Lecture 9: Scope

L98: Introduction to Computational Semantics

Weiwei Sun

Department of Computer Science and Technology
University of Cambridge

Michaelmas 2024/25

Every cat loves a cat.

Lecture 9: Scope

1. What is scope?
2. Type-driven analysis
3. Generalised quantifiers

What Is Scope?

Scope

$$\forall x(\text{cat'}(x) \rightarrow \exists y(\text{cat'}(y) \wedge \text{love'}(x, y)))$$

$$\exists y(\text{cat'}(y) \wedge \forall x(\text{cat'}(x) \rightarrow \text{love'}(x, y)))$$

Scope is an effect in syntax and semantics

- where a scopal lexical item casts its semantic effect over a particular part of the clause or phrase
- the entire part of the clause is then said to be in the scope of the scopal element

Types of scope

- negative scope
- modal scope
- "only" scope
- comparative scope
- contrastive scope (rather than)
- hypothetical scope
- attributive scope (she said that...)
- quotation scope (so-called...)
- ...

"Only" scope

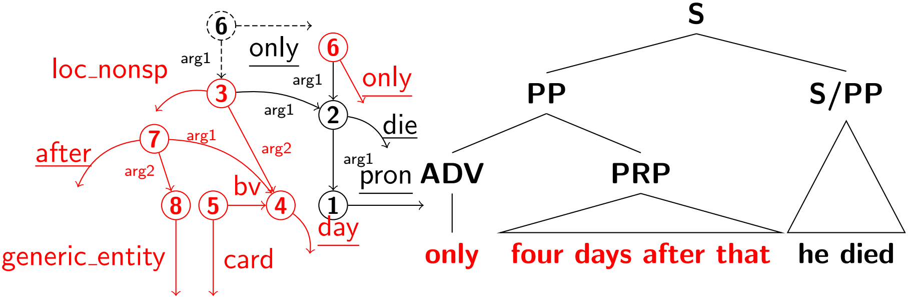
(1) a. Kim loved her cats.

b. Only Kim loved her cats.

c. Kim only loved her cats.

d. Kim loved only her cats.

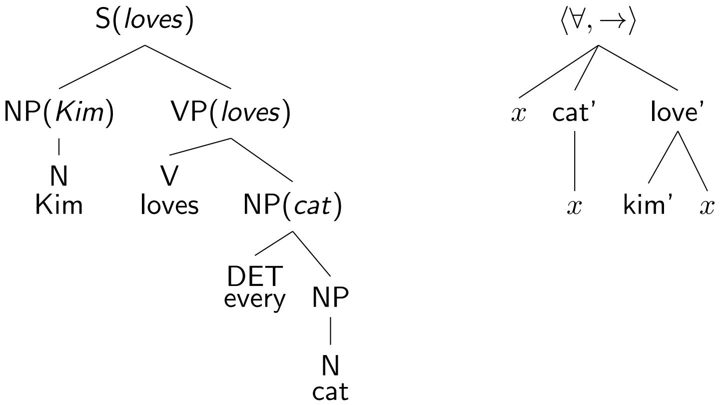
How to identify scope in graph?



Semantic graphs are not hierarchical, therefore we can't use a single node to identify scope.

Syntax–semantics mismatch

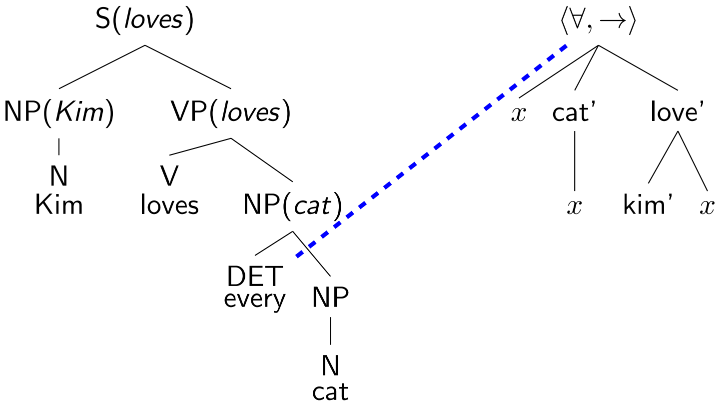
(2) Kim loves every cat $\triangleright \forall x(\text{cat'}(x) \rightarrow \text{love'}(\text{kim'}, x))$



An alternative analysis of noun phrases: DET is the syntactic head.

