5. Typen und indirekte Deutung

In diesem Kapitel werden wir eine Methode kennen lernen, die es ermöglicht, Inhalte, Extensionen und Intensionen systematisch darzustellen, und uns damit in den folgenden Kapiteln die Beschreibung komplexerer semantischer Operationen erleichtern wird: die *typenbasierte indirekte Deutung*. 'Indirekt' heißt dabei, dass die zu interpretierenden sprachlichen Ausdrücke und Konstruktionen systematisch in eine Kunstsprache, die sog. (zweisortige funktionale) *Typenlogik*, übersetzt werden. Die Typenlogik ist eine aus der mathematischen Logik stammende Formelsprache, die sich in der Semantik als eine Art Hilfssprache etabliert hat. Wir werden sie hier in erster Linie für die Darstellung von Extensionen und Intensionen verwenden; am Schluss des Kapitels werden wir sehen, wie man sie prinzipiell auch in der inhaltsbasierten Semantik einsetzen kann.

5.1 Typen

Bevor wir zu den Formeln der Typenlogik kommen, führen wir eine Klassifikation der Extensionen sprachlicher Ausdrücke ein, auf die diese Formeln Bezug nehmen: die (*zweisortigen funktionalen*) *Typen*. Im Wesentlichen geht es darum, Extensionen danach zu sortieren, ob sie einfache Objekte oder verschachtelte Funktionen sind, und wenn letzteres, welcher Art die Verschachtelung ist. Die folgenden Arten von Extensionen haben wir bisher kennen gelernt:

(1)

Art von Ausdruck	*Beispiel*	*Art von Extension*
Satz	**Fritz schläft**	Wahrheitswert
Eigenname	**Fritz**	Individuum
intransitives Verb / Verbalphrase	**schläft**	Funktion von Individuen in Wahrheitswerte (= Prädikatsextension)
transitives Verb	**küsst**	Funktion von Individuen in Prädikatsextensionen
ditransitives Verb	**übergibt**	Funktion von Individuen in Funktionen von Individuen in Prädikatsextensionen
quantifizierende Nominalphrase	**kein Kind**	Funktion von Prädikatsextensionen in Wahrheitswerte (= Quantorenextension)
Substantiv	**Kind**	Prädikatsextension
Determinator	**kein**	Funktion von Prädikatsextensionen in Quantorenextensionen
Einstellungsverb	**meint**	Funktion von Satzintensionen (= Funktionen von möglichen Situationen in Wahrheitswerte) in Prädikatsextensionen

Mit Ausnahme der ersten beiden Arten – Wahrheitswerte und Individuen – handelt es sich bei den Extensionen immer um Funktionen. Unter einer Funktion *von A*s *in B*s ist dabei eine solche zu verstehen, deren Argumente die *A*s bilden und deren Funktionswerte *B*s sind; dabei sind *A* und *B* auch wieder Arten von Extensionen oder, wie wir ab jetzt sagen werden: *Typen*. Um nun die obigen und weitere, noch einzuführende Typen systematisch zu erfassen, schreiben wir für den Extensionstyp 'Funktionen von *A* nach *B*' ab jetzt kürzer '***(AB)***' und kürzen die *primitiven* (= nicht-funktionalen) Typen *Individuum* und *Wahrheitswert* mit '***e***' bzw. '***t***'

ab; zudem benutzen wir '***s***' als Bezeichnung für die Menge der möglichen Situationen.[85] Die Tabelle (1) lässt sich mit dieser Notation in (2) umschreiben (wobei wir die äußersten Klammern um die Typen weglassen):

(2)

Art von Ausdruck	*Beispiel*	*(semantischer) Typ*
Satz	**Fritz schläft**	***t***
Eigenname	**Fritz**	***e***
intransitives Verb	**schläft**	***et***
transitives Verb	**küsst**	***e(et)***
ditransitives Verb	**übergibt**	***e(e(et))***
quantifizierende Nominalphrase	**kein Kind**	***(et)t***
Substantiv	**Kind**	***et***
Determinator	**kein**	***(et)((et)t)***
Einstellungsverb	**meint**	***(st)(et)***

Die einzigen bisher betrachteten Extensionen, die sich nicht ohne weiteres in dieses Schema einpassen, sind die der koordinierenden Konjunktionen, die wir in Abschnitt 1.7 als Funktionen von Paaren von Wahrheitswerten in Wahrheitswerten analysiert hatten: *Paare* von Extensionen sind in der Typen-Notation nicht vorgesehen. Wir könnten dafür eine eigene Notation einführen. Stattdessen werden wir die Analyse der koordinierenden Kojunktionen so umformulieren, dass sie sich in das Schema (2) einfügen lässt. Abgesehen von einer gewissen Vereinheitlichung wird sich die Methode der Umformulierung im Folgenden als sehr nützlich erweisen. Rekapitulieren wir zunächst die Analyse der Koordination aus Kapitel 1:

(3a) $[\![S \textbf{ und } S']\!]^s = [\![S]\!]^s \cdot [\![S']\!]^s$ [= (32b) aus 1.7]

(b) $[\![S \textbf{ oder } S']\!]^s = [\![S]\!]^s + [\![S']\!]^s \pm [\![S]\!]^s \cdot [\![S']\!]^s$ [= (33b) aus 1.7]

(4) $[\![S\ K\ S']\!]^s = [\![S]\!]^s\ [\![K]\!]^s\ [\![S']\!]^s$ [= (34b) aus 1.7]

Nach dieser Analyse können die Extensionen von **und** und **oder** als Funktionen aufgefasst werden, die jeweils zwei Wahrheitswerte *auf einmal* als Argumente nehmen und ihnen als Wert wieder einen Wahrheitswert zuweisen. Diese Funktionen lassen sich auch als Tabellen darstellen:

---

85 Es gilt also: $s = LR$. Das '***e***' erinnert an *Entität* (engl. *entity*), was im philosophischen Jargon so viel heißt wie 'Gegenstand'; '***t***' erinnert natürlich an *truth value*, die englische Übersetzung von *Wahrheitswert*; '***s***' kommt nicht von 'Situation', sondenrn von *sense* 'Sinn', und soll an Freges Intensionsbegriff erinnern. ***e*** und ***s*** sind werden auch als *Sorten* bezeichnet; der Begriff stammt ursprünglich aus der Prädikatenlogik und meint 'nicht mit außerlogischen Mitteln zu charakterisierende Arten'. Die Bezeichnungen für die Typen gehen auf Montagues *Universal Grammar* (vgl. Fn. 15) zurück und sind heute in der Semantik allgemein üblich.

(5)

$[\![\textbf{und}]\!]^s$ =

(0,0)	0
(0,1)	0
(1,0)	0
(1,1)	1

(6)

$[\![\textbf{oder}]\!]^s$ =

(0,0)	0
(0,1)	1
(1,0)	1
(1,1)	1

Gegenüber der üblichen Darstellung der Extensionen von **und** und **oder** durch Wahrheitstafeln, wie wir sie in Kapitel 1 kennengelernt haben, beanspruchen die beiden Argumente in (5) und (6) nur eine Spalte. Dieser Unterschied zwischen den Darstellungen ist rein kosmetisch. Die dargestellten Funktionen sind stets dieselben. Sie weisen Paaren von Wahrheitswerten Wahrheitswerte zu. Durch Anwendung eines formalen Kniffs[86] lassen sich aber Funktionen, deren Argumente (ausnahmslos) Paare sind, in Verschachtelungen von Funktionen transformieren, die auf die einzelnen Komponenten – genauer: nacheinander auf die Komponenten einzeln – angewandt werden. Mit dieser Ersetzung der *simultanen* Anwendung durch eine *sukzessive* erspart man sich einen eigenen Typ für Paare von Wahrheitswerten (oder anderen Extensionen). Man muss dafür nur die Extension der Konjunktion so modifizieren, dass sie zunächst das erste und dann das zweite Argument nehmen kann, womit das Ergebnis der ersten Anwendung selbst wieder eine Funktion sein muss – eine Verschachtelung, wie wir sie schon bei der Analyse transitiver Verben kennen gelernt haben. Für die Konjunktionen **und** und **oder** kommt nach dieser Idee das folgende heraus:

(7a)

$[\![\textbf{und}]\!]^s$ =

0	0 0 1 0
1	0 0 1 1

(b)

$[\![\textbf{oder}]\!]^s$ =

0	0 0 1 1
1	0 1 1 1

(7a) und (b) lassen sich wieder auf gewohnte Weise mit λ-Termen abkürzen:

(8) $[\![\textbf{und}]\!]^s = \lambda q.\lambda p.\ p \cdot q$

Und für die Disjunktion ergibt sich entsprechend:

(9) $[\![\textbf{oder}]\!]^s = \lambda q.\lambda p.\ p + q \pm pq$

Für die Extensionen in (8) und (9) lässt sich jetzt zwar ein semantischer Typ im Stil von Tabelle (2) angeben, aber die Kompositionsregel (4) funktioniert natürlich nicht mehr; denn was dort rechts vom Gleichheitszeichen steht ('$[\![S]\!]^s\ [\![K]\!]^s\ [\![S']\!]^s$'), setzt ja voraus, das die Konjunktionsextension simultan auf zwei Argumente angewandt werden kann. Doch lässt sich (4) leicht an die in (8) und (9) gegebenen Extensionen anpassen, wie in einer Übungsaufgabe gezeigt wird.

Fassen wir zusammen. Extensionen lassen sich nach ihrem Typ klassifizieren. Dabei gibt es zwei Arten von Typen: *primitive* und *funktionale* Typen. Die primitiven Typen sind der Typ ***e*** der Individuen, der Typ ***t*** der Wahrheitswerte und der Typ ***s*** der Situationen. Funktionale Typen haben die Gestalt ***(ab)***, wobei *a* und *b* selbst wieder (primitive oder funktionale) Typen sind; die Extensionen eines funktionalen Typs ***(ab)*** sind die Funktionen, deren Argumente Extensionen des Typs *a* und deren Werte Extensionen vom Typ *b* sind.

---

86 Das allgemeine Verfahren wird in der Literatur als *Schönfinkeln* (nach dem russischen Mathematiker Moses Schönfinkel [1889 – ca. 1942]) oder *Currying* (nach dem US-amerikanischen Mathermatiker Haskell Curry [1900 – 1982]) bezeichnet.

5.2 Typenlogische Formeln

Wie die Extensionen teilen sich auch die Formeln der Typenlogik in Typen ein, die jeweils angeben, was für eine Art von Objekt sie bezeichnen: Formeln des Typs ***e*** bezeichnen Individuen, Formeln des Typs ***(et)*** stehen für Prädikatsextensionen, usw. Dementsprechend werden bei der indirekten Deutung Eigennamen in Formeln des Typs ***e*** übersetzt, intransitive Verben in Formeln des Typs ***(et)***; etc. Die Übersetzungen komplexer Ausdrücke setzen sich dabei aus den Übersetzungen ihrer unmittelbaren Teile zusammen – was garantiert, dass das Kompositionalitätsprinzip im Rahmen der indirekten Deutung quasi automatisch erfüllt wird. Ein Satz wie (10) wird zum Beispiel übersetzt, indem man die Übersetzung des Subjekts **Fritz** mit der des Prädikats **schläft** auf geeignete Weise kombiniert:

(10) **Fritz schläft.**

Beide Teile sind lexikalische Ausdrücke, die sich syntaktisch nicht weiter zerlegen lassen. Ihre Übersetzungen werden – ganz wie die entsprechenden Bedeutungen – durch *lexikalische Gleichungen* angegeben. Für Subjekt und Prädikat von (10) hatten wir die folgenden Extensionen angesetzt:

(11a) $[\![\textbf{Fritz}]\!]^s$ = Fritz [vgl. (31) in 2.6]

(b) $[\![\textbf{schläft}]\!]^s = \lambda x.\ \vdash x$ schläft in $s\dashv$ [vgl. (35) in 2.6]

Die Übersetzungen der lexikalischen Ausdrücke sind typenlogische Formeln, die wesentliche logisch-strukturelle Merkmale der entsprechenden lexikalischen Gleichungen widerspiegeln. In der Gleichung (11a) wird die Extension des Eigennamens durch Benennung eines Objekts angegeben; diese Benennung ist ein (homophoner) Name der Metasprache und besitzt insofern keine interne Struktur.[87] Die typenlogische Übersetzung des Namens **Fritz** wird dementsprechend eine unzusammengesetzte Formel sein, eine sog. *Konstante*, die wir durch einen fett gedruckten Buchstaben bezeichnen, der an das zu interpretierende Wort erinnert: **f**. Wie alle typenlogischen Formel gehört die Konstante **f** einem Typ an: da sie für ein Individuum steht, ist sie eine Formel des Typs ***e***. Bei der indirekten Deutung geht man davon aus, dass die Sprache der Typenlogik für jeden Typ genügend Konstanten enthält, um die Extensionen der lexikalischen Ausdrücke zu benennen. Wie viele es sind und wie sie genau aussehen, kann dabei weitgehend offen bleiben – solange sich zwei Konstanten verschiedenen Typs immer von einander unterscheiden. Eine Konstante ist also immer nur von *einem* Typ; das wird für alle typenlogischen Formeln gelten.

Im Prinzip könnten wir auch für die typenlogische Übersetzung des Prädikats **schläft** eine Konstante bereitstellen, die dann natürlich vom Typ ***(et)*** sein muss.[88] Stattdessen werden wir aber die in (11b) explizite Abhängigkeit der Extension von der betrachteten Situation *s* auch in der Übersetzung zum Ausdruck bringen. Dazu setzen wir zunächst eine der *Intension* von **schläft** entsprechende Konstante **S** an; da diese Intension Situationen Prädikatsextensionen zuweist, ist **S** vom Typ ***(s(et))***. Außerdem brauchen wir dann eine dem '*s*' in (11b) entsprechende Bezeichnung für die jeweilige Situation. Wir verwenden dafür das Symbol ***i***, das nun allerdings keine Konstante, sondern eine *Variable* ist – und zwar eine des Typs ***s*** – klar, denn sie steht ja für eine Situation. Auf die Unterschiede zwischen Konstanten und Variablen kommen wir gleich noch zu sprechen. Dass ***i*** keine Konstante ist, hat unter anderem den Grund, dass sich die Formel nicht auf ein festes Objekt bezieht.

---

87 *homophon* ist ein vornehmer Ausdruck für 'gleichlautend'. Dass der metasprachliche Name und der objektsprachliche gleich ausgesprochen (und geschrieben) werden, liegt daran, dass wir das Deutsche analysieren und uns dabei des Deutschen bedienen. Dieser Umstand ist aus theoretischer Sicht irrelevant, kann aber in der Praxis gelegentlich zu Verwirrungen führen.

88 Das entspricht der Vorgehensweise von Montague, führt aber zu unangenehmen logischen Komplikationen. Auf die Montaguesche Methode kommen wir in Abschnitt 5.7 zu sprechen.

Die typenlogische Übersetzung des Prädikats **schläft** kombiniert nun die Konstante **S** und die Variable ***i*** so, dass insgesamt die Extension von **schläft** (in der betrachteten, durch ***i*** bezeichneten Situation) bezeichnet wird. Da sich diese Extension ergibt, indem man die von **S** bezeichnete Intension auf die von ***i*** bezeichnete Situation *anwendet*, muss die Gesamtformel die beiden Bestandteile so kombinieren, dass insgesamt dieser Funktionswert bezeichnet wird. Das geschieht mittels der folgenden Konstruktionsregel:

(App) Wenn **α** eine (typenlogische) Formel eines Typs ***(ab)*** ist und **β** eine Formel des Typs *a*, dann ist **α(β)** eine Formel des Typs *b*.

α(β) ist dabei die Formel, die sich ergibt, wenn man die Formel **β** mit (fett gedruckten) runden Klammern umgibt und vor das Ganze die Formel **α** schreibt; *a* und *b* sind beliebige Typen, die komplex sein können, aber nicht müssen; und **α** und **β** sind beliebige Ausdrücke, die ihrerseits wieder komplex sein können, aber nicht müssen. Insbesondere lässt sich *(App)* also auf die Konstanten **S** [für **α**] und die Variable ***i*** [für **β**] anwenden, die ja Ausdrücke des Typs ***s(et)*** [= ***(ab)***] bzw. ***s*** [= *a*] sind. Die Regel besagt für diesen Fall, dass die folgende Formel vom Typ ***t*** ist:

(12) **S(*i*)**

Die Bezeichnung '*(App)*' soll natürlich an [*Funktional-*] *Applikation* erinnern, denn in diesem Sinn sollen die nach dieser Regel kombinierten Formeln ja zu verstehen sein. Streng genommen betrifft *(App)* allerdings nur die *Bildung* typenlogischer Formeln und weder ihre Interpretation noch ihren Einsatz in der indirekten Deutung. Es ist zwar klar, dass eine Formel wie (12) für die Extension von **schläft** steht – und Formeln der Gestalt **α(β)** im allgemeinen für den Wert der durch **α** bezeichneten Funktion für das durch **β** bezeichnete Argument; aber das folgt nicht aus *(App)*, sondern erst aus der im folgenden Abschnitt anzugebenden *Deutung* der typenlogischen Formeln (die auch erst offziell festlegt, wofür die Konstanten **f** und **S** stehen). Ebenso sollte es zwar nicht überraschen, dass bei der typenlogischen Übersetzung (10*) von Sätzen wie (10) die Übersetzungen der Bestandteile per *(App)* kombiniert werden:

(10*) **S(*i*)(f)**

Aber auch das folgt nicht aus *(App)*, sondern ergibt sich erst aus der im Rahmen der indirekten Deutung in Abschnitt 5.5 gegebenen *Übersetzung* der natürlichen Sprache in die Typenlogik.

