Grammar Implementation with Tree Adjoining Grammar: LTAG Semantics

Laura Kallmeyer, Simon Petitjean
Heinrich-Heine-Universität Düsseldorf

Winter 2015/2016

Overview

1. LTAG Semantics using STAG
2. Unification-Based LTAG Semantics
3. Generalized quantifiers
4. Quantifiers in STAG
5. Unification-based LTAG semantics and quantifiers

Introduction

Goal: an LTAG architecture of the syntax-semantics interface that

- is compositional: the meaning of a complex expression can be computed from the meaning of its subparts and its composition operation.
- allows to account for underspecification in quantifier scope

(1) Some student likes every course.

Two principal approaches:

1. unification based LTAG semantics [Kallmeyer and Joshi, 2003, Gardent and Kallmeyer, 2003, Kallmeyer and Romero, 2008]
2. LTAG semantics with synchronous TAG (STAG) [Shieber, 1994, Nesson and Shieber, 2006, Nesson and Shieber, 2008].

Introduction

Both use typed first-order predicate-logic with λ-abstraction for semantics with

- basic types e (= entity, individual), t (= truth value), and sometimes s (= situations/worlds).
- complex types $\langle \tau_1, \tau_2 \rangle$ built from types τ_1, τ_2
- finite sets of constants C_τ for every possible type τ, and
- sets of variables V_τ for basic types τ.

Formulas are built in the usual way, involving functional application (with appropriate types), binary connectives and negation (for types t) and quantifications $\forall$, $\exists$ and λ-abstraction.

The formulas get a standard model-theoretic interpretation.

Introduction

Observations:

- Elementary trees for lexcial predicates contain slots for all arguments of the predicate.
- These slots get filled via adjunction or substitution.

$\Rightarrow$ semantic representations are assigned to whole elementary trees (not to single nodes) and the adjunctions and substitutions determine semantic composition.

This holds for both approaches presented here.

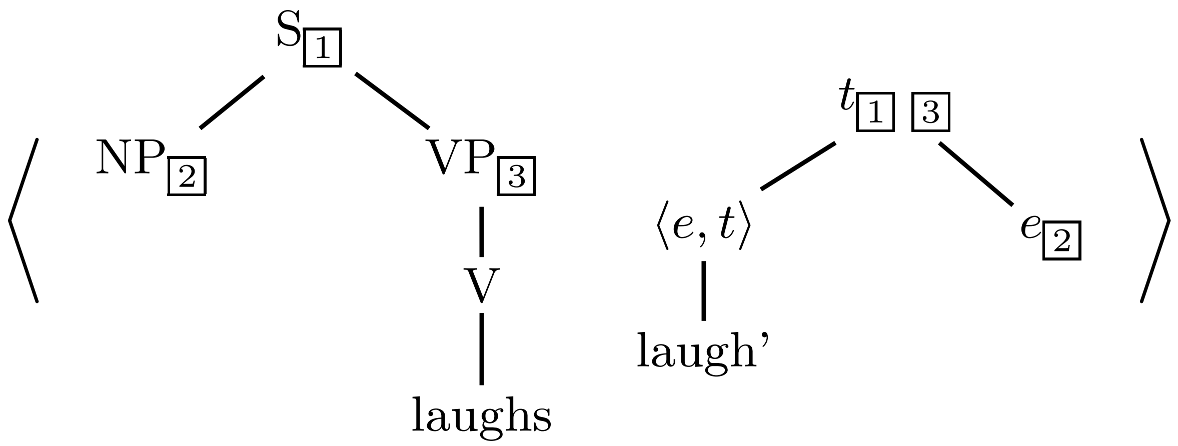
LTAG Semantics using STAG

Idea: pair two TAGs, one for syntax and one for L(ogical) F(orm) (= typed predicate logic), and do derivations in parallel.

Formalism used for this: *synchronous TAG (STAG)* [Shieber and Schabes, 1990, Shieber, 1994].

STAG = two TAGs G_1, G_2 whose trees are related to each other. More precisely, it contains pairs $\langle \gamma_1, \gamma_2, \textit{link} \rangle$ where γ_1 is an elementary tree from G_1, γ_2 an elementary tree from G_2, and *link* is a set of pairs of node addresses from γ_1 and γ_2 respectively.

LTAG Semantics using STAG



(The links are depicted with boxed numbers.)

