
Semantische Sprachverarbeitung

Carola Eschenbach

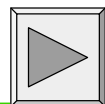
Universität Hamburg, MIN-Fakultät, Dept. Informatik
AB Wissens- und Sprachverarbeitung (WSV)

Sommersemester 2009

Semantische Sprachverarbeitung

Sitzung 7: Typen und Lambda-Kalkül

- Motivation, Gegliedertes Lexikon
- Typen
- Typisierter Lambda-Kalkül



Korrespondenz Syntax - Semantik

Syntax

- Ausdrücke (Wörter, Konstituenten, Phrasen) gehören **syntaktischen Kategorien** an, die ihren grammatischen Positionsmöglichkeiten entsprechen.
 - Verb (intransitiv, transitiv, di-transitiv, ...), Verbalphrase
 - Nomen, Nominalphrase, Artikel, Determinator, Pronomen

Semantik

- Die Bedeutungen der Ausdrücke haben unterschiedliche **semantische Typen**, die ihrer Funktion im Bedeutungsaufbau entsprechen
 - Proposition, Menge / Eigenschaft, Relation, Quantor, ...

Montagues These

Starke Korrespondenz

- Jeder syntaktischen Kategorie entspricht genau ein semantischer Typ auf der Bedeutungsebene
- Bedeutungskombination besteht im wesentlichen aus der Anwendung von Funktionen auf Argumente

Beispiel (vereinfacht)

- Nomen: einstellige Prädikate / Eigenschaften
- Nominalphrasen: Quantoren: Eigenschaften von Eigenschaften (.. später mehr)

Bedeutungen von Lexemen kombinieren

- eigentlichen Inhalt (Prädikat, Relation) und
- Typ-relevante Anteile

Grundidee (Sitzung 6)

die Bedeutung eines Ausdrucks

- beinhaltet eine eigentlichen, inhaltlichen **Beitrag**
- weist (wohldefinierte) **Lücken** auf, die der Kontext spezifiziert
- **Beitrag** und **Lücken** werden zu prädikatenlogischen Fragmenten zusammengesetzt

der Typ des Ausdrucks

- determiniert, welche **Lücken** vom Kontext zu füllen sind

Regeln

- determinieren, wie die **Lücken** gefüllt werden

Lambda-Kalkül

- Abstraktion und Applikation als **Lücken-Bildung** und **Lücken-Füllung**

Typen

Semantische Basistypen

- **e**: Entity: Ausdruck denotiert ein Objekt (im weiten Sinn)
- **t**: Truth-value: Ausdruck gibt eine Proposition wieder

Komplexe Typen

- Wenn α und β Typen sind, dann ist $\langle \alpha, \beta \rangle$ ein komplexer Typ.
 - Ausdrücke von Typ $\langle \alpha, \beta \rangle$ denotieren Abbildungen von α -Denotaten auf β -Denotate.
- Das sind alle Typen.

Beispiel

- $\langle e, t \rangle$: Prädikat, Eigenschaft
- $\langle e, \langle e, t \rangle \rangle$: zweistellige Relation
- $\langle \langle e, t \rangle, t \rangle$: gen. Quantor, Abbildung von Eigenschaften auf Propositionen

Interpretation von Typen

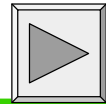
Sei D eine Menge

Semantische Basistypen

- e : Entity: Ausdruck denotiert ein Objekt (im weiten Sinn)
 - $D_e = D$
- t : Truth-value: Ausdruck gibt eine Proposition wieder
 - $D_t = \{\text{wahr, falsch}\}$

Komplexe Typen

- Wenn α und β Typen sind, dann ist $\langle \alpha, \beta \rangle$ ein komplexer Typ.
 - Ausdrücke von Typ $\langle \alpha, \beta \rangle$ denotieren Abbildungen von α -Denotaten auf β -Denotate.
 - $D_{\langle \alpha, \beta \rangle} = D_\alpha \rightarrow D_\beta$



Typen-gesteuerte Grammatik

Typenzuordnung

- jedem Symbol wird ein Typ zugeordnet

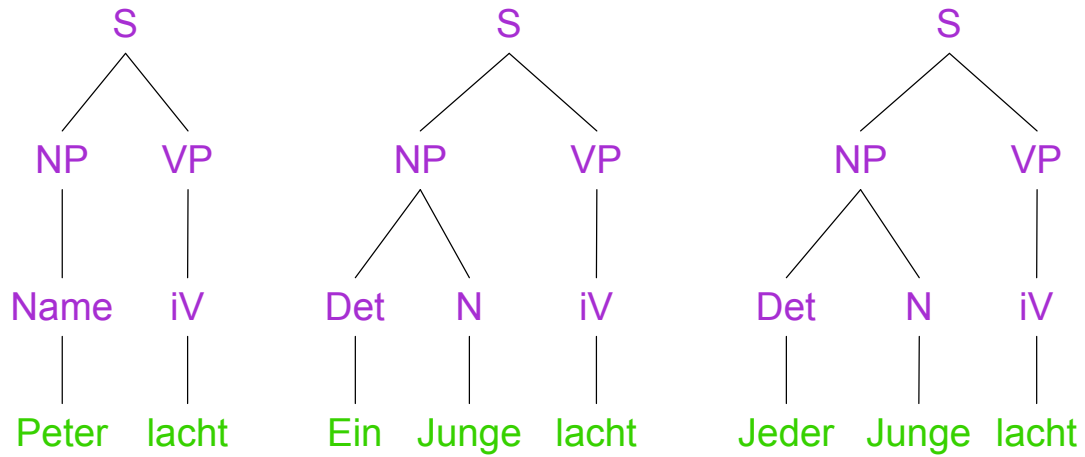
Bildung komplexer Ausdrücke

- Wenn α und β Typen sind, a ein Ausdruck vom Typ α und f ein Ausdruck vom Typ $\langle \alpha, \beta \rangle$, dann ist die Applikation von f auf a , geschrieben $f(a)$, ein (wohlgeformter) Ausdruck vom Typ β .
 - [Zur besseren Lesbarkeit werden aber normalerweise alternative Klammerregeln verwendet]

Beispiel

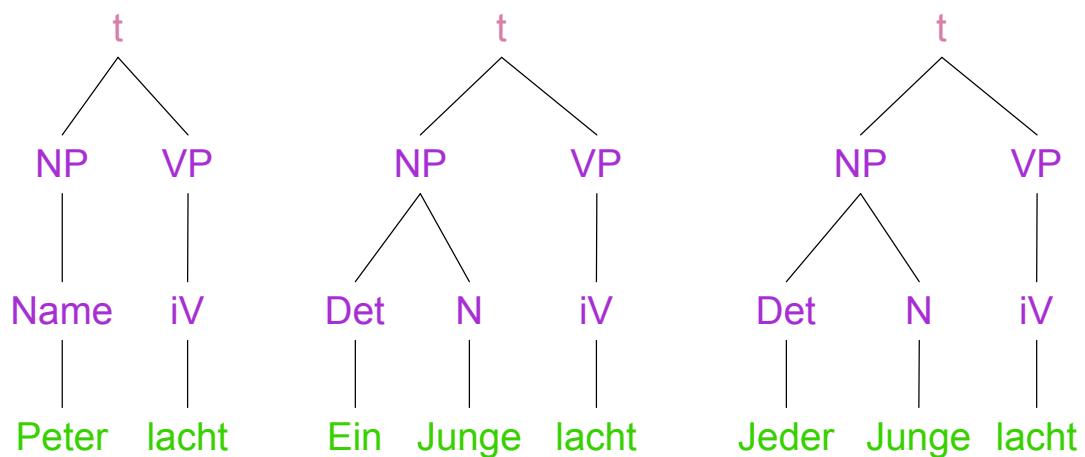
- Prädikat ($\langle e, t \rangle$) und Terme (e) bilden Formeln (t).
 - Negation ($\langle t, t \rangle$) und Formel (t) bilden eine Formel (t).
- ➔ Signatur-gesteuerte Sprachdefinition

