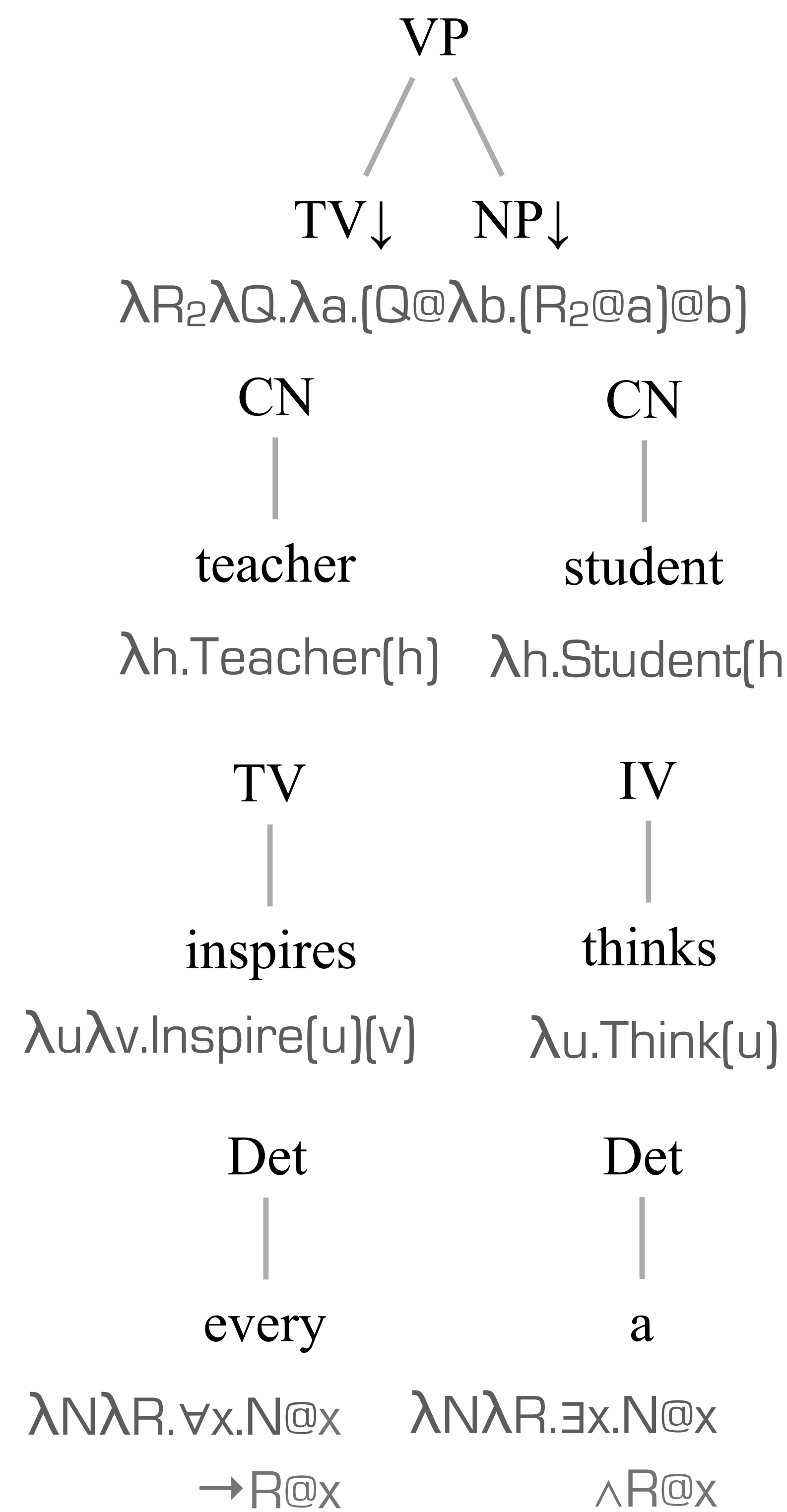
3. Think real hard

Transitive verbs

Every teacher inspires a student



$\forall y.Teacher(y) \rightarrow (\exists x.Student(x) \wedge Inspire(y)(x))$



Also covered in Montague 1973 😲

1. Pronouns
2. Tricky small words
    - meaning of “the”
    - meaning of “is”
3. Tense
4. Modal expressions
    - “possibly”
    - “necessarily”

Also covered in Montague 1973 🤯

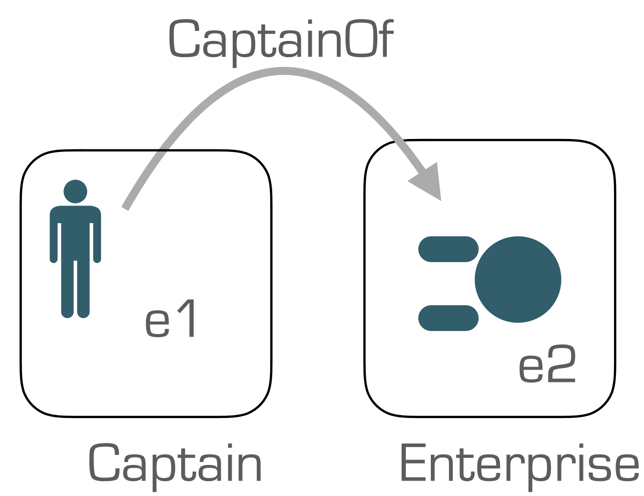
5. Scope ambiguities
   Every teacher inspires a student
   ...His name is Bart

6. Intension vs extension
   The Borg assimilated the captain of the Enterprise

7. Intensional verbs
   John finds a unicorn
   John seeks a unicorn
   John tries to find a unicorn

⟦the captain of the Enterprise⟧

intension

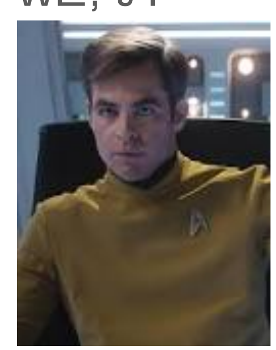


extension

w1, t1

w1, t2

w2, t1



Things to read

Representation and Inference for Natural Language

A First Course in Computational Semantics

Patrick Blackburn and Johan Bos

Teaching material

- Montague Semantics (Symbolic Systems 100 course)
  Christopher Potts
- Formal Semantics Teaching Material: https://people.umass.edu/partee/Teaching.htm
  Barbara Partee

Montague Papers

- English as a formal language (1970)
- Universal grammar (1970)
- The proper treatment of quantification in ordinary English (1973)

Things to read

Haskell + Natural Language

- Continuations: natural language meaning as computation
  Chris Barker and Chung-chieh Shan
- Grammatical Framework
  Aarne Ranta
- Haskell Wiki on Linguistics
  https://wiki.haskell.org/Applications_and_libraries/Linguistics