

## 3 Theorie der $\lambda$ -Repräsentation

### 3.1 Der $\lambda$ -Operator

In der ‚reinen‘ **Typenlogik** wird jeder Ausdruck einem Typ zugewiesen.

Ein funktionaler Typ gibt dabei zu erkennen, von welchem Typ das geforderte Argument und von welchem Typ der Funktionswert ist.

Den Ausdrücken eines funktionalen Typs selbst kann so allerdings noch nicht angesehen werden, mit **welchem Argument** sie **welchen Funktionswert** ergeben.

Dieser Mangel kann überwunden werden, wenn die Typenlogik um den **Lambda-Operator**  $\lambda$  erweitert wird.

Der **typisierte Lambda-Kalkül** ist ein logisches System, das die Eigenschaften dieses Operators definiert und die Basis aller funktionalen Programmiersprachen bildet.

Er wurde von Alonzo Church (1940), einem der Begründer der theoretischen Informatik, entwickelt.

Der  $\lambda$ -Operator ist ebenso wie der  $\exists$ - und der  $\forall$ -Quantor ein Variablenbinder.

Mit ihm wird über die jeweils gebundene Variable aber nicht quantifiziert, sondern abstrahiert.

Er wird deshalb auch als  $(\lambda)$ -**Abstraktor** bezeichnet.

Im Ergebnis einer Anwendung des  $\lambda$ -Operators entsteht ein Ausdruck, der ein  **$\lambda$ -Abstrakt** oder ein  **$\lambda$ -Term** genannt wird.

Allgemein hat ein  $\lambda$ -Term die Form  $\lambda v[\alpha]$ , wobei  $v$  eine Variable und  $\alpha$  ein Ausdruck von jeweils beliebigem Typ ist.

$\lambda v$  ist das  $\lambda$ -Präfix des Ausdrucks, das alle freien Vorkommen von  $v$  in  $\alpha$  bindet, und  $\alpha$  ist der Skopus des  $\lambda$ -Operators.

Werden in einem beliebigen Ausdruck beliebige freie Variablen durch einen  $\lambda$ -Operator gebunden, dann ist das Ergebnis ein Funktionsausdruck.

$\lambda v[\alpha]$  steht für diejenige Funktion, die jedem Argument  $v$  den Funktionswert zuordnet, der durch  $\alpha$  beschrieben wird.

## Beispiel:

$\lambda x[\textit{schlafen}'(x)]$

- $\lambda x[\textit{schlafen}'(x)]$  steht für eine Funktion, die jedem Argument  $x$  den Funktionswert zuordnet, der durch  $\textit{schlafen}'(x)$  beschrieben wird.
- $\lambda x[\textit{schlafen}'(x)]$  ist ebenso wie  $\textit{schlafen}'$  ein Ausdruck vom Typ  $\langle e, t \rangle$ .

Dabei sieht man  $\lambda x[\textit{schlafen}'(x)]$  den Typ insofern an, als  $x$  ein Ausdruck vom Typ  $e$  und  $\textit{schlafen}'(x)$  ein Ausdruck vom Typ  $t$  ist.

- $\lambda x[\textit{schlafen}'(x)]$  hat jeweils dieselbe Denotation wie  $\textit{schlafen}'$ .

Die Denotation von  $\lambda x[\textit{schlafen}'(x)]$  ist also eine charakteristische Funktion, die jedem Argument  $x$ , das Element der Menge der schlafenden Individuen ist, den Wahrheitswert 1, und ansonsten den Wahrheitswert 0 zuordnet.

- $\lambda x[\textit{schlafen}'(x)]$  kann gelesen werden als: „ist ein  $x$  derart, dass  $x$  schläft“.

### 3.1.1 $\lambda$ -Abstraktion

Für die Ableitung von  $\lambda$ -Termen gilt die folgende **Regel der  $\lambda$ -Abstraktion**:

Wenn  $v$  eine Variable vom Typ  $a$  und  $\alpha$  ein Ausdruck vom Typ  $b$  ist, dann ist  $\lambda v[\alpha]$  ein Ausdruck vom Typ  $\langle a, b \rangle$ .

Die Denotation von  $\lambda v[\alpha]$  ist damit eine Funktion von  $D_a$  in  $D_b$ .

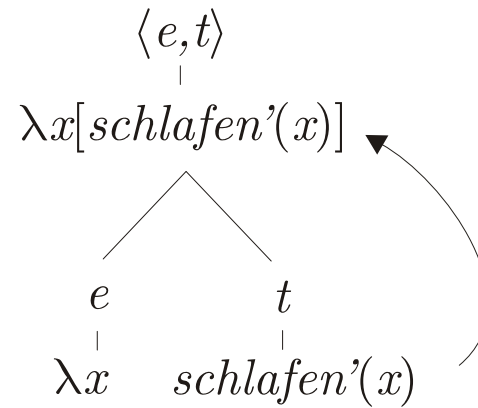
Speziell ein  $\lambda$ -Term vom Typ  $\langle e, t \rangle$  kann dadurch gewonnen werden, dass in einer Formel über eine Variable vom Typ  $e$  abstrahiert wird.

Es gilt also:

Wenn  $\phi$  ein Ausdruck vom Typ  $t$  und  $x$  eine Variable vom Typ  $e$  ist, dann ist  $\lambda x[\phi]$  ein Ausdruck vom Typ  $\langle e, t \rangle$ .

