

3 Theorie der λ -Repräsentation

[Dowty 98-111, Gamut 102-116, Partee 338-371, Chierchia 391-429]

3.1 Der λ -Operator

In der ‚reinen‘ Typenlogik wird jedem Ausdruck ein 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** λ 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 λ -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 (λ -)**Abstraktor** bezeichnet. Im Ergebnis einer Anwendung des λ -Operators entsteht ein Ausdruck, der ein **λ -Abstrakt** oder ein **λ -Term** genannt wird.

Allgemein hat ein λ -Term die Form $\lambda v[\alpha]$, wobei v eine Variable und α ein Ausdruck von jeweils beliebigem Typ ist. λv ist das λ -Präfix des Ausdrucks, das alle freien Vorkommen von v in α bindet, und α ist der Skopus des λ -Operators.

Werden in einem beliebigen Ausdruck beliebige freie Variablen durch einen λ -Operator gebunden, dann ist das Ergebnis ein Funktionsausdruck. $\lambda v[\alpha]$ steht für diejenige Funktion, die jedem Argument v den Funktionswert zuordnet, der durch α beschrieben wird.

Beispiel: $\lambda x[\text{schlafen}'(x)]$

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

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

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

Die Denotation von $\lambda x[\text{schlafen}'(x)]$ 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.

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

3.1.1 λ -Abstraktion

Für die Ableitung von λ -Termen gilt die folgende **Regel der λ -Abstraktion**:

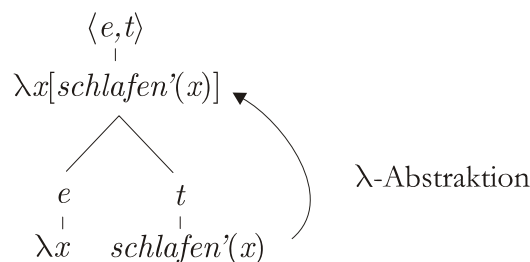
Wenn v eine Variable vom Typ a und α ein Ausdruck vom Typ b ist, dann ist $\lambda v[\alpha]$ ein Ausdruck vom Typ $\langle a, b \rangle$.

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

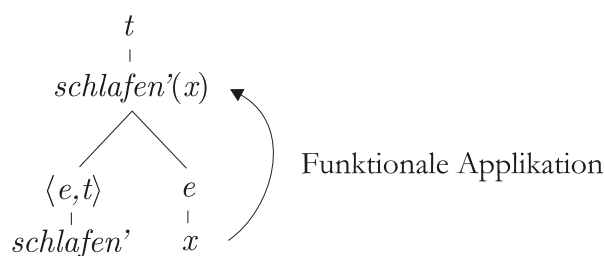
Speziell ein λ -Term vom Typ $\langle e, t \rangle$ kann dadurch gewonnen werden, dass in einer Formel über eine Variable des Typs e abstrahiert wird. Es gilt also:

Wenn ϕ 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 $\lambda x[\text{schlafen}'(x)]$



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



Ein λ -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 λ -Term vom Typ $\langle e, t \rangle$, durch anschließende Abstraktion über die zweite Variable der λ -Term vom Typ $\langle e, \langle e, t \rangle \rangle$ abgeleitet.

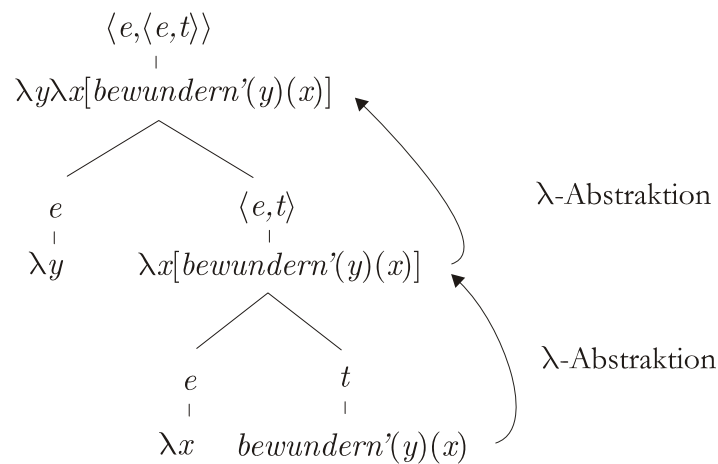
Die Reihenfolge der λ -Abstraktionen ist unter rein logischem Gesichtspunkt beliebig, d.h. durch keine logischen Prinzipien festgelegt.