Syntax–semantics mismatch

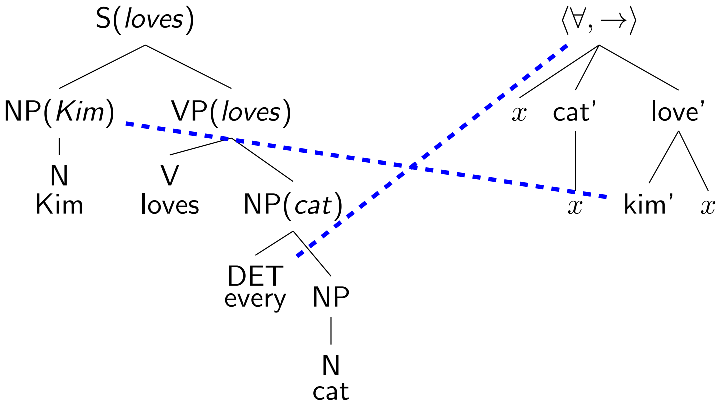
(2) Kim loves every cat $\triangleright \forall x(\text{cat'}(x) \rightarrow \text{love'}(\text{kim'}, x))$



An alternative analysis of noun phrases: DET is the syntactic head.

Syntax–semantics mismatch

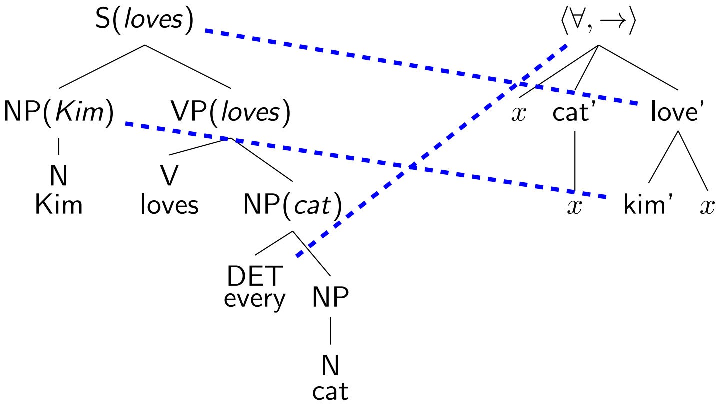
(2) Kim loves every cat $\triangleright \forall x(\text{cat'}(x) \rightarrow \text{love'}(\text{kim'}, x))$



An alternative analysis of noun phrases: DET is the syntactic head.

Syntax–semantics mismatch

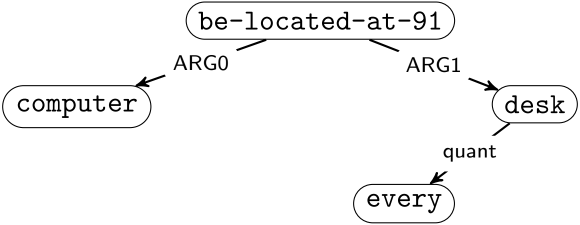
(2) Kim loves every cat $\triangleright \forall x(\text{cat'}(x) \rightarrow \text{love'}(\text{kim'}, x))$



An alternative analysis of noun phrases: DET is the syntactic head.

Three types of graphs

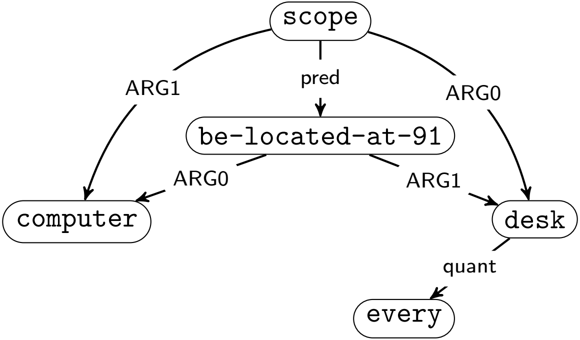
Abstract Meaning Representation



A computer is on every desk.

Three types of graphs

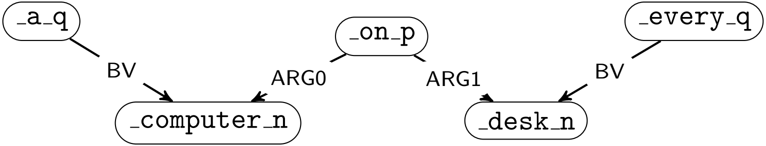
Abstract Meaning Representation



A computer is on every desk.