Beispiel: Typen in der natürlichen Sprache



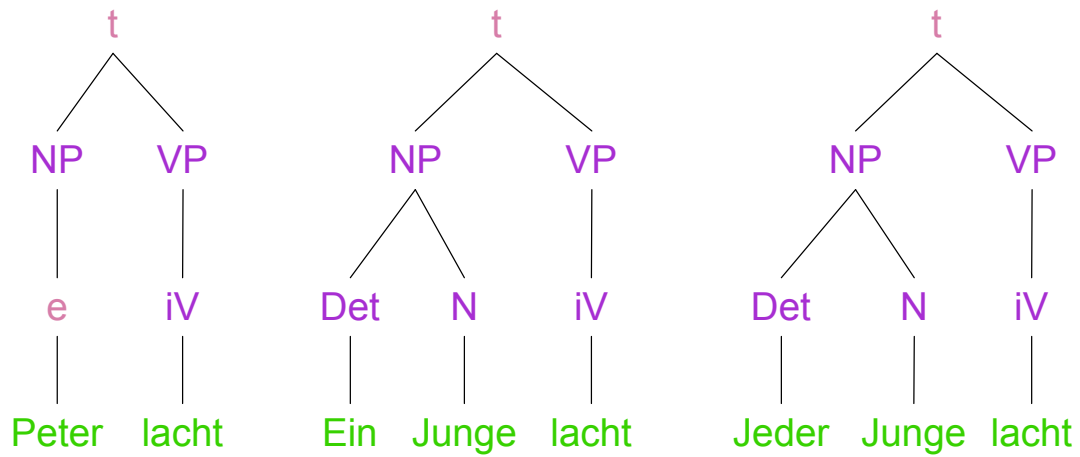
Beispiel: Typen in der natürlichen Sprache

S(ätze) sind vom Typ t



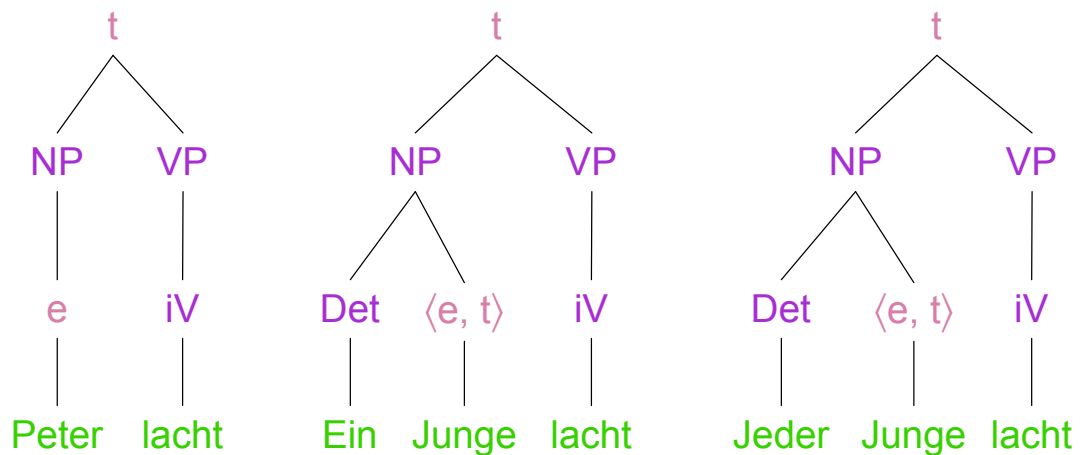
Beispiel: Typen in der natürlichen Sprache

Namen sind vom Typ e



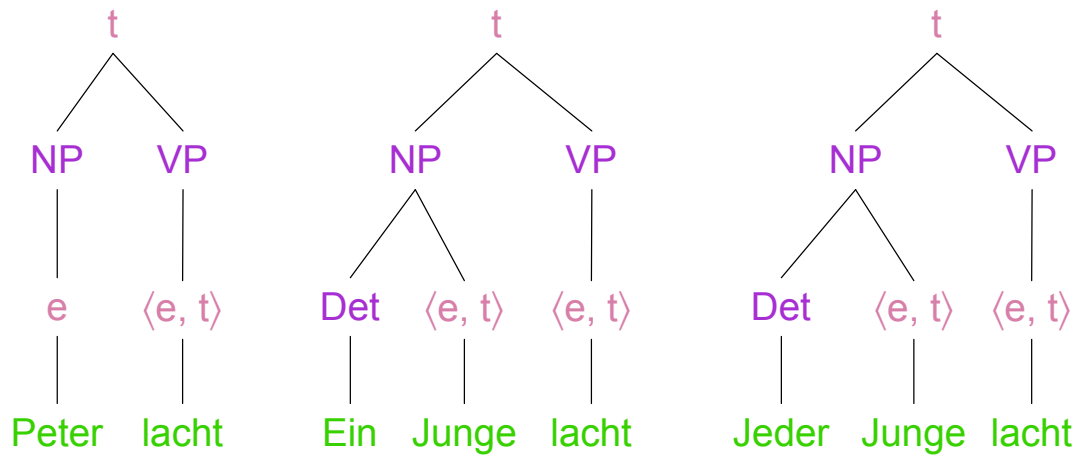
Beispiel: Typen in der natürlichen Sprache

N(omen) sind vom Typ $\langle e, t \rangle$



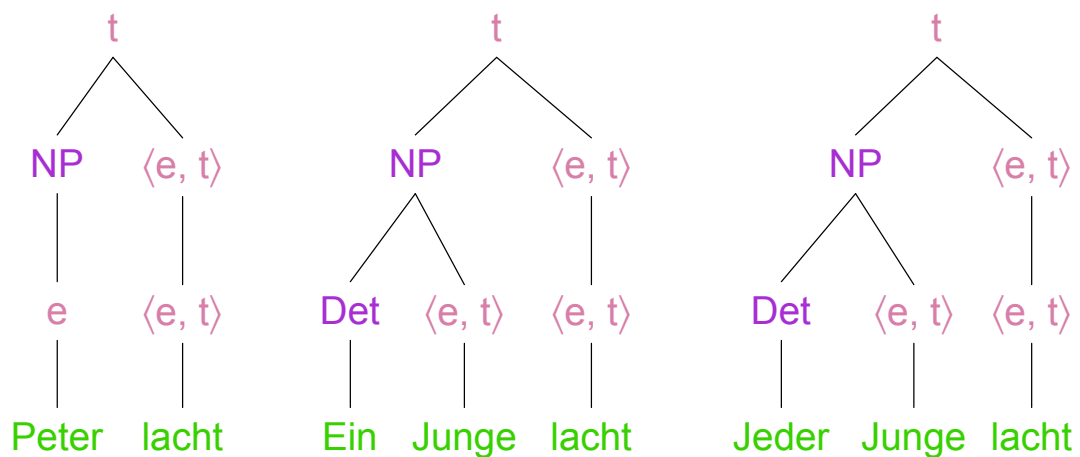
Beispiel: Typen in der natürlichen Sprache