Die mit *(App)* aus Konstanten gebildeten Formeln reichen für die indirekte Deutung vieler Konstruktionen aus – wie z.B. der Objekt-Anbindung in (13):

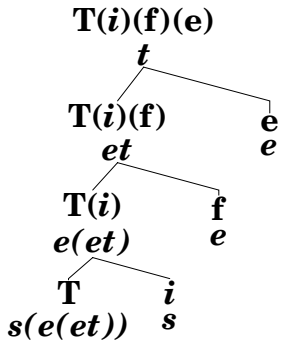
(13) **trifft Fritz**

Wenn eine lexikalische Regel sichert, dass **trifft** in Analogie zu (12) durch eine Formel der gestalt **T(*i*)** übersetzt wird (wobei **T** diesmal eine Konstante des Typs ***s(e(et))*** ist), dann lassen sich die Übersetzungen von Verb und Objekt in (13) wieder mit *(App)* kombinieren. Der gesamte Satz (14) lässt sich dementsprechend mit (15) in die Typenlogik übersetzen (wobei wir unterstellen, dass **e** die Übersetzung des Namens **Eike** ist):

(14) **Eike trifft Fritz.**
(15) **T(*i*)(f)(e)**

Der unterstrichene Teil in (15) ist die Übersetzung des Prädikats, die durch Anwendung von *(App)* auf **T(*i*)** [für **α**] und **f** [für **β**] entsteht; die Formel (15) erhält man, wenn man die unterstrichene Prädikatsübersetzung wie schon in (12) per *(App)* mit der Übersetzung des Subjekts kombiniert. Der Aufbau der Formel (15) lässt sich mit einem Baumdiagramm veranschaulichen:

(15')



Die untere Zeile gibt jeweils den Typ der Formel in der oberen Zeile an. Der Baum verdeutlicht, dass *(App)* in dem Sinne eine *Kürzungsregel* ist, als der Typ einer nach dieser Regel gebildeten Formel immer kürzer ist als der der linken Teilformel.[89]

Als Mittel zum Ausdruck von Extensionskombinationen stößt die Regel *(App)* genau dort auf ihre Grenzen, wo auch schon die direkte Deutung mit Funktionalapplikation nicht weiter kam. In Abschnitt 3.4 haben wir gesehen, dass dies bei der Anbindung *quantifizierender Objekte* der Fall war. Nach der dafür einschlägigen Extensionskombination ermittelt sich die Bedeutung des Prädikats von (16) wie in (17):

(16) **Eike trifft niemanden.**

(17) $[\![\textbf{trifft niemanden}]\!]^s = \lambda x.\ [\![\textbf{niemand}]\!]^s\ (\lambda y.\ [\![\textbf{trifft}]\!]^s(y)(x))$

Um diese Kombination durch eine typenlogische Formel auszudrücken, benötigt man neben der Möglichkeit *(App)*, Funktionswerte zu benennen, offenbar auch noch eine Möglichkeit, Funktionen per (Lambda-) Abstraktion zu definieren. Genau diese bietet die zweite Konstruktionsregel:[90]

(Abs) Wenn $\boldsymbol{x}$ eine Variable eines Typs a ist und $\boldsymbol{\alpha}$ eine Formel des Typs b, dann ist $\boldsymbol{(\lambda x.\ \alpha)}$ eine Formel des Typs $\boldsymbol{(ab)}$.

Die Formel $\boldsymbol{(\lambda x.\ \alpha)}$ entsteht, indem man die Variable $\boldsymbol{x}$ von einem (fett gedruckten) $\boldsymbol{\lambda}$ und einem (fett gedruckten) Punkt umschließt, darauf die Formel $\boldsymbol{\alpha}$ folgen lässt und das Ganze (fett) einklammert; a und b sind wieder beliebige Typen; $\boldsymbol{\alpha}$ ist ein beliebiger Ausdruck, der komplex sein kann, aber nicht muss. Die Regel *(Abs)* setzt voraus, dass es in der typenlogischen Sprache Variablen gibt, die wie die Konstanten Grundausdrücke sind. Auch für Variablen gilt, dass sie jeweils Ausdrücke eines – und nur eines – Typen sind. Aber im Unterschied zu den Konstanten nehmen wir es im Fall der Variablen mit ihrer Anzahl genauer und setzen voraus, dass es *für jeden Typ unendlich viele Variablen* dieses Typs gibt. Diese Voraussetzung wird bei der indirekten Deutung nützlich sein.[91]

---

89 Die Analogie zur arithmetischen Kürzung könnte man noch weiter treiben, wenn man Typen $\boldsymbol{(ab)}$ als $\frac{b}{a}$ notieren würde. *(FA)* kombiniert dann $\frac{b}{a}$ und a zu b – analog zur Multiplikation $\frac{b}{a} \cdot a = b$.

90 Um sie von den in der Metasprache benutzten Variablen zu unterscheiden, beziehen wir uns auf Variablen der Typenlogik immer mit fett und kursiv gesetzten lateinischen Buchstaben. Man beachte, dass diese Buchstaben nicht selbst die Variablen sind, sondern metasprachliche Variablen für sie: in *(Abs)* steht ja '$\boldsymbol{x}$' nicht für eine bestimmte typenlogische Variable, sondern für *beliebige* Variablen eines ebenso beliebigen Typs a. Wie die Variablen 'wirklich' aussehen, bleibt dabei offen. Im Bezug auf typenlogische Variablen verfolgen wir die übliche, wenn auch nicht immer explizit gemachte Konvention, dass – solange nichts Gegenteiliges gesagt wird – innerhalb einer Formel (wie $\mathbf{T(\boldsymbol{i})(\boldsymbol{x})(\boldsymbol{y})}$) verschiedene Variablennamen ($\boldsymbol{x}$ vs. $\boldsymbol{y}$) für verschiedene Variablen (wie immer sie aussehen) stehen; das schließt nicht aus, dass sich verschiedene Variablen auf ein und dasselbe Objekt (z.B. Fritz) beziehen.

Nach *(Abs)* und *(App)* sind die Ausdrücke (18) – (20) typenlogische Formeln:

(18) $(\lambda P^{et}.\ P(\mathbf{f}))$

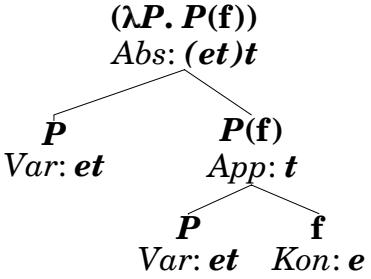
(19) $(\lambda p^{t}.\ (\lambda q^{t}.\ (\lambda R^{t(tt)}.\ R(p)(q))))$

(20) $(\lambda x^{e}.\ \mathbf{T}(i)(\mathbf{f})(x))$

In (18) – (20) haben wir uns an die Konvention gehalten, dass eine Variable ihren Typ als (oberen) Index mit sich führt, wenn sie das erste Mal in der Formel erscheint. Das werden wir auch im Folgenden in der Regel so halten; nur bei der Variablen ***i***, die immer für die gerade betrachtete Situation steht, machen wir eine Ausnahme: wir schreiben sie stets ohne Typenindex .

Wenn die Typen der in ihr vorkommenden Konstanten und Variablen bekannt sind, liegt der Typ einer nach *(Abs)* und *(App)* konstruierten typenlogischen Formel eindeutig fest. Die Formel (18) ist z.B. vom Typ ***(et)t***; denn nach *(App)* ist ***P*(f)** vom Typ ***t***. Die Struktur der Formel lässt sich wieder in Baumform darstellen:

(18')



Der Baum gibt jetzt auch noch an, nach welcher Konstruktionsregel die einzelnen Teilformeln zusammengesetzt sind; diese Information geht jeweils dem Typ voran. Die zweite Zeile des obersten Knotens besagt also: nach *(Abs)* ist die Formel in der darüber stehenden Zeile vom Typ ***(et)t***. Bei einfachen, nicht zusammengesetzten Formeln wird anstelle der Konstruktionsregel angegeben, um welche Art von Formel – Variable oder Konstante – es sich handelt. Nach dem Vorbild von (18') kann nun jede(r) die Typen der Formeln (19) und (20) in einer Übungsaufgabe selbst bestimmen.

(App) und *(Abs)* sind die einzigen Konstruktionsregeln der Typenlogik: alle Formeln werden aus Konstanten und Variablen mit Hilfe dieser beiden Operationen zusammen gesetzt. Damit ergibt sich die folgende:

(21) *Syntax der (zweisortigen funktionalen) Typenlogik*
Für alle Typen *a* und *b* gilt:

(Var) Variablen des Typs *a* sind Formeln des Typs *a*.

(Kon) Konstanten des Typs *a* sind Formeln des Typs *a*.

(App) Wenn **α** eine Formel eines Typs *(ab)* ist und **β** eine Formel des Typs *a*, dann ist **α(β)** eine Formel des Typs *b*.

(Abs) Wenn ***x*** eine Variable eines Typs *a* ist und **α** eine Formel des Typs *b*, dann ist **(λ*x*. α)** eine Formel des Typs *(ab)*.

(21) ist so zu verstehen, dass etwas nur dann als typenlogische Formel eines Typs *a* gilt, wenn es nach einer dieser Regeln hergeleitet wurde. Mit (21) ist die Definition der typenlogischen Formeln eigentlich abgeschlossen. Doch bevor wir zu ihrer Deutung kommen – denn (21) gibt ja

91 Anmerkung für HobbymathematikerInnen: 'Unendlich viele' ist keine genaue Angabe. Vielmehr unterscheidet man in der Mathematik verschiedene Größenordnungen der Unendlichkeit. Für Variablen nimmt man gemeinhin an, dass es (pro Typ und insgesamt) *abzählbar* viele von ihnen gibt – was innerhalb des Unendlichen relativ wenig ist. (Zum Vergleich: die natürliche Zahlen sind abzählbar, die reellen Zahlen nicht.)

nur an, wie die Formeln aufgebaut sind, nicht wofür sie stehen – führen wir noch vier sog. *logische* Konstanten ein, die eine zentrale Rolle in der Praxis der indirekten Deutung spielen werden, weil sie bestimmten *Rechengesetzen* unterliegen, die eine quasi mechanische Reduktion langer und unübersichtlicher Formeln gestatten werden.

Die erste logische Konstante ist ein spezielles Symbol für die Satz-Konjunktion (im semantischen Sinn), die sich, wie wir im vorangehenden Abschnitt gesehen haben, durch eine Funktion des Typs ***t(tt)*** 'simulieren' lässt. In der Typenlogik wird diese Funktion durch eine Konstante des Typs ***t(tt)*** bezeichnet, für die wir das aus der Aussagenlogik stammenden Symbol **∧** verwenden.[92] In Analogie zur natürlichen Sprache setzt man dieses Symbol zwischen die konjugierten Formeln; wenn also **φ** und **ψ** Formeln des Typs ***t*** sind, schreiben wir **[φ ∧ ψ]** statt **∧(ψ) (φ)** und lesen die Formel '**φ** und **ψ**'.

Die Konstante **∧** wird nicht nur für die Deutung des Wortes **und** benötigt, sie spielt überhaupt eine wichtige Rolle in der indirekten Deutung und in der Typenlogik im Allgemeinen. So bildet sie den Kern der Semantik der im nächsten Kapitel betrachteten Modifikations-Konstruktionen (Relativsätze, Adjektive) und kommt bereits in diesem Kapitel in mehreren lexikalischen Gleichungen zum Einsatz – wie z.B. bei der Übersetzung des indefiniten Artikels, den wir im vorangehenden Kapitel wie folgt analysiert hatten:

(22) $[\![\textbf{ein-}_{indef}]\!]^s = \lambda Q.\ \lambda P.\ \vdash \downarrow P \cap \downarrow Q \neq \varnothing \dashv$ [= (60_i) aus 3.2]

Die Konjunktion **∧** wird uns – gemeinsam mit einer weiteren Konstanten – helfen, die rechts vom Gleichheitszeichen in (22) angegebene Extension typenlogisch darzustellen. Um zu sehen, wie das geht, formulieren wir die Gleichung zunächst etwas um:

(22') $[\![\textbf{ein-}_{indef}]\!]^s = \lambda Q.\ \lambda P.\ \vdash \downarrow[\lambda x.P(x) \cdot Q(x)] \neq \varnothing \dashv$

Wo in (22) vom Schnitt der von P und Q charakterisierten Mengen die Rede ist, wird in (22') die durch den Lambda-Term $\lambda x.P(x) \cdot Q(x)$ charakterisierte Menge betrachtet. Dabei handelt es sich aber um dieselbe Menge – denn:

(23)
$\downarrow[\lambda x.P(x) \cdot Q(x)]$
$= \downarrow[\lambda x.\ \vdash P(x) = 1 \text{ und } Q(x) = 1 \dashv]$ weil $P(x) \cdot Q(x) = 1$ gdw. $P(x) = 1$ & $Q(x) = 1$
$= \{x \mid P(x) = 1 \text{ und } Q(x) = 1\}$ mit (38) aus 2.6
$= \{x \mid x \in \downarrow P \text{ und } x \in \downarrow Q\}$ Def. '↓'
$= \downarrow P \cap \downarrow Q$ Def. '∩'

Um die rechte Seite von (22') in der Typenlogik auszudrücken, verwendet man eine Konstante des Typs ***(et)t***, die zum Ausdruck bringt, dass eine (als Argument) gegebene Prädikatsextension eine nicht-leere Menge charakterisiert. Aus der Prädikatenlogik übernehmen wir für diese Konstante das Symbol **∃** und bezeichnen sie als *Existenzquantor*.[93] In Anlehnung an die dort übliche Notation lassen wir das Lambda-Präfix weg und klammern um, wenn **∃** mit einem (typenlogischen) Lambda-Term kombiniert wird: wenn **φ** eine Formel des Typs ***t*** ist, schreiben wir **(∃*x*) φ** statt **∃(λ*x*. φ)** und lesen 'es gibt ein x, so dass **φ** gilt'. – Obwohl wir die indirekte Deutung der Determinatoren ausführlich und im Zusammenhang einführen werden, nachdem

92 Die Aussagenlogik, deren Anfänge in die Antike zurückreichen, ist der Bereich der formalen Logik, der die Zusammenhänge zwischen der Konjunktion und der sogleich einzuführenden Negation betrifft.

93 Die Prädikatenlogik (die auf Freges *Begriffsschrift* aus dem Jahre 1879 zurückgeht) ist der Kernbereich der formalen Logik; sie umfasst die in der vorangehenden Fußnote genannte Aussagenlogik, ist aber weniger ausdrucksstark und flexibel als die Typenlogik.

wir die Deutung der typenlogischen Formeln kennen gelernt haben, geben wir hier schon einmal die Übersetzung des indefiniten Artikels an:

(24) $(\lambda Q^{et}.\ (\lambda P^{et}.\ (\exists x)\ [Q(x) \wedge P(x)]))$

Expandiert man (24) im Sinne der Abkürzungskonventionen für Konjunktion und Existenzquantor, kann man leicht zeigen (Übungsaufgabe), dass es sich dabei um eine Formel des Typs ***(et)((et)t)*** handelt.

Die dritte logische Konstante ist die aussagenlogische *Negation*, die mit ¬ symbolisiert wird. Die Negation ist vom Typ ***tt*** und dient dazu, einen Wahrheitswert umzukehren: wenn eine Formel φ des Typs ***t*** den Wahrheitswert 1 hat, hat ¬ (φ) den Wahrheitswert – und umgekehrt. Wie in der Aussagenlogik lassen wir die Klammern um das Argument von ¬ für gewöhnlich weg und schreiben statt ¬ (φ) einfach ¬ φ, was als 'es ist nicht der Fall, dass φ' gelesen werden kann. Mit dieser Konvention sieht die typenlogische Übersetzung des Determinators **kein-** folgendermaßen aus:

(25) $(\lambda Q^{et}.\ (\lambda P^{et}.\ \neg\ (\exists x)\ [Q(x) \wedge P(x)]))$

Die vierte und letzte logische Konstante werden wir hier nicht eigens motivieren; ihre Interpretation bereitet keinerlei Schwierigkeiten, und wir werden später ihren Nutzen für die indirekte Deutung kennen lernen. Es handelt sich um die *Identität* zwischen Individuen, einer Konstanten vom Typ ***e(et)***, die wir mit einem fetten Gleichheitszeichen notieren, das wir für gewöhnlich zwischen die Terme schreiben und einklammern: **(α = β)** steht also für **=(α)(β)**, wenn **α** und **β** vom Typ ***e*** sind.