Three types of graphs

English Resource Semantics

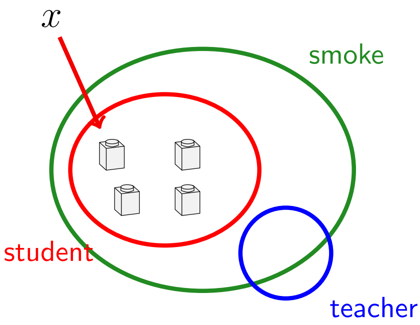


A computer is on every desk.

Type-Driven Analysis

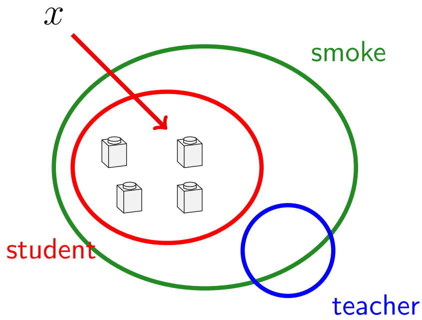
Quantification over individuals/sets

- What is ⟦*every student smokes*⟧? $\forall x(\text{student'}(x) \rightarrow \text{smoke'}(x))$
- What is ⟦*some students smoke*⟧? $\exists x(\text{student'}(x) \wedge \text{smoke'}(x))$



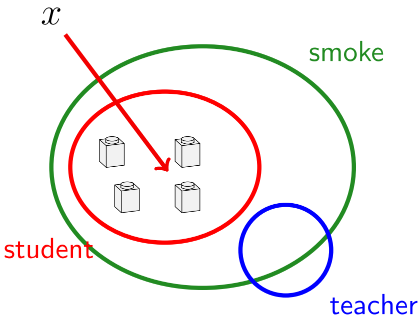
Quantification over individuals/sets

- What is ⟦*every student smokes*⟧? $\forall x(\text{student'}(x) \rightarrow \text{smoke'}(x))$
- What is ⟦*some students smoke*⟧? $\exists x(\text{student'}(x) \wedge \text{smoke'}(x))$



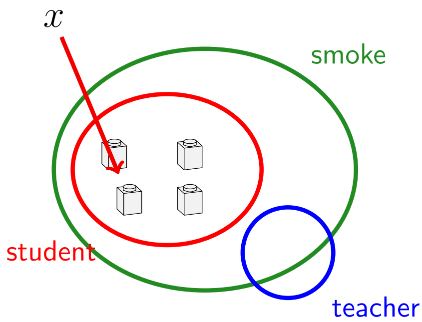
Quantification over individuals/sets

- What is ⟦*every student smokes*⟧? $\forall x(\text{student'}(x) \rightarrow \text{smoke'}(x))$
- What is ⟦*some students smoke*⟧? $\exists x(\text{student'}(x) \wedge \text{smoke'}(x))$



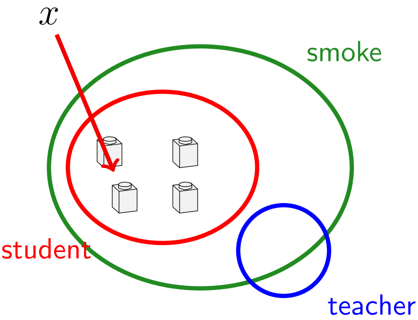
Quantification over individuals/sets

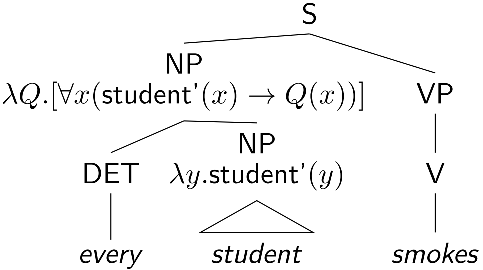
- What is ⟦*every student smokes*⟧? $\forall x(\text{student'}(x) \rightarrow \text{smoke'}(x))$
- What is ⟦*some students smoke*⟧? $\exists x(\text{student'}(x) \wedge \text{smoke'}(x))$