i(ntransitive) V(erben) sind vom Typ $\langle e, t \rangle$



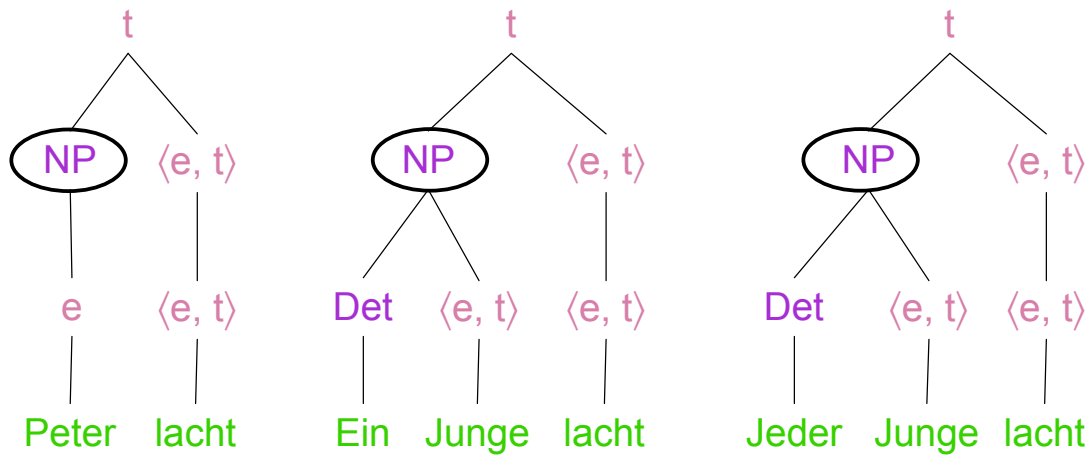
Beispiel: Typen in der natürlichen Sprache

V(erbale)P(hrasen) sind vom Typ $\langle e, t \rangle$



Beispiel: Typen in der natürlichen Sprache

Sind N(ominal)P(hrasen) vom Typ e ?



AN IMPORTANT JOB

This is a story about four people named Everybody, Somebody, Anybody and Nobody.

There was an important job to be done and Everybody was sure that Somebody would do it.

Anybody could have done it, but Nobody did it.

Somebody got angry about that, because it was Everybody's job.

Everybody thought Anybody could do it, but Nobody realised that Everybody wouldn't do it.

It ended up that Everybody blamed Somebody when Nobody did what Anybody could have.

Nominalphrasen: Generalisierte Quantoren

Jeder Junge lacht

- Jeder Junge → die Eigenschaften, die jeder Junge hat
- die Eigenschaft zu lachen gehört zu den Eigenschaften, die jeder Junge hat

Ein Junge lacht

- Ein Junge → die Eigenschaften, die mind. ein Junge hat
- die Eigenschaft zu lachen gehört zu den Eigenschaften, die mind. ein Junge hat

Viele Jungen lachen

- Ein Junge → die Eigenschaften, die viele Jungen haben
- die Eigenschaft zu lachen gehört zu den Eigenschaften, die viele Jungen haben

Typenanhebung

ohne Anhebung

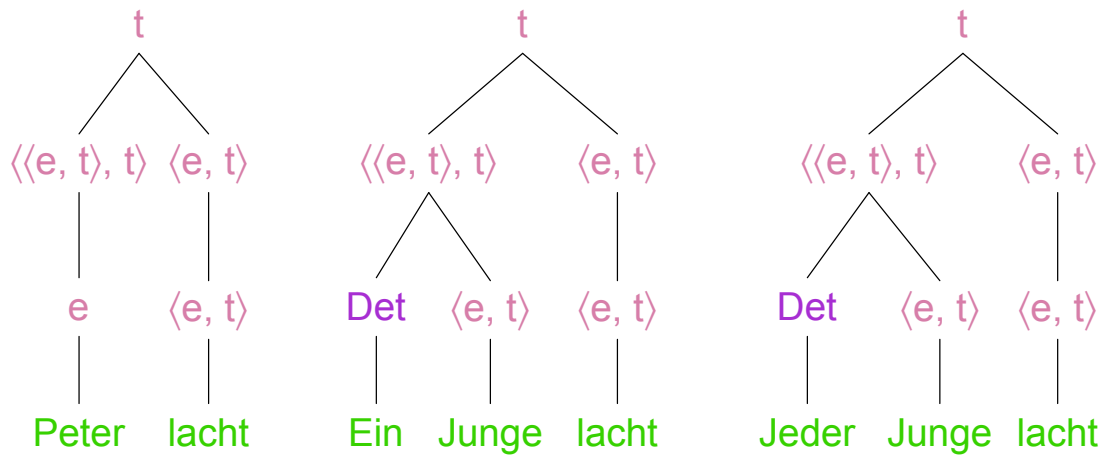
- Peter → das Individuum namens ‚Peter‘ e
- lacht → die Eigenschaft zu lachen ⟨e, t⟩
- Peter lacht → Peter hat die Eigenschaft zu lachen t

mit Anhebung

- Peter → die Eigenschaft, eine Eigenschaft von Peter zu sein ⟨⟨e, t⟩, t⟩
- lacht → die Eigenschaft zu lachen ⟨e, t⟩
- Peter lacht → die Eigenschaft zu lachen gehört zu den Eigenschaften von Peter t

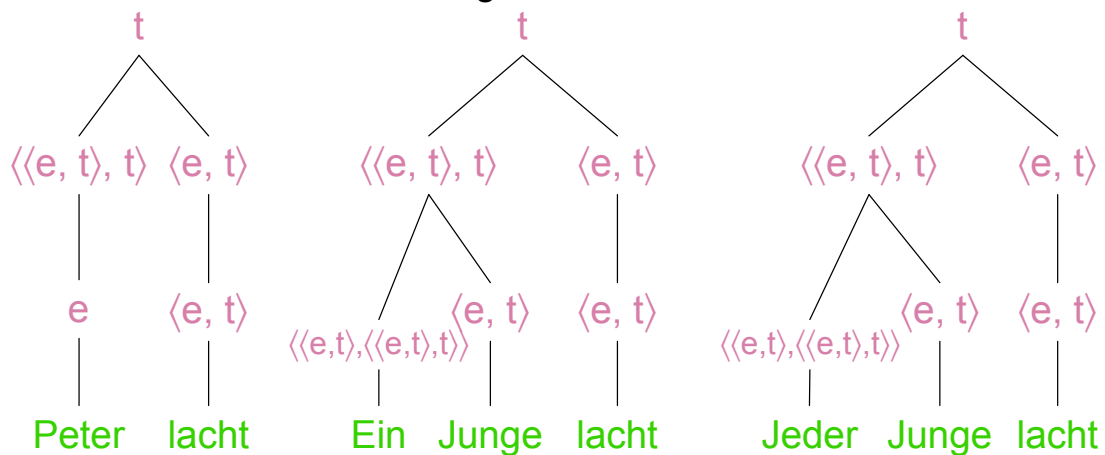
Beispiel: Typen in der natürlichen Sprache