Die Konstruktionsregeln (21) und die logischen Konstanten ∧, ∃, ¬ und = (samt ihren Typen und den Notationskonventionen) machen die syntaktische Seite der Typenlogik aus, d.h. die Definition der typenlogischen Formeln. Jetzt geht es an ihre Interpretation.

5.3 Die Deutung der Typenlogik

Wir haben zwar schon gesehen, wie die typenlogischen Formeln zu verstehen sind, doch eine *systematische* Deutung steht noch aus. Ganz wie bei der (direkten) Deutung der natürlichen Sprache werden wir dabei jeder Formel **α** einen semantischen Wert $[\![\alpha]\!]$ zuordnen. Bei den einfachen Formeln – Konstanten und Variablen – geschieht dies direkt, d.h. in Form von Gleichungen '$[\![\alpha]\!]$ = …', die diesen Wert jeweils spezifizieren; der Wert komplexer Formeln ergibt sich dagegen systematisch aus den Werten ihrer (unmittelbaren) Teile und der Art der Zusammensetzung: Applikation oder Abstraktion.

Es gibt zwei Arten von unzusammengesetzten Formeln in der Typenlogik: Variablen und Konstanten. Die Deutung der Konstanten ist, wie wir gleich sehen werden, eine verhältnismäßig einfache Angelegenheit. Die Variablen dagegen bereiten ernsthafte Schwierigkeiten für eine kompositionelle Deutung. Grob gesprochen ist das Problem, dass eine Variable für sich genommen gar keine feste Extension hat. Aber im Zusammenhang einer komplexen Formel kann eine Variable durchaus zu deren Extension beitragen. So bezeichnet eine Formel der Gestalt **λ*x*.** … eine Funktion, die *beliebigen* Objekten eines bestimmten Typs irgendwelche Werte zuordnet – aber was sind dabei *beliebige* Objekte? Wie man in (25) sieht, kann ¬ **(∃*x*) [*Q*(*x*) ∧ *P*(*x*)]** zum Ausdruck bringen, dass die von ***P*** und ***Q*** charakterisierten Mengen disjunkt sind, aber worauf bezieht sich dabei das ***x***? Auf beliebige Objekte im Schnitt der genannten Mengen – also auf nichts (wenn sie wirklich disjunkt sind)? Worauf immer sich das ***x*** beziehen mag: für die mit der Formel gemachte (Disjunktheits-) Aussage ist sie letztlich entbehrlich; denn in der unterstrichenen Formulierung kommt man ohne Bezugnahme auf diese 'beliebigen' Objekte aus.

Die Probleme mit der Funktion von Variablen und ihrer kompositionellen Deutung weisen weit über die Typenlogik hinaus und sind ohne einen gewissen Aufwand nicht lösbar. Die Lösung, der wir uns hier anschließen werden, basiert auf der Idee, dass eine Variable simultan für alle Objekte ihres Typs steht.[94] Eine Variable $\boldsymbol{x}$ des Typs *e* kann sich danach auf jedes Individuum beziehen, d.h. jedes Individuum kann Extension von $\boldsymbol{x}$ sein; ebenso kann jede Prädikatsextension – also jede charakteristische Funktion einer Menge von Individuen – Extension einer Variablen $\boldsymbol{P}$ des Typs ***et*** sein. Variablen haben danach (in einer gegebenen Situation) mehr als eine Extension. Dasselbe gilt für komplexe Formeln, in denen Variablen vorkommen:

(26) $\boldsymbol{P(x)}$

Die Formel (26) kann sich auf jeden Wahrheitswert beziehen, der sich ergibt, wenn man den – beliebigen – semantischen Wert von $\boldsymbol{P}$ auf den – beliebigen – semantischen Wert von $\boldsymbol{x}$ anwendet.[95] Wenn also (Fall 1) $\boldsymbol{P}$ für die Prädikatsextension P_1 steht und $\boldsymbol{x}$ für das Individuum x_1, dann steht (26) für $P_1(x_1)$; stehen dagegen (Fall 2) $\boldsymbol{P}$ und $\boldsymbol{x}$ für P_2 bzw. x_2, ist $P_2(x_2)$ der semantische Wert von (26); aber $\boldsymbol{P}$ kann ebensogut (Fall 3) für P_1 stehen, während $\boldsymbol{x}$ für x_2 steht, womit der semantische Wert von (26) natürlich $P_1(x_2)$ wäre; etc. pp. Die Fälle, von denen es abhängt, wofür die einzelnen Variablen stehen können, bezeichnet man in der logischen Semantik als *Variablenbelegungen* oder einfach *Belegungen* (engl. [*variable*] *assignments*). Da eine Belegung für jede Variable jedes Typs sagt, worauf sie sich bezieht, kann man sie als eine Funktion verstehen, die Variablen ihre semantischen Werte zuordnet:

(27) *Definition*
Eine *Belegung* ist eine Funktion *g*, deren Argumente die Variablen sind und für die gilt: wenn $\boldsymbol{x}$ eine Variable eines Typs *a* ist, dann ist $g(\boldsymbol{x})$ ein Objekt des Typs *a*.

Die Fälle 1 – 3 entsprechen demnach jeweils ganz vielen Belegungen: Fall 1 liegt z. B. mit jeder Belegung *g* vor, nach der $\boldsymbol{P}$ und $\boldsymbol{x}$ die semantischen Werte P_1 und x_1 haben, d.h. sobald $g(\boldsymbol{P}) = P_1$ und $g(\boldsymbol{x}) = x_1$, egal welche Werte *g* den anderen Variablen zuweist.

Wie die Betrachtung zu (26) gezeigt hat, genügt es nicht, nur für Variablen anzunehmen, dass sie mehrere semantische Werte haben; auch der Wert eines komplexen Ausdrucks kann von einer Belegung abhängen. In der Semantik der Typenlogik muss man daher jedem Ausdruck $\boldsymbol{\alpha}$ nicht nur *einen* semantischen Wert zuordnen, sondern einen Wert $[\![\boldsymbol{\alpha}]\!]^g$ für jede Belegung *g*. Liegt etwa Fall 2 vor – d.h.: haben wir es mit einer Belegung *h* zu tun, bei der $h(\boldsymbol{P}) = P_2$ und $h(\boldsymbol{x}) = x_2$ –, dann ermittelt sich der Wert von (26) so: $[\![\boldsymbol{P(x)}]\!]^h = P_2(x_2)$. Allgemein gesprochen müssen wir also bei der Bestimmung der Bedeutung typenlogischer Formeln für jede Belegung *g* angeben, was der semantische Wert *bei der Belegung g* ist. Anders als bei der direkten Deutung der natürlichen Sprache in den voran gehenden Kapiteln werden wir also für die Typenlogik eine ganze 'Familie' semantischer Werte kompositionell definieren; und diese Familie von Werten macht dann die eigentliche Bedeutung der Formel aus. Dennoch werden in der Praxis nur die einzelnen Werte eine Rolle spielen. Warum das so ist, werden wir gleich sehen.

Nach diesen grundsätzlichen Erläuterungen zur Architektur der Deutung typenlogischer Formeln erklären sich die lexikalischen Gleichungen für Variablen von selbst:

---

94 Diese Methode der Deutung von Variablen – die sog. *Belegungssemantik* – geht auf den polnisch-US-amerikanischen Logiker Alfred Tarski (1901–1983) zurück und ist in der logischen Semantik sehr beliebt. Eine äquivalente Alternative, die *Substitutionssemantik*, wird in Abschnitt 5.7 skizziert.

95 Wir setzen bei dieser Betrachtung voraus, dass die Konstruktion *(App)* im Sinne der Funktionalapplikation gedeutet wird – wie wir es gleich festlegen werden.

(28) *Deutung der Variablen (Var)*
Wenn g eine Belegung ist und $\boldsymbol{x}$ eine Variable, dann gilt:
$[\![\boldsymbol{x}]\!]^g = g(\boldsymbol{x})$.

Da die später anzugebenden typenlogischen Übersetzungen natürlichsprachlicher Ausdrücke für ihre Extensionen stehen, enthalten sie – wie wir schon gesehen haben – stets eine Variable, die sich auf eine mögliche Situation bezieht. Dabei handelt es sich stets um dieselbe, feste Variable $\boldsymbol{i}$ des Typs $\boldsymbol{s}$; warum das so ist, wird noch klar werden. Da der Wert dieser Variablen $\boldsymbol{i}$ von der Belegung bestimmt wird, kommt der Variablenbelegung unter Anderem die Funktion zu, die jeweils betrachtete Situation zu spezifizieren.

Kommen wir nun zu den Konstanten. Da wir offen gelassen haben, wie viele und welche Konstanten es genau gibt, müssen wir auch ihre Deutung weitgehend offen lassen. Die meisten Konstanten werden wir im Rahmen der indirekten Deutung bei Bedarf einführen und dann an Ort und Stelle interpretieren. Allerdings gibt es zwei Einschränkungen, denen die lexikalischen Gleichungen genügen müssen. Zum einen hat jede Konstante einen (und nur einen) Typ, der den Typ der Bedeutung dieser Konstanten festlegt; das ist analog zu der in (27) gegebenen Beschränkung für Variablenbelegungen. Zum anderen hängt der semantische Wert einer Konstanten natürlich nicht von den semantischen Werten irgendwelcher Variablen ab; bei zwei verschiedenen Belegungen erhält die Konstante m.a.W. denselben Wert. Wir halten diese Einschränkungen fest als:

(29) *Konstantenprinzipien (Kon)*
(i) Wenn g eine Belegung und $\mathbf{c}$ eine Konstante eines Typs a ist, dann gilt:
$[\![\mathbf{c}]\!]^g$ ist ein Objekt des Typs a.
(ii) Wenn g und h Belegungen sind und $\mathbf{c}$ eine Konstante (irgendeines Typs), dann ist
$[\![\mathbf{c}]\!]^g = [\![\mathbf{c}]\!]^h$.

Lexikalische Gleichungen müssen und werden im Folgenden stets den Prinzipien in (29) genügen. Zur Illustration geben wir ein beliebig gewähltes Beispiel:

(30) Wenn g eine Belegung ist, ist $[\![\mathbf{f}]\!]^g$ = Fritz.

Unter der (syntaktischen) Voraussetzung, dass es sich bei $\mathbf{f}$ um eine Konstante des Typs $\boldsymbol{e}$ handelt, erfüllt (30) die Bedingung (29i); denn Fritz ist ein Individuum. Auch in den später anzugebenden Gleichungen wird (29i) stets erfüllt sein, ohne dass wir eigens darauf hinweisen. (29ii) gilt ebenfalls; denn (30) ist so zu verstehen, dass die Gleichung für alle Belegungen g gilt.

Setzt man (29ii) voraus, ist der Index an den semantischen Klammern in den lexikalischen Gleichungen für Konstanten redundant und wird deswegen in Zukunft weggelassen. (30) kann man nach dieser Konvention umschreiben in:

(30') $[\![\mathbf{f}]\!]$ = Fritz

Abgesehen von diesem illustrativen Beispiel sind die einzigen spezifischen lexikalischen Gleichungen für Konstanten diejenigen für die vier logischen Konstanten, die wie folgt lauten:

(31) $[\![\wedge]\!] = \lambda v.\, \lambda u.\, u \cdot v$
(32) $[\![\exists]\!] = \lambda P.\, !\downarrow P \neq \varnothing!$
(33) $[\![\neg]\!] = \lambda v.\, 1 - v$
(34) $[\![=]\!] = \lambda x.\, \lambda y.\, \vdash x = y \dashv$

In (31) – (34) haben wir von der obigen Konvention Gebrauch gemacht und die Belegungsindizes weggelassen; es sei noch einmal darauf hingewiesen, dass dies deshalb geht, weil es sich um Konstanten handelt, für deren Bewertung die Belegung keine Rolle spielt.

Rechts von den Gleichheitszeichen in (31) – (34) stehen jeweils Lambda-Terme. Diese Terme sind keine typenlogischen Formeln, sondern Abkürzungen für Beschreibungen von Funktionen, wie wir sie auch schon in den voran gehenden Kapiteln verwendet haben. Auch die in (31) – (34) benutzten Variablen sind keine typenlogischen Variablen, sondern Variablen der Metasprache, die für beliebige Wahrheitswerte (‘*u*’, ‘*v*’), Prädikatsextensionen (‘*P*’) bzw. Individuen (‘*x*’, ‘*y*’) stehen. Dass diese Meta-Variablen für Objekte der Typen ***t***, ***et*** bzw. ***e*** stehen, geht aus diesen Gleichungen nicht explizit hervor, folgt aber aus dem Konstantenprinzip (29i). Danach muss z.B. $[\![\wedge]\!]$ eine Funktion des Typs ***t(tt)*** sein und somit Wahrheitswerte als Argumente nehmen; und da die Variable ‘*v*’ nach dem Lambda sich auf das (erste) Argument von $[\![\wedge]\!]$ bezieht, kann sie nur für Wahrheitswerte stehen. Umständlicher, aber eindeutiger hätten wir die Gleichungen so formulieren können:

(31') $[\![\wedge]\!]$ = diejenige Funktion des Typs ***t(tt)***, die jedem Wahrheitswert v diejenige Funktion des Typs ***tt*** zuordnet, deren Wert für jeden Wahrheitswert u das Produkt uv ist

(32') $[\![\exists]\!]$ = diejenige Funktion des Typs ***(et)t***, die jeder Prädikatsextension P genau dann den Wahrheitswert 1 zuordnet, wenn P nicht die leere Menge charakterisiert

(33') $[\![\neg]\!]$ = diejenige Funktion des Typs ***tt***, die jedem Wahrheitswert v den Wahrheitswert $1-v$ zuweist

(34') $[\![=]\!]$ = diejenige Funktion des Typs ***e(et)***, die jedem Individuum x diejenige Funktion des Typs ***et*** zuordnet, deren Wert für ein beliebiges Individuum y genau dann 1 ist, wenn x mit y identisch ist

Wir kommen nun zur Interpretation komplexer Formeln. Wie schon angekündigt, werden diese kompositionell gedeutet, d.h. wir setzen jeweils die Bedeutungen der unmittelbaren Teilformeln voraus und geben an, wie sich diese zur Bedeutung der Gesamtformel kombinieren. Entsprechend den zwei Konstruktionsregeln für typenlogische Formeln müssen wir dabei zwei Fälle unterscheiden. Wir beginnen mit dem deutlich einfacheren Fall, in dem zwei Ausdrücke **α** und **β** zu **α(β)** kombiniert werden. Nach (21*App*) geht das nur, wenn **α** eine Formel eines Typs ab ist und **β** vom Typ a. Wir können dann voraussetzen, dass die semantischen Werte $[\![\boldsymbol{\alpha}]\!]^g$ und $[\![\boldsymbol{\beta}]\!]^g$ immer (also bei jeder Belegung g) Objekte der Typen ab bzw. a sind. $[\![\boldsymbol{\beta}]\!]^g$ ist dann insbesondere ein Argument, dem $[\![\boldsymbol{\alpha}]\!]^g$ einen Wert zuweist – und dieser Wert ist der Wert der Gesamtformel:

(35) *Deutung der Applikation (App)*
Wenn g eine Belegung ist, und **α** und **β** Formeln des Typs ab und a sind, dann gilt:
$[\![\boldsymbol{\alpha}(\boldsymbol{\beta})]\!]^g = [\![\boldsymbol{\alpha}]\!]^g([\![\boldsymbol{\beta}]\!]^g)$.