Quantification over individuals/sets

- What is ⟦*every student smokes*⟧? $\forall x(\text{student'}(x) \rightarrow \text{smoke'}(x))$
- What is ⟦*some students smoke*⟧? $\exists x(\text{student'}(x) \wedge \text{smoke'}(x))$



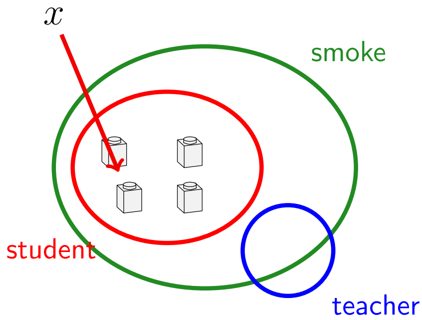


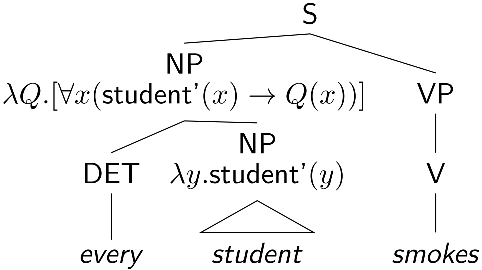
$$[\![\textit{every}]\!] = \lambda P.[\lambda Q.[\forall x(P(x) \rightarrow Q(x))]]$$

$$[\![\textit{some}]\!] = \lambda P.[\lambda Q.[\exists x(P(x) \wedge Q(x))]]$$

Quantification over individuals/sets

- What is ⟦*every student smokes*⟧? $\forall x(\text{student'}(x) \rightarrow \text{smoke'}(x))$
- What is ⟦*some students smoke*⟧? $\exists x(\text{student'}(x) \wedge \text{smoke'}(x))$



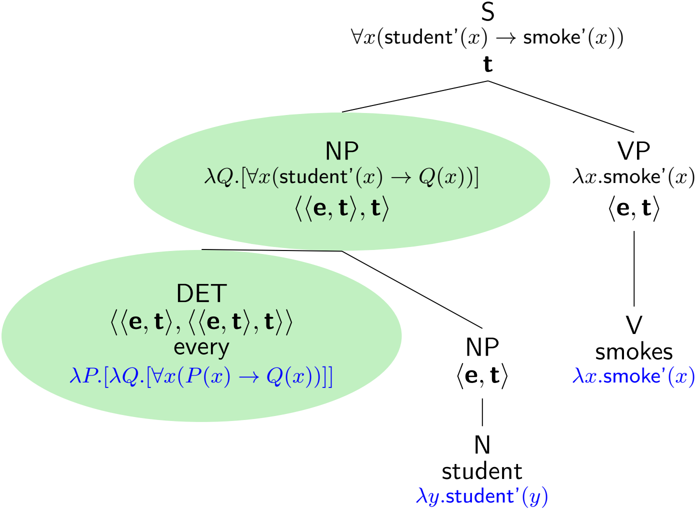


$$[\![\textit{every}]\!] = \lambda P.[\lambda Q.[\forall x(P(x) \rightarrow Q(x))]]$$

$$[\![\textit{some}]\!] = \lambda P.[\lambda Q.[\exists x(P(x) \wedge Q(x))]]$$

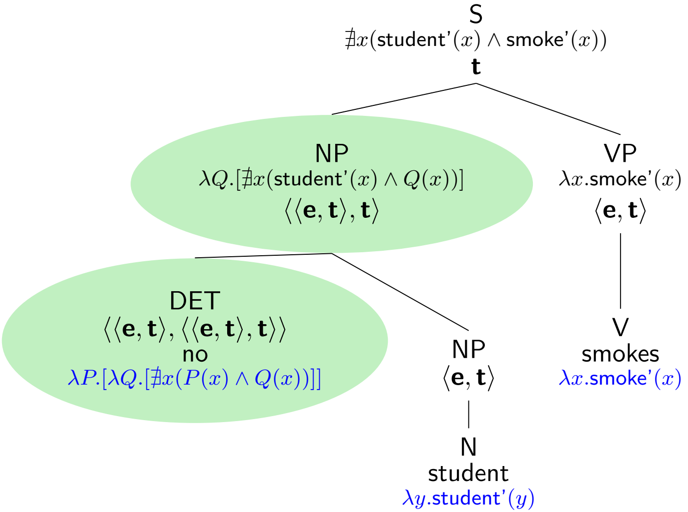
In order to do what they need to do (namely return a quantified NP of type $\langle\langle \mathbf{e}, \mathbf{t}\rangle, \mathbf{t}\rangle$), such quantifiers must be of type $\langle\langle \mathbf{e}, \mathbf{t}\rangle, \langle\langle \mathbf{e}, \mathbf{t}\rangle, \mathbf{t}\rangle\rangle$, which indicates that a quantifier identifies a relation between two sets.