N(ominal)P(hrasen) sind vom Typ $\langle\langle e, t \rangle, t\rangle$
 Generalisierte Quantoren



Beispiel: Typen in der natürlichen Sprache

Det(erminatoren) sind vom Typ $\langle\langle e, t \rangle, \langle\langle e, t \rangle, t\rangle\rangle$
 Relationen zwischen Eigenschaften



Determinatoren: Relationen zwischen Eigenschaften bzw. Mengen

ein

- die beiden Eigenschaften sind an einem gemeinsamen Objekt realisiert
- die beiden Mengen haben einen nicht-leeren Durchschnitt

jeder

- die N-Eigenschaft ist spezifischer als die V-Eigenschaft
- die N-Menge ist in der V-Menge enthalten

kein

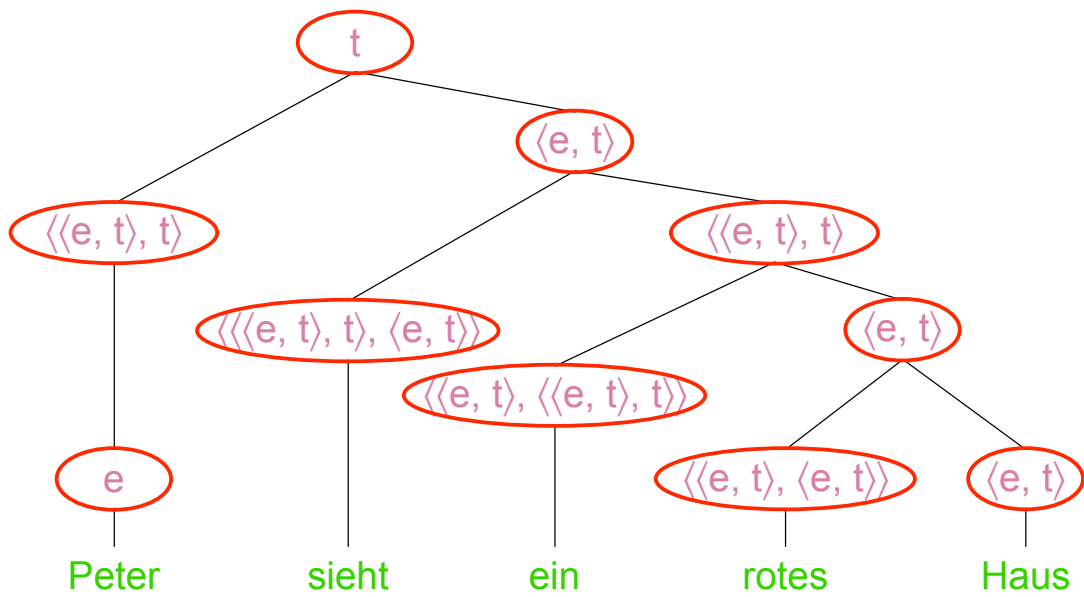
- die beiden Eigenschaften sind nicht an einem gemeinsamen Objekt realisiert
- der Durchschnitt der beiden Mengen ist leer

Wozu der Aufwand?

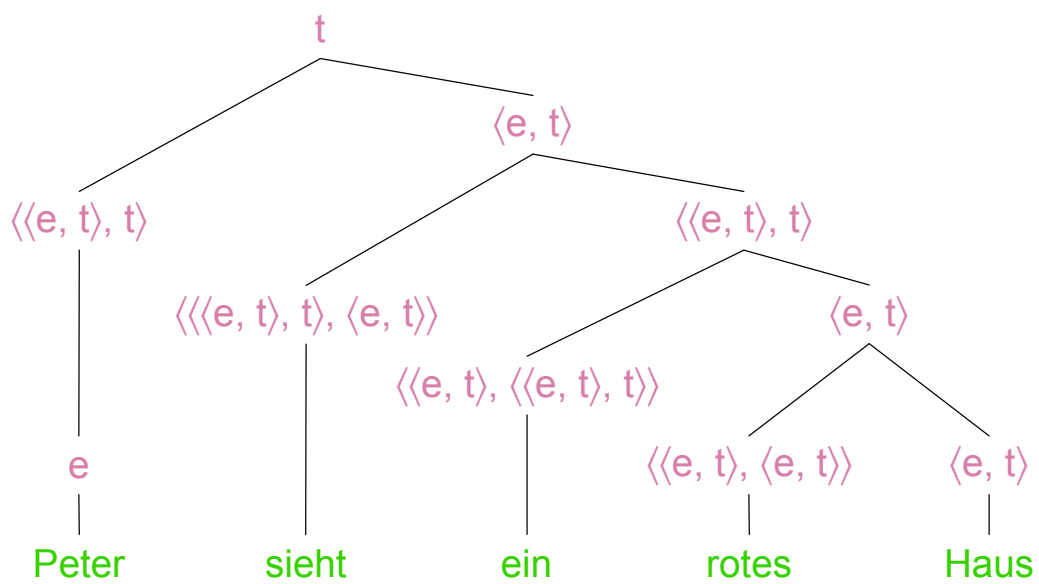
Generalisierung

- Einheitlichen syntaktischen Strukturen ($[_S[_{NP}][_VP]]$) entsprechen einheitlichen semantischen Kombinationsregeln (Anwendung der NP-Bedeutung auf VP-Bedeutung)

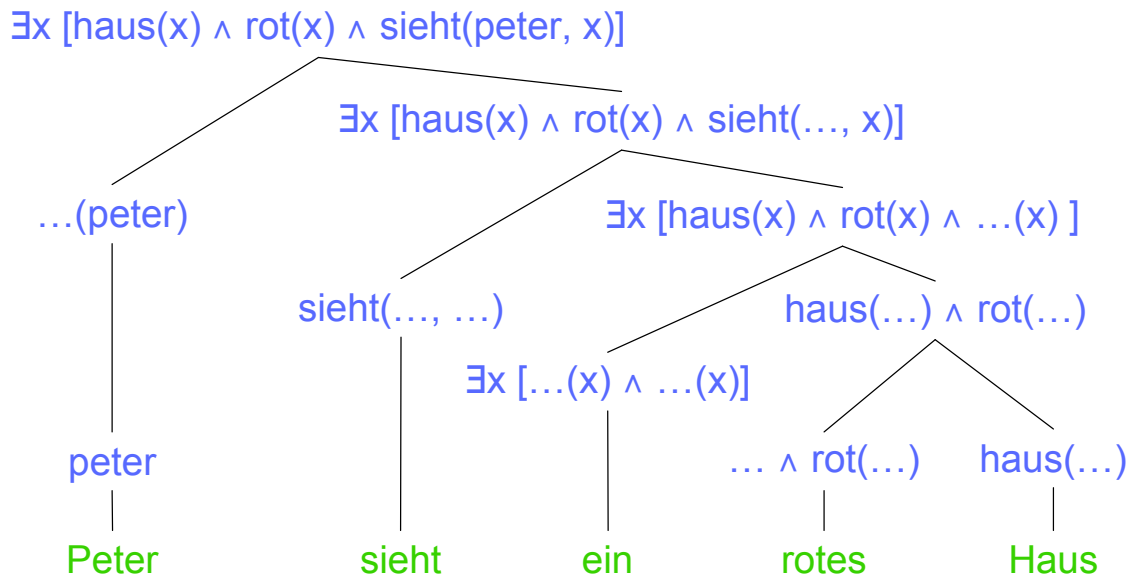
Typen: Peter sieht ein rotes Haus.