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

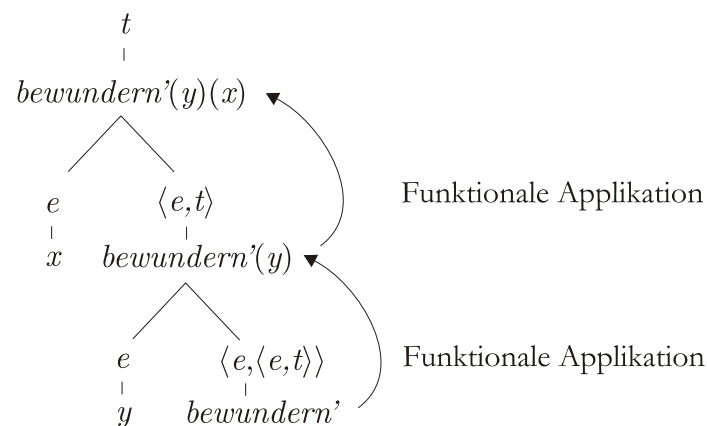
Um zu gewährleisten, dass die semantische Struktur von natürlichsprachlichen Ausdrücken parallel zu ihrer syntaktischen Struktur aufgebaut ist, müssen die λ -Abstraktionen in einer bestimmten Reihenfolge erfolgen.

Für den Fall eines λ -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 λ -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 λ -Konversion

Ein λ -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 β 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 β 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 ϕ die Vorkommen der λ -gebundenen Variablen v durch β ersetzt werden.

Beispiel: $\lambda x[\text{schlafen}'(x)](\text{Peter}')$

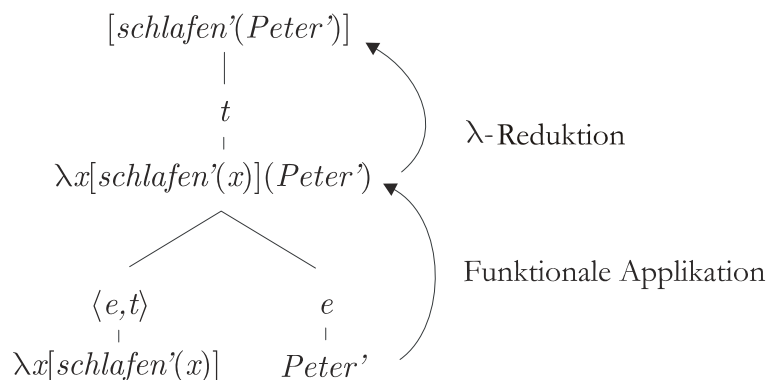
- Die Formel $\lambda x[\text{schlafen}'(x)](\text{Peter}')$ kann gelesen werden als: „Peter ist ein x derart, dass x schläft“.
- $\lambda x[\text{schlafen}'(x)](\text{Peter}')$ besagt das Gleiche wie die Formel $\text{schlafen}'(\text{Peter}')$, die gelesen wird als: „Peter schläft“.

Allgemein gilt das folgende **Prinzip der λ -Konversion** (vorläufige Fassung):

Wenn v eine Variable und β ein Ausdruck vom selben Typ sind, dann gilt: $\lambda v[\alpha](\beta) = \alpha[\beta/v]$, wobei $\alpha[\beta/v]$ aus einer Substitution von β für alle freien Vorkommen von v in α hervorgeht.