Analysis by function application (*every student*)



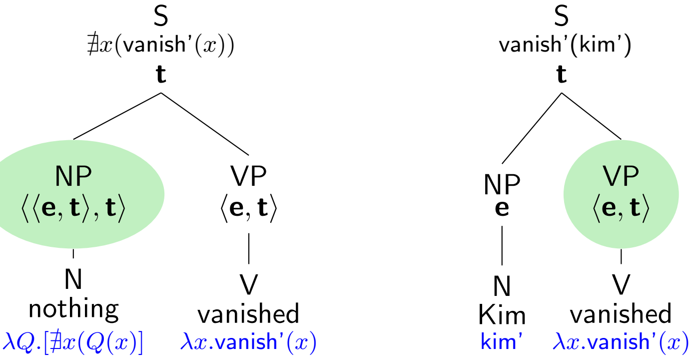
Only Function Application used

Analysis by function application (*no student*)



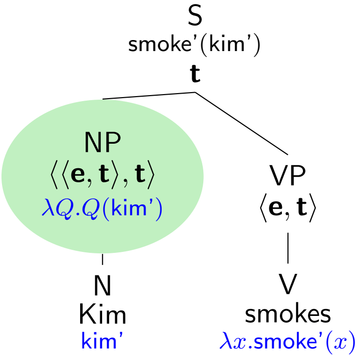
Only Function Application used

Nothing



FUNCTOR

Type raising



- Type raising is a unary rule.
- Type raising is systematic.
- Type shifting is more like a free change.
- Karl Marx: *human nature is formed by the totality of social relations*.

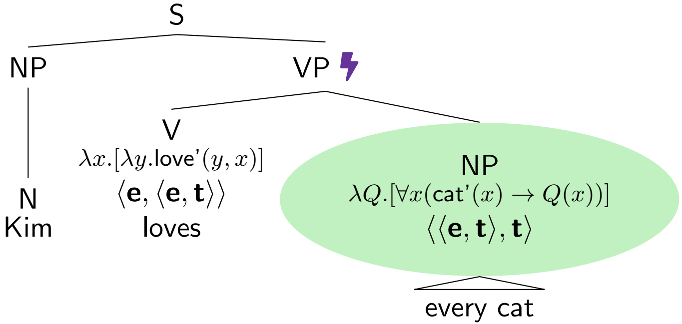
Semantic interpretation

$$\langle\langle \mathbf{e}, \mathbf{t}\rangle, \mathbf{t}\rangle$$

- *Every student* smokes.
  the bucket associated with *student* is the only element in the bucket associated with *every student*.
- Assume we have two students in our world model:

$$[\![\textit{every student}]\!] = \begin{bmatrix} \begin{bmatrix} t & \mapsto & 1 \\ j & \mapsto & 1 \end{bmatrix} & \mapsto & 1 \\ \begin{bmatrix} t & \mapsto & 1 \\ j & \mapsto & 0 \end{bmatrix} & \mapsto & 0 \\ \begin{bmatrix} t & \mapsto & 0 \\ j & \mapsto & 1 \end{bmatrix} & \mapsto & 0 \\ \begin{bmatrix} t & \mapsto & 0 \\ j & \mapsto & 0 \end{bmatrix} & \mapsto & 0 \end{bmatrix}$$

Problem with quantified NPs in object position



- Type mismatch
- VP: $\forall x(\text{cat'}(x) \rightarrow \lambda y.\text{love'}(y, x))$