Typen: Peter sieht ein rotes Haus.



Wahrheitsbedingungen: Peter sieht ein rotes Haus.



Typen: Peter sieht ein rotes Haus.

Peter	e	peter
sieht	$\langle\langle\langle e, t \rangle, t \rangle, \langle e, t \rangle\rangle$	sieht(..., ...)
ein	$\langle\langle e, t \rangle, \langle\langle e, t \rangle, t \rangle\rangle$	$\exists x [\dots(x) \wedge \dots(x)]$
rotes	$\langle\langle e, t \rangle, \langle e, t \rangle\rangle$	rotes(...) ∧ ...
Haus	$\langle e, t \rangle$	haus(...)

Typisierte λ -Ausdrücke

Typenzuordnung

- jedem Symbol (auch den Variablen) wird ein Typ zugeordnet

Beispiel

	Symbol	Typ
Variablen	x	e
	P, R, S	$\langle e, t \rangle$
	Q, T	$\langle \langle e, t \rangle, t \rangle$
freie Symbole	junge	$\langle e, t \rangle$
	lacht	$\langle e, t \rangle$

Typisierte λ -Ausdrücke

Bildung komplexer Ausdrücke

- Wenn α und β Typen sind, a ein Ausdruck vom Typ α und f ein Ausdruck vom Typ $\langle \alpha, \beta \rangle$, dann ist die Applikation von f auf a , geschrieben $f(a)$, ein (wohlgeformter) Ausdruck vom Typ β .
- Bei binären Junktoren op (Typ $\langle t, \langle t, t \rangle \rangle$) und Formeln F_1, F_2 (Typ t) schreiben wir statt $op(F_1)(F_2)$ auch $(F_1 op F_2)$.

Beispiel

Ausdruck	Typ
junge(x)	t
lacht(x)	t
$P(x)$	t
$R(x)$	t

Ausdruck	Typ
$(junge(x) \wedge lacht(x))$	t
$(junge(x) \wedge R(x))$	t
$(P(x) \wedge R(x))$	t

Typisierte λ -Ausdrücke

Bildung komplexer Ausdrücke

- Wenn α und β Typen sind, x eine Variable vom Typ α und b ein Ausdruck vom Typ β , dann ist $\lambda x [b]$, ein (wohlgeformter) Ausdruck (Abstraktion) vom Typ $\langle \alpha, \beta \rangle$.
- Statt $\lambda x [\lambda y [F]]$ schreiben wir auch $\lambda x \lambda y [F]$.

Beispiel

Ausdruck	Typ
$\lambda x [(junge(x) \wedge lacht(x))]$	$\langle e, t \rangle$
$\lambda R \lambda x [(junge(x) \wedge R(x))]$	$\langle \langle e, t \rangle, \langle e, t \rangle \rangle$
$\lambda P \lambda R \lambda x [(P(x) \wedge R(x))]$	$\langle \langle e, t \rangle, \langle \langle e, t \rangle, \langle e, t \rangle \rangle \rangle$

Typisierte λ -Ausdrücke

Typenzuordnung

- jedem Symbol (auch den Variablen) wird ein Typ zugeordnet

Bildung komplexer Ausdrücke

- Wenn α und β Typen sind, a ein Ausdruck vom Typ α und f ein Ausdruck vom Typ $\langle \alpha, \beta \rangle$, dann ist die Applikation von f auf a , geschrieben $f(a)$, ein (wohlgeformter) Ausdruck vom Typ β .
- Wenn α und β Typen sind, x eine Variable vom Typ α und b ein Ausdruck vom Typ β , dann ist $\lambda x [b]$, ein (wohlgeformter) Ausdruck (Abstraktion) vom Typ $\langle \alpha, \beta \rangle$.

Beispiel

$\lambda x [(junge(x) \wedge lacht(x))]$ ist vom Typ $\langle e, t \rangle$.

$\lambda R \lambda x [(junge(x) \wedge R(x))]$ ist vom Typ $\langle \langle e, t \rangle, \langle e, t \rangle \rangle$.

$\lambda P \lambda R \lambda x [(P(x) \wedge R(x))]$ ist vom Typ $\langle \langle e, t \rangle, \langle \langle e, t \rangle, \langle e, t \rangle \rangle \rangle$.

Interpretation von Applikation und Abstraktion

- Sei D eine Menge und I eine Interpretation in D
- Seien α und β Typen, a ein Ausdruck vom Typ α , x eine Variable vom Typ α , b ein Ausdruck vom Typ β und f ein Ausdruck vom Typ $\langle \alpha, \beta \rangle$

Applikation

- $I(f(a)) = I(f)(I(a))$
 - Funktionsausdruck und Argumentsausdruck werden durch I ausgewertet, dann wird die resultierende Funktion auf das resultierende Argument angewendet.

Abstraktion

- $I(\lambda x [b]) = f$, wobei für alle $d \in D_\alpha$ ist $f(d) = I_{[x \rightarrow d]}(b)$
 - Es resultiert eine Funktion f , die α -Objekte auf β -Objekte abbildet, wobei der Funktionswert danach variiert, wie b abhängig von der Belegung von x interpretiert wird.

Äquivalenzen

Definition

- Zwei λ -Terme A , B sind genau dann (*logisch*) *äquivalent* (\equiv), wenn für jede Interpretation I gilt: $I(A) = I(B)$

Äquivalenzen für λ -Terme

- Seien x und y Variablen und B ein Ausdruck vom selben Typ.
- Sei A ein Ausdruck, in dem y nicht vorkommt und der mit B keine gemeinsamen Variablen hat
- Gebundene Umbenennung
 - $\lambda x [A]$ ist äquivalent mit $\lambda y [A\{x / y\}]$
- Applikation nach Abstraktion
 - $\lambda x [A](B)$ ist äquivalent mit $A\{x / B\}$
- Abstraktion nach Applikation
 - $\lambda y [A(y)]$ ist äquivalent mit A

Substitution (1)

Definition

Seien x eine Variable und B ein Ausdruck vom selben Typ
Die **Substitution** $\{x / B\}$ ist eine Abbildung von λ -Termen auf λ -
Terme, bei der die freien Vorkommen von x durch den Term
 B ersetzt werden.

Üblicherweise werden Substitutionen hinter ihr Argument
geschrieben. Also heißt $F \{x / B\}$, dass die Substitution
 $\{x / B\}$ auf den Ausdruck F angewendet wird.

Substitution (2)

Definition

Seien x eine Variable und B ein Ausdruck vom selben Typ.