Die zweite Art komplexer Formeln sind nach (21*Abs*) gebildet und haben die Form **(λ*x*. α)** mit einer Variablen ***x*** eines Typs a und irgendeiner Formel **α** eines Typs b. Um zu sehen, wie sich diese Formeln kompositionell deuten lassen, betrachten wir zunächst ein Beispiel:

(36) **(λ*y*^{e}. (λ*x*^{e}. S(*i*)(*x*)(*y*)))**

Dabei soll **S** eine Konstante des Typs ***s(e(et))*** sein, die die Intension von **sieht** bezeichnet:

(37) $[\![\mathbf{S}]\!] = \lambda s.\ \lambda y.\ \lambda x.\ \vdash x$ sieht y in $s\dashv$

Zunächst machen wir uns klar, dass der Wert der Formel (36) eine Funktion sein sollte, die angewandt auf Eike die Prädikatsextension von **wird von Eike gesehen** (in der von $\boldsymbol{i}$ bezeichneten Situation) liefert – also die Funktion, die jedem Individuum x den Wahrheitswert 1 zuweist, wenn x in $g(\boldsymbol{i})$ von Eike gesehen wird; denn wenn man die durch $\mathbf{S}(\boldsymbol{i})$ bezeichnete Extension von **sieht** auf x anwendet und dann das Ergebnis auf Eike, erhält man nach (37) gerade den Wahrheitswert 1, wenn Eike x sieht. (36) kehrt also sozusagen die durch $\mathbf{S}(\boldsymbol{i})$ bezeichnete Beziehung des Sehens in die des Gesehen-Werdens um. Um (36) kompositionell zu deuten, muss die Bedeutung dieser Formel aus den Bedeutungen ihrer beiden unmittelbaren Bestandteile ermittelt werden – der Variablen $\boldsymbol{y}$ und der komplexen Formel (38):

(38) $(\lambda \boldsymbol{x}^{e}.\ \mathbf{S}(\boldsymbol{i})(\boldsymbol{x})(\boldsymbol{y}))$

Die Bedeutung von (38) muss sich wiederum systematisch aus denen der Variablen $\boldsymbol{x}$ und der Bedeutung der Formel (39) ergeben:

(39) $\mathbf{S}(\boldsymbol{i})(\boldsymbol{x})(\boldsymbol{y})$

Der Wert von (39) lässt sich sukzessiv mit Hilfe der obigen Deutung (35) der Applikation ermitteln. Danach gilt für beliebige Belegungen g:

(40) $[\![\mathbf{S}(\boldsymbol{i})(\boldsymbol{x})(\boldsymbol{y})]\!]^{g}$
$= [\![\mathbf{S}(\boldsymbol{i})(\boldsymbol{x})]\!]^{g}([\![\boldsymbol{y}]\!]^{g})$
$= [\![\mathbf{S}(\boldsymbol{i})]\!]^{g}([\![\boldsymbol{x}]\!]^{g})([\![\boldsymbol{y}]\!]^{g})$
$= [\![\mathbf{S}]\!]^{g}([\![\boldsymbol{i}]\!]^{g})([\![\boldsymbol{x}]\!]^{g})([\![\boldsymbol{y}]\!]^{g})$
$= [\![\mathbf{S}]\!]^{g}(g(\boldsymbol{i}))\ (g(\boldsymbol{x}))\ (g(\boldsymbol{y}))$

Der letzte Übergang in (40) macht von der Variablendeutung (28) Gebrauch, wonach der Wert einer Variablen von der jeweiligen Belegung bestimmt wird. Diese Belegungsabhängigkeit überträgt sich nach (40) auf die komplexe Formel (39). Wenn also g_1 den Variablen $\boldsymbol{x}$, $\boldsymbol{y}$ und $\boldsymbol{i}$ als Werte Fritz, Eike und s_0 zuweist – d.h. $g_1(\boldsymbol{x})$ = Fritz, $g_1(\boldsymbol{y})$ = Eike und $g_1(\boldsymbol{i}) = s_0$ – ist der Wert von (39) bei dieser Belegung 1, falls Eike Fritz in s_0 sieht; bei einer Belegung g_2, nach der $g_2(\boldsymbol{i}) = g_1(\boldsymbol{i})$ und $g_2(\boldsymbol{x})$ = Fritz = $g_2(\boldsymbol{y})$, ist hingegen $[\![(39)]\!]^{g_2} = 1$, wenn Fritz sich in s_0 selbst sieht; usw.:

(41)

Bel.	Wert von $\boldsymbol{i}$	Wert von $\boldsymbol{y}$	Wert von $\boldsymbol{x}$	Wert von $\mathbf{S}(\boldsymbol{i})(\boldsymbol{x})(\boldsymbol{y})$
g_1	s_0	Eike	Fritz	⊢ Eike sieht Fritz in s_0 ⊣
g_2	s_0	Fritz	Fritz	⊢ Fritz sieht sich in s_0 ⊣
g_3	s_0	Eike	Eike	⊢ Eike sieht sich in s_0 ⊣
g_4	s_1	Fritz	Eike	⊢ Fritz sieht Eike in s_1 ⊣
g_5	s_1	Fritz	Fritz	⊢ Fritz sieht sich in s_1 ⊣
…	…	…	…	…
g	$g(\boldsymbol{i})$	$g(\boldsymbol{y})$	$g(\boldsymbol{x})$	⊢ g($\boldsymbol{y}$) sieht $g(\boldsymbol{x})$ in $g(\boldsymbol{i})$⊣
…	…	…	…	…

Beim Übergang von (39) zu (38) wird nun diese Belegungsabhängigkeit durch den Lambda-Operator reduziert. Denn (38) bezieht sich auf eine Funktion, die *beliebigen* Individuen x den Wahrheitswert 1 zuordnet, wenn x das durch die Variable ***y*** bezeichnete Individuum in der durch die Variable ***i*** bezeichneten Situation sieht:

(42)

Bel.	Wert von ***i***	Wert von ***y***	Wert von ***x***	Wert von **(λ*x*. S(*i*)(*x*)(*y*))**
g_1	s_0	Eike	Fritz	λ*x*. ⊢ Eike sieht *x* in s_0 ⊣
g_2	s_0	Fritz	Fritz	λ*x*. ⊢ Fritz sieht *x* in s_0 ⊣
g_3	s_0	Eike	Eike	λ*x*. ⊢ Eike sieht *x* in s_0 ⊣
g_4	s_1	Fritz	Eike	λ*x*. ⊢ Fritz sieht *x* in s_1 ⊣
g_5	s_1	Fritz	Fritz	λ*x*. ⊢ Fritz sieht *x* in s_1 ⊣
…	…	…	…	…
g	*g*(***i***)	*g*(***y***)	*g*(***x***)	λ*x*. ⊢ g(***y***) sieht *x* in *g*(***i***)⊣
…	…	…	…	…

Obwohl also der Wert von (39) davon abhängt, auf wen sich die Variablen ***x***, ***y*** und ***i*** beziehen, kommt es in (38) nur noch auf die Werte von ***y*** und ***i*** an: sobald zwei Belegungen in diesen Variablen-Werten übereinstimmen (was in der Tabelle durch gleiche Färbung angedeutet ist), stimmen sie auch im Wert der Gesamtformel überein. Gegenüber der in (41) dargestellten Deutung von (39) wird in (38) der Wert von ***x*** durch das Lambda also 'neutralisiert'. In der Logik bezeichnet man diese Form der Neutralisierung von Variablenwerten als *Bindung*: die zunächst *freie* Variable im *Skopus*(≈ Bereich) des Lambda-Operators wird bei der Abstraktion von diesem *gebunden*. Der Skopus (manchmal auch als *Matrix* bezeichnet) ist dabei gerade der Teil der Formel, auf den sich der Operator bezieht – zwischen Punkt und Klammer zu. Man beachte, dass der *Status* einer Variablen – ob sie frei oder gebunden ist – etwas Relatives ist; eine Variable ist niemals 'an sich' frei oder gebunden, sondern nur in Bezug auf eine Formel: ***x*** ist z.B. frei in (39), aber nicht in (38).

Die Bindung ist ein sehr allgemeiner Vorgang, der immer dort stattfindet, wo man Variablen scheinbar dazu benutzt, um sich auf beliebige Objekte zu beziehen. Um zu sehen, wie dieser Prozess funktioniert, sehen wir uns den Übergang von (41) zu (42) im Detail an. Dabei konzentrieren wir uns zunächst auf die Belegung g_1 . Der Wert, den die Formel (38) bei dieser Belegung nach (42) erhält, ist eine Funktion, die sich selbst wieder in Tabellenform darstellen lässt:

(42_1) $[\![\textbf{(λ}\boldsymbol{x}\textbf{. S(}\boldsymbol{i}\textbf{)(}\boldsymbol{x}\textbf{)(}\boldsymbol{y}\textbf{))}]\!]^{g_1}$ =

Argument	*Funktionswert*
Fritz	⊢ Eike sieht Fritz in s_0 ⊣
Eike	⊢ Eike sieht sich in s_0 ⊣
…	…
NN	⊢ Eike sieht *NN* in s_0 ⊣

(42_1) gibt den Wert in der ersten Zeile von (42) an. Da es das Ziel der gegenwärtigen Betrachtungen ist, eine allgemeine Bedeutungskombination zu finden, die die Werte von Formeln der

Gestalt $(\lambda \boldsymbol{z}.\ \alpha)$ aus denen der gebundenen Variablen $\boldsymbol{z}$ und der Matrix-Formel α zusammensetzt, gilt es herauszufinden, wie man die in (42_1) dargestellte Funktion aus der in (41) dargestellten Bedeutung der Matrix $\mathbf{S}(\boldsymbol{i})(\boldsymbol{x})(\boldsymbol{y})$ erhalten kann. Doch das ist ganz offensichtlich. Denn (42_1) ergibt sich, wenn man in den beiden rechten Spalten von (41) die Zeilen herausnimmt, die anders gefärbt sind als die erste, der Belegung g_1 entsprechende Zeile – wobei wir die Färbung aus (42) übernehmen.

(41_2)

Bel.	Wert von $\boldsymbol{i}$	Wert von $\boldsymbol{y}$	Wert von $\boldsymbol{x}$	Wert von $\mathbf{S}(\boldsymbol{i})(\boldsymbol{x})(\boldsymbol{y})$
g_1	s_0	Eike	Fritz	⊦Eike sieht Fritz in s_0⊣
~~g_2~~	~~s_0~~	~~Fritz~~	~~Fritz~~	~~⊦Fritz sieht x in s_0⊣~~
g_3	s_0	Eike	Eike	⊦Eike sieht sich in s_0⊣
~~g_4~~	~~s_1~~	~~Fritz~~	~~Eike~~	~~⊦Fritz sieht Eike in s_1⊣~~
~~g_5~~	~~s_1~~	~~Fritz~~	~~Fritz~~	~~⊦Fritz sieht sich in s_1⊣~~
…	…	…	…	…

⊢⊣Dasselbe Vorgehen führt auch von (41) zum Wert von $(\lambda \boldsymbol{x}_e.\ \mathbf{S}(\boldsymbol{x})(\boldsymbol{y}))$ an der Belegung g_3; denn diese Belegung hat dieselbe Farbe – bei Ausschluss aller andersfarbigen Zeilen kommt also wieder (41_2) heraus, und nach (42) ist ja tatsächlich $[\![(\lambda \boldsymbol{x}.\ \mathbf{S}(\boldsymbol{x})(\boldsymbol{y}))]\!]^{g_1} = [\![(\lambda \boldsymbol{x}.\ \mathbf{S}(\boldsymbol{x})(\boldsymbol{y}))]\!]^{g_3}$. Und ganz analog ergibt sich der Wert von $(\lambda \boldsymbol{x}.\ \mathbf{S}(\boldsymbol{x})(\boldsymbol{y}))$ an den Belegungen g_2 und g_5, wenn man in (41) nur die dunklen Belegungen betrachtet und dann die Zuordnung in den rechten beiden Zeilen nimmt. Das Malen entsprechender Tabellen ist Gegenstand einer Übungsaufgabe.

Insgesamt ergibt sich damit das folgende allgemeine Muster: der Wert einer Formel der Gestalt $(\lambda \boldsymbol{z}.\ \alpha)$ an einer Belegung g ist eine Funktion f, die den Werten von $\boldsymbol{z}$ an den zu g gleichfarbigen Belegungen den dortigen Wert von α zuordnet. Wenn also h eine solche gleichfarbige Belegung ist, dann ordnet f einem Objekt $u = h(\boldsymbol{z})$ den Wert von α an h zu:

(43) $[\![(\lambda \boldsymbol{z}.\ \alpha)]\!]^g (h(\boldsymbol{z})) = [\![\alpha]\!]^h$

(43) erfasst bereits im wesentlichen die gesuchte, der Lambda-Abstraktion entsprechende Bedeutungskombination. Allerdings muss man dafür die Gleichung auf *beliebige* einander *gleichfarbige* Belegungen g und h beziehen. Doch worin besteht eigentlich *Gleichfarbigkeit* im allgemeinen? In den obigen Tabellen erhielten zwei Belegungen immer dann die gleiche Schattierung, wenn sie in den Werten für $\boldsymbol{i}$ und $\boldsymbol{y}$ übereinstimmten – oder besser (weil leichter verallgemeinerbar): *wenn sie sich allenfalls im $\boldsymbol{x}$-Wert unterschieden*. Die durch den Lambda-Term bezeichnete Funktion ließ sich gerade deswegen aus den gleichfarbigen Belegungen konstruieren, weil jedes Individuum irgendwann einmal als $\boldsymbol{x}$-Wert auftaucht, ohne dass die anderen Variablenwerte davon berührt werden. In der Logik nennt man zwei Belegungen, die sich – wenn überhaupt – nur im Wert einer einzigen Variablen $\boldsymbol{x}$ unterscheiden *$\boldsymbol{x}$-Alternativen* voneinander. Mit Hilfe dieses Begriffs lässt sich (43) in die folgende allgemeine Bedeutungsregel für Lambda-Formeln umschreiben:

(44) *Deutung der Abstraktion (Abs)* [1. Formulierung]
Wenn g eine Belegung ist, $\boldsymbol{x}$ eine Variable eines Typs a und $\boldsymbol{\alpha}$ eine Formel eines Typs b, dann ist $[\![\boldsymbol{(\lambda x.\alpha)}]\!]^g$ diejenige Funktion von Typ (ab), so dass für jede $\boldsymbol{x}$-Alternative h *von* g gilt:
$[\![\boldsymbol{(\lambda x.\ \alpha)}]\!]^g\,(h(\boldsymbol{x})) = [\![\boldsymbol{\alpha}]\!]^h$.

Nach (44) ist der Wert des Lambda-Terms wirklich eine Funktion, die jedem Objekt des Typs a ein Objekt des Typs b zuordnet. Denn jedes Objekt u vom Typ a ist der Wert einer $\boldsymbol{x}$-Alternative zu g [für das Argument $\boldsymbol{x}$], nämlich *der Funktion, die wie g ist, außer dass sie der Variablen* $\boldsymbol{x}$ *das Objekt u zuordnet*. Wie man sich leicht überlegen kann, gibt es *eine einzige* $\boldsymbol{x}$-Alternative, die dies tut.[96] Wir werden sie '$g^{[x/u]}$' nennen und als *die an der Stelle* $\boldsymbol{x}$ *um u modizierte Belegung* bezeichnen. Mit dieser Notation lässt sich (44) wie folgt reformulieren:

(44') *Deutung der Abstraktion (Abs)* [Standardformulierung]
Wenn g eine Belegung ist, $\boldsymbol{x}$ eine Variable eines Typs a und $\boldsymbol{\alpha}$ eine Formel eines Typs b, dann ist $[\![\boldsymbol{(\lambda x.\alpha)}]\!]^g$ diejenige Funktion vom Typ (ab), so dass für jedes Objekt u vom Typ a gilt:
$[\![\boldsymbol{(\lambda x.\ \alpha)}]\!]^g\,(u) = [\![\boldsymbol{\alpha}]\!]^{g[x/u]}$.

Man mache sich klar, dass (44') genau dasselbe besagt wie (44); aber aus der Reformulierung geht klarer hervor, dass es sich bei $[\![\boldsymbol{(\lambda x.\alpha)}]\!]^g$ in der Tat um ein Objekt des Typs (ab) handelt, also eine Funktion von Objekten des Typs a in Objekte des Typs b. (44') ist deswegen die in Lehrbüchern übliche Formulierung.

Unter der (bislang immer impliziten) Voraussetzung, dass die in der Metasprache benutzte Variable 's' für beliebige mögliche Situationen – und nur für diese – steht, dann lässt sich (44') auch mit dem metasprachlichen Lambda-Operator reformulieren:

(44'') *Deutung der Abstraktion (Abs)* [Schnellversion]
Wenn g eine Belegung ist, $\boldsymbol{x}$ eine Variable eines Typs a und $\boldsymbol{\alpha}$ eine Formel eines Typs b, dann ist:
$[\![\boldsymbol{(\lambda x_a.\ \alpha)}]\!]^g = \lambda u.\ [\![\boldsymbol{\alpha}]\!]^{g[x/u]}$.

(44'') macht deutlich, dass und inwiefern es sich bei dem $\boldsymbol{\lambda}$-Operator der Typenlogik um das Pendant zur Lambda-Notation in der Metasprache handelt.

Sind (44) – (44'') kompositionell? Ja und nein. Denn der jeweilige *Wert* einer Lambda-Formel (bei einer gegebenen Belegung g) setzt sich danach nicht aus den Werten ihrer unmittelbaren Teile zusammen. Vielmehr benötigt man für seine Konstruktion die Werte der Teile bei anderen Belegungen, den x-Alternativen oder Modifikationen. Andererseits lässt sich nach (44) und (44') die *Gesamtheit aller Werte* einer Lambda-Formel aus der Gesamtheit aller Werte ihrer Teile ermitteln – und diese Gesamtheit, die sich wieder als Funktion auffassen lässt, die Belegungen Werte zuweise, ist gerade die *Bedeutung*.

