Semantic Theory
Lecture 7: Lambda Calculus and NL Quantifiers

Manfred Pinkal
FR 4.7 Computational Linguistics and Phonetics

Summer 2014

Lambda Abstraction

- Syntax:

  If $\alpha \in WE\tau$ and $v \in VAR_\sigma$, then $\lambda v \alpha \in WE_{\langle\sigma,\tau\rangle}$.

- Interpretation:

  $[\![\lambda v \alpha]\!]^{M,g}$ is that function $f : D_\sigma \rightarrow D_\tau$ such that

  for all $a \in D_\sigma$, $f(a) = [\![\alpha]\!]^{M,g[v/a]}$ (for $\alpha \in WE_\tau$, $v \in VAR_\sigma$)

Example

- *Someone drinks and drives*

  ➔ someone’(λx(drink’(x) ∧ drive’(x))

- $[\![\text{someone'}(\lambda x(\text{drink'}(x) \wedge \text{drive'}(x)))]\!]^{M,g} =$

  $[\![\text{someone'}]\!]^{M,g}([\![\lambda x(\text{drink'}(x) \wedge \text{drive'}(x))]\!]^{M,g})$

- $[\![\text{someone'}]\!]^{M,g}(S) = 1$ iff $S \neq \emptyset$

- $[\![\lambda x(\text{drink'}(x) \wedge \text{drive'}(x))]\!]^{M,g} = V_M(\text{drink'}) \cap V_M(\text{drive'})$

- $[\![\text{someone'}]\!]^{M,g}([\![\lambda x(\text{drink'}(x) \wedge \text{drive'}(x))]\!]^{M,g}) = 1$

  iff $V_M(\text{drink'}) \cap V_M(\text{drive'}) \neq \emptyset$

Example, Continued

- If the λ-expression is applied to an overt argument, we can simplify the interpretation:

  $[\![\lambda v \alpha(\beta)]\!]^{M,g} = [\![\alpha]\!]^{M,g[v/[\![\beta]\!]^{M,g}]}$

- Example:

  $[\![\lambda x(\text{drink}'(x) \wedge \text{drive}'(x))(j^*)]\!]^{M,g} = 1$

  iff $[\![\text{drink}'(\mathbf{x}) \wedge \text{drive}'(\mathbf{x})]\!]^{M,\mathbf{g[x/[\![j^*]\!]^{M,g}]}} = 1$

  iff $V_M(j^*) \in V_M(\text{drink}') \cap V_M(\text{drive}')$

Lambda Conversion

- $[\![\lambda v \alpha(\beta)]\!]^{M,g} = [\![\alpha]\!]^{M,g[v/[\![\beta]\!]^{M,g}]}$

  $= [\![[\beta/v]\alpha]\!]^{M,g}$

- **$\lambda v \alpha(\beta) \Leftrightarrow [\beta/v]\alpha$**

Lambda Conversion: Examples

- *drinks and drives* ➔ λx[drink‘(x)∧drive‘(x)] : ⟨e,t⟩
- *John* ➔ j*:e
- *John drinks and drives* ➔ λx[drink‘(x)∧drive‘(x)](j*) : t

  ⇔ [j*/x] drink‘(x)∧drive‘(x)

  = drink‘(j*) ∧ drive‘(j*)
- *John* ➔ λ*F*.*F*(j*): ⟨⟨e,t⟩,t⟩
- *John drinks and drives* ➔ λ*F*[*F*(j*)](λx[drink‘(x)∧drive‘(x)]) : t

  ⇔ λx[drink‘(x)∧drive‘(x)](j*)

  ⇔ drink‘(j*) ∧ drive‘(j*)