Die **Substitution** $\{x / B\}$ ist rekursiv definiert durch:

x wird auf B abgebildet:

- $x \{x / B\} = B$

andere Variablen und Konstanten werden auf sich selbst
abgebildet:

- $y \{x / B\} = y$, falls y eine Variable und $y \neq x$

- $c \{x / B\} = c$, falls c eine Konstante ist

Bei der Applikation wird die Substitution auf Funktor und
Argument angewendet:

- $[A(C)] \{x / B\} = A\{x / B\}(C\{x / B\})$

Substitution (3)

Definition

Seien x eine Variable und B ein Ausdruck vom selben Typ

Die **Substitution** $\{x / B\}$ ist rekursiv definiert durch:

Ist x durch eine Abstraktor oder Quantor gebunden, dann stoppt die Ersetzung:

- $(\lambda x [A])\{x / B\} = \lambda x [A]$
- $(\exists x [A])\{x / B\} = \exists x [A]$ $[\forall x [A]]\{x / B\} = \forall x [A]$

Ansonsten wird die Ersetzung bei dem eingebetteten Ausdruck fortgesetzt

- $(\lambda y [A])\{x / B\} = \lambda y [A\{x / B\}]$, falls $y \neq x$
- $(\exists y [A])\{x / B\} = \exists y [A\{x / B\}]$, falls $y \neq x$
- $(\forall y [A])\{x / B\} = \forall y [A\{x / B\}]$, falls $y \neq x$

Äquivalenzen

Definition

- Zwei λ -Terme A , B sind genau dann *(logisch) äquivalent* (\equiv), wenn für jede Interpretation I gilt: $I(A) = I(B)$

Äquivalenzen für λ -Terme

- Seien x und y Variablen und B ein Ausdruck vom selben Typ.
- Sei A ein Ausdruck, in dem y nicht vorkommt und der mit B keine gemeinsamen Variablen hat
- Gebundene Umbenennung
 - $\lambda x [A]$ ist äquivalent mit $\lambda y [A\{x / y\}]$
- Applikation nach Abstraktion
 - $\lambda x [A](B)$ ist äquivalent mit $A\{x / B\}$
- Abstraktion nach Applikation
 - $\lambda y [A(y)]$ ist äquivalent mit A

Typen: Peter sieht ein rotes Haus.

Peter	e	peter
sieht	$\langle\langle e, t \rangle, t \rangle, \langle e, t \rangle$	sieht(..., ...)
ein	$\langle\langle e, t \rangle, \langle\langle e, t \rangle, t \rangle\rangle$	$\exists x [\dots(x) \wedge \dots(x)]$
rotes	$\langle\langle e, t \rangle, \langle e, t \rangle\rangle$	rotes(...) \wedge ...
Haus	$\langle e, t \rangle$	haus(...)

Peter sieht ein rotes Haus.

Peter	e	peter
sieht	$\langle\langle\langle e, t \rangle, t \rangle, \langle e, t \rangle\rangle$	$\lambda Q \lambda u [Q(\lambda w [sieht(u, w)])]$
ein	$\langle\langle e, t \rangle, \langle\langle e, t \rangle, t \rangle\rangle$	$\lambda R \lambda S \exists x [R(x) \wedge S(x)]$
rotes	$\langle\langle e, t \rangle, \langle e, t \rangle\rangle$	$\lambda P \lambda z [rotes(z) \wedge P(z)]$
Haus	$\langle e, t \rangle$	$\lambda y [haus(y)]$

rotes Haus

$$\begin{aligned}
 & \lambda P \lambda z [rotes(z) \wedge P(z) (\lambda y [haus(y)])] \\
 \equiv & \lambda z [rotes(z) \wedge (\lambda y [haus(y)]) (z)] \\
 \equiv & \lambda z [rotes(z) \wedge haus(z)]
 \end{aligned}$$

Peter sieht ein rotes Haus.

Peter	e	peter
sieht	$\langle\langle\langle e, t \rangle, t \rangle, \langle e, t \rangle\rangle$	$\lambda Q \lambda u [Q(\lambda w [\text{sieht}(u, w)])]$
ein	$\langle\langle e, t \rangle, \langle\langle e, t \rangle, t \rangle\rangle$	$\lambda R \lambda S \exists x [R(x) \wedge S(x)]$
rotes	$\langle\langle e, t \rangle, \langle e, t \rangle\rangle$	$\lambda P \lambda z [\text{rotes}(z) \wedge P(z)]$
Haus	$\langle e, t \rangle$	$\lambda y [\text{haus}(y)]$

ein rotes Haus

$$\begin{aligned}
 & \lambda R \lambda S \exists x [R(x) \wedge S(x)] \lambda z [\text{rotes}(z) \wedge \text{haus}(z)] \\
 \equiv & \lambda S \exists x [\lambda z [\text{rotes}(z) \wedge \text{haus}(z)](x) \wedge S(x)] \\
 \equiv & \lambda S \exists x [\text{rotes}(x) \wedge \text{haus}(x) \wedge S(x)]
 \end{aligned}$$

Peter sieht ein rotes Haus.

Peter	e	peter
sieht	$\langle\langle\langle e, t \rangle, t \rangle, \langle e, t \rangle\rangle$	$\lambda Q \lambda u [Q(\lambda w [\text{sieht}(u, w)])]$
ein	$\langle\langle e, t \rangle, \langle\langle e, t \rangle, t \rangle\rangle$	$\lambda R \lambda S \exists x [R(x) \wedge S(x)]$
rotes	$\langle\langle e, t \rangle, \langle e, t \rangle\rangle$	$\lambda P \lambda z [\text{rotes}(z) \wedge P(z)]$
Haus	$\langle e, t \rangle$	$\lambda y [\text{haus}(y)]$

sieht ein rotes Haus

$$\begin{aligned}
 & \lambda Q \lambda u [Q(\lambda w [\text{sieht}(u, w)])] \lambda S \exists x [\text{rotes}(x) \wedge \text{haus}(x) \wedge S(x)] \\
 \equiv & \lambda u [\lambda S \exists x [\text{rotes}(x) \wedge \text{haus}(x) \wedge S(x)](\lambda w [\text{sieht}(u, w)])] \\
 \equiv & \lambda u \exists x [\text{rotes}(x) \wedge \text{haus}(x) \wedge \lambda w [\text{sieht}(u, w)](x)]
 \end{aligned}$$

Peter sieht ein rotes Haus.

Peter	e	peter
sieht	$\langle\langle\langle e, t \rangle, t \rangle, \langle e, t \rangle\rangle$	$\lambda Q \lambda u [Q(\lambda w [sieht(u, w)])]$
ein	$\langle\langle e, t \rangle, \langle\langle e, t \rangle, t \rangle\rangle$	$\lambda R \lambda S \exists x [R(x) \wedge S(x)]$
rotes	$\langle\langle e, t \rangle, \langle e, t \rangle\rangle$	$\lambda P \lambda z [rotes(z) \wedge P(z)]$
Haus	$\langle e, t \rangle$	$\lambda y [haus(y)]$

$$\begin{aligned}
 \text{sieht ein rotes Haus} &\equiv \lambda u \exists x [rotes(x) \wedge haus(x) \wedge \lambda w [sieht(u, w)(x)]] \\
 &\equiv \lambda u \exists x [rotes(x) \wedge haus(x) \wedge sieht(u, x)]
 \end{aligned}$$

Peter sieht ein rotes Haus.