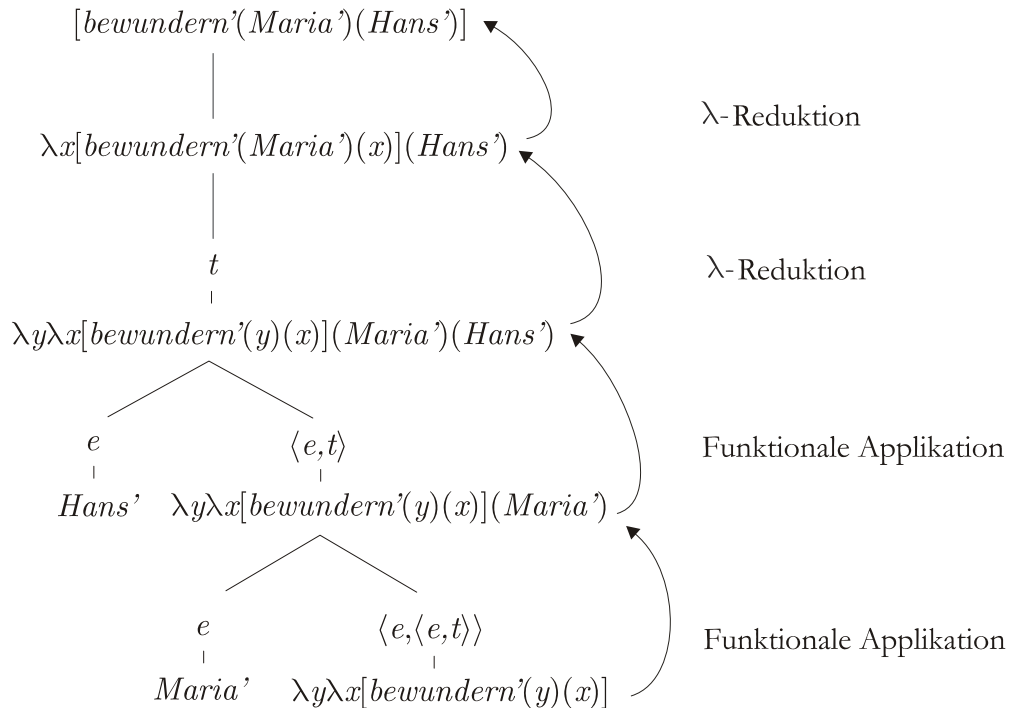
Wird auf Grund des Prinzips der λ -Konversion in einem Ausdruck $\lambda v[\alpha](\beta)$ durch $\alpha[\beta/v]$ ersetzt, spricht man auch von **λ -Reduktion**.

Beispiel: $\lambda x[\text{schlafen}'(x)](\text{Peter}') = \text{schlafen}'(\text{Peter}')$

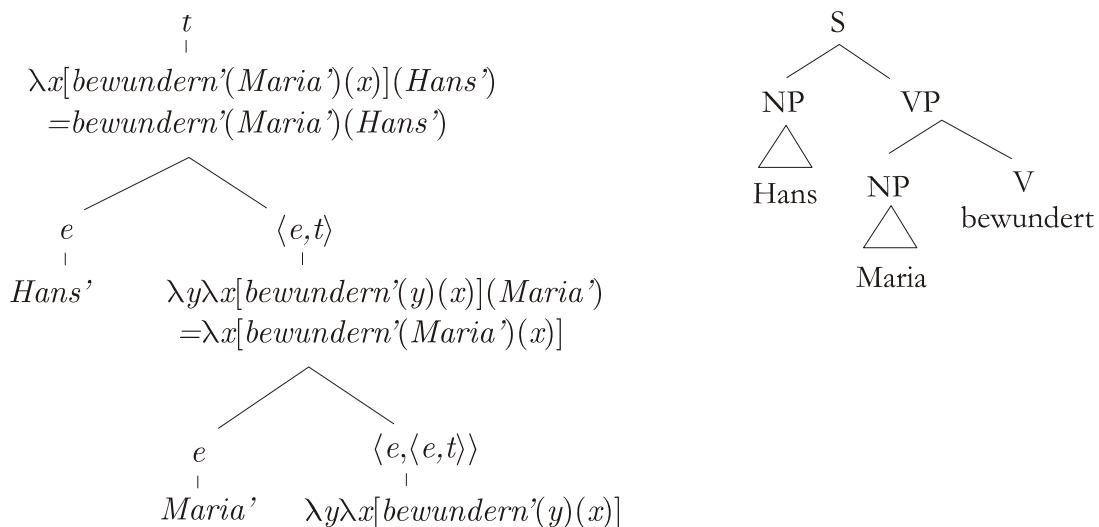


Bei der Anwendung eines λ -Terms vom Typ $\langle e, \langle e, t \rangle \rangle$ auf zwei Argumente vom Typ e wird entsprechend verfahren, d.h. es werden zwei funktionale Applikationen und, falls möglich, zwei λ -Reduktionen vorgenommen.

Beispiel: $\lambda y \lambda x [bewundern'(y)(x)](Maria')(Hans') = bewundern'(Maria')(Hans')$



Bei der Anwendung des Prädikats $\lambda y \lambda x [bewundern'(y)(x)]$ auf die Argumente *Maria'* und *Hans'* kann somit die Parallelität der semantischen Struktur des natürlichsprachlichen Satzes *Hans bewundert Maria* zu seiner syntaktischen Struktur gewährleistet werden.