Nicht die semantischen Werte der Teilformeln kombinieren sich bei der Variablenbindung, sondern die gesamten Bedeutungen. (44) und (44') zeigen allerdings nur an, wie die Bedeutung des rechten Teils, also der Matrix $\boldsymbol{\alpha}$, bei der Bestimmung der semantischen Werte des Lambda-Terms eingeht. Welche Rolle dabei die diversen Werte von $\boldsymbol{x}$ spielen, wird in dieser (durchaus gängigen) Formulierung verschwiegen. In (44') ist von ihnen zumindest explizit nicht die Rede; in (44) kommen sie zwar in der entscheidenden Gleichung (43) vor, aber dennoch wird nicht

96 Wir setzen hier voraus, dass h und h' dieselbe Funktion sind, wenn sie auf allen Variablen übereinstimmen. Diese Voraussetzung wird in der Mengenlehre als *Extensionalitätsprinzip* bezeichnet und akzeptiert.

deutlich inwiefern der Wert der Gesamtformel sich *nur* aus ihnen und den Werten der Variablen ermittelt. Um die Kompositionalität der Deutung der Variablenbindung mit letzter Sicherheit nachzuweisen, müsste man (44) und (44') daher noch einmal reformulieren. Genauer gesagt müsste man dafür die Begriffe '$\boldsymbol{x}$-Alternative' bzw. 'an der Stelle $\boldsymbol{x}$ modifizierte Belegung' durch einen Begriff ersetzen, der allein auf die Bedeutung, die Gesamtheit der semantischen Werte, der Variablen $\boldsymbol{x}$, nicht aber auf die Variable selbst, Bezug nimmt. Dass dies möglich ist, wird in einer Übungsaufgabe nachgewiesen. – Die in (44) – (44'') definierte Bedeutungskombination lässt sich nun auch anwenden, um den Wert der hier noch einmal wiederholten Ausgangsformel (36) aus den in der Tabelle (42) gegebenen Werten der Teilformel $\boldsymbol{(\lambda x.\ S(x)(y))}$ zu ermitteln:

(36) $\boldsymbol{(\lambda y^{e}.\ (\lambda x^{e}.\ S(i)(x)(y)))}$

Wenn man jetzt noch die Situationsvariable $\boldsymbol{i}$ bindet, stellt sich (in einer Übungsaufgabe) heraus, dass der Wert der resultierenden Formel bei jeder Belegung derselbe ist. Kein Wunder: die Formel enthält dann keine freien Variablen mehr – sie ist, wie man in der Logik sagt, *geschlossen*.

5.4 Rechenregeln

Wie eingangs des Kapitels bereits angedeutet wurde, besteht ein immenser Vorteil der indirekten Deutung in einer gewissen Übersichtlichkeit. Je komplexer die zu bestimmenden und zu manipulierenden Bedeutungen werden, desto mehr wird sich diese Übersichtlichkeit in ihrer Darstellung als wünschenswert oder sogar notwendig erweisen. Die Übersichtlichkeit verdankt die Methode nicht zuletzt dem Umstand, dass die typenlogischen Formeln gewissen logischen Gesetzen unterliegen, die es erlauben, komplexe Formeln auf quasi mechanische Weise zu reduzieren, um dadurch ihre Lesbarkeit zu erhöhen. Um diese Reduktionen geht es im vorliegenden Abschnitt. Bei den genannten Gesetzen handelt es sich dabei keineswegs um zusätzliche oder gar willkürliche Festlegungen. Ganz im Gegenteil: sie ergeben sich zwingend aus der im vorangehenden Abschnitt gegebenen Deutung der Formeln. Denn aus ihr folgt in vielen Fällen, dass zwei gegebene Formeln in dem Sinne *(logisch) äquivalent* sind[97], als sie stets – d.h. bei beliebigen Belegungen – denselben Wert besitzen. Hier ist ein einfaches Beispiel:

(45) Wenn $\boldsymbol{\varphi}$ und $\boldsymbol{\psi}$ Formeln des Typs $\boldsymbol{t}$ sind, dann gilt:
$$\boldsymbol{[\varphi \wedge \psi] \equiv [\psi \wedge \varphi]}$$

Das Symbol '$\equiv$' steht für die genannte logische Äquivalenz. Man beachte, dass es sich dabei nicht um ein Symbol der Typenlogik handelt, sondern um eine metasprachliche Abkürzung. Insbesondere ist also die in (45) herausgestellte Zeile selbst keine typenlogische Formel, sondern steht für die folgenden metasprachliche Aussage:

(45') Für alle Belegungen g gilt: $[\![\boldsymbol{[\varphi \wedge \psi]}]\!]^{g} = [\![\boldsymbol{[\psi \wedge \varphi]}]\!]^{g}$.

Inwiefern folgt (45) aus der obigen Deutung der typenlogischen Formeln? Ganz einfach: man kann die Behauptung mit ihrer Hilfe *beweisen*, es handelt sich also (etwas pompös gesprochen) um einen mathematischen *Satz*. Um einen Eindruck zu vermitteln, wie man solche Rechenregeln wie (45) prinzipiell rechtfertigt, und dass sie sich dabei insbesondere einzig und allein auf die kompositionelle Deutung der Formeln verlässt, führen wir den Beweis von (45) exemplarisch vor; danach werden wir auf detaillierte Nachweise verzichten.

Nehmen wir also einmal an, wir hätten es mit irgendwelchen Formeln $\boldsymbol{\varphi}$ und $\boldsymbol{\psi}$ des Typs π zu tun. Um die in (45) behauptete Äquivalenz zu zeigen, müssen wir dann eine beliebige Belegung

97 Die Verwendung dieses Begriffs weicht von der üblichen Praxis in der Logik ab, wo er etwas enger gefasst ist.

g betrachten und die Werte von **[φ ∧ ψ]** und **[ψ ∧ φ]** an g – also $[\![\mathbf{[\varphi \wedge \psi]}]\!]^g$ und $[\![\mathbf{[\psi \wedge \varphi]}]\!]^g$ – miteinander vergleichen. Zunächst einmal sollten wir dazu die notationellen Vereinfachungen rückgängig machen; denn '**[φ ∧ ψ]** ' steht ja für die typenlogische Formel **∧(ψ)(φ)** , die mit Hilfe der logischen Konstanten **∧** von Typ ***t(tt)*** gebildet wurde, deren Bedeutung in der Gleichung (31) gegeben wurde, die wir hier noch einmal wiederholen:

(31) $[\![\wedge]\!] = \lambda v.\, \lambda u.\, u \cdot v$

Des Weiteren nehmen wir zur Kenntnis, dass es sich bei **∧(ψ)(φ)** um das Ergebnis einer zweifachen Anwendung der syntaktischen Regel *(App)* handelt: zunächst wird aus der Konstanten **∧** – die ja nach *(Kon)* insbesondere eine Formel des Typs ***t(tt)*** ist – und der Formel **ψ** die Formel **∧(ψ)** von Typ pp gebildet; dann entsteht aus letzterer und der Formel **φ** das Endergebnis. (Malen Sie sich einen syntaktischen Baum, wenn Sie Probleme haben, das nachzuvollziehen!) Und beide Male ist dafür, wie gesagt, die Konstruktionsregel *(App)* verantwortlich. Da dies so ist, lässt sich der Wert von **∧(ψ)(φ)** schrittweise mit Hilfe der im vorangehenden Abschnitt eingeführten und hier wiederholten Deutung (35) von nach *(App)* gebildeten Formeln ermitteln:

(35) *Deutung der Applikation (App)*
Wenn g eine Belegung ist, und **α** und **β** Formeln des Typs ab und b, dann gilt:
$[\![\boldsymbol{\alpha(\beta)}]\!]^g = [\![\boldsymbol{\alpha}]\!]^g([\![\boldsymbol{\beta}]\!]^g)$.

Und das geht so:[98]

(46) $[\![\mathbf{\wedge(\psi)(\varphi)}]\!]^g$

= $[\![\mathbf{\wedge(\psi)}]\!]^g([\![\boldsymbol{\varphi}]\!]^g)$ — nach *(App)*: **∧(ψ)** für **α** , **φ** für **β**

= $[\![\boldsymbol{\wedge}]\!]^g([\![\boldsymbol{\psi}]\!]^g)([\![\boldsymbol{\varphi}]\!]^g)$ — nach *(App)*: **∧** für **α** , **ψ** für **β**

= $[\lambda v.\, \lambda u.\, u \cdot v]([\![\boldsymbol{\psi}]\!]^g)([\![\boldsymbol{\varphi}]\!]^g)$ — nach (31)

= $[\lambda u.\, u \cdot [\![\boldsymbol{\psi}]\!]^g]([\![\boldsymbol{\varphi}]\!]^g)$ — **λ**-Konversion

= $[\![\boldsymbol{\varphi}]\!]^g \cdot [\![\boldsymbol{\psi}]\!]^g$ — **λ**-Konversion

Ganz analog lässt sich nun mit Hilfe derselben Regeln und Festlegungen der semantische Wert von **[ψ ∧ φ]** bei der Belegung g bestimmen. Das Ergebnis ist:

(47) $[\![\mathbf{\wedge(\varphi)(\psi)}]\!]^g = [\![\boldsymbol{\psi}]\!]^g \cdot [\![\boldsymbol{\varphi}]\!]^g$

Aber natürlich handelt es sich dabei um denselben Wert wie den in (46) ermittelten – nämlich dem (arithmetischen) Produkt der Wahrheitswerte $[\![\varphi]\!]^g$ und $[\![\psi]\!]^g$. Damit ist die in (45) behauptete Äquivalenz – die sog. *Kommutativität der Konjunktion* – in der Tat nachgewiesen.

Die Rechenregel (45) ist gänzlich trivial; man sieht sie auch ohne einen umständlichen Beweis ein. Und unter dem Aspekt der Übersichtlichkeit ist sie nicht einmal besonders interessant; denn die in ihr gleichgesetzten Formeln sind gleich lang. Der Zweck des obigen expliziten Äqui-

98 Wir machen bei diesem Nachweis zweimal von der λ-Konversion Gebrauch. Sicherheitshalber sei darauf hingewiesen, dass dies nicht notwendig ist. Stattdessen kann man sich darauf besinnen, dass es sich bei den metasprachlichen Lambda-Termen um Abkürzungen für Beschreibungen von Funktionen handelt, und dann diese Beschreibungen selbst einsetzen. So steht z.B. der λ-Term '$[\lambda u.\, u \cdot [\![\boldsymbol{\psi}]\!]^g]$ ' für 'diejenige Funktion f, die jedem Wahrheitswert u als Wert $u \cdot [\![\boldsymbol{\psi}]\!]^g$ zuweist', und '$[\lambda u.\, u \cdot [\![\boldsymbol{\psi}]\!]^g]([\![\boldsymbol{\varphi}]\!]^g)$ ' benennt den Wert der so beschriebenen Funktion für das spezielle Argument $u = [\![\boldsymbol{\varphi}]\!]^g$ – also $[\![\boldsymbol{\varphi}]\!]^g \cdot [\![\boldsymbol{\psi}]\!]^g$.

valenznachweises lag ja auch nicht im Ergebnis, sondern im Weg: wir sehen, dass sich logische Äquivalenzen im Prinzip aufgrund der Festlegungen im vorangehenden Abschnitt beweisen lassen.

Viele der wichtigsten Rechenregeln nehmen auf den bereits angesprochenen Unterschied zwischen freien und gebundenen Variablen Bezug. Ein wesentliches Charakteristikum der Bindung ist es, dass der Wert der Variablen durch sie unabhägnig von der Belegung gemacht wird. Diese Beobachtung ist Inhalt eines grundlegenden Hilfssatzes (oder *Lemmas*):[99]

(48) *Koinzidenzlemma*
Es sei $\boldsymbol{\alpha}$ eine typenlogische Formel. Dann gilt für alle Belegungen g und h, die auf den in $\boldsymbol{\alpha}$ freien Variablen übereinstimmen:
$[\![\boldsymbol{\alpha}]\!]^g = [\![\boldsymbol{\alpha}]\!]^h$.

Dass zwei Belegungen g und h auf einer Variablen $\boldsymbol{x}$ *übereinstimmen*, heißt dabei natürlich, dass $g(\boldsymbol{x}) = h(\boldsymbol{x})$. Das Koinzidenzlemma besagt danach, dass sich die Übereinstimmung im Wert von den freien Variablen auf die gesamte Formel überträgt. Im Klartext bedeutet das, dass der Wert der Formel $\boldsymbol{\alpha}$ nur von der Belegung der in $\boldsymbol{\alpha}$ freien Variablen abhängt – also weder von den Variablen, die gar nicht in $\boldsymbol{\alpha}$ vorkommen, noch von denen, die (in $\boldsymbol{\alpha}$) nur gebunden vorkommen. Und genau letzteres ist die Folge der oben beobachten Neutralisierung der Belegungsabhängigkeit durch Bindung.

Der Beweis von (48) erfolgt schrittweise, indem man ihn zunächst für einfache, nicht zusammengesetzte Formeln führt, und dann bei komplexen Formeln zeigt, wie sich die im Koinzidenzlemma behauptete Belegungsunabhängigkeit gebundener Variablen von den unmittelbaren Teilen auf die Gesamtformel vererbt. Die Strategie ist also das aus der Arithmetik bekannte Prinzip der *vollständigen Induktion*.[100] Wir ersparen uns diesen Beweis, machen uns aber klar, dass er nur gegeben werden kann auf dem Hintergrund einer präzisen Definition des Begriffs der *freien Variablen*. Dieser Begriff lässt sich ebenfalls schrittweise definieren, indem man für jede Formel $\boldsymbol{\alpha}$ eine Menge $Fr(\boldsymbol{\alpha})$ angibt:

(49) *Definition der in einer Formel* $\boldsymbol{\alpha}$ *frei vorkommenden Variablen*

Formel $\boldsymbol{\alpha}$	*Bildungsregel*	$Fr(\boldsymbol{\alpha})$
$\mathbf{c}$	*Kon*	$\emptyset$
$\boldsymbol{x}$	*Var*	$\{\boldsymbol{x}\}$
$\boldsymbol{\beta}(\boldsymbol{\gamma})$	*App*	$Fr(\boldsymbol{\beta}) \cup Fr(\boldsymbol{\gamma})$
$(\boldsymbol{\lambda x}.\boldsymbol{\beta})$	*Abs*	$Fr(\boldsymbol{\beta}) \setminus \{\boldsymbol{x}\}$

Der Tabelle (49) kann man entnehmen, welche Variablen in einer gegebenen typenlogischen Formel $\boldsymbol{\alpha}$ frei vorkommen. Handelt es sich bei $\boldsymbol{\alpha}$ um eine Konstante (Zeile 1), ist die Menge der in $\boldsymbol{\alpha}$ frei vorkommenden Variablen leer – denn in einer Konstanten kommt keine Variable vor. Ist dagegen $\boldsymbol{\alpha}$ selbst eine Variable (Zeile 2), kommt diese natürlich auch in $\boldsymbol{\alpha}$ vor – und zwar frei, denn es gibt in $\boldsymbol{\alpha}$ kein $\boldsymbol{\lambda}$, das die Variable binden könnte. In einer Applikation $\boldsymbol{\beta}(\boldsymbol{\gamma})$ kommt eine Variable frei vor, wenn sie in (mindestens) einem der Teile frei vorkommt; die Menge der in $\boldsymbol{\beta}(\boldsymbol{\gamma})$ freien Variablen ergibt sich somit durch Vereinigung von $Fr(\boldsymbol{\beta})$ mit $Fr(\boldsymbol{\gamma})$. Und die freien

99 Statt vom Koinzidenz*lemma* spricht man auch vom Koinzidenz*theorem*. In den meisten Logiklehrbüchern findet man eine etwas allgemeinere ('modelltheoretische') Version dieses Hilfssatzes.

100 Das ist das Prinzip, nach dem jede Zahl eine gegebene Eigenschaft E hat, sobald die 0 E hat und sich E von einer Zahl (n) auf die nächste (n+1) überträgt.

Variablen einer Abstraktion **(λ*x*.β)** sind dieselben wie die in der Matrix **β** – außer natürlich dem vom **λ** gebundenen ***x***.

Dass eine Variable in einer Formel **α** frei ist, heißt nicht, dass sie nicht gleichzeitig auch in **α** gebunden sein kann; denn die Variable kann ja an mehreren Stellen vorkommen. Umgekehrt kommen Variablen, die nicht in *Fr*(**α**) auftauchen, entweder in **α** gar nicht vor oder nur als gebundene Variablen. Eine Formel, die überhaupt keine freien Variablen enthält, nennt man *geschlossen*; entsprechend sind *offene* Formeln solche mit freien Variablen. Aus dem Koinzidenzlemma folgt unmittelbar, dass die semantischen Werte geschlossener Formeln immer belegungsunabhängig sind; denn wenn *g* und *h* irgendwelche Belegungen sind und **α** eine geschlossene Formel ist, dann stimmen *g* und *h* trivialerweise auf allen freien Variablen von **α** überein – *Fr*(**α**) ist ja leer. Bei geschlossenen Formeln kann man also den Wert einfach als '$[\![\alpha]\!]$' notieren, wie wir das schon bei Konstanten gemacht haben (die ja auch geschlossene Formeln sind). Wir werden reichhaltig Gelegenheit haben, von dieser Notationskonvention Gebrauch zu machen; denn die typenlogischen Übersetzungen natürlichsprachlicher Ausdrücke sind in aller Regel geschlossene Formeln. Ausnahmen lernen wir erst im nächsten Kapitel kennen.