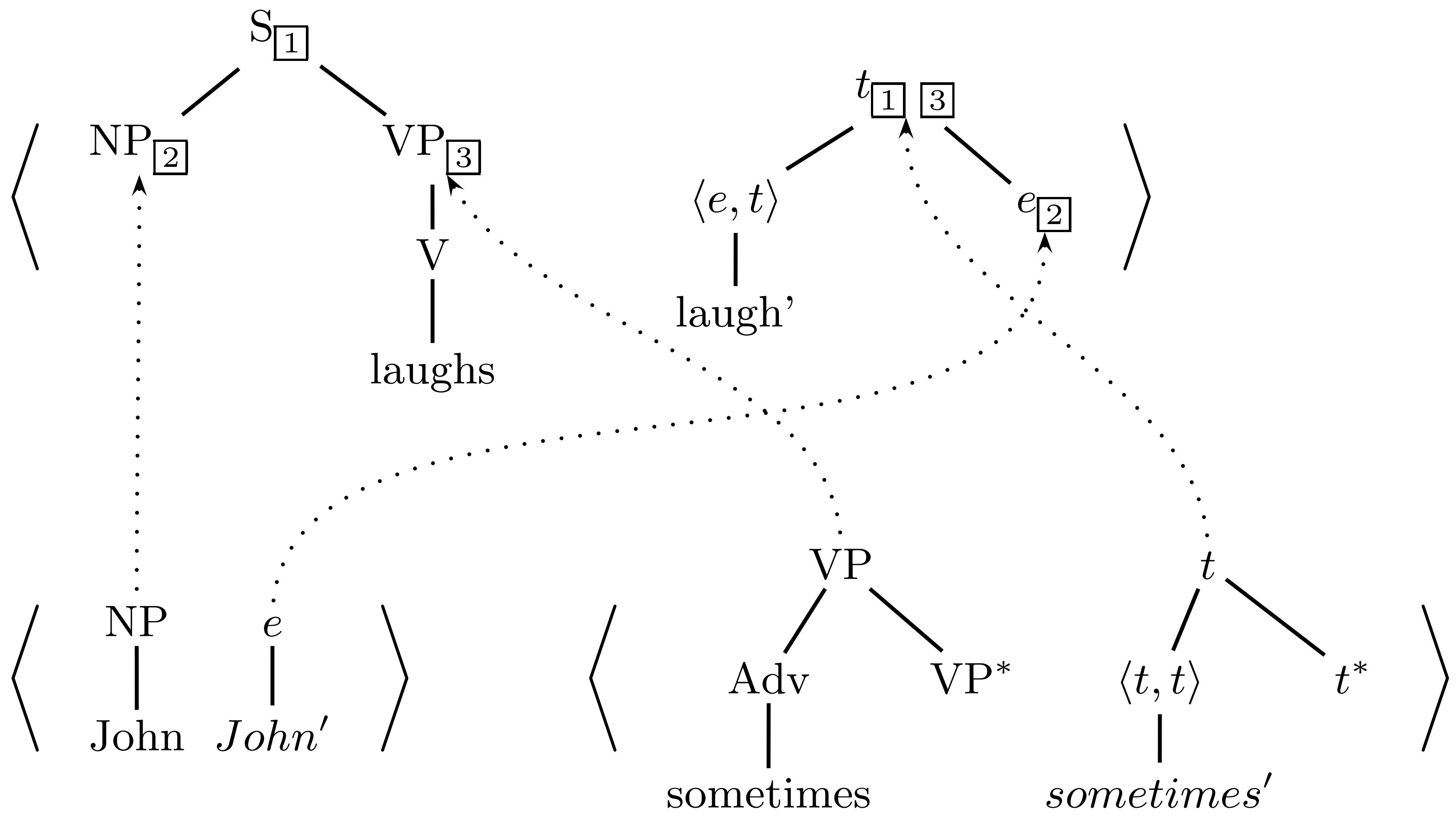
- The non-terminals of the semantic TAG are types $t, e, \langle e, t\rangle, \ldots$.
- The links in this example tell us, for instance, that the subject NP corresponds to the e argument of $laugh'$.

LTAG Semantics using STAG

STAG derivation proceeds as in TAG, except that all operations must be paired: In every derivation step:

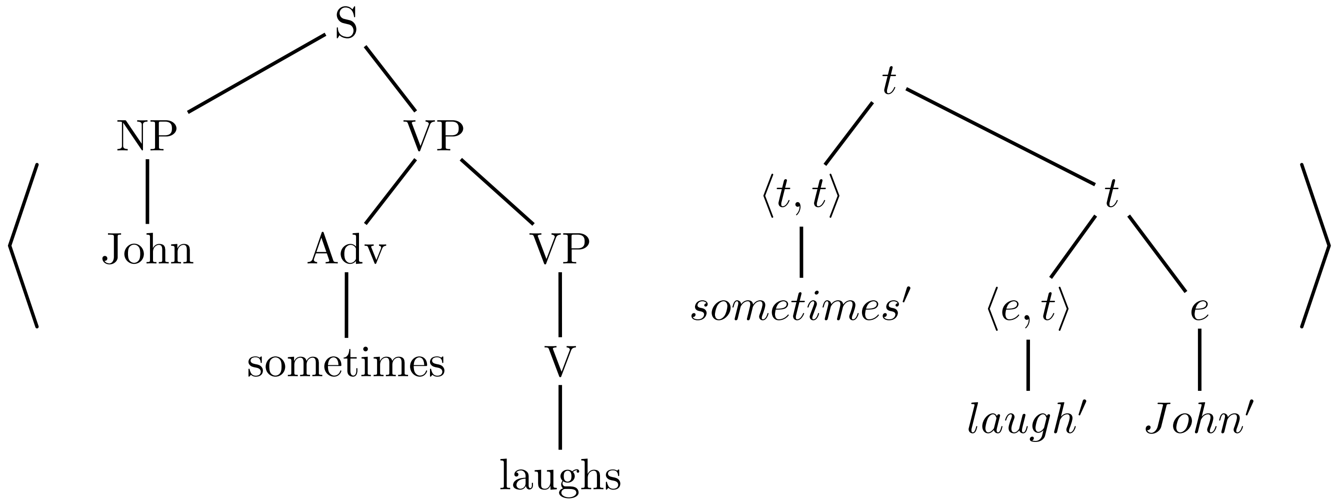
- A new elementary tree pair $\langle\gamma_1, \gamma_2\rangle$ is picked.
- γ_1 is attached (substituted or adjoined) to the syntactic tree while γ_2 is attached to the semantic tree.
- The nodes that the two trees attach to must be linked.
- The link that is used in this derivation step disappears while all other links involving the attachment sites are inherited by the root of the attaching tree.