Problem with quantified NPs in object position

$$\forall x(\text{cat'}(x) \rightarrow \text{love'}(\text{kim'}, x))$$

"slot" for the expected subject

"semantic material" corresponding to *every cat*

"semantic material" corresponding to *loves*

$[\![$*every cat*$]\!]$ is separated into two parts

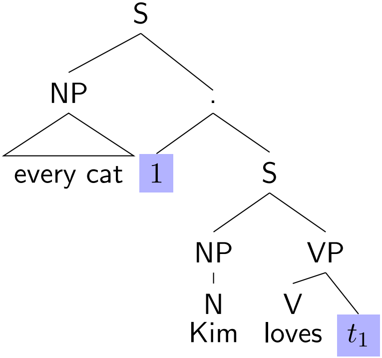
- an unbound variable x
- universal quantifier $\forall x(\text{cat'}(x) \rightarrow \ldots)$

We now need some heavy machinery

- Movement
- Traces
- Predicate abstraction rule for binding of traces
- Different shaped trees

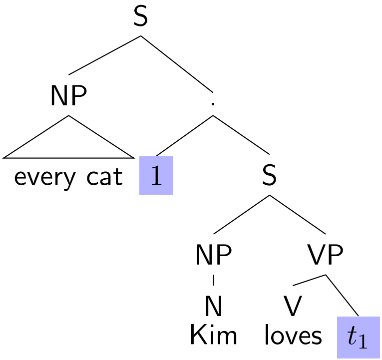
Movement and traces

What if in reality the tree looks like this:



Movement and traces

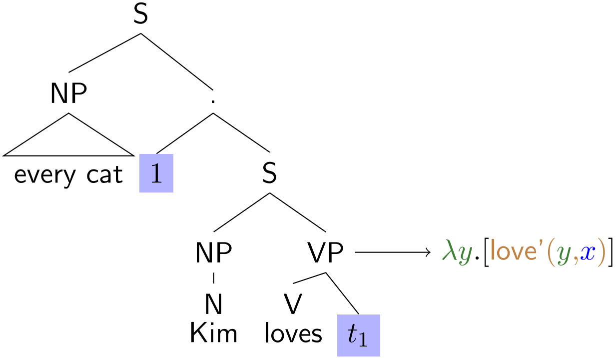
What if in reality the tree looks like this:



- When a constituent is moved, a trace (here: t_1) is left in its place. It's bound to its index (here: 1).

Movement and traces

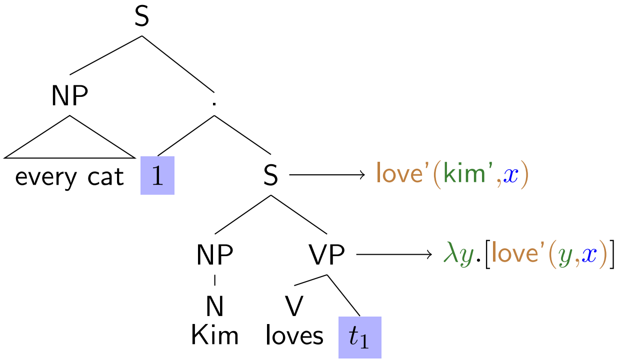
What if in reality the tree looks like this:



- When a constituent is moved, a trace (here: t_1) is left in its place. It's bound to its index (here: 1).

Movement and traces

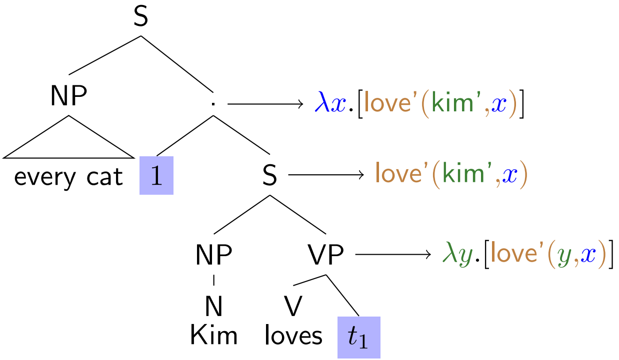
What if in reality the tree looks like this:



- When a constituent is moved, a trace (here: t_1) is left in its place. It's bound to its index (here: 1).