Peter	e	peter
sieht	$\langle\langle\langle e, t \rangle, t \rangle, \langle e, t \rangle\rangle$	$\lambda Q \lambda u [Q(\lambda w [sieht(u, w)])]$
ein	$\langle\langle e, t \rangle, \langle\langle e, t \rangle, t \rangle\rangle$	$\lambda R \lambda S \exists x [R(x) \wedge S(x)]$
rotes	$\langle\langle e, t \rangle, \langle e, t \rangle\rangle$	$\lambda P \lambda z [rotes(z) \wedge P(z)]$
Haus	$\langle e, t \rangle$	$\lambda y [haus(y)]$

$$\begin{aligned}
 [_{NP} [_{PN} \text{peter}]] \quad & \lambda x \lambda P [P(x)] \text{peter} \\
 & \equiv \lambda P [P(\text{peter})]
 \end{aligned}$$

Type Shifting

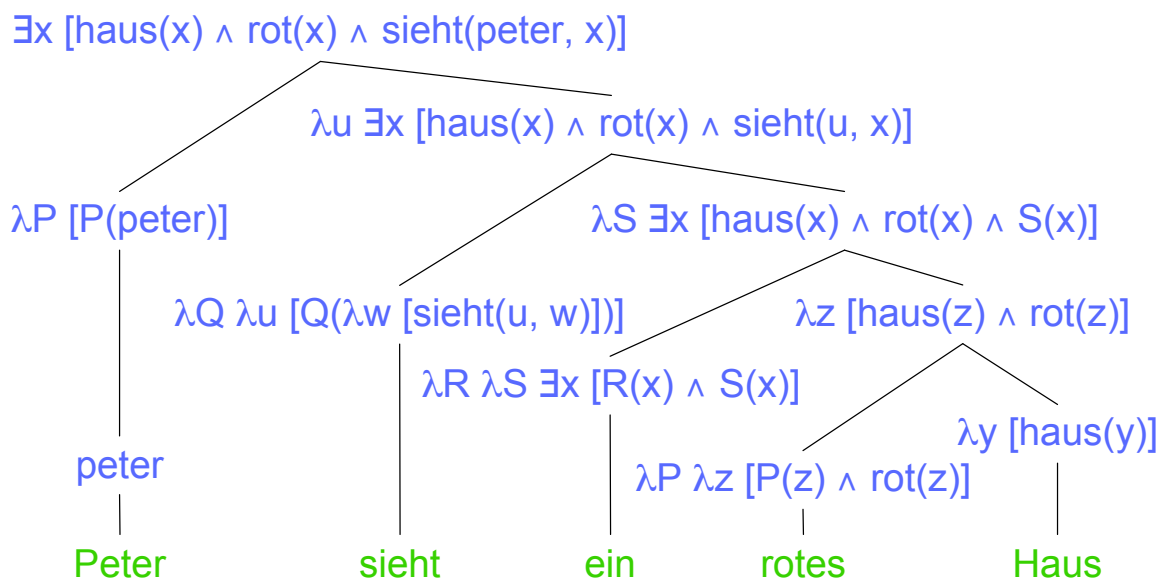
Peter sieht ein rotes Haus.

Peter	e	peter
sieht	$\langle\langle\langle e, t \rangle, t \rangle, \langle e, t \rangle\rangle$	$\lambda Q \lambda u [Q(\lambda w [sieht(u, w)])]$
ein	$\langle\langle e, t \rangle, \langle\langle e, t \rangle, t \rangle\rangle$	$\lambda R \lambda S \exists x [R(x) \wedge S(x)]$
rotes	$\langle\langle e, t \rangle, \langle e, t \rangle\rangle$	$\lambda P \lambda z [rotes(z) \wedge P(z)]$
Haus	$\langle e, t \rangle$	$\lambda y [haus(y)]$

peter sieht
ein rotes Haus

$$\begin{aligned}
 & \lambda P [P(peter)] (\lambda u \exists x [rotes(x) \wedge haus(x) \wedge sieht(u, x)]) \\
 \equiv & \lambda u \exists x [rotes(x) \wedge haus(x) \wedge sieht(u, x)] (peter) \\
 \equiv & \exists x [rotes(x) \wedge haus(x) \wedge sieht(peter, x)]
 \end{aligned}$$

Wahrheitsbedingungen: Peter sieht ein rotes Haus.



λ -Terme in PROLOG

- λ -Terme werden durch PROLOG-Terme repräsentiert
- Variablen: PROLOG-Variablen

Kein Symbol-Zeichensatz in PROLOG

$\text{lambda}(X, F)$ anstelle von $\lambda x [F]$

Keine zusammengesetzten Funktionsausdrücke in PROLOG

$F@T$ anstelle von $F(T)$

Leider keine korrekte PROLOG -Syntax

Applikation wird repräsentiert, nicht 'ausgeführt'

Vereinfachung als Nachverarbeitung:

'Ausführung' der Applikation

- Beta-Konversion: Nutzung von
- $\lambda x [A](B)$ ist äquivalent mit $A\{x / B\}$

Grammatik von Fragment 1 mit Lambda-Ausdrücken

$\text{np}(\text{np}[\text{Det}, \text{N}], \text{Dsem@Nsem}) \text{ ---> } \text{det}(\text{Det}, \text{Dsem}), \text{n}(\text{N}, \text{Nsem}).$

$\text{n}(\text{n}[\text{Adj}, \text{N}], \text{Asem@Nsem}) \text{ ---> } \text{adj}(\text{Adj}, \text{Asem}), \text{n}(\text{N}, \text{Nsem}).$

$\text{vp}(\text{vp}[\text{TV}, \text{NP}], \text{Vsem@NPsem}) \text{ ---> } \text{tv}(\text{TV}, \text{Vsem}), \text{np}(\text{NP}, \text{NPsem}).$

$\text{s}(\text{s}[\text{NP}, \text{VP}], \text{NPsem@VPsem}) \text{ --->}$
 $\text{np}(\text{NP}, \text{NPsem}), \text{vp}(\text{VP}, \text{VPsem}).$

$\text{s}(\text{s}[\text{S1}, \text{C}, \text{S2}], \text{(Csem@S1sem)@S2sem}) \text{ --->}$
 $\text{s}(\text{S1}, \text{S1sem}), \text{conj}(\text{C}, \text{Csem}), \text{s}(\text{S2}, \text{S2sem}).$

$\text{vp}(\text{vp}[\text{IV}], \text{Vsem}) \text{ ---> } \text{iv}(\text{IV}, \text{Vsem}).$

$\text{np}(\text{np}[\text{PN}], \text{lambda}(\text{P}, \text{P@Nsem})) \text{ ---> } \text{pn}(\text{PN}, \text{Nsem}).$

$\text{vp}(\text{vp}[\text{VP1}, \text{C}, \text{VP2}],$
 $\text{lambda}(\text{X}, (\text{Cs@}(\text{VP1s@X}))@(\text{VP2s@X}))) \text{ --->}$
 $\text{vp}(\text{VP1}, \text{VP1s}), \text{conj}(\text{C}, \text{Cs}), \text{vp}(\text{VP2}, \text{VP2s}).$

Weiterer Nutzen für Verarbeitung

Trennung von

- Wortform-Information / Lexembeitrag
- Kategorienbeitrag: Argumentstrukturinformation / Kombinationspotential

lexem

- Wortform → Syntaktische Kategorie + Lexembeitrag

semMacro

- Syntaktische Kategorie + Lexembeitrag → Semantischer Beitrag

Kombination

- mit Grammatikregeln in der Schnittstelle zum Lexikon

Beispiel

Anstelle von

- Lexikon

`lexem(sieht, tv, lambda(Q, lambda(X, Q@lambda(Z, (sieht@X)@Z))))).`

`lexem(oeffnet, tv, lambda(Q, lambda(X, Q@lambda(Z, (oeffnet)@Z))))).`

`lexem(schliesst, tv, lambda(Q, lambda(X, Q@lambda(Z, (schliesst)@Z))))).`

- Lexikon-Grammatik-Schnittstelle

`tv(tv (-Lexem), Sem) --->
lexem(Lexem, tv, Sem).`

jetzt neu

- Lexikon

`lexem(sieht, tv, sieht).`

`lexem(oeffnet, tv, oeffnet).`

`lexem(schliesst, tv, schliesst).`

- Lexikon-Grammatik-Schnittstelle

`tv(tv/ (- Phrase), Sem) --->
lexem(Phrase, tv, LexSem),
semMacro(tv, LexSem, Sem).`