LTAG Semantics using STAG

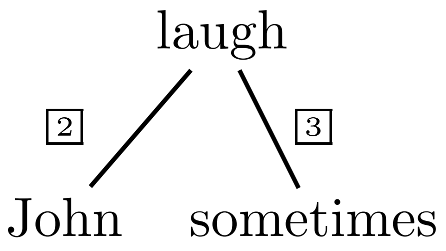


LTAG Semantics using STAG

Derived tree pair:



Derivation tree:



Logical Form:

$sometimes'(laugh'(John'))$

Unification-Based LTAG Semantics

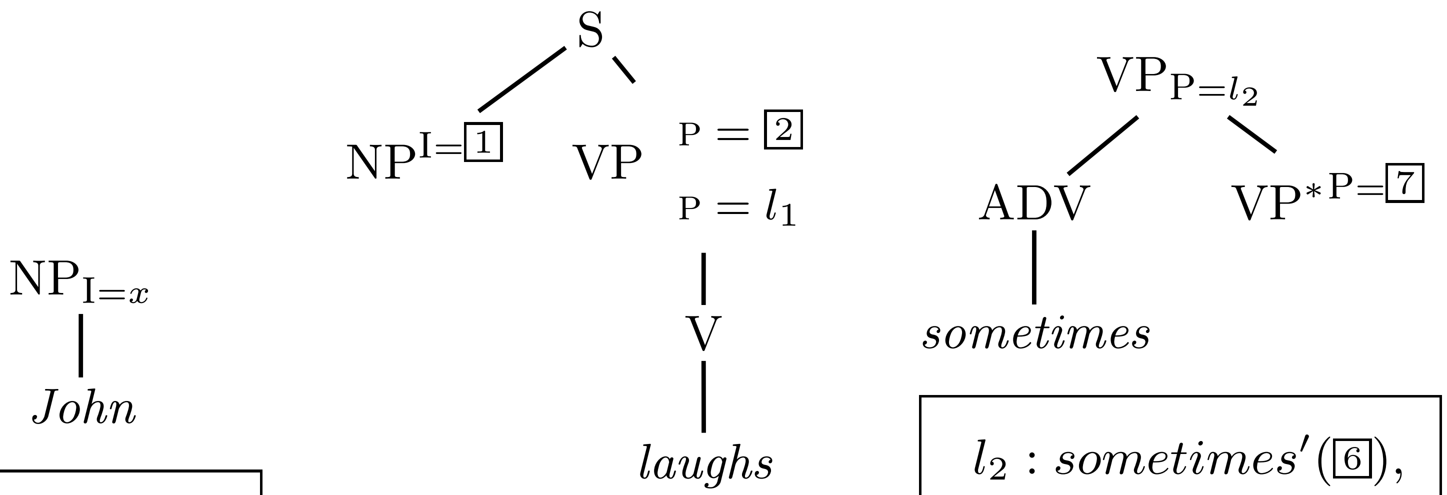
[Kallmeyer and Romero, 2008], [Gardent and Kallmeyer, 2003]:
Syntax-Semantics Interface for LTAG

Idea: Each elementary tree is paired with

- A set of typed predicate logic expressions and of scope constraints (i.e., constraints on sub-term relations)
- interface features that characterizes a) which arguments need to be filled, b) which elements are available as arguments for other elementary trees and c) the scope behaviour.

  The features are linked to positions in the elementary tree.