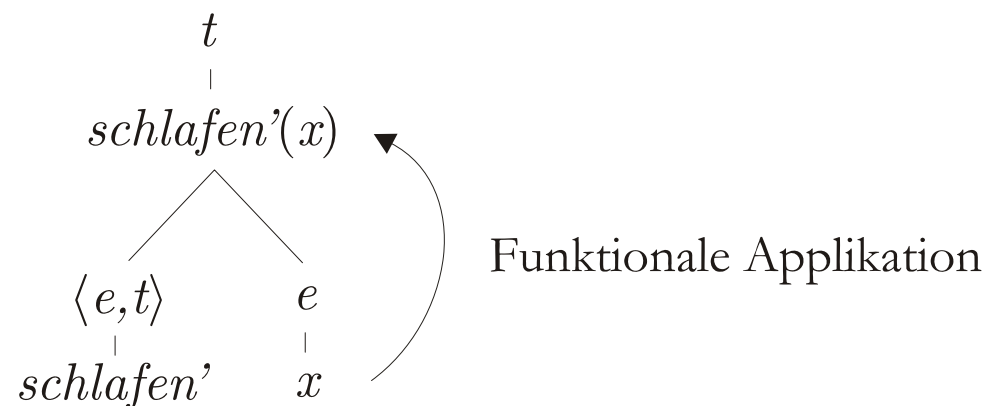
Beispiel:

Ableitung von  $\lambda x[\textit{schlafen}'(x)]$



$\lambda$ -Abstraktion

Ausgangsbasis für die  $\lambda$ -Abstraktion ist dabei die Formel  $schlafen'(x)$ , die ihrerseits durch funktionale Applikation der Prädikatskonstanten  $schlafen'$  auf die Individuenvariable  $x$  erhalten wird.



Ein  $\lambda$ -Term vom Typ  $\langle e, \langle e, t \rangle \rangle$  kann entsprechend dadurch gewonnen werden, dass in einem Ausdruck vom Typ  $t$  nacheinander über **zwei Variablen** vom Typ  $e$  abstrahiert wird.

Durch Abstraktion über die eine Variable wird zunächst ein  $\lambda$ -Term vom Typ  $\langle e, t \rangle$ , durch anschließende Abstraktion über die zweite Variable der  $\lambda$ -Term vom Typ  $\langle e, \langle e, t \rangle \rangle$  abgeleitet.

Die **Reihenfolge** der  $\lambda$ -Abstraktionen ist unter rein logischem Gesichtspunkt beliebig, d.h. durch keine logischen Prinzipien festgelegt.

Unter dem Gesichtspunkt der Verwendung von  $\lambda$ -Termen bei der semantischen Analyse der natürlichen Sprache gilt die folgende Beschränkung:

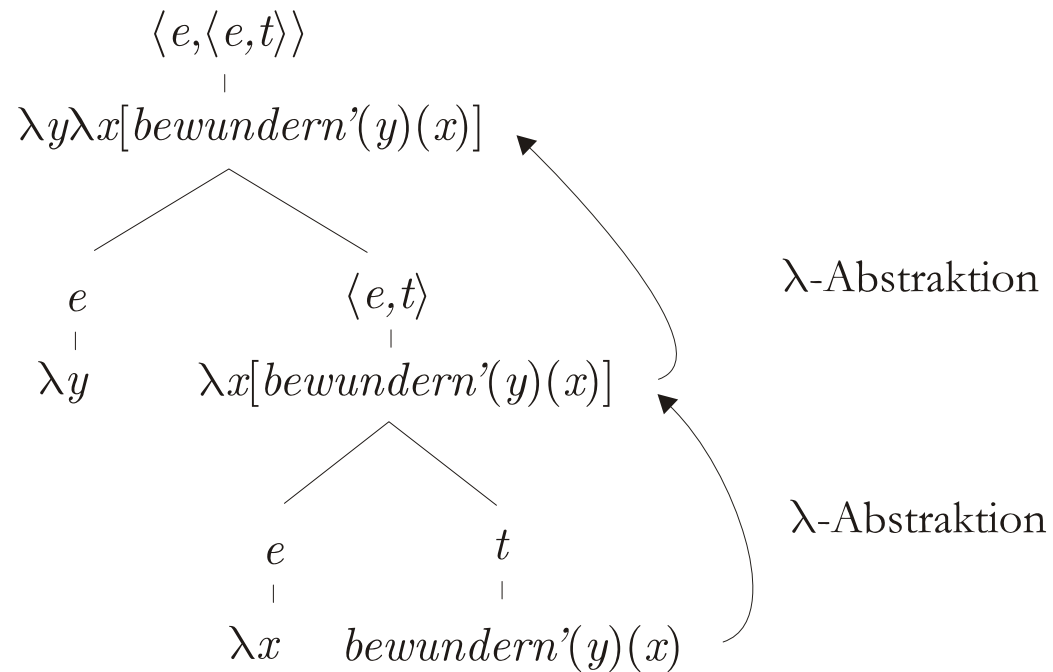
Um zu gewährleisten, dass die semantische Struktur von natürlichsprachlichen Ausdrücken **parallel** zu ihrer syntaktischen Struktur aufgebaut ist, müssen die  $\lambda$ -Abstraktionen in einer bestimmten Reihenfolge erfolgen.