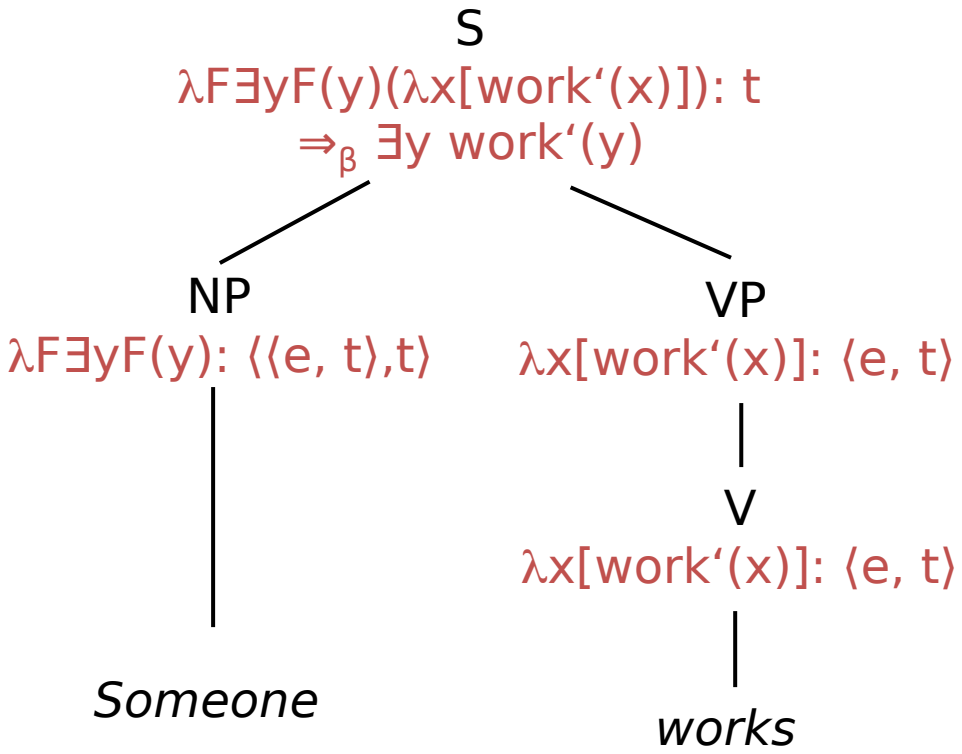
β-Reduction in Semantic Construction

- *someone* ➔ λ*F*∃y*F*(y)
- *work* ➔ λx[work‘(x)]
- *someone works*

  ➔ λ*F*∃y*F*(y)(λx [work‘(x)])

  ⇔$_\beta$ ∃yλx[work‘(x)](y)

  ⇔$_\beta$ ∃y [work‘(y)]



Variable Capturing

- Are λvα(β) and [β/v]α always equivalent?
  - λx[drive’(x) ∧ drink’(x)](j*) ⇔ drive’(j*) ∧ drink’(j*)
  - λx[drive’(x) ∧ drink’(x)](y) ⇔ drive’(y) ∧ drink’(y)
  - λx[∀y know’(x)(y)](j*) ⇔ ∀y know(j*)(y)
  - **NOT:** λx[∀y know’(x)(y)](y) ⇔ ∀y know(y)(y)
- Let v, v’ be variables of the same type, α any well-formed expression:

  v’ is free for v in α iff no free occurrence of v in α is in the scope of a quantifier or a λ-operator that binds v’.

Lambda Calculus: Conversion Rules

- **β-conversion:** λvα(β) ⇔ [β/v]α
  - if all free variables in β are free for v in α.
- **α-conversion:** λvα ⇔ λw[w/v]α
  - if w is free for v in α.
- **η-conversion:** λv(α(v)) ⇔ α
- In semantic construction, the by far most important rule is β-conversion, and its only important direction left-to-right, called **β-reduction**. λ-conversion, β-conversion and β-reduction are used as quasi-synonyms (in NL semantics).