3.1.3 Komplexe λ -Terme

Eine wesentliche Eigenschaft des λ -Operators ist, dass man mit ihm beliebig komplexe λ -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) $\mathbf{SR}(\text{*sich bewundern*}) = \lambda x[\text{*bewundern'*}(x)(x)]$
- (2) $\mathbf{SR}(\text{*bewundert werden (von)*}) = \lambda x \lambda y[\text{*bewundern'*}(y)(x)]$
- (3) $\mathbf{SR}(\text{*bewundert werden*}) = \lambda x \exists y[\text{*bewundern'*}(x)(y)]$
- (4) $\mathbf{SR}(\text{*schlafen und bewundert werden*}) = \lambda x[\text{*schlafen'*}(x) \wedge \exists y[\text{*bewundern'*}(x)(y)]]$ =
- (5) $\mathbf{SR}(\text{*unverheiratet*}) = \lambda x[\neg \text{*verheiratet'*}(x)]$
- (6) $\mathbf{SR}(\text{*unverheiratete Frau*}) = \lambda x[\neg \text{*verheiratet'*}(x) \wedge \text{*Frau'*}(x)]$
- (7) $\mathbf{SR}(\text{*eine unverheiratete Frau*}) = \lambda P \exists x[\neg \text{*verheiratet'*}(x) \wedge \text{*Frau'*}(x) \wedge P(x)]$

☐ Gewinne aus den folgenden Formeln mögliche λ -Terme durch λ -Abstraktion über vorkommende Variablen:

- (1) $\text{*krank'*}(x) \wedge \text{*hilflos'*}(x)$
- (2) $\text{*krank'*}(x) \wedge P(x)$
- (3) $P(x) \wedge Q(x)$
- (4) $\exists x[P(x) \wedge Q(x)]$

☐ Von welchem Typ sind die jeweiligen λ -Terme?

☐ Appliziere die λ -Terme auf passende Argumente und nimm mögliche Vereinfachungen durch λ -Reduktion vor.

3.2 λ -Typenlogik (TL λ)

3.2.1 Syntax von TL λ

Die Menge T der Typen von TL λ ist identisch mit der Menge der Typen von TL.

D3.1 Typen von TL λ

- (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 λ -Operator erweitert.

D3.2 Vokabular von TL λ

Das Vokabular von TL λ 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: λ ,
- (7) die technischen Hilfszeichen: $(,), [,]$.

Syntaktische Regeln von TL λ

Die syntaktischen Regeln von TL λ legen für jeden Typ a die Menge WFA_a der wohlgeformten Ausdrücke wfa des Typs a fest. Ihre Vereinigung über alle Typen a ist die Menge WFA der wfae von TL λ .

D3.3 Wohlgeformte Ausdrücke von TL λ

- (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 λ -Operator (Regel der λ -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 λ

Die Menge der Domänen der Typen von TL λ ist identisch mit der Menge der Domänen der Typen von TL.

D3.4 Domänen der Typen von TL λ

- (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 λ .

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

Semantische Regeln von TL λ

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

D3.5 Denotation eines wfa von TL λ 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

$$\begin{aligned} \llbracket \neg \phi \rrbracket^{M,g} = 1 & \text{ gdw } \llbracket \phi \rrbracket^{M,g} = 0, \\ \llbracket \phi \wedge \psi \rrbracket^{M,g} = 1 & \text{ gdw } \llbracket \phi \rrbracket^{M,g} = \llbracket \psi \rrbracket^{M,g} = 1, \\ \llbracket \phi \vee \psi \rrbracket^{M,g} = 1 & \text{ gdw } \llbracket \phi \rrbracket^{M,g} = 1 \text{ oder } \llbracket \psi \rrbracket^{M,g} = 1, \\ \llbracket \phi \rightarrow \psi \rrbracket^{M,g} = 1 & \text{ gdw } \llbracket \phi \rrbracket^{M,g} = 0 \text{ oder } \llbracket \psi \rrbracket^{M,g} = 1, \\ \llbracket \phi \leftrightarrow \psi \rrbracket^{M,g} = 1 & \text{ gdw } \llbracket \phi \rrbracket^{M,g} = \llbracket \psi \rrbracket^{M,g}. \end{aligned}$$

(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

$$\begin{aligned} \llbracket \forall v[\phi] \rrbracket^{M,g} = 1 & \text{ gdw f\"ur jedes } d \in D_a \text{ gilt: } \llbracket \phi \rrbracket^{M,g[v \rightarrow d]} = 1, \\ \llbracket \exists v[\phi] \rrbracket^{M,g} = 1 & \text{ gdw es mindestens ein } d \in D_a \text{ gibt, so dass gilt: } \llbracket \phi \rrbracket^{M,g[v \rightarrow d]} = 1. \end{aligned}$$