Für den Fall eines  $\lambda$ -Terms vom Typ  $\langle e, \langle e, t \rangle \rangle$  heißt das:

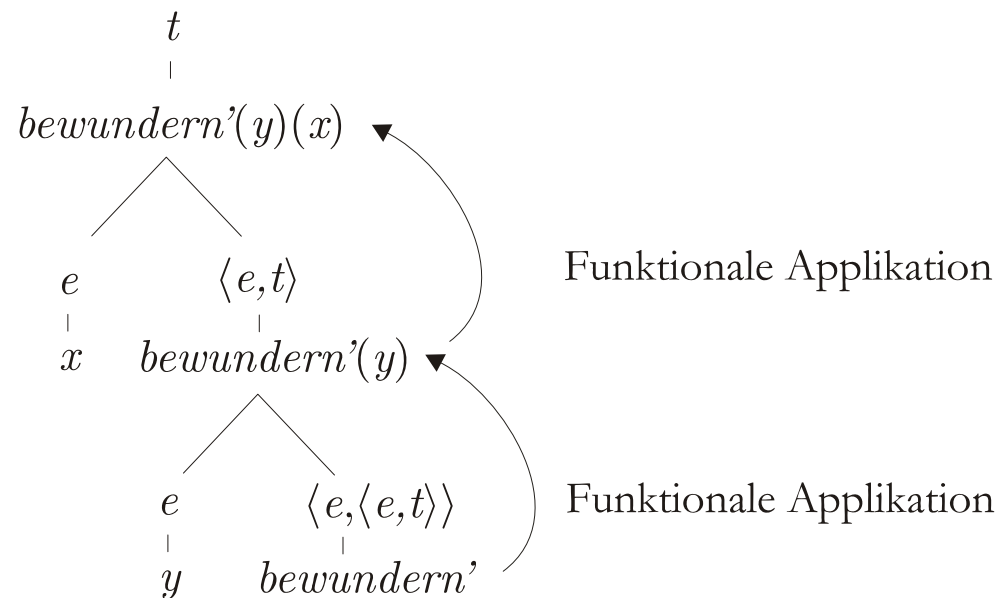
Zunächst wird über die Variable abstrahiert, die dem syntaktischen **Subjekt**, und danach über die Variable abstrahiert, die dem syntaktischen **Objekt** entspricht.

Beispiel:

Ableitung von  $\lambda y \lambda x [bewundern'(y)(x)]$



Ausgangsbasis für die  $\lambda$ -Abstraktionen ist die Formel  $bewundern'(y)(x)$ , die selbst aus einer zweimaligen funktionalen Applikation hervorgeht, und zwar dadurch, dass zunächst die Prädikatskonstante  $bewundern'$  auf die Variable  $y$  (für das syntaktische Objekt) und danach das komplexe Prädikat  $bewundern'(y)$  auf die Variable  $x$  (für das syntaktische Subjekt) appliziert wird.



### 3.1.2 $\lambda$ -Konversion

Ein  $\lambda$ -Term kann auf Argumente des jeweils geforderten Typs appliziert werden.

Gemäß der **Regel der funktionalen Applikation (FA-Regel)** gilt:

Wenn  $\lambda v[\alpha]$  ein Ausdruck vom Typ  $\langle a, b \rangle$  und  $\beta$  ein Ausdruck vom Typ  $a$  ist, dann ist  $\lambda v[\alpha](\beta)$  ein Ausdruck vom Typ  $b$ .

Insbesondere gilt:

Wenn  $\lambda v[\phi]$  ein Ausdruck vom Typ  $\langle e, t \rangle$  und  $\beta$  ein Ausdruck vom Typ  $e$  ist, dann ist  $\lambda v[\phi](\beta)$  ein Ausdruck vom Typ  $t$ .

Unter bestimmten Bedingungen kann  $\lambda v[\phi](\beta)$  dadurch **vereinfacht** werden, dass in  $\phi$  die Vorkommen der  $\lambda$ -gebundenen Variablen  $v$  durch  $\beta$  ersetzt werden.

Beispiel:

$\lambda x[\textit{schlafen}'(x)](\textit{Peter}'')$

- Die Formel  $\lambda x[\textit{schlafen}'(x)](\textit{Peter}'')$  kann gelesen werden als: „Peter ist ein  $x$  derart, dass  $x$  schläft“.
- $\lambda x[\textit{schlafen}'(x)](\textit{Peter}'')$  besagt das Gleiche wie die Formel  $\textit{schlafen}'(\textit{Peter}'')$ , die gelesen wird als: „Peter schläft“.

Allgemein gilt das folgende **Prinzip der  $\lambda$ -Konversion**  
(vorläufige Fassung):

Wenn  $v$  eine Variable und  $\beta$  ein Ausdruck vom selben Typ sind,  
dann gilt:

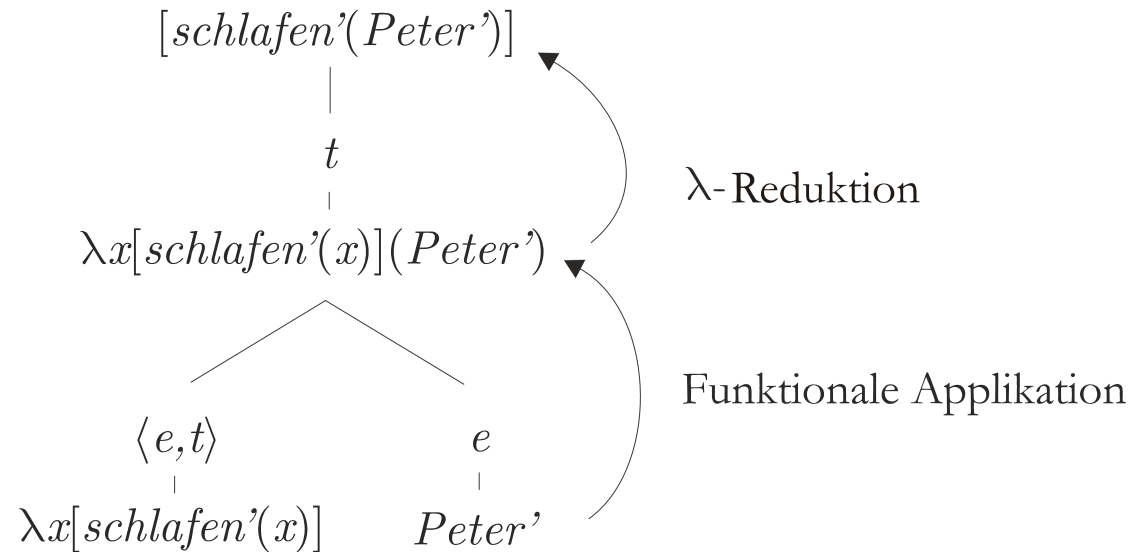
$$\lambda v[\alpha](\beta) = \alpha[\beta/v],$$

wobei  $\alpha[\beta/v]$  aus einer Substitution von  $\beta$  für alle freien  
Vorkommen von  $v$  in  $\alpha$  hervorgeht.

Wird auf Grund des Prinzips der  $\lambda$ -Konversion in einem Ausdruck  
 $\lambda v[\alpha](\beta)$  durch  $\alpha[\beta/v]$  ersetzt, spricht man auch von  **$\lambda$ -Reduktion**.

Beispiel:

$$\lambda x[\textit{schlafen}'(x)](\textit{Peter}') = \textit{schlafen}'(\textit{Peter}')$$