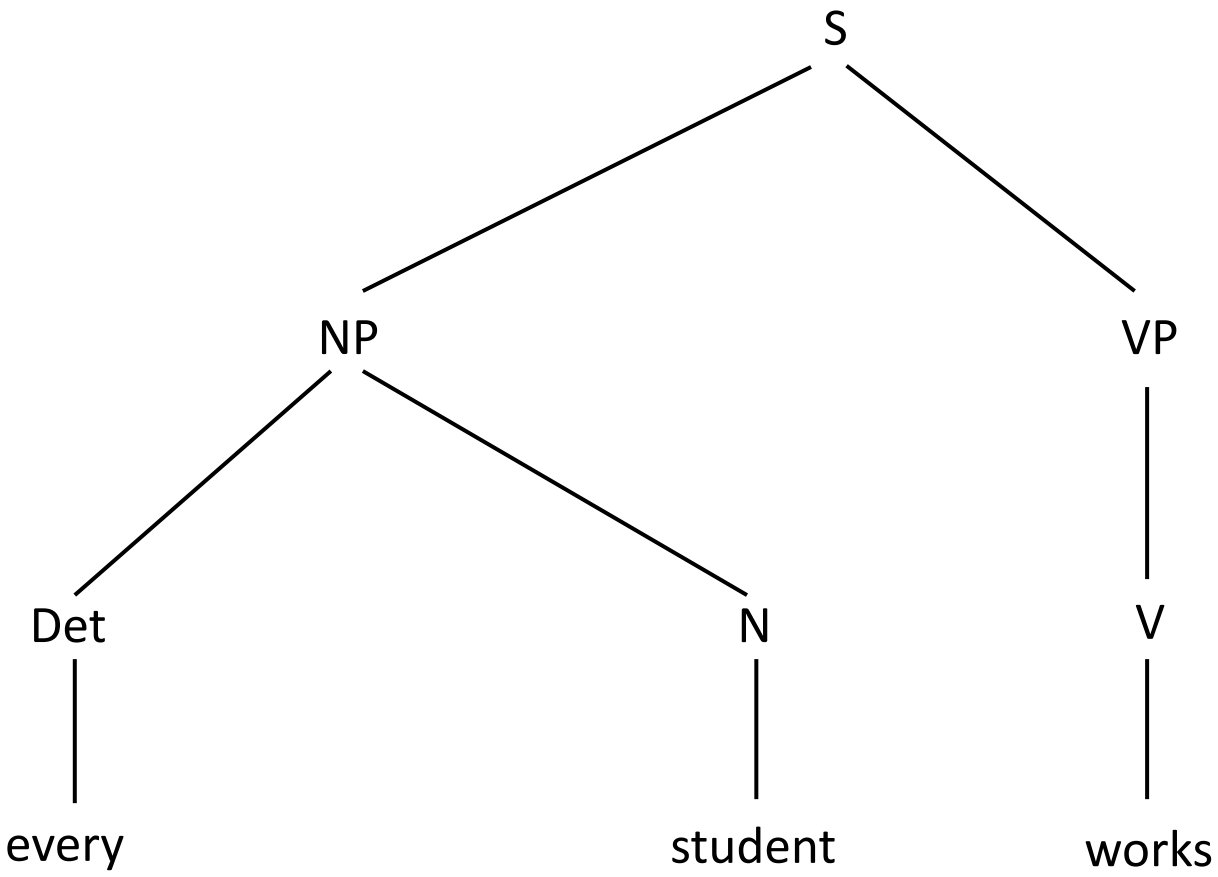
Determiners as λ-Expressions

- *a student* → $\lambda P \exists x(\text{student}'(x) \wedge P(x)) : \langle\langle e,t\rangle,t\rangle$
- *a, some* → $\lambda F \lambda G \exists x(F(x) \wedge G(x)) : \langle\langle e,t\rangle,\langle\langle e,t\rangle,t\rangle\rangle$

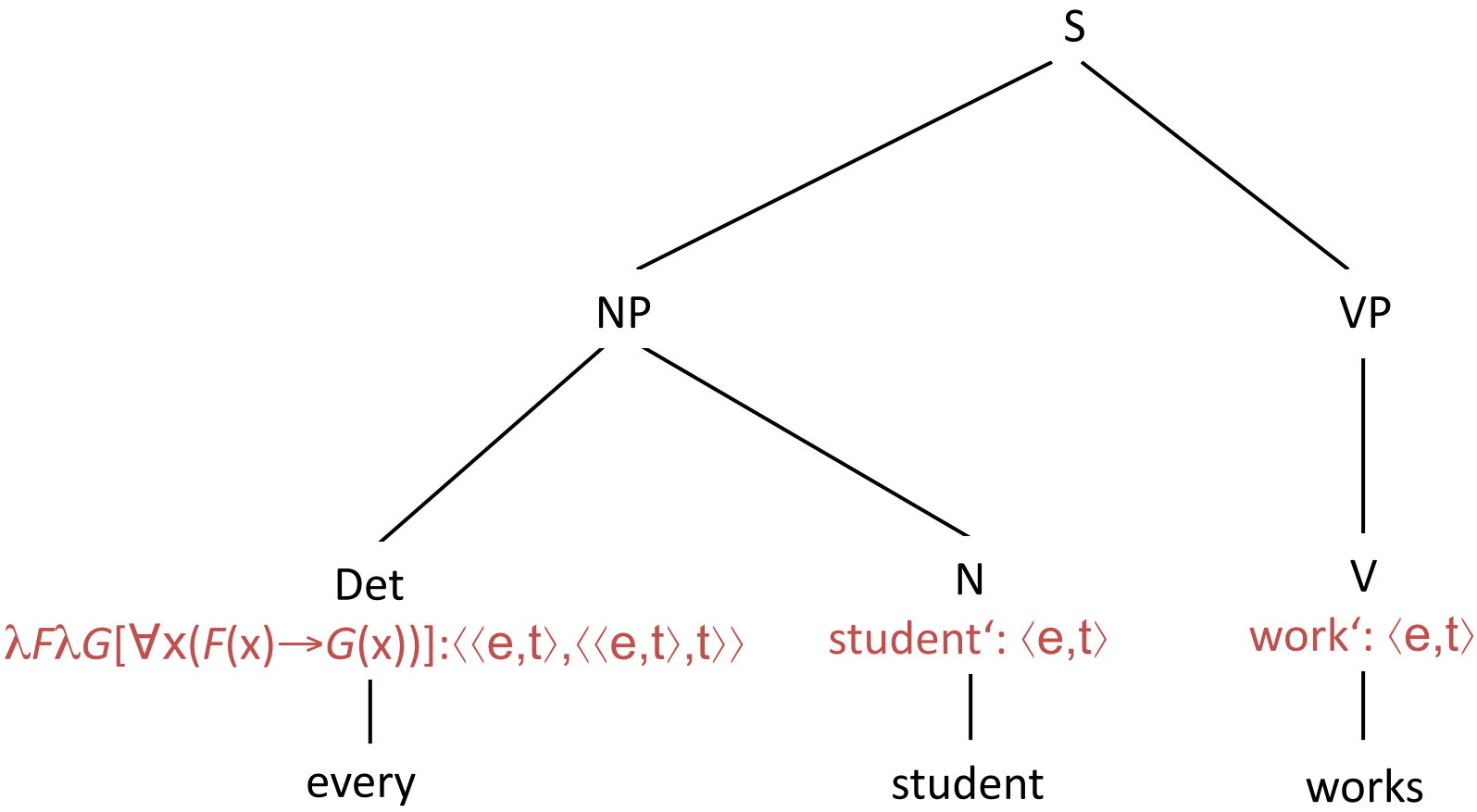
- *every student* → $\lambda P \forall x(\text{student}'(x) \rightarrow P(x)) : \langle\langle e,t\rangle,t\rangle$
- *every* → $\lambda F \lambda G \forall x(F(x) \rightarrow G(x)) : \langle\langle e,t\rangle,\langle\langle e,t\rangle,t\rangle\rangle$