Unification-Based LTAG Semantics



$\boxed{l_3 : john'(x)}$

$\boxed{l_1 : laugh'(\boxed{1})}$

$l_2 : sometimes'(\boxed{6}),$
$\boxed{6} \geq \boxed{7}$

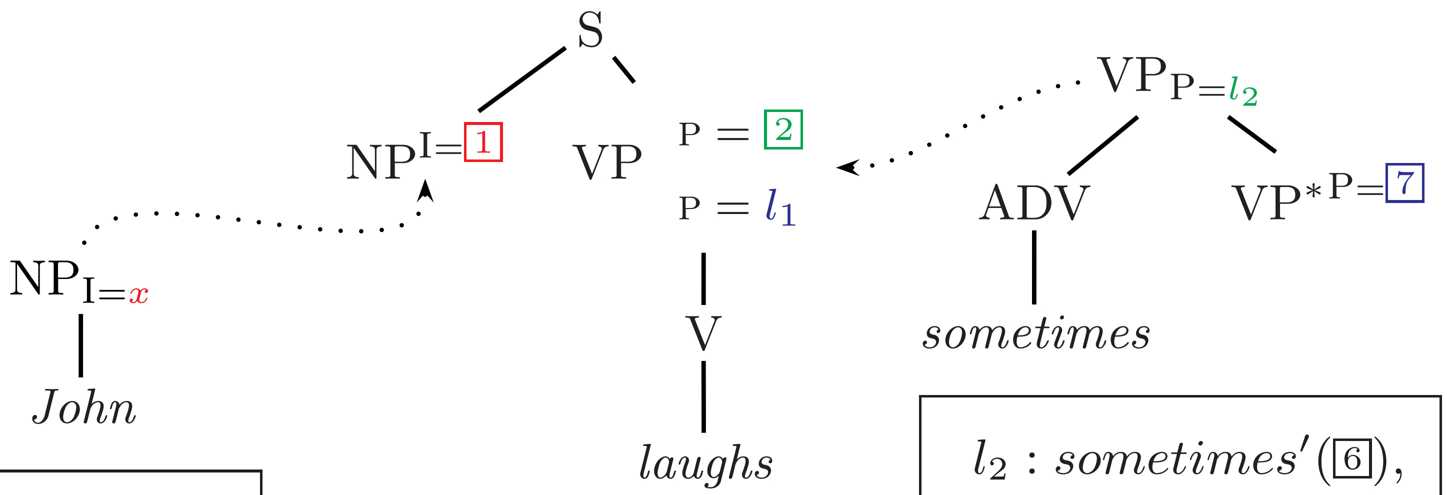
Unification-Based LTAG Semantics

- There is no proper functional application.

  Instead, depending on substitutions and adjunctions, we perform unifications that lead to assignments for the metavariables in the semantic representations.

- The result can be underspecified. Therefore, a further disambiguation is needed in order to obtain the different readings.

Unification-Based LTAG Semantics



$$\boxed{l_3 : john'(x)}$$

$$\boxed{l_1 : laugh'(\boxed{1})}$$

$$\boxed{\begin{array}{l} l_2 : sometimes'(\boxed{6}), \\ \boxed{6} \geq \boxed{7} \end{array}}$$

$$\rightsquigarrow \boxed{\begin{array}{l} l_3 : john(x)',\ l_1 : laugh'(x),\ l_2 : sometimes'(\boxed{6}) \\ \boxed{6} \geq l_1 \end{array}}$$

Unification-Based LTAG Semantics

Disambiguation: Function assigning propositional labels to the remaining propositional metavariables while respecting the scope constraints.

$$\boxed{\begin{array}{l} l_1 : \mathit{laugh}'(x),\ l_0 : \mathit{john}'(x),\ l_2 : \mathit{sometimes}'(\boxed{3}), \\ \boxed{3} \geq l_1 \end{array}}$$

Only one disambiguation: $\boxed{3} \rightarrow l_1$. Leads to

$$\boxed{l_0 : \mathit{john}'(x),\ l_2 : \mathit{sometimes}'(l_1 : \mathit{laugh}'(x))}$$

The resulting set is interpreted conjunctively.

This yields $\mathit{john}'(x) \wedge \mathit{sometimes}'(\mathit{laugh}'(x))$

Generalized quantifiers

Quantificational NPs can in principle scope freely; their scope is not directly linked to their surface position.

(2) a. Exactly one student admires every professor
  b. $\exists > \forall, \forall > \exists$

(3) a. Two policemen spy on someone from every city
  b. $\forall > \exists > 2$ (among others)

(4) a. John seems to have visited everybody
  b. $seem > \forall, \forall > seem$

Attitude verbs block the scope of embedded quantificational NPs:

(5) a. Mary thinks John likes everybody
  b. thinks > everybody, *everybody > thinks

Generalized quantifiers

The following must be guaranteed:

1. the proposition to which a quantifier attaches must be in its nuclear scope
2. a quantifier cannot scope higher than the next finite clause
3. besides its scope, the quantifier contributes an argument to some predicate

Generalized quantifiers

Frage: Was denotieren Quantoren, bzw. von welchem Typ sind sie?

Solange man es nur mit All- und Existenzquantifikation zu tun hat, könnte man diese als spezielle Mechanismen in die Syntax unserer Prädikatenlogik einbauen.

(6) a. every student is sleeping.
    b. $\forall x[\mathit{student}'(x) \rightarrow \mathit{sleep}'(x)]$

(7) a. some student is sleeping.
    b. $\exists x[\mathit{student}'(x) \wedge \mathit{sleep}'(x)]$

Problem: Was passiert mit anderen quantifizierenden Determinierern?

Generalized quantifiers

Lösung: Generalisierte Quantoren [Barwise and Cooper, 1981].