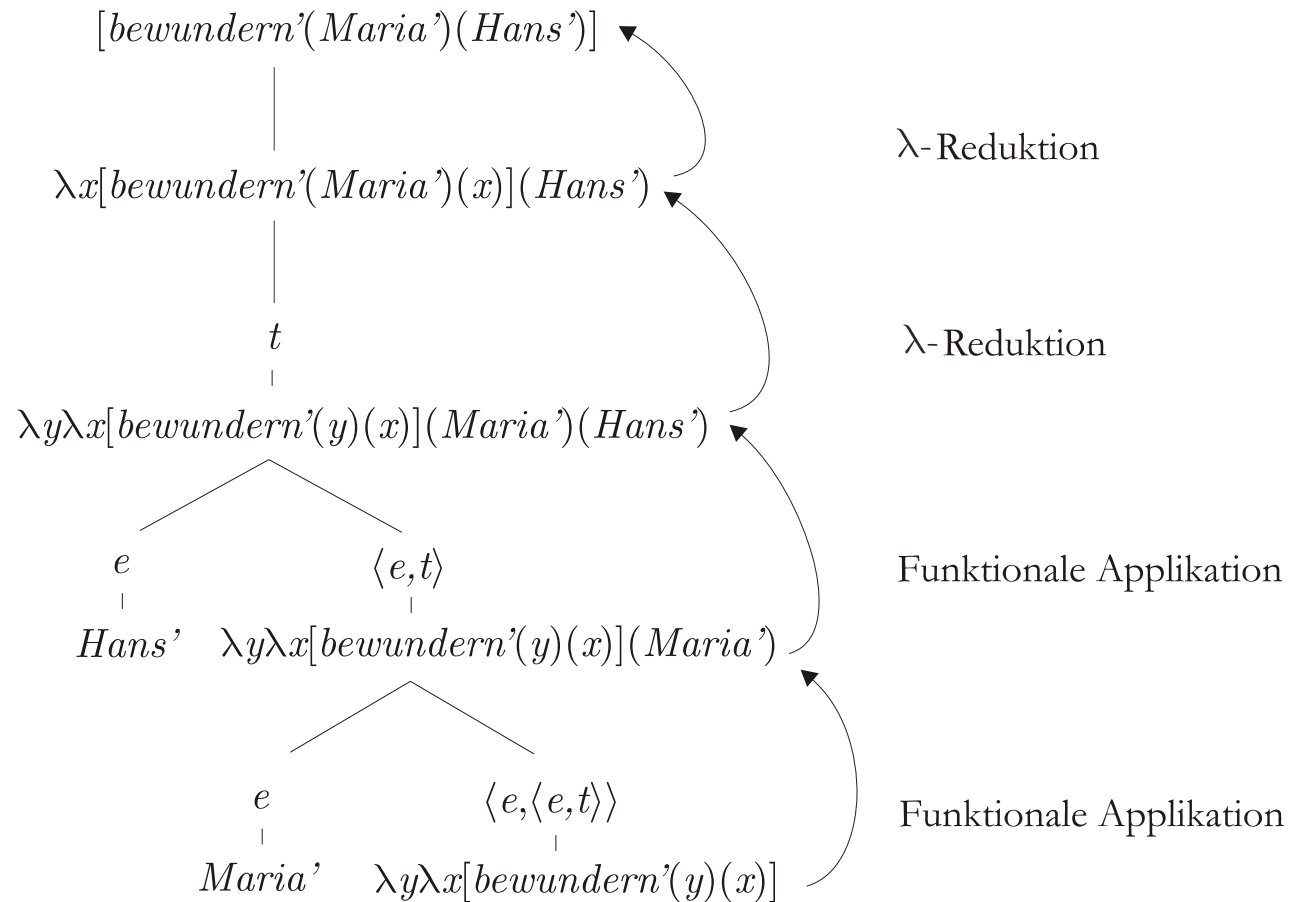


Die Methode der  $\lambda$ -Reduktion wird hierbei faktisch so angewandt, dass das  $\lambda$ -Präfix weggestrichen und in  $\textit{schlafen}'(x)$  an die Stelle von  $x$  das Argument  $\textit{Peter}'$  geschrieben wird.

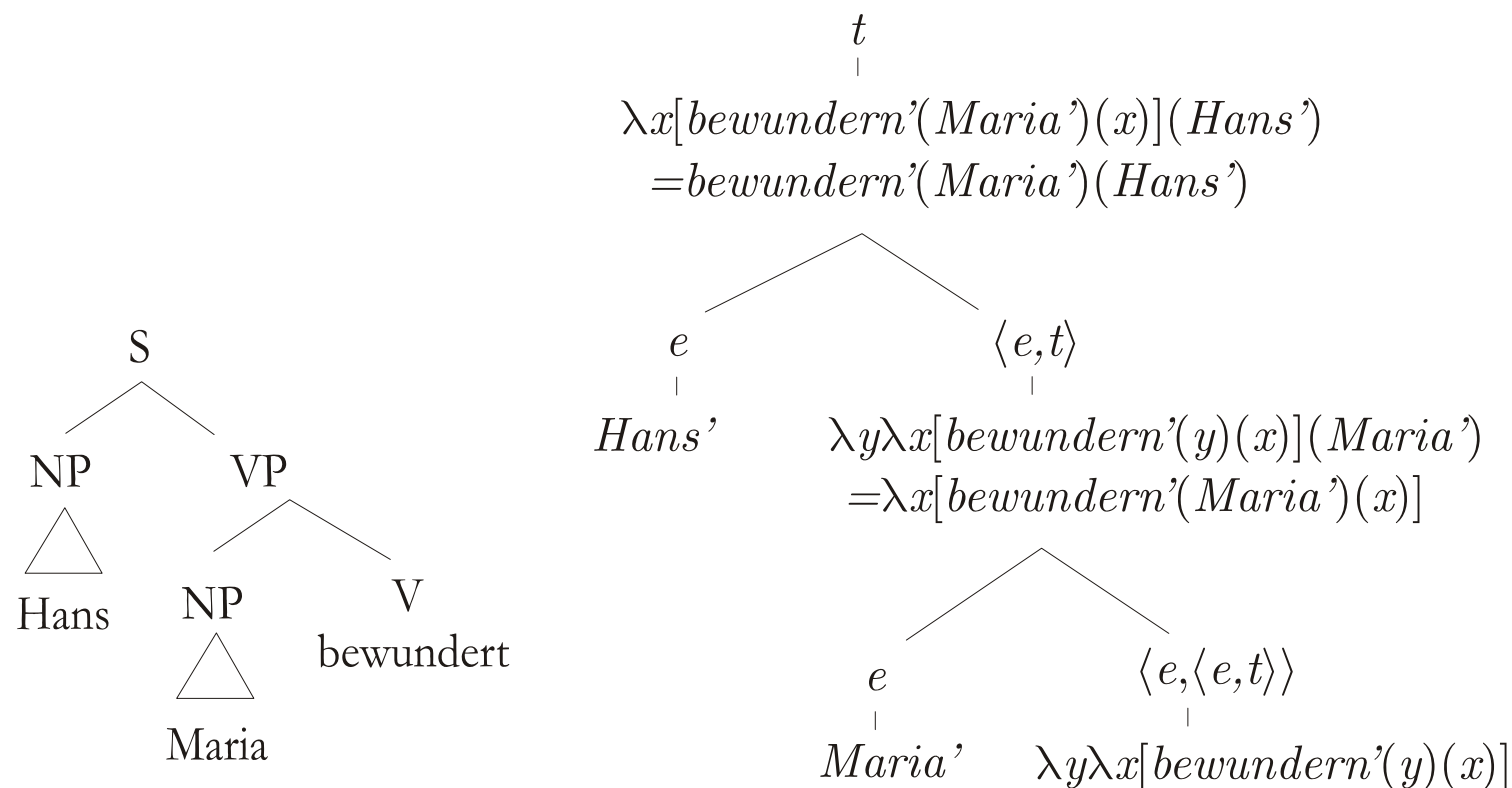
Bei der Anwendung eines  $\lambda$ -Terms vom Typ  $\langle e, \langle e, t \rangle \rangle$  auf zwei Argumente vom Typ  $e$  wird entsprechend verfahren, d.h. es werden zwei funktionale Applikationen und, falls möglich, zwei  $\lambda$ -Reduktionen vorgenommen.