- *no* → $\lambda F \lambda G \neg\exists x(F(x) \wedge G(x)) : \langle\langle e,t\rangle,\langle\langle e,t\rangle,t\rangle\rangle$

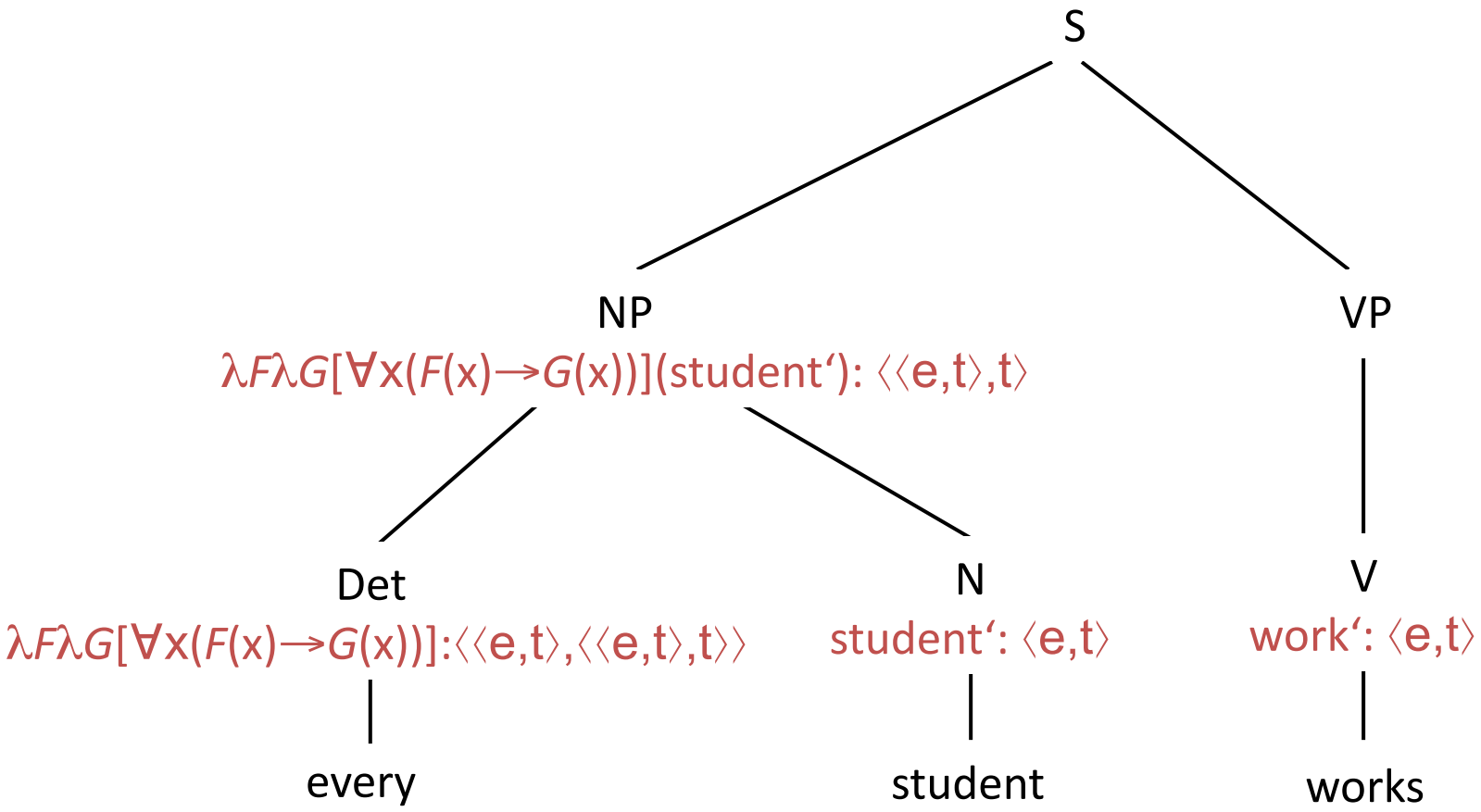
Semantic Construction: An Example



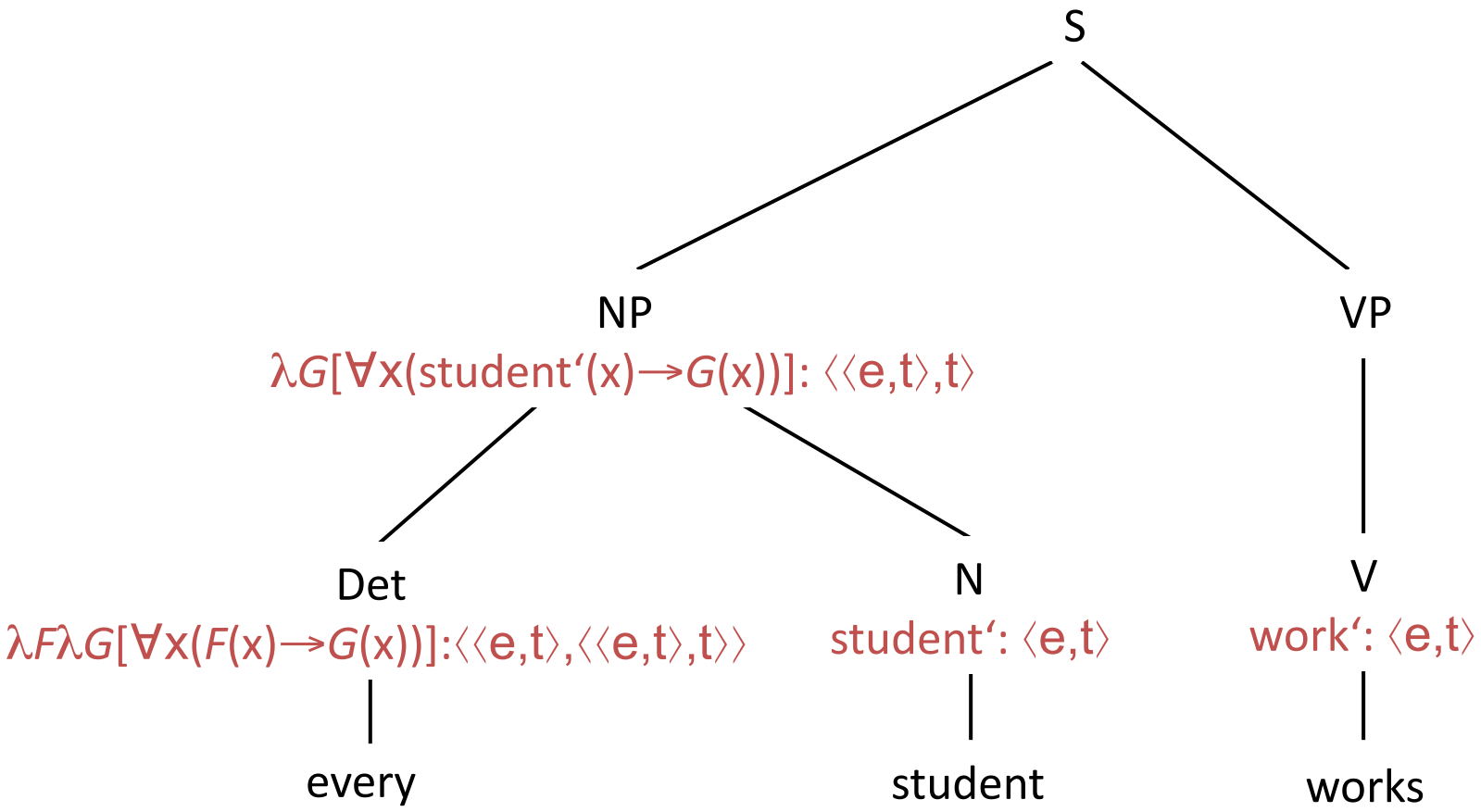
Semantic Construction: Lexicon



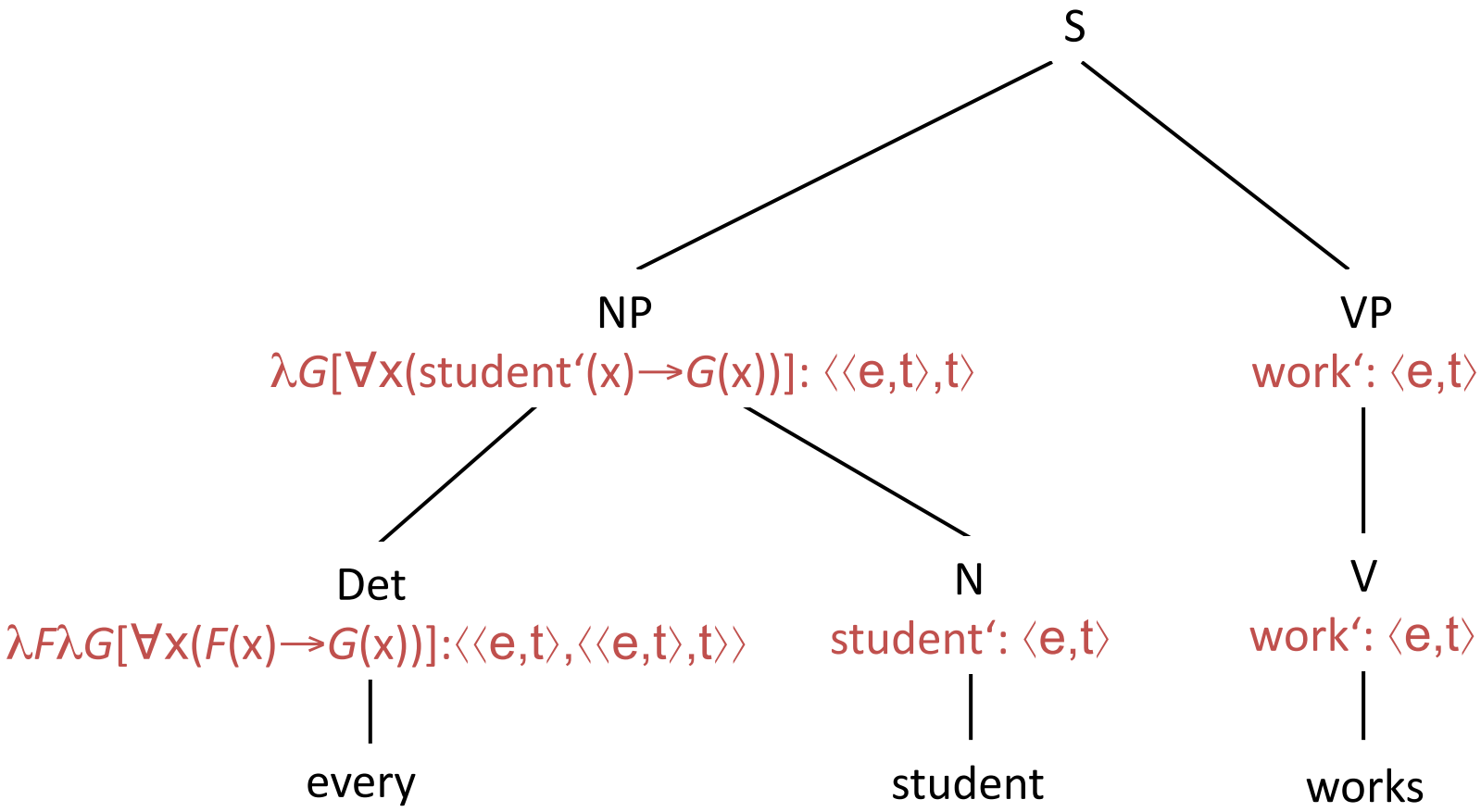
Semantic Construction: Application



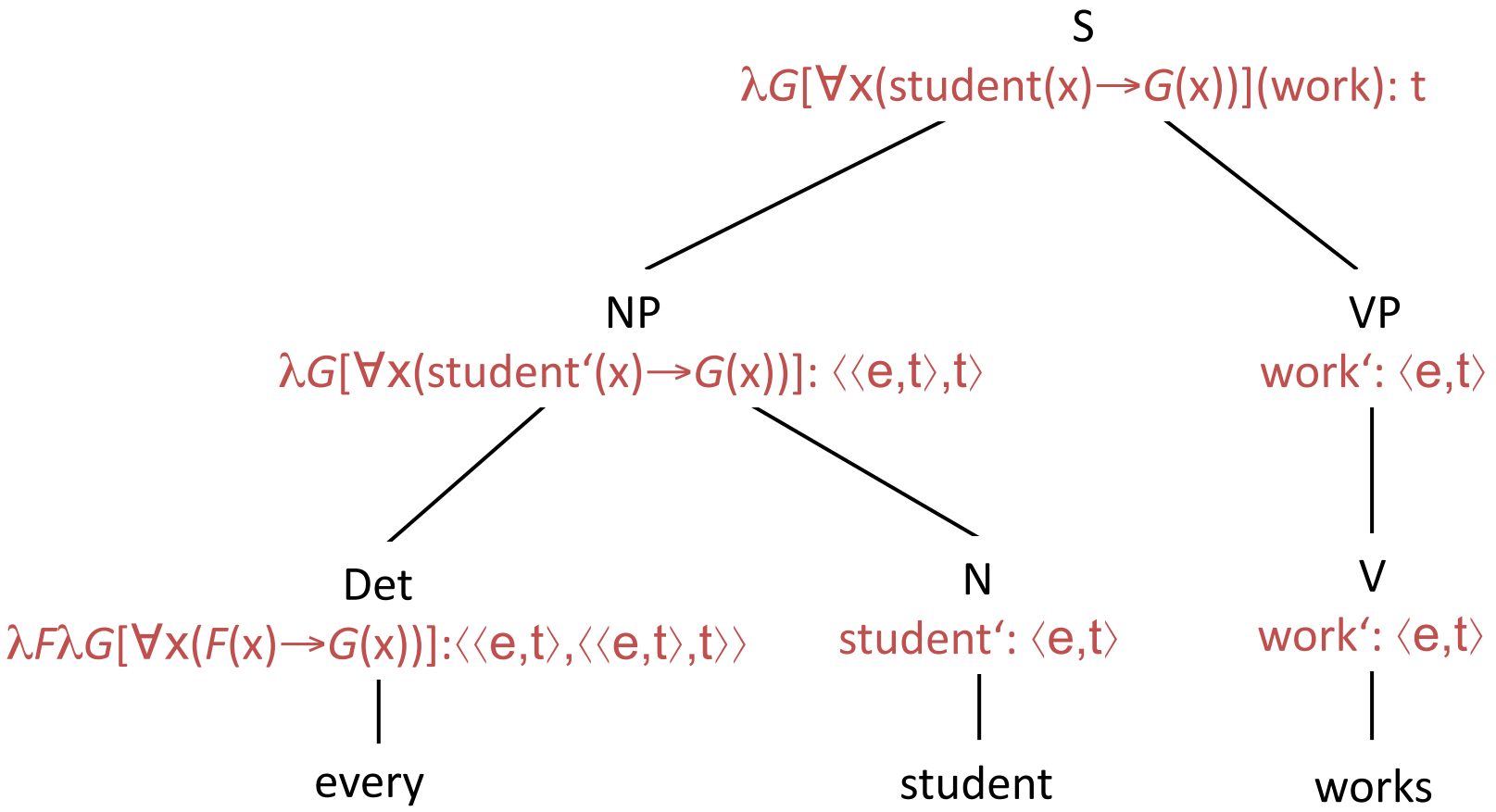
Semantic Construction: Reduction



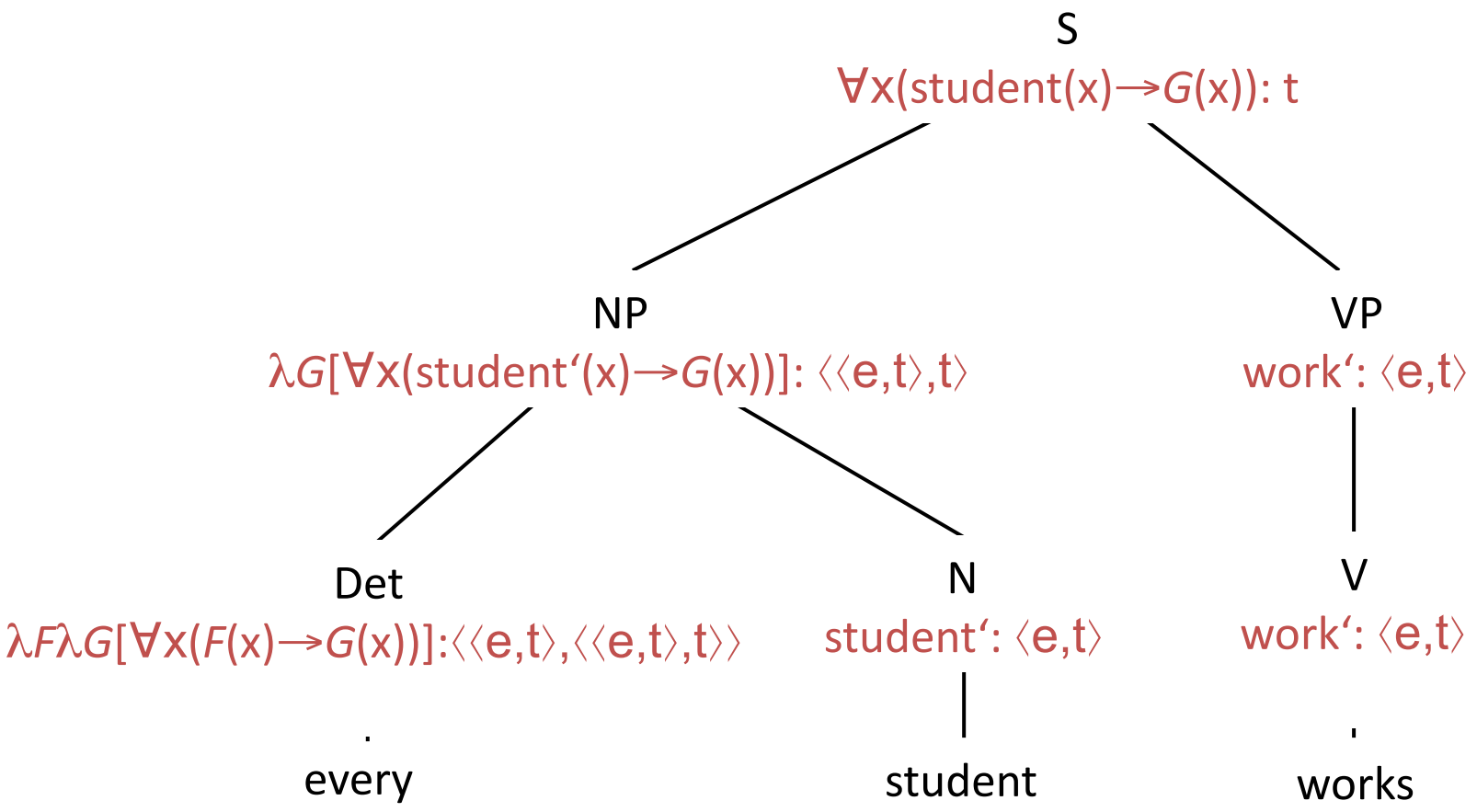
Semantic Construction: Projection



Semantic Construction: Application



Semantic Construction: Reduction



Transitive Verbs: Type Clash

Every student presented a paper