(8) every student is sleeping.

- Nomen denotieren einstellige Prädikate, die man als Mengen von Individuen auffassen kann.

  $[[student']] = \{$Mary, Anna, Bill, Max, $\dots\}$.

  Typ: $\langle e, t\rangle$

- Einstellige (intransitive) Verben denotieren ebenfalls einstellige Prädikate.

  $[[sleep']] = \{$Mary, Anna, Bill, John, $\dots\}$.

  Typ: $\langle e, t\rangle$

Generalized quantifiers

Der Determinierer beschreibt die Relation, die zwischen diesen beiden Mengen vorliegen muss. D.h., er nimmt zwei Argumente vom Typ $\langle e, t\rangle$ und liefert einen Wahrheitswert, ist also vom Typ

$$\langle\langle e, t\rangle, \langle\langle e, t\rangle, t\rangle\rangle$$

$$\begin{aligned}
& [[every'(student')(sleep')]] \\
= \quad & [[every']]([[student']])([[sleep']]) \\
= \quad & t \text{ if } [[student']] \subseteq [[sleep']] \\
& f \text{ else}
\end{aligned}$$

Solche Quantoren werden als *generalized quantifiers* bezeichnet.

Generalized quantifiers

(9) some student is sleeping.

$$
\begin{aligned}
& [[\mathit{some}'(\mathit{student}')(\mathit{sleep}')]] \\
= \quad & [[\mathit{some}']]([[\mathit{student}']])([[\mathit{sleep}']]) \\
= \quad & t \text{ if } [[\mathit{student}']] \cap [[\mathit{sleep}']] \neq \emptyset \\
& f \text{ else}
\end{aligned}
$$

(10) at least half of the students are sleeping.

$$
\begin{aligned}
& [[\mathit{at\_least\_half\_of}'(\mathit{student}')(\mathit{sleep}')]] \\
= \quad & [[\mathit{at\_least\_half\_of}']]([[\mathit{student}']])([[\mathit{sleep}']]) \\
= \quad & t \text{ if } |[[\mathit{student}']] \cap [[\mathit{sleep}']]| \geq |[[\mathit{student}']] \setminus [[\mathit{sleep}']]| \\
& f \text{ else}
\end{aligned}
$$