Beispiel:

$$\lambda y \lambda x [\text{bewundern}'(y)(x)](\text{Maria}')(\text{Hans}') = \text{bewundern}'(\text{Maria}')(\text{Hans}')$$



Bei der Anwendung des Prädikats  $\lambda y\lambda x[bewundern'(y)(x)]$  auf die Argumente  $Mar\acute{a}$ ' und  $Hans$ ' kann somit die Parallelitat der semantischen Struktur des naturlichsprachlichen Satzes *Hans bewundert Mar\acute{a}* zu seiner syntaktischen Struktur gewahrleistet werden.



### 3.1.3 Komplexe $\lambda$ -Terme

Eine wesentliche Eigenschaft des  $\lambda$ -Operators ist, dass man mit ihm beliebig **komplexe**  $\lambda$ -Terme bilden kann.

Dadurch kann auch für viele **komplexe** natürlichsprachliche Ausdrücke eine **semantische Repräsentation SR** angegeben werden, für die dies bisher nicht möglich war.

## Beispiele:

$$(1) \text{ SR}(\textit{sich bewundern}) = \lambda x[\textit{bewundern}'(x)(x)]$$

$$(2) \text{ SR}(\textit{bewundert werden (von)}) = \lambda x\lambda y[\textit{bewundern}'(y)(x)]$$

$$(3) \text{ SR}(\textit{bewundert werden}) = \lambda x\exists y[\textit{bewundern}'(x)(y)]$$

$$(4) \text{ SR}(\textit{schlafen und bewundert werden}) = \lambda x[\textit{schlafen}'(x)\wedge\exists y[\textit{bewundern}'(x)(y)]]$$

$$(5) \text{ SR}(\textit{unverheiratet}) = \lambda x[\neg\textit{verheiratet}'(x)]$$

$$(6) \text{ SR}(\textit{unverheiratete Frau}) = \lambda x[\neg\textit{verheiratet}'(x)\wedge\textit{Frau}'(x)]$$

$$(7) \text{ SR}(\textit{eine unverheiratete Frau}) = \lambda P\exists x[\neg\textit{verheiratet}'(x)\wedge\textit{Frau}'(x)\wedge P(x)]$$

[?] Gewinne aus den folgenden Formeln mögliche  $\lambda$ -Terme durch  $\lambda$ -Abstraktion über vorkommende Variablen:

(1)  $krank'(x) \wedge hilflos'(x)$

(2)  $krank'(x) \wedge P(x)$

(3)  $P(x) \wedge Q(x)$

(4)  $\exists x[P(x) \wedge Q(x)]$

[?] Von welchem Typ sind die jeweiligen  $\lambda$ -Terme?

[?] Appliziere die  $\lambda$ -Terme auf passende Argumente und nimm mögliche Vereinfachungen durch  $\lambda$ -Reduktion vor.

## 3.2 $\lambda$ -Typenlogik (TL $\lambda$ )

### 3.2.1 Syntax von TL $\lambda$

Die Menge  $T$  der Typen von TL $\lambda$  ist **identisch** mit der Menge der Typen von TL.

#### **D3.1 Typen von TL $\lambda$**

- (1)  $e \in T$ .
- (2)  $t \in T$ .
- (3) Wenn  $a, b \in T$ , dann  $\langle a, b \rangle \in T$ .
- (4) Nichts sonst ist in  $T$ .

Das Vokabular von TL wird um den  $\lambda$ -Operator **erweitert**.

### **D3.2 Vokabular von TL $\lambda$**

Das Vokabular von TL $\lambda$  enthält:

- (1) für jeden Typ  $a$  eine Menge  $VAR_a$  von Variablen des Typs  $a$ :  
 $v_{n,a}$  ( $n \geq 1$ ),
- (2) für jeden Typ  $a$  eine Menge  $CON_a$  von Konstanten des Typs  $a$ :  
 $c_{n,a}$  ( $n \geq 1$ ),
- (3) die Konnektoren:  $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$ ,
- (4) das Identitätsprädikat:  $=$ ,
- (5) die Quantoren:  $\forall, \exists$ ,
- (6) den Lambda-Operator:  $\lambda$ ,
- (7) die technischen Hilfszeichen:  $(, ), [, ]$ .

## Syntaktische Regeln von $TL\lambda$

Die syntaktischen Regeln von  $TL\lambda$  legen für jeden Typ  $a$  die Menge  $WFA_a$  der wohlgeformten Ausdrücke des Typs  $a$  fest.

Ihre Vereinigung über alle Typen  $a$  ist die Menge  $WFA$  der wfAe von  $TL\lambda$ .

### D3.3 Wohlgeformte Ausdrücke von TL $\lambda$

- (1) Wenn  $\alpha \in VAR_a$  oder  $\alpha \in CON_a$ , dann  $\alpha \in WFA_a$ .
- (2) Wenn  $v \in VAR_a$  und  $\alpha \in WFA_b$ , dann  $\lambda v[\alpha] \in WFA_{\langle a,b \rangle}$ .
- (3) Wenn  $\alpha \in WFA_{\langle a,b \rangle}$  und  $\beta \in WFA_a$ , dann  $\alpha(\beta) \in WFA_b$ .
- (4) Wenn  $\phi, \psi \in WFA_t$ , dann  $\neg\phi, (\phi \wedge \psi), (\phi \vee \psi), (\phi \rightarrow \psi), (\phi \leftrightarrow \psi) \in WFA_t$ .
- (5) Wenn  $\alpha, \beta \in WFA_a$ , dann  $(\alpha = \beta) \in WFA_t$ .
- (6) Wenn  $\phi \in WFA_t$  und  $v \in VAR_a$ , dann  $\forall v[\phi], \exists v[\phi] \in WFA_t$ .
- (7) Nichts sonst ist in  $WFA$ .