present': **⟨e, ⟨e,t⟩⟩** a-paper': **⟨⟨e,t⟩,t⟩**

every-student': ⟨⟨e,t⟩,t⟩ **?:?**

?: t

Solution: reverse functor-argument relation (again):

present' $\in$ $CON_{\langle\langle\langle e,t\rangle,t\rangle, \langle e,t\rangle\rangle}$

Transitive Verbs (1)

- *read* ➔ read’ $\in$ $CON_{\langle\langle e,t\rangle, t\rangle, \langle e, t\rangle\rangle}$
- *read a book* ➔ read’($\lambda P \exists y$(book’(y) $\wedge$ P(y)) $\in$ $WE_{\langle e, t\rangle}$
- *every student reads a book*

  ➔ $\lambda R \forall x$(student’(x) $\rightarrow$ R(x))(read’($\lambda P \exists y$(book’(y) $\wedge$ P(y)))

  $\Leftrightarrow_{\beta}$ $\forall x$(student’(x) $\rightarrow$ read’($\lambda P \exists y$(book’(y) $\wedge$ P(y)))(x))
- Inappropriate analysis for standard transitive verbs. The analysis doesn’t support the entailment:

  Every student reads a book $\models$ *There are books*
- **Solution:** Use a more explicit λ-term to translate *read.*

Transitive Verbs (2)

- *read* → $\lambda Q \lambda z Q(\lambda x(\text{read}^*(x)(z))) \in \text{WE}_{\langle\langle e,t\rangle, t\rangle, \langle e, t\rangle\rangle}$
  - $\text{read}^* \in \text{WE}_{\langle e, \langle e, t\rangle\rangle}$ is the "underlying" first-order relation

- *read a book* → $\lambda Q \lambda z Q(\lambda x(\text{read}^*(x)(z)))(\lambda P \exists y(\text{book}'(y) \wedge P(y)))$