Mit D3.5 (2) wird die Menge der semantischen Regeln von TL um eine semantische Regel f\"ur den λ -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 f\"ur 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 f\"ur alle $d \in D$ gilt:

$$\begin{aligned} h(d) &= \llbracket \text{schlafen}'(x) \rrbracket^{M,g[x \rightarrow d]} && \text{(nach D3.5 (2))} \\ &= \llbracket \text{schlafen}' \rrbracket^{M,g[x \rightarrow d]}(\llbracket x \rrbracket^{M,g[x \rightarrow d]}) && \text{(nach D3.5 (3))} \\ &= I(\text{schlafen}')(g[x \rightarrow d](x)) && \text{(nach D3.5 (1))} \\ &= I(\text{schlafen}')(d) && \text{(Variablenbelegung)} \end{aligned}$$

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 *schlafen'* ist.

Angenommen, es gelte:

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

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

Problem:

Das in 3.1.2 eingeführte Prinzip der λ -Konversion muss präzisiert werden, weil seine uneingeschränkte Anwendung zu falschen Ergebnissen führt.

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.

Definition:

D3.6 Eine Variable v ist in einem Ausdruck α frei für einen Ausdruck β gdw v und β vom selben Typ sind und kein freies Vorkommen von v in α im Skopus eines Quantors $\forall v'$ oder $\exists v'$ oder eines λ -Operators $\lambda v'$ steht und v' frei in β 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 λ -Konversion** wie folgt formuliert:

Wenn v eine Variable und β ein Ausdruck vom selben Typ sind und v in α frei für β ist, dann gilt: $\lambda v[\alpha](\beta) = \alpha[\beta/v]$, wobei $\alpha[\beta/v]$ aus einer Substitution von β für alle freien Vorkommen von v in α hervorgeht.

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

Übungen

Ü3.1 Konstruieren Sie aus den folgenden Ausdrücken λ -Terme vom Typ $\langle e, \langle e, t \rangle \rangle$, so dass der Abfolge Objekt $<$ Subjekt entsprochen wird:

- (a) *umarmen'*
- (b) *vorstellen'(Paula')*
- (c) $\exists z[\text{erzählen}'(z)(y)(x)]$

Ü3.2 Repräsentieren Sie die Bedeutungen der folgenden natürlichsprachlichen Ausdrücke mit passenden λ -Termen:

- (a) *nicht rauchen*
- (b) *sich loben*
- (c) *sich schämen*
- (d) *rennen und sich loben*
- (e) *jedes Buch lesen*

Ü3.3 Vereinfachen Sie die folgenden Ausdrücke mit Hilfe von λ -Reduktion so weit wie möglich:

- (a) $\lambda x[\text{träumen}'(x)](\text{Peter}')$
- (b) $\lambda y \lambda x[\text{verfolgen}'(y)(x)](\text{Tweety}')(\text{Sylvester}')$
- (c) $\lambda P \exists x[\text{Einhorn}'(x) \wedge P(x)](\lambda y[\text{schnaufen}'(y)])$

Ü3.4 Warum ist in den folgenden Fällen keine λ -Reduktion möglich?

- (a) $\lambda x \forall y[\text{lieben}'(y)(x)](y)$
- (b) $\lambda y \lambda x[\text{waschen}'(y)(x)](x)(\text{Anna}')$

Ü3.5 Von welchem Typ sind die folgenden Ausdrücke?

- (a) $\lambda Q[Q(\text{Ulf}')]$
- (b) $\lambda p \lambda x[\text{wissen}'(p)(x)]$
- (c) $\lambda x[\lambda y \lambda x[\text{bewundern}'(y)(x)](\text{Fritz}')](x) \vee Q(x)$
- (d) $\lambda y \lambda Q \lambda x[\text{streicheln}'(y)(x) \wedge Q(y)](\text{Peter}')(\lambda z[\text{lachen}'(z)])(\text{Rita}')$
- (e) $\lambda \mathcal{R}[\mathcal{R}(\lambda z[\text{Obst}'(z)])]$

Für die vorkommenden Variablen gelte dabei:

$$x, y, z \in \text{VAR}_e, \quad p \in \text{VAR}_t, \quad Q \in \text{VAR}_{\langle e, t \rangle}, \quad \mathcal{R} \in \text{VAR}_{\langle \langle e, t \rangle, t \rangle}.$$