- semantisches Macro

`semMacro(tv, Rel,
lambda(Q, lambda(X,
Q@lambda(Z, (Rel@X)@Z))))).`

Kombination von Lexem- und Kategorien-Beitrag

$n(n/ (- \text{Phrase}), \text{Sem}) \text{ ---> } \text{lexem}(\text{Phrase}, n, \text{LexSem}),$
 $\text{semMacro}(n, \text{LexSem}, \text{Sem}).$

$\text{adj}(\text{adj}/ (- \text{Phrase}), \text{Sem}) \text{ ---> } \text{lexem}(\text{Phrase}, \text{adj}, \text{LexSem}),$
 $\text{semMacro}(\text{adj}, \text{LexSem}, \text{Sem}).$

$\text{det}(\text{det}/ (- \text{Phrase}), \text{Sem}) \text{ ---> } \text{lexem}(\text{Phrase}, \text{det}, \text{LexSem}),$
 $\text{semMacro}(\text{det}, \text{LexSem}, \text{Sem}).$

$\text{pn}(\text{pn}/ (- \text{Phrase}), \text{Sem}) \text{ ---> } \text{lexem}(\text{Phrase}, \text{pn}, \text{LexSem}),$
 $\text{semMacro}(\text{pn}, \text{LexSem}, \text{Sem}).$

$\text{iv}(\text{iv}/ (- \text{Phrase}), \text{Sem}) \text{ ---> } \text{lexem}(\text{Phrase}, \text{iv}, \text{LexSem}),$
 $\text{semMacro}(\text{iv}, \text{LexSem}, \text{Sem}).$

$\text{tv}(\text{tv}/ (- \text{Phrase}), \text{Sem}) \text{ ---> } \text{lexem}(\text{Phrase}, \text{tv}, \text{LexSem}),$
 $\text{semMacro}(\text{tv}, \text{LexSem}, \text{Sem}).$

$\text{conj}(\text{conj}/(- \text{Phrase}), \text{Sem}) \text{ ---> } \text{lexem}(\text{Phrase}, \text{coord}, \text{LexSem}),$
 $\text{semMacro}(\text{coord}, \text{LexSem}, \text{Sem}).$

Formenlexikon

$\text{lexem}(\text{haus}, n, \text{haus}).$	$\text{lexem}(\text{roter}, \text{adj}, \text{intersective}(\text{rot})).$
$\text{lexem}(\text{junge}, n, \text{junge}).$	$\text{lexem}(\text{rote}, \text{adj}, \text{intersective}(\text{rot})).$
$\text{lexem}(\text{peter}, \text{pn}, \text{peter}).$	$\text{lexem}(\text{rotes}, \text{adj}, \text{intersective}(\text{rot})).$
$\text{lexem}(\text{laura}, \text{pn}, \text{laura}).$	$\text{lexem}(\text{roten}, \text{adj}, \text{intersective}(\text{rot})).$
$\text{lexem}(\text{sieht}, \text{tv}, \text{sieht}).$	$\text{lexem}(\text{rotem}, \text{adj}, \text{intersective}(\text{rot})).$
$\text{lexem}(\text{lacht}, \text{iv}, \text{lacht}).$	$\text{lexem}(\text{ein}, \text{det}, \text{indef}).$
$\text{lexem}(\text{winkt}, \text{iv}, \text{winkt}).$	$\text{lexem}(\text{eine}, \text{det}, \text{indef}).$
$\text{lexem}(\text{und}, \text{coord}, \text{conj}).$	$\text{lexem}(\text{eines}, \text{det}, \text{indef}).$
$\text{lexem}(\text{oder}, \text{coord}, \text{disj}).$	$\text{lexem}(\text{einen}, \text{det}, \text{indef}).$

	$\text{lexem}(\text{kein}, \text{det}, \text{negindef}).$
	...
	$\text{lexem}(\text{jeder}, \text{det}, \text{uni}).$
	...

Zusammenfassung

typisierter Lambda-Kalkül

- Basis: Korrespondenz von syntaktischer Kategorie und semantischem Typ
- Systematische Spezifikation
 - des Kombinationspotentials der Lexeme
 - des semantischen Beitrags der Grammatik-Regeln
- Anwendung von Funktionen auf Argumente als basales Kombinationsmittel
- ergänzt um Typen-Anhebung

Ausprobieren ?

Paket Lambda

main.pl

- Grammatik mit Erzeugung der prädikatenlogischen Formeln unter Verwendung des Lambda-Kalküls
-

Treiber-Prädikate

```
go(Ex):- example_S(Ex, Phrase), % Waehle Beispiel
         write(Ex), write(': '), % Gib Beispiel-Name aus
         writeList(Phrase), nl, nl, % Gib Wortfolge aus
         analyze(Phrase).
```

```
parse:- readLine(Sentence), analyze(Sentence).
```

```
analyze(Phrase):- recognise(s(T, Sem),Phrase), % Parse
                  writeFormula(Sem),
                  pp_syn_sem(T, Sem),
                  betaConvert(Sem, SimpleSem), % vereinfache
                  writeFormula(SimpleSem),
                  pp_syn_sem(T, SimpleSem).
```

Diskussion

Jeder Junge winkt.

jeder	$\langle\langle e, t \rangle, \langle\langle e, t \rangle, t \rangle\rangle$	$\lambda R \lambda S \forall z [R(z) \Rightarrow S(z)]$
Junge	$\langle e, t \rangle$	$\lambda y [junge(y)]$
winkt	$\langle e, t \rangle$	$\lambda x [winkt(x)]$

Jeder Junge zeigt Laura ein Haus.

jeder	$\langle\langle e, t \rangle, \langle\langle e, t \rangle, t \rangle\rangle$	$\lambda R \lambda S \forall z [R(z) \Rightarrow S(z)]$
Junge	$\langle e, t \rangle$	$\lambda y [junge(y)]$
zeigt	$\langle\langle\langle e, t \rangle, t \rangle, \langle\langle e, t \rangle, t \rangle, \langle e, t \rangle\rangle\rangle$	$\lambda T \lambda Q \lambda u [T(\lambda v [Q(\lambda w [zeigt(u, v, w)])))]$ <div style="display: flex; justify-content: space-around; width: 100px; margin-top: 5px;"> dat akk </div>
Laura	e	laura
ein	$\langle\langle e, t \rangle, \langle\langle e, t \rangle, t \rangle\rangle$	$\lambda R \lambda S \exists x [R(x) \wedge S(x)]$
Haus	$\langle e, t \rangle$	$\lambda y [haus(y)]$

Literatur

- Blackburn, Patrick & Johan Bos (1999). *Representation and Inference for Natural Language. A First Course in Computational Semantics*. Kapitel 2. Ms. **Auf Anfragen**