Quantifiers in STAG

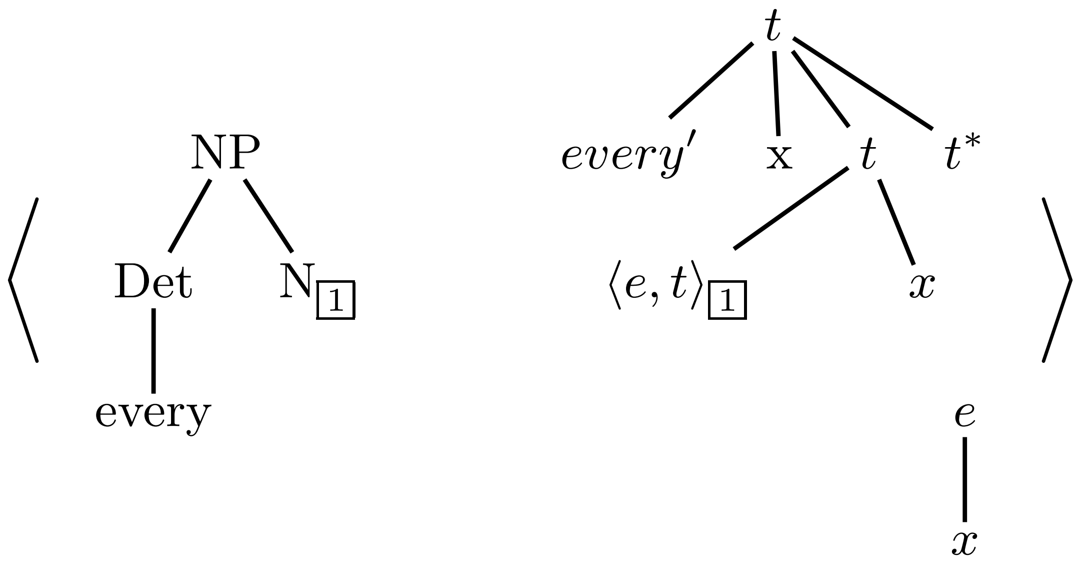
(11) a. every man laughs

b. $every'(x, man'(x), laugh'(x))$

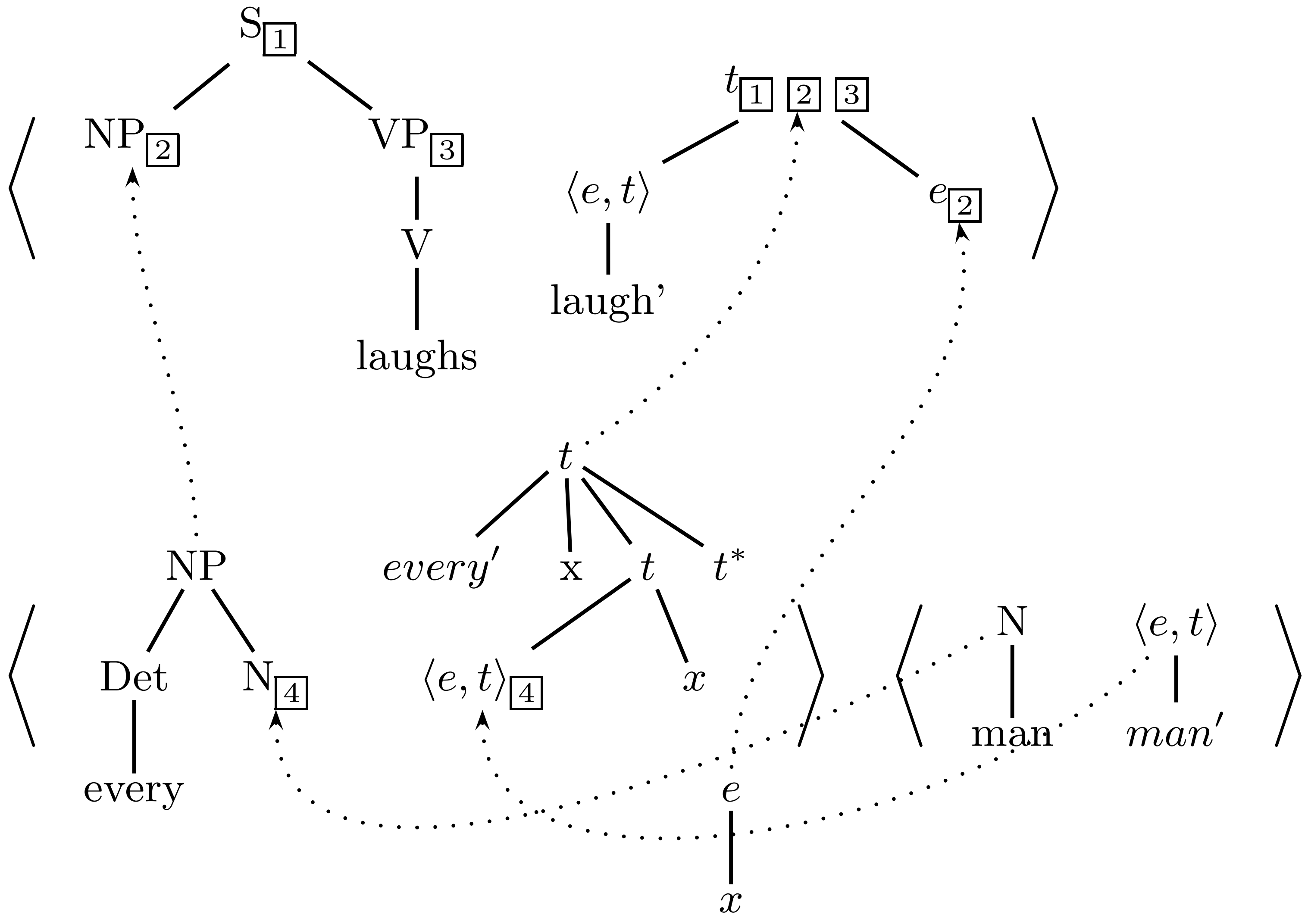
Since the argument of $laugh'$ is separated from the quantifier, [Nesson and Shieber, 2006] separate the semantic contribution of *every* into two trees:

- a scopal tree that determines the scope position of the quantifier, and
- an individual e tree that contributes the argument of $laugh'$.