$\Leftrightarrow_\beta \lambda z(\lambda P \exists y(\text{book}'(y) \wedge P(y))(\lambda x(\text{read}^*(x)(z))))$

$\Leftrightarrow_\beta \lambda z(\exists y(\text{book}'(y) \wedge \lambda x(\text{read}^*(x)(z))(y)))$

$\Leftrightarrow_\beta \lambda z(\exists y(\text{book}'(y) \wedge \text{read}^*(y)(z)))$

Transitive Verbs (3)

- *read a book*
  - ➔ λz∃y(book'(y) ∧ read*(y)(z))
- *every student*
  - ➔ λR∀x(student'(x) → R(x))
- *every student reads a book*
  - ➔ λR∀x(student'(x) → R(x))(λz∃y(book'(y) ∧ read*(y)(z)))
  - ⇔$_\beta$ ∀x(student'(x) → λz∃y(book'(y) ∧ read*(y)(z))(x))
  - ⇔$_\beta$ **∀x(student'(x) → ∃y(book'(y) ∧ read*(y)(x)))**

Non-Referential Arguments

- *John seeks a unicorn* $\not\models$ *There is a unicorn*
- The higher-order analysis of transitive verbs covers non-referential verbs such as *seek:*
- *seek* ➔ seek' $\in$ $CON_{\langle\langle e,t\rangle, t\rangle, \langle e, t\rangle\rangle}$
- *seek a unicorn* ➔ seek'($\lambda P \exists y$(unicorn'(y) $\wedge$ P(y)) $\in$ $WE_{\langle e, t\rangle}$
- *John seeks a unicorn*

  ➔ $\lambda F[F(j^*)]$(seek'($\lambda P \exists y$(unicorn'(y) $\wedge$ P(y)))

  $\Leftrightarrow_\beta$ seek'($\lambda P \exists y$(unicorn'(y) $\wedge$ P(y)))(j^*)
- No seek*-based translation in this case, therefore no existential entailment.