Mit D3.3 (2) wird die Menge der syntaktischen Regeln von TL um eine syntaktische Regel für den  $\lambda$ -Operator (**Regel der  $\lambda$ -Abstraktion**) erweitert.

## Notationskonventionen:

- Eckige Klammern zur Markierung des Skopus eines Operators können weggelassen werden; als Skopus des Operators ist dann der unmittelbar auf ihn folgende wfa zu verstehen.

Beispiel:       $\lambda y[\lambda x[R(y)(x)]]$   
                   $= \lambda y\lambda x[R(y)(x)]$

- Runde Klammern zur Markierung der funktionalen Applikation auf Argumente können weggelassen werden; die Reihenfolge der Applikationen ist dann von links nach rechts zu verstehen.

Beispiel:       $(\lambda x\lambda y[Q(x) \rightarrow P(y)](a))(b)$   
                   $= \lambda x\lambda y[Q(x) \rightarrow P(y)](a)(b)$

### 3.2.2 Semantik von $TL\lambda$

Die Menge der Domänen der Typen von  $TL\lambda$  ist **identisch** mit der Menge der Domänen der Typen von  $TL$ .

#### **D3.4 Domänen der Typen von $TL\lambda$**

- (1)  $D_e = D$
- (2)  $D_t = \{0,1\}$
- (3) Für beliebige  $a, b \in T$  gilt:  $D_{\langle a, b \rangle} = D_b^{D_a}$ .
- (4) Nichts sonst ist Domäne eines Typs von  $TL\lambda$ .

Die Begriffe des Modells und der Variablenbelegung sind gegenüber  $TL$  unverändert.

## Semantische Regeln von TL $\lambda$

Die semantischen Regeln von TL $\lambda$  legen **parallel** zu den syntaktischen Regeln von TL $\lambda$  für jeden Typ  $a$  die Denotation eines beliebigen Ausdrucks  $\alpha \in WFA_a$  fest.

### D3.5 Denotation eines wfA von TL $\lambda$ bzgl. $M$ und $g$

- (1) Wenn  $\alpha \in CON_a$ , dann  $\llbracket \alpha \rrbracket^{M,g} = I(\alpha)$ .  
Wenn  $\alpha \in VAR_a$ , dann  $\llbracket \alpha \rrbracket^{M,g} = g(\alpha)$ .
- (2) Wenn  $v \in VAR_a$  und  $\alpha \in WFA_b$ , dann ist  $\llbracket \lambda v[\alpha] \rrbracket^{M,g}$  diejenige Funktion  $h$  von  $D_a$  in  $D_b$ , so dass für alle  $d \in D_a$  gilt:  $h(d) = \llbracket \alpha \rrbracket^{M,g[v \rightarrow d]}$ .
- (3) Wenn  $\alpha \in WFA_{\langle a,b \rangle}$  und  $\beta \in WFA_a$ , dann  $\llbracket \alpha(\beta) \rrbracket^{M,g} = \llbracket \alpha \rrbracket^{M,g}(\llbracket \beta \rrbracket^{M,g})$ .

(4) Wenn  $\phi, \psi \in WFA_t$ , dann

$$\llbracket \neg \phi \rrbracket^{M,g} = 1 \text{ gdw } \llbracket \phi \rrbracket^{M,g} = 0,$$

$$\llbracket \phi \wedge \psi \rrbracket^{M,g} = 1 \text{ gdw } \llbracket \phi \rrbracket^{M,g} = \llbracket \psi \rrbracket^{M,g} = 1,$$

$$\llbracket \phi \vee \psi \rrbracket^{M,g} = 1 \text{ gdw } \llbracket \phi \rrbracket^{M,g} = 1 \text{ oder } \llbracket \psi \rrbracket^{M,g} = 1,$$

$$\llbracket \phi \rightarrow \psi \rrbracket^{M,g} = 1 \text{ gdw } \llbracket \phi \rrbracket^{M,g} = 0 \text{ oder } \llbracket \psi \rrbracket^{M,g} = 1,$$

$$\llbracket \phi \leftrightarrow \psi \rrbracket^{M,g} = 1 \text{ gdw } \llbracket \phi \rrbracket^{M,g} = \llbracket \psi \rrbracket^{M,g}.$$

(5) Wenn  $\alpha, \beta \in WFA_a$ , dann  $\llbracket \alpha = \beta \rrbracket^{M,g} = 1$  gdw  $\llbracket \alpha \rrbracket^{M,g} = \llbracket \beta \rrbracket^{M,g}$ .

(6) Wenn  $\phi \in WFA_t$  und  $v \in VAR_a$ , dann

$$\llbracket \forall v[\phi] \rrbracket^{M,g} = 1 \text{ gdw f\u00fcr jedes } d \in D_a \text{ gilt: } \llbracket \phi \rrbracket^{M,g[v \rightarrow d]} = 1,$$

$$\llbracket \exists v[\phi] \rrbracket^{M,g} = 1 \text{ gdw es mindestens ein } d \in D_a \text{ gibt, so dass gilt:}$$

$$\llbracket \phi \rrbracket^{M,g[v \rightarrow d]} = 1.$$

Mit **D3.5** (2) wird die Menge der semantischen Regeln von TL um eine semantische Regel f\u00fcr den  $\lambda$ -Operator **erweitert**.