Quantifiers in STAG



Quantifiers in STAG



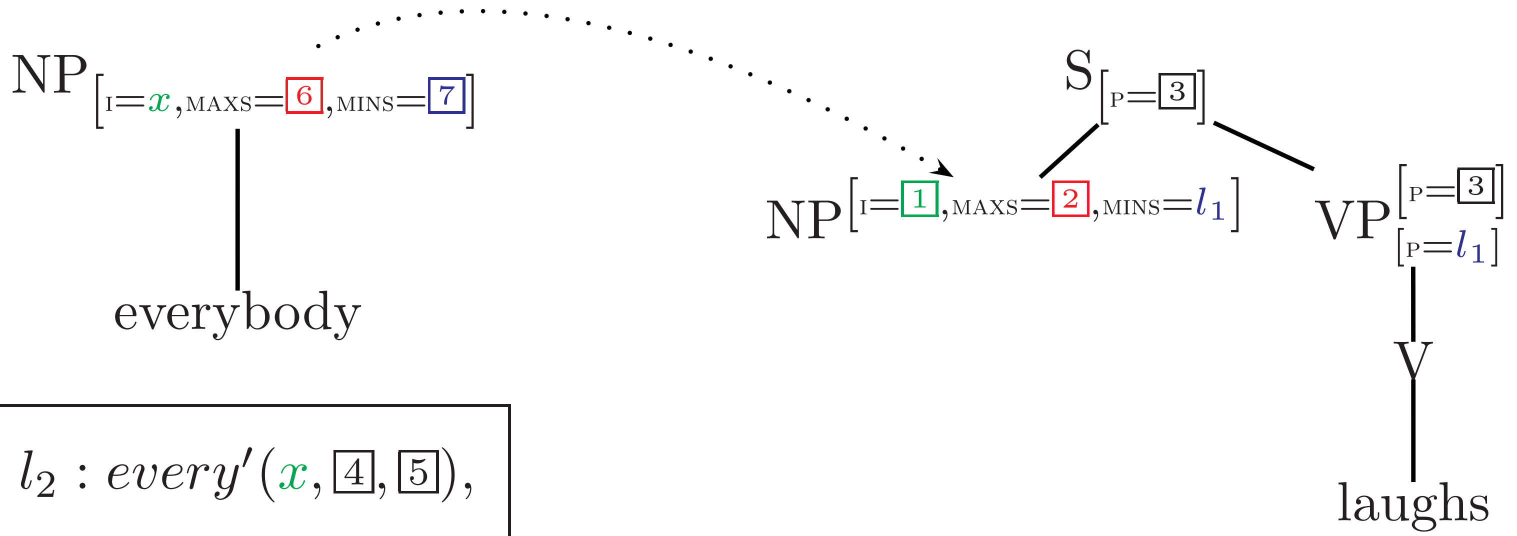
Quantifiers in STAG

(12) Some student likes every course.

- We have two semantic tree sets for the two quantifiers whose scopal auxiliary trees both adjoin to the t root node of the verbal predicate.
- In standard TAG, this would not be possible; we therefore have to allow for *multiple adjunctions*.
- If two trees adjoin to the same node, the order of the adjunctions determines the derived tree.
- The derivation tree, as long as it is unordered, does no longer specify the derived tree uniquely. Instead, it is an underspecified representation of several (here 2) derived trees.
- This is how, within STAG, one can generate underspecified representations for scope ambiguities.

Unification-based LTAG semantics and quantifiers

Idea: scope window delimited by some maximal scope MAXS and some minimal scope MINS for a quantifier.



$\text{NP}_{[\text{I}=x, \text{MAXS}=\boxed{6}, \text{MINS}=\boxed{7}]}$
everybody
$\text{S}_{[\text{P}=\boxed{3}]}$
$\text{NP}^{[\text{I}=\boxed{1}, \text{MAXS}=\boxed{2}, \text{MINS}=l_1]}$
$\text{VP}^{[\text{P}=\boxed{3}]}_{[\text{P}=l_1]}$
V
laughs

$l_2 : every'(x, \boxed{4}, \boxed{5}),$

$l_3 : person'(x),$

$\boxed{4} \geq l_3,$

$\boxed{6} \geq l_2, \boxed{5} \geq \boxed{7}$

$l_1 : laugh'(\boxed{1}),$

$\boxed{2} \geq \boxed{3}$

Unification-based LTAG semantics and quantifiers

Result:

$$\boxed{\begin{array}{l} l_1 : laugh'(x), \\ l_2 : every'(x, \boxed{4}, \boxed{5}),\ l_3 : person'(x) \\ \boxed{2} \geq l_1, \boxed{2} \geq l_2, \\ \boxed{4} \geq l_3, \boxed{5} \geq l_1 \end{array}}$$

Disambiguation:

$\boxed{2} \rightarrow l_2$, $\boxed{4} \rightarrow l_3, \boxed{5} \rightarrow l_1$