Movement and traces

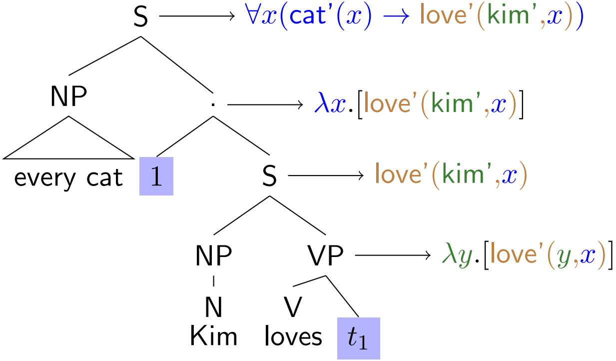
What if in reality the tree looks like this:



- When a constituent is moved, a trace (here: t_1) is left in its place. It's bound to its index (here: 1).
- What is the functionality of 1 ?
  Binding x – adding λx. This is function abstraction in λ-calculus.

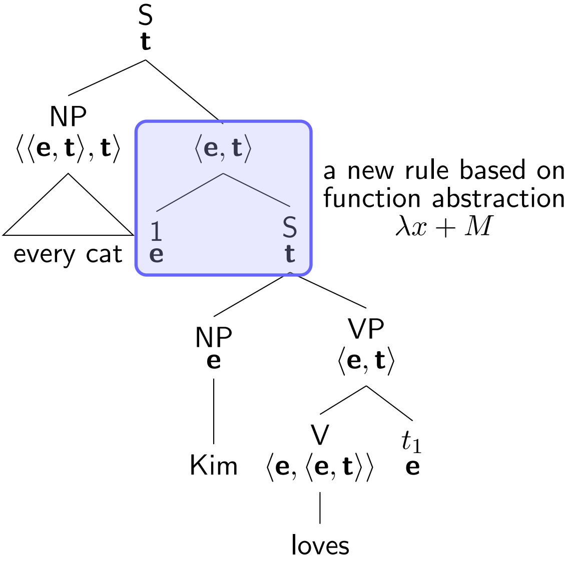
Movement and traces

What if in reality the tree looks like this:



- When a constituent is moved, a trace (here: t_1) is left in its place. It's bound to its index (here: 1).
- What is the functionality of 1 ?
  Binding x – adding λx. This is function abstraction in λ-calculus.

Now our types work out



Heim and Kratzer, p. 112 and chapter 5.4 on Variable binding

In-situ analysis vs. Movement analysis

- What we have just seen here is the movement analysis favoured by many Chomskyan Generative Linguists
- There is also an "in-situ" analysis
- In-situ means that the quantified NPs stay in their place
- The solution then involves two different types for quantified subject and object NPs
- Combinatory Categorial Grammar uses an in-situ analysis
- Minimal Recursion Semantics solves the problem with underspecification
- Contentious issue in Computational Linguistics
- Advantages and disadvantages for either

Generalised Quantifiers

Generalised quantifiers

- *At least three students* smoke.
  every bucket in the bucket associated with *at least three students* contains at least three students.
- *nothing*, *most*, *many*, *half* ...
- FOPL is not expressive enough.

A convenient notation

- $\forall x(\text{student'}(x) \rightarrow \text{smoke'}(x))$
- $\text{every'}(x, \text{student'}(x), \text{smoke'}(x))$
- $\exists x(\text{student'}(x) \wedge \text{smoke'}(x))$
- $\text{some'}(x, \text{student'}(x), \text{smoke'}(x))$

$$\text{at\_least\_three'}(x, \text{student'}(x), \text{smoke'}(x))$$

Truth conditions for generalized determiners

Determiner	Truth conditions
$[\![\textit{every}]\!](P)(Q)$	$P \subseteq Q$
$[\![\textit{some}]\!](P)(Q)$	$P \cap Q \neq \emptyset$
$[\![\textit{no}]\!](P)(Q)$	$P \cap Q = \emptyset$
$[\![\textit{three}]\!](P)(Q)$	$\|P \cap Q\| = 3$
$[\![\textit{less than three}]\!](P)(Q)$	$\|P \cap Q\| < 3$
$[\![\textit{at least three}]\!](P)(Q)$	$\|P \cap Q\| \geq 3$
$[\![\textit{most}]\!](P)(Q)$	$\|P \cap Q\| \geq \|P - Q\|$
$[\![\textit{few}]\!](P)(Q)$	$\|P \cap Q\| \ll \|P - Q\|$

Reading

- Heim and Kratzer (1999):
  - Chapter 6 and 7 for quantifiers and scope
  - Chapter 5 for traces and Predicate Abstraction