Variablenbindung schafft Belegungsunabhängigkeit. Das ist die erste von zwei zentralen Eigenschaften dieses Vorgangs. Die zweite ist der ebenfalls aus dem vortheoretischen Umgang mit Variablen vertraute Sachverhalt, dass die Identität einer gebundenen Variablen keine Rolle spielt: ob man von einem beliebigen *x* spricht oder einem beliebigen *y*, ist egal – gebundene Variablen kann man ohne Änderung der Gesamtaussage (einigermaßen) beliebig umbenennen. Diese Einsicht werden wir in einer Rechenregel festhalten, die auf einem Prinzip beruht, das ebenso fundamental ist für das Verständnis der Variablenbindung wie das Koinzidenzlemma. Das Prinzip betrifft die Ersetzung *freier* Variablen durch beliebige Formeln gleichen Typs und besagt, dass das Ergebnis einer solchen Ersetzung den Wert der Gesamtformel (in der die Ersetzung vorgenommen wurde) nicht verändert, wenn die ersetzte Variable denselben Wert hat wie die Formel, die für sie eingesetzt wird. Das klingt komplizierter, als es ist. Betrachten wir lieber ein Beispiel:

(50) $\mathbf{T}(\boldsymbol{i})(\boldsymbol{x}^{e})(\mathbf{e})$

In (50) ist **T** die schon weiter oben benutzte Konstante des Typs ***s(e(et))***, die als Übersetzung des transitiven Verbs **trifft** fungiert, und **e** ist eine Konstante des Typs ***e***, deren (belegungsunabhängiger) Wert Eike ist. (50) ist also eine offene Formel vom Typ ***t*** – ***x*** und ***i*** kommen frei vor – und der Wert dieser Proposition bei einer Belegung g hängt offenbar davon ab, welches Individuum *g* der Variablen ***x*** zuweist und was die betrachtete Situation $g(\boldsymbol{i})$ ist: wenn $g(\boldsymbol{x})$ = Eike und $g(\boldsymbol{i}) = s^{*}$, ist $[\![\mathbf{T}(\boldsymbol{i})(\boldsymbol{x})(\mathbf{e})]\!]^{g}$ der Wahrheitswert 1, wenn Eike sich in s^{*} selbst trifft (also eine kleine Menge abstruser Situationen);[101] wenn $h(\boldsymbol{x})$ = Fritz und $h(\boldsymbol{i}) = s^{+}$, ist $[\![\mathbf{T}(\boldsymbol{i})(\boldsymbol{x})(\mathbf{e})]\!]^{h}$ der Wert 1, wenn Eike Fritz in s^{+} trifft; etc. Wenn nun **f** eine Konstante des Typs ***e*** ist, deren Wert Fritz ist, dann ist klar, dass die Formel (50) bei der soeben betrachteten Belegung *h* denselben Wert hat wie:

(51) $\mathbf{T}(\boldsymbol{i})(\mathbf{f})(\mathbf{e})$

(51) geht aus (50) hervor, indem die freie Variable ***x*** durch die Konstante **f** ersetzt wurde. Da die beiden an der Belegung *h* denselben Wert – Fritz – haben, überträgt sich diese Wertgleichheit auf die Gesamtformeln. Dahinter steckt ein allgemeines Prinzip, das *Substitutionslemma*.

101 Dabei ist die (wörtliche) Lesart von **treffen** gemeint, die nahezu synonym ist mit **begegnen**: treffen in dem Sinn kann sich jemand selbst wohl nur im Rahmen von Zeitreisen (– oder?). Bei einer anderen Lesart von [**jemanden mit etwas**] **treffen** werden die Situationen weniger abstrus, dafür aber teilweise makaber.

Um es zu formulieren, greift man auf einen allgemeinen Begriff der Substitution freier Variablen zurück: wenn $\boldsymbol{\alpha}$ irgendeine Formel ist, bezeichnet man mit '$[\![\boldsymbol{\alpha}[^{\boldsymbol{x}}/_{\boldsymbol{\delta}}]]\!]^{h}$' die Formel, die entsteht, indem man alle freien Vorkommen der Variablen $\boldsymbol{x}$ durch die Formel $\boldsymbol{\delta}$ (des gleichen Typs) ersetzt. Mit dieser Notation lässt sich der Übergang von (50) zu (51) beschreiben als: $\mathbf{T}(\boldsymbol{i})(\boldsymbol{x}^{e})(\mathbf{e})\,[^{\boldsymbol{x}}/_{\mathbf{f}}] = \mathbf{T}(\boldsymbol{i})(\mathbf{f})(\mathbf{e})$. Auch die Substitution kann man schrittweise, also induktiv, definieren:

(52) *Definition der Ersetzung der freien $\boldsymbol{x}$ in einer Formel $\boldsymbol{\alpha}$ durch $\boldsymbol{\delta}$*

Formel $\boldsymbol{\alpha}$	*Bildungsregel*	$\boldsymbol{\alpha}[^{\boldsymbol{x}}/_{\boldsymbol{\delta}}]$
$\mathbf{c}$	*Kon*	$\mathbf{c}$
$\boldsymbol{x}$	*Var*	$\boldsymbol{\delta}$
$\boldsymbol{y}$ $[\neq \boldsymbol{x}]$	*Var*	$\boldsymbol{y}$
$\boldsymbol{\beta}(\boldsymbol{\gamma})$	*App*	$\boldsymbol{\beta}[^{\boldsymbol{x}}/_{\boldsymbol{\delta}}](\boldsymbol{\gamma}[^{\boldsymbol{x}}/_{\boldsymbol{\delta}}])$
$(\boldsymbol{\lambda x}.\boldsymbol{\beta})$	*Abs*	$(\boldsymbol{\lambda x}.\boldsymbol{\beta})$
$(\boldsymbol{\lambda y}.\boldsymbol{\beta})$ $[\boldsymbol{y} \neq \boldsymbol{x}]$	*Abs*	$(\boldsymbol{\lambda y}.\boldsymbol{\beta}[^{\boldsymbol{x}}/_{\boldsymbol{\delta}}])$

Die Tabelle (52) zeigt, was passiert, wenn man alle freien Vorkommen einer Variablen $\boldsymbol{x}$ in einer gegebenen typenlogischen Formel $\boldsymbol{\alpha}$ durch eine Formel $\boldsymbol{\delta}$ ersetzt. Damit das Ergebnis $\boldsymbol{\alpha}[^{\boldsymbol{x}}/_{\boldsymbol{\beta}}]$ dieser Ersetzung überhaupt eine typenlogische Formel ist, wird dabei vorausgesetzt, dass $\boldsymbol{x}$ und $\boldsymbol{\delta}$ Formeln desselben Typs sind; $\boldsymbol{\alpha}$ kann dagegen von einem anderen Typ sein, wie das Beispiel (51) zeigt, wo in der Gesamtformel (50) vom Typ $\boldsymbol{t}$ das freie $\boldsymbol{x}$ vom Typ e durch die Konstante $\mathbf{f}$ (vom selben Typ) ersetzt wurde. Handelt es sich bei $\boldsymbol{\alpha}$ um eine Konstante (Zeile 1), passiert bei der Ersetzung der freien $\boldsymbol{x}$ in $\boldsymbol{\alpha}$ gar nichts – denn weder $\boldsymbol{x}$ noch sonst eine Variable kommt in $\boldsymbol{\alpha}$ frei vor; das Ergebnis der Ersetzung ist also $\boldsymbol{\alpha}$ selbst. Ist dagegen $\boldsymbol{\alpha}$ selbst die Variable $\boldsymbol{x}$ (Zeile 2a), kommt diese natürlich auch in $\boldsymbol{\alpha}$ vor, und wenn man dieses eine Vorkommen durch $\boldsymbol{\delta}$ ersetzt, ist $\boldsymbol{\delta}$ eben das Ergebnis $\boldsymbol{x}[^{\boldsymbol{x}}/_{\boldsymbol{\delta}}]$ dieser Ersetzung. Handelt es sich bei $\boldsymbol{\alpha}$ dagegen um eine andere, von $\boldsymbol{x}$ verschiedene Variable, passiert bei der Substitution das Gleiche wie im ersten Fall (Zeile 1) – nämlich gar nichts. Will man in einer Applikation $\boldsymbol{\beta}(\boldsymbol{\gamma})$ (Zeile 3) die Variable $\boldsymbol{x}$ überall, wo sie frei vorkommt, durch $\boldsymbol{\delta}$ ersetzen, muss man dies in den beiden unmittelbaren Teilen der Formel – dem Funktor $\boldsymbol{\beta}$ und dem Argumentterm $\boldsymbol{\gamma}$ – tun und die Ergebnisse dieser Ersetzung wieder per Funktionalapplikation kombinieren. Analog verfährt man bei Abstraktionen $(\boldsymbol{\lambda y}.\boldsymbol{\beta})$ (Zeile 4b), in denen die gebundene Variable nicht das $\boldsymbol{x}$ ist: hier muss man die Ersetzung in der Matrix $\boldsymbol{\beta}$ vornehmen und dann die (von $\boldsymbol{x}$ verschiedene) Variable abstrahieren. Wird dagegen $\boldsymbol{x}$ selbst vom $\boldsymbol{\lambda}$ gebunden (Zeile 4a), passiert bei der Substitution gar nichts; denn das $\boldsymbol{\lambda}$ lässt keine zu ersetzenden freien Vorkommen von $\boldsymbol{x}$ übrig.

Man mache sich klar, dass nach der in Tabelle (52) gegebenen Definition der Ersetzung in der Tat gilt: $\mathbf{T}(\boldsymbol{i})(\boldsymbol{x}^{e})(\mathbf{e})\,[^{\boldsymbol{x}}/_{\mathbf{f}}] = \mathbf{T}(\boldsymbol{i})(\mathbf{f})(\mathbf{e})$. Wenn die zu ersetzende Variable bei einer gegebenen Belegung denselben Wert hat wie die sie ersetzende Konstante, sollte sich diese Wertgleichheit also auf die beiden Gesamtformeln (vor und nach der Substitution) übertragen. Allerdings gilt diese Übertragung der Wertgleichheit nicht immer; d.h. das folgende Prinzip wäre in seiner Allgemeinheit falsch:

(53) Wenn $[\![\boldsymbol{x}]\!]^{g} = [\![\boldsymbol{\delta}]\!]^{g}$, dann ist $[\![\boldsymbol{\alpha}]\!]^{g} = [\![\boldsymbol{\alpha}[^{\boldsymbol{x}}/_{\boldsymbol{\delta}}]]\!]^{g}$

Hier ist ein simples Gegenbeispiel. Man wähle als $\boldsymbol{\alpha}$ die Formel $(\boldsymbol{\lambda y^e.T(i)(y)(x)})$ und betrachte eine Belegung g, nach der $g(\boldsymbol{x}) = g(\boldsymbol{y})$ = Eike. Dann ist auch $[\![\boldsymbol{x}]\!]^g = [\![\boldsymbol{y}]\!]^g$, und nach (53) – mit $\boldsymbol{y}$ für $\boldsymbol{\delta}$ – würde jetzt folgen: $[\![(\boldsymbol{\lambda y^e.T(i)(y)(x)})]\!]^g = [\![(\boldsymbol{\lambda y^e.T(i)(y)(x)})[{}^{x}\!/_{y}]]\!]^g$. Aber $(\boldsymbol{\lambda y^e.T(i)(y)(x)})[{}^{x}\!/_{y}] = (\boldsymbol{\lambda y^e.T(i)(y)(y)})$ – was trotz der Gleichbelegung von $\boldsymbol{x}$ und $\boldsymbol{y}$ nicht denselben Wert haben muss wie $(\boldsymbol{\lambda y^e.T(i)(y)(x)})$: wendet man $[\![(\boldsymbol{\lambda y^e.T(i)(y)(x)})]\!]^g$ auf Fritz an, erhielte man für Situationen $g(\boldsymbol{i})$, in denen Eike (= $g(\boldsymbol{x})$) Fritz trifft, den Wahrheitswert 1; diesen Wert liefert $[\![(\boldsymbol{\lambda y^e.T(i)(y)(y)})]\!]^g$ für das Argument Fritz nur in den absurden Situationen $g(\boldsymbol{i})$, in denen Fritz sich selbst begegnet.

Das Gegenbeispiel lebt davon, dass die Variable $\boldsymbol{y}$ bei der Einsetzung für $\boldsymbol{x}$ in den Skopus eines '$\boldsymbol{\lambda y}$' gerät und somit gebunden wird: da Bindung Belegungsunabhängigkeit schafft, wird damit die Tatsache, dass $\boldsymbol{x}$ und $\boldsymbol{y}$ bei der betrachteten Belegung g wertgleich sind, unerheblich; das gebundene $\boldsymbol{y}$ hat hat ja gar keinen eigenständigen Wert. Insbesondere kann sich also in diesem Fall keine Wertgleichheit von Teilausdrücken $\boldsymbol{x}$ und $\boldsymbol{y}$ auf die Gesamtformel übertragen. Das Prinzip (53) muss also dahingehend eingeschränkt werden, dass ganz allgemein keine ehemals freien Variablen $\boldsymbol{z}$ 'versehentlich' gebunden werden – also in den Skopus des bindenen '$\boldsymbol{\lambda z}$' geraten – wenn man sie für die Variable $\boldsymbol{x}$ einsetzt. Die Gefahr der versehentlichen Bindung besteht dabei nicht nur in den Fällen, wo die einzusetzende Formel $\boldsymbol{\delta}$ selber eine Variable ist, sondern auch dann, wenn $\boldsymbol{\delta}$ eine komplexe Formel ist, die eine freie Variable enthält, welche beim Einsetzen gebunden würde; ein entsprechendes Beispiel wird in den Übungsaufgaben betrachtet. Um das in (53) angestrebte allgemeine Prinzip einigermaßen griffig formulieren zu können, empfiehlt sich eine eigene Benennung für den Sachverhalt, dass eine Variable versehentlich gebunden würde:

(54) *Definition*
Es seien $\boldsymbol{\alpha}$ und $\boldsymbol{\delta}$ typenlogische Formeln und $\boldsymbol{x}$ eine Variable desselben Typs wie $\boldsymbol{\delta}$. Dann ist $\boldsymbol{x}$ *frei für [die Einsetzung von]* $\boldsymbol{\delta}$ *in* $\boldsymbol{\alpha}$, falls $\boldsymbol{\alpha}$ für kein $\boldsymbol{z} \in Fr(\boldsymbol{\delta})$ einen Teil der Gestalt '$(\boldsymbol{\lambda z.}\ \boldsymbol{\beta})$' enthält, in dem $\boldsymbol{x}$ frei vorkommt.

Diese Definition wird leichter verständlich, wenn man sich klar macht, unter welchen Umständen eine Variable $\boldsymbol{x}$ für eine Formel $\boldsymbol{\delta}$ *nicht* frei ist in einer Formel $\boldsymbol{\alpha}$. Nach (54) liegt ein solcher Fall gerade dann vor, wenn $\boldsymbol{\delta}$ eine freie Variable $\boldsymbol{z}$ enthält und es in $\boldsymbol{\alpha}$ einen Teil der Gestalt '$(\boldsymbol{\lambda z.}\ \boldsymbol{\beta})$' gibt, in dem $\boldsymbol{x}$ frei vorkommt. Wenn man dann (die freien Vorkommen von) $\boldsymbol{x}$ in $\boldsymbol{\alpha}$ durch $\boldsymbol{\delta}$s ersetzt, müsste man insbesondere das $\boldsymbol{x}$ in '$(\boldsymbol{\lambda z.}\ \boldsymbol{\beta})$' ersetzen. Aber dann geriete das eingesetzte $\boldsymbol{\delta}$ in den Skopus des '$\boldsymbol{\lambda z}$' – und mit ihm die in $\boldsymbol{\delta}$ freien $\boldsymbol{z}$, die somit versehentlich gebunden würden. Genau das passiert, wenn $\boldsymbol{x}$ *nicht* frei ist für $\boldsymbol{\delta}$ in $\boldsymbol{\alpha}$; und genau das muss verhindert werden. Wir gelangen damit zu der folgenden eingeschränkten, aber korrekten Version von (53):

(55) *Substitutionslemma*
Es seien $\boldsymbol{\alpha}$ und $\boldsymbol{\delta}$ typenlogische Formeln und $\boldsymbol{x}$ eine Variable (desselben Typs wie $\boldsymbol{\delta}$), die frei ist für $\boldsymbol{\delta}$ in $\boldsymbol{\alpha}$. Dann gilt für alle Belegungen g:
Wenn $[\![\boldsymbol{x}]\!]^g = [\![\boldsymbol{\delta}]\!]^g$, dann ist $[\![\boldsymbol{\alpha}]\!]^g = [\![\boldsymbol{\alpha}[{}^{x}\!/_{\delta}]]\!]^g$.