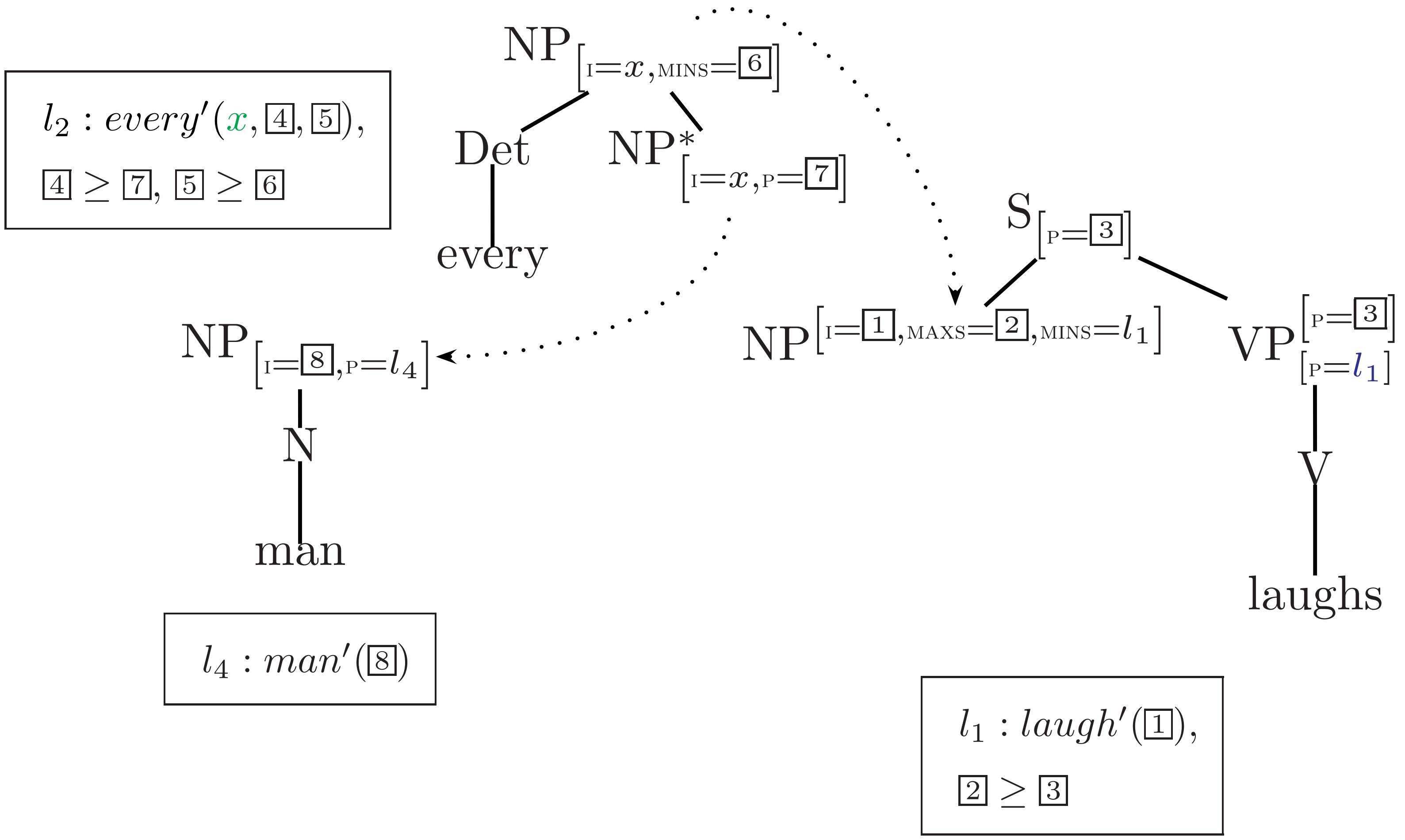
yields $every'(x, person'(x), laugh'(x))$

This analysis generates an underspecified representation for

(12) Some student likes every course.

Unification-based LTAG semantics and quantifiers

(13) every man laughs


$l_2 : every'(x, \boxed{4}, \boxed{5}),$
$\boxed{4} \geq \boxed{7}, \boxed{5} \geq \boxed{6}$
$\text{NP}_{[\text{I}=x, \text{MINS}=\boxed{6}]}$
Det
every
$\text{NP}^*_{[\text{I}=x, \text{P}=\boxed{7}]}$
$\text{S}_{[\text{P}=\boxed{3}]}$
$\text{NP}_{[\text{I}=\boxed{8}, \text{P}=l_4]}$
N
man
$\text{NP}^{[\text{I}=\boxed{1}, \text{MAXS}=\boxed{2}, \text{MINS}=l_1]}$
$\text{VP}^{[\text{P}=\boxed{3}]}_{[\text{P}=l_1]}$
V
laughs
$l_4 : man'(\boxed{8})$
$l_1 : laugh'(\boxed{1}),$
$\boxed{2} \geq \boxed{3}$

References

[Barwise and Cooper, 1981] Barwise, J. and Cooper, R. (1981). Generalized quantifiers and natural language. *Linguistics and Philosophy*, 4:159–219.

[Gardent and Kallmeyer, 2003] Gardent, C. and Kallmeyer, L. (2003). Semantic Construction in FTAG. In *Proceedings of EACL 2003*, pages 123–130, Budapest.

[Kallmeyer and Joshi, 2003] Kallmeyer, L. and Joshi, A. K. (2003). Factoring Predicate Argument and Scope Semantics: Underspecified Semantics with LTAG. *Research on Language and Computation*, 1(1–2):3–58.

[Kallmeyer and Romero, 2008] Kallmeyer, L. and Romero, M. (2008). Scope and situation binding in LTAG using semantic unification. *Research on Language and Computation*, 6(1):3–52.

[Nesson and Shieber, 2006] Nesson, R. and Shieber, S. M. (2006). Simpler TAG semantics through synchronization. In *Proceedings of the 11th Conference on Formal Grammar*, Malaga, Spain.

[Nesson and Shieber, 2008] Nesson, R. and Shieber, S. M. (2008). Synchronous vector tag for syntax and semantics: Control verbs, relative clauses, and inverse linking. In *Proceedings of the Ninth International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+ 9)*, Tübingen, Germany.

[Shieber, 1994] Shieber, S. M. (1994). Restricting the weak-generative capacity of synchronous Tree-Adjoining Grammars. *Computational Intelligence*, 10(4):271–385.

[Shieber and Schabes, 1990] Shieber, S. M. and Schabes, Y. (1990). Synchronous Tree-Adjoining Grammars. In *Proceedings of COLING*, pages 253–258.