Ein **Spezialfall** von **D3.5** (2) ist die folgende Regel:

Wenn  $v \in VAR_e$  und  $\phi \in WFA_t$ , dann ist  $\llbracket \lambda v[\phi] \rrbracket^{M,g}$  diejenige Funktion  $h$  von  $D$  in  $\{0,1\}$ , so dass für alle  $d \in D$  gilt:

$$h(d) = \llbracket \phi \rrbracket^{M,g[v \rightarrow d]},$$

d.h.  $h(d) = 1$  gdw  $\llbracket \phi \rrbracket^{M,g[v \rightarrow d]} = 1$ .

### Beispiel:

$\llbracket \lambda x[\text{schlafen}'(x)] \rrbracket^{M,g}$  ist diejenige Funktion  $h$  von  $D$  in  $\{0,1\}$ , so dass für alle  $d \in D$  gilt:

$$h(d) = \llbracket \text{schlafen}'(x) \rrbracket^{M,g[x \rightarrow d]} \quad (\text{nach D3.5 (2)})$$

$$= \llbracket \text{schlafen}' \rrbracket^{M,g[x \rightarrow d]}(\llbracket x \rrbracket^{M,g[x \rightarrow d]}) \quad (\text{nach D3.5 (3)})$$

$$= I(\text{schlafen}')(g[x \rightarrow d](x)) \quad (\text{nach D3.5 (1)})$$

$$= I(\text{schlafen}')(d) \quad (\text{Variablenbelegung})$$

Die (charakteristische) Funktion  $h$  ordnet also jedem Individuum aus  $D$  den Wahrheitswert zu, den das Individuum als Argument jener (charakteristischen) Funktion zugeordnet bekommt, die die Interpretation von  $\text{schlafen}'$  ist.

Angenommen, es gelte:

$$I(\textit{schlafen}') = \begin{bmatrix} \text{Peter} \rightarrow 0 \\ \text{Maria} \rightarrow 0 \\ \text{Hans} \rightarrow 1 \end{bmatrix} \text{ und } d = \text{Maria}.$$

Dann gilt:  $I(\textit{schlafen}')(d) = 0$ .

## Problem:

Die in 3.1.2 angeführte **vorläufige** Fassung des **Prinzips der  $\lambda$ -Konversion** muss präzisiert werden, weil seine uneingeschränkte Anwendung zu falschen Ergebnissen führt.

Wenn  $v$  eine Variable und  $\beta$  ein Ausdruck vom selben Typ sind, dann gilt:

$$\lambda v[\alpha](\beta) = \alpha[\beta/v],$$

wobei  $\alpha[\beta/v]$  aus einer Substitution von  $\beta$  für alle freien Vorkommen von  $v$  in  $\alpha$  hervorgeht.

Beispiel:  $\lambda x \exists y [kennen'(y)(x)](y) \neq \exists y [kennen'(y)(y)]$ ,  
wobei  $\exists y [kennen'(y)(y)]$  aus einer Substitution von  $y$  für alle freien Vorkommen von  $x$  in  $\exists y [kennen'(y)(x)]$  hervorgeht.

### Einschränkende Bedingung:

Eine freie Variable **darf nicht** in den Skopus eines Operators substituiert werden, der sie bindet.

D.h. die Variable, für die substituiert wird, **muss frei** für den zu substituierenden Ausdruck sein.

**D3.6** Eine Variable  $v$  ist in einem Ausdruck  $\alpha$  frei für einen Ausdruck  $\beta$  gdw kein freies Vorkommen von  $v$  in  $\alpha$  im Skopus eines Quantors  $\forall v'$  oder  $\exists v'$  oder eines  $\lambda$ -Operators  $\lambda v'$  steht und  $v'$  frei in  $\beta$  vorkommt.

Beispiel:  $\lambda p \exists x [P(x) \wedge p](Q(x))$ ,  
wobei  $p, Q(x) \in WFA_t$   
und  $p$  in  $\exists x [P(x) \wedge p]$  nicht frei ist für  $Q(x)$ , weil  $p$  im  
Skopus von  $\exists x$  steht.

Spezialfall:  $\beta = v'$

Beispiel:  $\lambda x \exists y [kennen'(y)(x)](y)$ ,  
wobei  $x, y \in VAR_e$   
und  $x$  in  $\exists y [kennen'(y)(x)]$  nicht frei ist für  $y$ , weil  $x$  im  
Skopus von  $\exists y$  steht.

In einer präzisierten Fassung wird das **Prinzip der  $\lambda$ -Konversion** wie folgt formuliert:

Wenn  $v$  eine Variable und  $\beta$  ein Ausdruck vom selben Typ sind und  $v$  in  $\alpha$  **frei für**  $\beta$  ist, dann gilt:

$$\lambda v[\alpha](\beta) = \alpha[\beta/v],$$

wobei  $\alpha[\beta/v]$  aus einer Substitution von  $\beta$  für alle freien Vorkommen von  $v$  in  $\alpha$  hervorgeht.

- [?] Wie kann das Prinzip der  $\lambda$ -Konversion auch in jenen Fällen angewandt werden, in denen die einschränkende Bedingung zunächst nicht erfüllt wird?

# Übungen

Ü3.1 – Ü3.5

Termin: nächstes Tutorium