Auch in diesem Fall ersparen wir uns den Beweis, der sich wieder schrittweise von einfachen zu immer komplexeren Formel führen lässt. Stattdessen sehen wir uns an, wie das Substitutionslemma gemeinsam mit dem Koinzidenzlemma die oben erwähnte Beliebigkeit der Variablenbenennung impliziert. Unter dieser Beliebigkeit ist zu verstehen, dass man eine Bindung der Gestalt $(\boldsymbol{\lambda x.}\ \ldots)$ umschreiben kann in $(\boldsymbol{\lambda y.}\ \ldots)$, wenn man gleichzeitig in '…' alle durch das Lambda gebundenen $\boldsymbol{x}$ durch $\boldsymbol{y}$ ersetzt. Allerdings darf diese Variable $\boldsymbol{y}$ nicht

schon in '…' frei vorkommen; sonst würde ja das **(λ*y*.** …**)** diese(s) Vorkommen binden. (In einer Übungsaufgabe kann man sich dies an dem Beispiel **(λ*x*. *x*=*y*)** klar machen.) Außerdem muss wieder verhindert werden, dass das ***y*** beim Einsetzen für ***x*** durch ein '**λ*y***' gebunden wird; ***x*** muss also frei für ***y*** sein.

(56) *Prinzip der gebundenen Umbenennung*
Es sei **α** eine typenlogische Formeln und ***x*** eine Variable. Wenn dann ***y*** eine Variable (desselben Typs wie ***x***) ist, die nicht frei ist in **α** und für die ***x*** frei ist in **α**, dann gilt:

$$(\lambda x.\alpha) \equiv (\lambda y.\alpha[x/y]) \ .$$

Um einzusehen, dass (56) im Rahmen der in Abschnitt 5.3 gegebenen Deutung der Lambda-Abstraktion korrekt ist, kann man irgendeine Belegung *g* betrachten und zeigen, dass die Werte der beiden in (56) angegebenen Formeln unter den dort genannten Voraussetzungen gleich sind. Da die Lambda-Abstraktion immer zu Formeln eines Typs der Gestalt *ab* führt, handelt es sich bei diesen Werten um Funktionen von Objekten des Typs ***a*** der Variablen ***x*** und ***y*** in Objekte des Typs ***b*** von **α**. Und diese Funktionen sind gleich, wenn sie für alle Objekte *u* des Typs *a* denselben Wert liefern. Es muss also gelten, dass $[\![(\lambda x.\alpha)]\!]^g(u) = [\![(\lambda y.\alpha[x/y])]\!]^g(u)$. Und das ist in der Tat so:

(57)

$$[\![(\lambda x.\alpha)]\!]^g(u)$$

nach der Deutung (44') der Abstraktion

$$= [\![\alpha]\!]^{g[x/u]}$$

nach dem Koinzidenzlemma; denn $y \notin Fr(\alpha)$

$$= [\![\alpha]\!]^{g[x/u][y/u]}$$

nach dem Substitutionslemma; denn ***x*** ist frei für ***y*** in **α**, und $[\![x]\!]^{g[x/u][y/u]} = u = [\![y]\!]^{g[x/u][y/u]}$

$$= [\![\alpha[x/y]]\!]^{g[x/u][y/u]}$$

nach dem Koinzidenzlemma; denn $x \notin Fr(\alpha[x/y])$

$$= [\![\alpha[x/y]]\!]^{g[y/u]}$$

nach der Deutung (44') der Abstraktion

$$= [\![(\lambda y.\alpha[x/y])]\!]^g(u)$$

Die in (56) angegebene logische Äquivalenz ist eine Rechenregel, die wir bei der indirekten Deutung häufig anwenden werden. Dabei wird wichtig sein, dass es immer genügend Variablen ***y*** gibt, die die dort gestellte Bedingung erfüllen – für die also das zu ersetzende ***x*** frei ist und die im Skopus des '**λ*x***' nicht frei vorkommen; denn es gibt unendlich viele Variablen jedes Typs.

Die für die indirekte Deutung wichtigste Rechenregel ist zweifellos die Lambda-Konversion, die wir bereits in der Metasprache kennen gelernt haben. Als typenlogische Äquivalenz können wir sie jetzt allerdings wesentlich genauer fassen:

(58) *Prinzip der Lambda-Konversion*
Es seien **α** und **β** typenlogische Formeln und ***x*** eine Variable (desselben Typs wie **β**), die frei ist für **β** in **α**. Dann gilt:

$$(\lambda x.\alpha)(\beta) \equiv \alpha[x/\beta] \ .$$

Natürlich muss man auch bei der Lambda-Konversion vermeiden, dass versehentlich Variablen gebunden werden, die vorher frei waren. So ist z.B. **(λ*x*.(∃*y*) T(*i*)(*x*)(*y*)) (*y*)** nicht äquivalent zu **(λ*x*.(∃*y*) T(*i*)(*y*)(*y*))**, wie man sich leicht überlegen kann (und soll – in einer Übungsaufgabe). Die Nebenbedingung über die Variable schließt diesen Fall gerade aus.

(58) basiert im wesentlichen auf Koinzidenz- und Substitutionslemma, wobei auch das Prinzip (56) der gebundenen Umbenennung seine Rolle spielt. Sehen wir uns den Nachweis der in (58) behaupteten Äquivalenz (unter den dort genannten Bedingungen) genauer an! Zunächst gilt aufgrund der Deutung (35) der Applikation für beliebige Belegungen *g*:

(59) $[\![(\lambda x.\alpha)(\beta)]\!]^g = [\![(\lambda x.\alpha)]\!]^g([\![\beta]\!]^g)$

Um den rechts vom Gleichheitszeichen stehenden Wert zu bestimmen, empfiehlt sich eine gebundene Umbenennung. Dazu betrachtet man eine Variable z des gemeinsamen Typs von x und β, die weder in α noch in β vorkommt, und zwar weder frei noch gebunden noch in einem Lambda-Präfix. So ein z muss es geben; schließlich gibt es unendlich viele Variablen jedes Typs. Da z in β nicht vorkommt, kann es beim Substituieren für x auch nicht versehentlich gebunden werden, d.h. x ist frei für z in α; da z außerdem nicht in α vorkommt, sind die Bedingungen für die gebundene Umbenennung (56) erfüllt, d.h. die Gleichung (59) lässt sich wie folgt verlängern:

(60) $[\![(\lambda x.\alpha)(\beta)]\!]^g = [\![(\lambda x.\alpha)]\!]^g([\![\beta]\!]^g) = [\![(\lambda z.\alpha[x/z])]\!]^g([\![\beta]\!]^g) = [\![\alpha[x/z]]\!]^{g[z/[\![\beta]\!]^g]}$

Der letzte Übergang ist eine einfache Anwendung der Deutung (44') der Abstraktion. Um jetzt das Substitutionslemma anzuwenden, müssen wir garantieren, dass einerseits (i) z frei ist für β in $\alpha[x/z]$ und andererseits (ii) z und β bei der Belegung $g[z/[\![\beta]\!]^g]$ denselben Wert haben. (i) ist der Fall, weil z in $\alpha[x/z]$ gerade die Stellen besetzt, an denen in α das x stand; und an diesen Stellen wird keine der Variablen von β versehentlich gebunden – denn wir setzen bei der Lambda-Konversion (58) voraus, dass x frei ist für β in α. Aber auch (ii) gilt, denn $[\![z]\!]^{g[z/[\![\beta]\!]^g]} = g[z/[\![\beta]\!]^g](z) = [\![\beta]\!]^g = [\![\beta]\!]^{g[z/[\![\beta]\!]^g]}$ – letzteres aufgrund des Koinzidenzlemmas und weil ja $z \notin Fr(\beta)$. Wir erhalten also aus (60) durch Anwendung des Substitutionslemmas:

(60') $[\![(\lambda x.\alpha)(\beta)]\!]^g = \ldots = [\![\alpha[x/z]]\!]^{g[z/[\![\beta]\!]^g]} = [\![\alpha[x/z][z/\beta]]\!]^{g[z/[\![\beta]\!]^g]}$

Aber offenbar ist $\alpha[x/z][z/\beta] = \alpha[x/\beta]$: die zwischendurch eingesetzten z stehen genau dort, wo ursprünglich ein x stand – andere z kommen ja in α nicht vor.[102] (60') reduziert sich somit auf:

(60") $[\![(\lambda x.\alpha)(\beta)]\!]^g = \ldots [\![\alpha[x/\beta]]\!]^{g[z/[\![\beta]\!]^g]}$

Da nun aber z weder in α noch in β vorkam, kann es auch nicht in $\alpha[x/\beta]$ vorkommen; und da sich g und $g[z/[\![\beta]\!]^g]$ nur im Wert für dieses z unterscheiden, erhält $\alpha[x/\beta]$ an diesen beiden Belegungen laut Koinzidenzlemma denselben Wert. (60") läuft also auf die nachzuweisende Äquivalenz im Prinzip (58) der Lambda-Konversion hinaus:

(61) $[\![(\lambda x.\alpha)(\beta)]\!]^g = \ldots [\![\alpha[x/\beta]]\!]^{g[z/[\![\beta]\!]^g]} = [\![\alpha[x/\beta]]\!]^g$

Die Lambda-Konversion wird in der Praxis vor allem von links nach rechts angewandt, also um eine Formel der Gestalt $(\lambda x.\alpha)(\beta)$ zu *reduzieren*. In der Regel ist die reduzierte Formel kürzer und vor allem leichter lesbar. Da sich die genannte Konstellation bei der indirekten Deutung ausgesprochen häufig ergibt, dabei aber nicht immer die Variablenbedingung erfüllt ist, ist es wichtig zu sehen, dass man letztere durch Anwendung des Prinzips (56) der gebundenen Umbenennung immer umgehen kann. Denn sollte das Argument β eine Variable y enthalten, die beim Einsetzen für x in α versehentlich gebunden würde, kann man *die gebundenen y in α* einfach umbenennen – und zwar durch eine Variable, die weder in α noch in β vorkommt und so keinerlei Gefahren birgt. Da es unendlich viele Variablen jedes Typs gibt, lässt sich eine solche Umbenennung immer – notfalls auch mehrfach – durchführen. Wir werden das Abwechseln von gebundener Umbenennung und Lambda-Konversion bereits im nächsten Abschnitt anhand von Beispielen studieren.

---

[102] Allgemeiner und genauer gilt für beliebige Formeln β und Variablen x und z desselben Typs: wenn x frei ist für z in einer (beliebigen) Formel α und $z \notin Fr(\alpha)$, dann ist: $\mathfrak{a}[x/z][z/\mathfrak{b}] = \mathfrak{a}[x/\mathfrak{b}]$. Das lässt sich wieder schrittweise (induktiv) zeigen.

Der vielleicht etwas kompliziert anmutende Nachweis von (58) sollte deutlich machen, dass es sich bei der Lambda-Konversion keineswegs um eine willkürliche formale Festlegung handelt, sondern um eine Rechenregel, die sich zwingend aus der Deutung der typenlogischen Formeln ergibt, wie wir sie im vorangehenden Abschnitt angegeben haben. Auch wenn es später manchmal den Anschein haben mag: nicht die Reduktion der Formeln gibt ihnen ihren Sinn, sondern ihre kompositionelle Deutung; die Reduktion erhält die Bedeutung der Formel und macht sie in der Regel allerdings leichter fassbar.

Ein spezieller, häufiger und wichtiger Fall der Lambda-Konversion liegt vor, wenn es sich bei dem Argument β in der Konstellation $(\lambda x.\alpha)(\beta)$ 'zufällig' um die im Präfix 'λx' gebundene Variable handelt: $(\lambda x.\alpha)(x)$. Zunächst scheint in diesem Fall die Variablenbedingung nicht ganz offensichtlich erfüllt zu sein; denn das Argument β ist die Variable x, was bedeutet, dass $Fr(\beta) \neq \emptyset$. Um die λ-Konversion anzuwenden, müsste man also zunächst überprüfen, ob eine in β freie Variable beim Einsetzen in α versehentlich gebunden würde. Da $Fr(\beta) = \{x\}$, genügt es zu überprüfen, ob dies für x der Fall wäre, ob also die durch das λ gebundene Variable x frei ist für x selbst in α. Nun werden aber bei der Ersetzung nur die *freien* Vorkommen von x in α erfasst, und wenn man für diese freien x wieder x einsetzt, ändert sich ganz offensichtlich nichts — $\alpha[x/x] = \alpha$; insbesondere bleiben die x frei, d.h. eine versehentliche Bindung ist ausgeschlossen. Die Konversion darf also durchgeführt werden, und weil $\alpha[x/x] = \alpha$, läuft sie auf eine Streichung des Präfixes '$\lambda x.$' und des Arguments '(x)' hinaus. Wir halten diesen Spezialfall der Lambda-Konversion als eigenes Prinzip fest:

(58*) *Prinzip der Eigen-Konversion*
Es seien α eine typenlogische Formel und x eine Variable. Dann gilt:
$(\lambda x.\alpha)(x) \equiv \alpha$.

Neben den bisher eingeführten, grundlegenden Äquivalenzen gibt es noch weitere Rechenregeln, die zwar nicht ganz so oft zum Einsatz kommen, die aber gelegentlich hilfreich sein können und deren Kenntnis auf alle Fälle für das Verständnis des Formalismus unerlässlich ist. Einige von ihnen listen wir hier einfach auf; weitere werden wir erwähnen, wenn wir sie benötigen. Alle lassen sich ganz leicht nachweisen:

(62) *Logische Äquivalenzen*
Für alle Formeln φ, ψ und χ des Typs t und alle Variablen x und y des Typs e gilt:

(a) $\neg \neg \varphi \equiv \varphi$

(b) $[\varphi \wedge [\psi \wedge \chi]] \equiv [[\varphi \wedge \psi] \wedge \chi]$

(c) $(\exists x)(\exists y)\varphi \equiv (\exists y)(\exists x)\varphi$

(d) $(\exists x)[\varphi \wedge (\exists y)\psi] \equiv (\exists x)(\exists y)[\varphi \wedge \psi]$ – *falls* $y \notin Fr(\varphi)$

(e) $(\exists x)[\varphi \vee \psi] \equiv [(\exists x)\varphi \vee (\exists x)\psi]$ – *auch sonst*

5.5 Indirekte Deutung

Wir sind jetzt in der Lage, die in den vorangehenden Kapiteln gegebene Interpretation natürlichsprachlicher Konstruktionen im Rahmen der indirekten Deutung zu rekonstruieren. Dabei werden wir für jeden natürlichsprachlichen Ausdruck A eine typenlogische Übersetzung $|A|$ angeben, deren semantischer Wert gerade die Extension ist, die er nach der zuvor gegebenen direkten Deutung hat. Nach den schon geleisteten Vorüberlegungen ist das nicht mehr sonderlich schwierig. Wir gehen kompositionell vor und müssen also zunächst einmal typenlogische Entsprechungen für die lexikalischen Ausdrücke angeben. In den meisten Fällen haben diese die Gestalt $c(i)$, was wir ab jetzt durch c_i abkürzen:

(63) $|\textbf{Eike}| = \textbf{e}; |\textbf{Fritz}| = \textbf{f}; |\textbf{Hans}| = \textbf{h}; \ldots$ [allesamt Konstanten des Typs $\boldsymbol{e}$]
$|\textbf{arbeitet}| = \textbf{A}_i; |\textbf{Mann}| = \textbf{M}_i; |\textbf{Frau}| = \textbf{F}_i; |\textbf{Porsche}| = \textbf{P}_i \ldots$
[wobei **A**, **M**, **F**, **P**,... Konstanten des Typs ***s(et)*** sind]
$|\textbf{trifft}| = \textbf{T}_i; |\textbf{besitzt}| = \textbf{B}_i; |\textbf{sieht}| = \textbf{B}_i; |\textbf{knutscht}| = \textbf{K}_i; \ldots$
[wobei **T**, **B**, **S**, **K**,... Konstanten des Typs ***s(e(et))*** sind]

Weitere lexikalische Übersetzungen werden wir bei Bedarf einführen. Dabei gehen wir immer davon aus, dass die Konstanten so gedeutet werden wie die entsprechenden lexikalischen Ausdrücke in der direkten Deutung: der (belegungsunabhängige) semantische Wert von **f** ist das Individuum Fritz; der Wert von **A** weist jeder Situation $s \in LR$ die charakteristische Funktion der Menge der Individuum zu, die in s arbeiten; etc.

Die Übersetzungen der (Satz-) koordinierenden Konjunktionen **und** und **oder** machen von den oben eingeführten *logischen Konstanten* Gebrauch. Erstere entspricht der Konstanten $\wedge$ und wird durch diese übersetzt: für Letztere benutzen wir dagegen eine komplexe Formel:

(64a) $|\textbf{und}| = \wedge$
(b) $|\textbf{oder}| = (\lambda q^t.\ (\lambda p^t.\ \neg [\ \neg\ p \wedge \neg\ q]))$

Wendet man den semantischen Wert von (64b) hintereinander auf zwei Wahrheitswerte q und p an, ergibt sich der Wert 1, wenn es nicht der Fall ist, dass $q = p = 0$ – also gerade dann, wenn mindestens einer der beiden Werte 0 ist. Natürlich hätten wir auch für **oder** eine logische Konstante '$\vee$' einführen und diese dann durch die in (9) in Abschnitt 5.1 definierte Funktion deuten können. Aber einerseits ist die Tatsache, dass man die Disjunktion auf die Konjunktion (und Negation) zurückführen kann, an sich interessant.[103] Andererseits können wir auch die Notation

$[\varphi \vee \psi]$

als Abkürzung für die folgende in (64b) verwendete Formelkombination

$\neg [\ \neg \varphi \wedge \neg \psi]$

auffassen – was wir ab jetzt auch tun werden.

Auch die meisten Determinatoren werden durch komplexe Formeln übersetzt. Zwei Beispiele, die Übersetzungen von **ein**- und **kein**-, haben wir schon in (24) und (25) kennen gelernt. Die anderen liegen angesichts der direkten Deutung aus Abschnitt 3.2 nahe:

(65a) $|\textbf{kein-}| = (\lambda Q^{et}.\ (\lambda P^{et}.\ \neg\ (\exists x^e)\ [Q(x) \wedge P(x)]))$
(b) $|\textbf{ein-}_{indef}| = (\lambda Q^{et}.\ (\lambda P^{et}.\ (\exists x^e)\ [Q(x) \wedge P(x)]))$
(c) $|\textbf{ein-}_{Num}| = (\lambda Q^{et}.\ (\lambda P^{et}.\ (\exists x^e)\ [Q(x) \wedge P(x) \wedge \neg\ (\exists y^e)\ [\neg(x = y) \wedge Q(y) \wedge P(y)]]))$
(d) $|\textbf{jed-}| = (\lambda Q^{et}.\ (\lambda P^{et}.\ \neg\ (\exists x^e)\ [Q(x) \wedge \neg\ P(x)]))$
(e) $|\textbf{die meisten}| = \textbf{MOST}$ [Konstante des Typs ***(et)((et)t)***]
(f) $|\textbf{d-}_{Russell}| = (\lambda Q^{et}.\ (\lambda P^{et}.\ (\exists x^e)\ [Q(x) \wedge \neg\ (\exists y^e)\ [\neg(x = y) \wedge Q(y)] \wedge P(x)]))$

Die Übersetzung (65c) von **ein**- als Zahlwort sieht sehr kompliziert aus, bewirkt aber nur, dass sich nach Kombination mit Substantiv und Prädikat der Wahrheitswert 1 ergibt, wenn der Schnitt ihrer (als Mengen aufgefassten) Extensionen gerade aus einem Individuum x besteht – in denen es also ein x gibt, das sowohl in der Substantiv-Extension als auch in der Prädikatsextension liegt, ohne dass es ein weiteres (= *von x verschiedenes*) Individuum y gibt, das

[103] Die Rückführung, die auch in der anderen Richtung funktioniert, ist bereits in der Antike bekannt gewesen und wird heute als *de Morgansches Gesetz* bezeichnet.

ebenfalls in diesem Schnitt liegt. Auch bei den Übersetzungen (65d) und (65f) überzeugt man sich leicht, dass ihre Werte gerade die in Kapitel 3 gegebenen Deutungen sind. In (65e) dagegen haben wir den Determinator **die meisten** mit einer nicht-logischen Konstanten übersetzt, von deren korrekter Deutung wir wieder ausgehen.

Kommen wir nun zu den komplexen Ausdrücken. Wir folgen in dem Sinne dem Kompositionalitätsprinzip, als wir bei jeder der in den vorangehenden Kapiteln betrachteten grammatischen Konstruktionen angeben, wie sich die typenlogische Übersetzung des Gesamtausdrucks aus den Übersetzungen seiner Teile ergibt. Auf diese Weise ergibt sich dann auch die Bedeutung des Gesamtausdrucks kompositionell aus den Bedeutungen der Übersetzungen seiner Teile. Das erste Beispiel ist die in Kapitel 1 analysierte Satzkoordination, die sich in der indirekten Deutung wie folgt ausnimmt:

(66) *Indirekte Deutung der Satzkoordination*
Wenn S und S' (Aussage-) Sätze sind und K eine koordinierende Konjunktion ist, gilt:
$|S\,K\,S'| = |K|(|S'|)(|S|)$.

Man mache sich klar, wie die Gleichung in (66) zu verstehen ist. Danach erhält man die Übersetzung zweier durch **und** bzw. **oder** (allgemein: durch eine Konjunktion K) koordinierter Sätze S und S' – also die Übersetzung von ‘ $S\,K\,S'$ ’; also: $|S\,K\,S'|$ – indem man zunächst die Konjunktion selbst in die Typenlogik übersetzt (das ist dann irgendeine Formel $|K|$) und dahinter jeweils in (fette) Klammern die Übersetzungen der Teile – also entsprechende Formeln $|S'|$ und $|S|$ – folgen lässt. Da $|S|$ und $|S'|$ selbst wieder Formeln des Typs $\boldsymbol{t}$ sind und die Konjunktion K durch eine Formel des Typs $\boldsymbol{t(tt)}$ übersetzt wird, erhält man insgesamt nach (66) eine Formel des Typs $\boldsymbol{t}$. Bevor wir die Regel auf ein konkretes Beispiel anwenden, übersetzen wir noch die Konstruktionen aus Kapitel 2:

(67) *Indirekte Deutung der Prädikation*
Wenn S ein Satz ist, an dessen Subjektstelle ein Eigenname NN steht und dessen Prädikat P ist, dann gilt:
$|S| = |P|(|NN|)$.

(68) *Indirekte Deutung der Anbindung von Eigennamen als Objekten*
Wenn VP ein Prädikat bestehend aus einem Verb V und einem Eigennamen NN als Objekt ist, dann gilt:
$|VP| = |V|(|NN|)$.

Testen wir diese Regeln anhand eines Beispiels:

(69) **Eike arbeitet und Fritz trifft Hans.**

Die Übersetzung folgt der offenkundigen Konstituentenstruktur des Satzes:

(70)
$|\textbf{Eike arbeitet und Fritz trifft Hans}|$ nach(66)
$= |\textbf{und}|(|\textbf{Fritz trifft Hans}|)(|\textbf{Eike arbeitet}|)$ nach (67) [2 x]
$= |\textbf{und}|(|\textbf{trifft Hans}|(|\textbf{Fritz}|))(|\textbf{arbeitet}|(|\textbf{Eike}|))$ nach (68)
$= |\textbf{und}|(|\textbf{trifft}|(|\textbf{Hans}|)(|\textbf{Fritz}|))\,(|\textbf{arbeitet}|(|\textbf{Eike}|))$ nach (63) und (64a)
$= \boldsymbol{\wedge}\,(\mathbf{T}_i(\mathbf{h})(\mathbf{f}))(\mathbf{A}_i(\mathbf{e}))$

Die resultierende Formel lässt sich mithilfe der für die Konjunktion eingeführten Notation umschreiben als $\mathbf{[A}_i\mathbf{(e)} \wedge \mathbf{T}_i\mathbf{(h)(f)]}$; und mit einer ähnlichen Konvention bringt man das rechte Konjunkt in die Oberflächenreihenfolge der natürlichen Sprache und erhält so:

(71) $\mathbf{[A}_i\mathbf{(e)} \wedge \mathbf{T}_i\mathbf{(f,h)]}$

Man beachte, dass es sich bei der Reformulierung (71) der in (70) berechneten Übersetzung von (69) lediglich um eine notationelle Variation handelt. Die typenlogische Formel ist haargenau dieselbe, sie wird nur anders dargestellt. Der Übergang stellt also keine logische Reduktion aufgrund irgendwelcher Rechenregeln dar. In der Tat: diese Formel lässt sich durch logische Umformung allenfalls verschlimmbessern. Bei der folgenden Variante liegt der Fall dagegen anders:

(72) **Eike arbeitet oder Fritz trifft Hans.**

Die Übersetzung funktioniert natürlich genauso wie in (70) – mit Ausnahme der letzten Zeile, wo statt der logischen Konstanten $\wedge$ die in (64b) gegebene komplexe Übersetzung von **oder** hereinkommt:

(73) $|\textbf{Eike arbeitet oder Fritz trifft Hans}|$ nach (66)
= $|\textbf{oder}|\,(|\textbf{Fritz trifft Hans}|)\,(|\textbf{Eike arbeitet}|)$ nach (67) [2 x]
= $|\textbf{oder}|\,(|\textbf{trifft Hans}|(|\textbf{Fritz}|))\,(|\textbf{arbeitet}|(|\textbf{Eike}|))$ nach (68)
= $|\textbf{oder}|\,(|\textbf{trifft}|(|\textbf{Hans}|)\,(|\textbf{Fritz}|))\,(|\textbf{arbeitet}|\,(|\textbf{Eike}|))$ nach (63) und (64b)
= $(\lambda q.\,(\lambda p.\,[\,p \vee q\,]))\,(\mathbf{T}_i(\mathbf{h})(\mathbf{f}))\,(\mathbf{A}_i(\mathbf{e}))$

Auch hier können wie die soeben eingeführte Konvention anwenden, um die Übersetzung des zweiten Teilsatzes etwas aufzumotzen – aber viel bringt das nicht:

(74) $(\lambda q.\,(\lambda p.\,[\,p \vee q\,]))\,(\mathbf{T}_i(\mathbf{f},\mathbf{h}))\,(\mathbf{A}_i(\mathbf{e}))$

Doch natürlich lässt sich diese Formel weiter reduzieren. Zwar ist die Gesamtformel nicht von der in (58) verlangten Gestalt; denn es handelt sich um eine Applikation $\gamma(\delta)$, wobei aber γ kein λ-Term ist, sondern selbst wieder eine Applikation. Aber dieser Teil γ erfüllt ganz offenkundig eine für die Lambda-Konversion einschlägige Konstellation: er hat die Gestalt '$(\lambda x.\alpha)(\beta)$' – α ist der Teil ab '$(\lambda q.$… ' und bis ausschließlich '$(\mathbf{T(f,h)})$', x ist q, und β ist gerade das Argument $\mathbf{T(f,h)}$. Da außerdem dieses β gar keine Variablen enthält – **T**, **h** und **f** sind nicht-logische Konstanten –, ist natürlich q frei für β in α, und die Konversion kann losgehen:

(75) $(\lambda p.\,[\,p \vee \mathbf{T}_i(\mathbf{f},\mathbf{h})])\,(\mathbf{A}_i(\mathbf{e}))$

Und das ist natürlich selbst wieder eine einschlägige Konstellation für eine weitere Konversion:

(76) $[\mathbf{A}_i(\mathbf{e}) \vee \mathbf{T}_i(\mathbf{f},\mathbf{h})]$

(76) ist eine (logisch) *reduzierte* Übersetzung von (72), d.h. eine Formel, die aufgrund irgendwelcher Rechenregeln der eigentlichen Übersetzung in (73) logisch äquivalent ist. Die Reduktion von (74) auf (76) ist aus zwei Gründen relativ einfach. Zum Einen gab es hier jeweils nur eine Möglichkeit, das Gesetz der Lambda-Konversion einzusetzen; sowohl in der Übersetzung (74) selbst als auch in der ersten Reduktion (75) lag die einschlägige Konstellation nur einmal vor. Zum Anderen war die Variablenbedingung in beiden Fällen trivialerweise erfüllt, da das einzusetzende Argument ohnehin keine Variablen enthielt. Beide Umstände sind nicht unbedingt der Regelfall, wie wir sehen werden, sobald wir die verbleibenden Konstruktionen übersetzen und miteinander interagieren lassen. Diese Konstruktionen führen quantifizierende Nominale an Subjekt- und Objektstelle ein. Bei der Subjektquantifikation kehren sich gegenüber der in (67) übersetzten Prädikation lediglich Funktion und Argument um:

(77) *Indirekte Deutung der Quantifikation*
Wenn S ein Satz ist, an dessen Subjektstelle eine quantifizierende Nominalphrase QN steht und dessen Prädikat VP ist, dann gilt:
$|S| = |QN|(|VP|)$.

Und auch im Fall von Quantoren an Objekstelle lässt sich die in Abschnitt 3.4 gegebene Deutung unmittelbar in typenlogische Formeln übertragen:

(78) *Indirekte Deutung der Anbindung quantifizierender Nominalphrasen als Objekte*
Wenn *VP* ein Prädikat ist, bestehend aus einem Verb *V* und einer quantifizierenden Nominalphrase *QN* als Objekt, dann gilt:[104]

$$|VP| = \boldsymbol{\lambda x^e.}\ |QN|\,(\boldsymbol{(\lambda y^e.}\ |V|\,\boldsymbol{(y)(x)))}$$

Wir setzen (77) und (78) gleich auf ein Beispiel an, was uns Gelegenheit geben wird, das Zusammenwirken von gebundener Umbenennung und Lambda-Konversion zu illustrieren. Zuvor müssen wir noch die Übersetzung von nicht-lexikalischen Nominalphrasen angeben. In Kapitel 3 hatten wir nicht eigens eine Regel dafür angegeben, weil sich diese implizit aus dem Umstand ergab, dass die Extensionen der Determinatoren per Differenz ermittelt wurden. Als Extensionskombination für die Konstruktion der quantifizierenden Nominale kommt damit nur die Funktionalapplikation in Frage, womit wir die folgende Übersetzungsregel erhalten:

(79) *Indirekte Deutung der quantifizierenden Nominalphrasen*
Wenn *QN* eine quantifizierende Nominalphrase ist, bestehend aus einem Determinator *D* und Nomen *N*, dann gilt:

$$|QN| = |D|\,(|N|)$$

Mit (79) haben wir den gesamten Bestand der in den Kapiteln 1–3 erarbeiteten Interpretation in die Methodologie der indirekten Deutung übertragen. Betrachten wir nun ein etwas komplexeres Beispiel:

(80) **Jeder Mann trifft eine Frau.**

Zunächst liefern die einschlägigen Übersetzungsregeln:

(81)

	$\boldsymbol{\lvert}$**Jeder Mann trifft eine Frau**$\boldsymbol{\rvert}$	nach(77)
=	$\boldsymbol{\lvert}$**jeder Mann**$\boldsymbol{\rvert}$ $\boldsymbol{(\lvert}$**trifft eine Frau**$\boldsymbol{\rvert)}$	nach (79)
=	$\boldsymbol{\lvert}$**jeder**$\boldsymbol{\rvert(\lvert}$**Mann**$\boldsymbol{\rvert)}$ $\boldsymbol{(\lvert}$**trifft eine Frau**$\boldsymbol{\rvert)}$	nach (78)
=	$\boldsymbol{\lvert}$**jeder**$\boldsymbol{\rvert(\lvert}$**Mann**$\boldsymbol{\rvert)}$ $\boldsymbol{(\lambda x.\ \lvert}$**eine Frau**$\boldsymbol{\rvert(\lambda y.\ \lvert}$**trifft**$\boldsymbol{\rvert(x,y)))}$	nach (79)
=	$\boldsymbol{\lvert}$**jeder**$\boldsymbol{\rvert(\lvert}$**Mann**$\boldsymbol{\rvert)}$ $\boldsymbol{(\lambda x.\ \lvert}$ **eine**$\boldsymbol{\rvert}$ $\boldsymbol{(\lvert}$**Frau**$\boldsymbol{\rvert)}$ $\boldsymbol{(\lambda y.\ \lvert}$**trifft**$\boldsymbol{\rvert(x,y)))}$	nach (63) [3x]
=	$\boldsymbol{\lvert}$**jeder**$\boldsymbol{\rvert(M_i)}$ $\boldsymbol{(\lambda x.\ \lvert}$ **eine**$\boldsymbol{\rvert}$ $\boldsymbol{(F_i)}$ $\boldsymbol{(\lambda y.\ T_i(x,y)))}$	nach (65d)
=	$\boldsymbol{(\lambda Q.\ (\lambda P.\ \neg\ (\exists x)\ [Q(x) \wedge \neg\ P(x)]))}$ $\boldsymbol{(M_i)}$ $\boldsymbol{(\lambda x.\ \lvert}$ **eine**$\boldsymbol{\rvert}$ $\boldsymbol{(F_i)}$ $\boldsymbol{(\lambda y.\ T_i(x,y)))}$	nach (65a)
=	$\boldsymbol{(\lambda Q.\ (\lambda P.\ \neg\ (\exists x)\ [Q(x) \wedge \neg\ P(x)]))}$ $\boldsymbol{(M_i)}$ $\boldsymbol{((\lambda x.\ (\lambda Q.\ (\lambda P.\ (\exists x)\ [Q(x) \wedge P(x)]))}$ $\boldsymbol{(F_i)}$ $\boldsymbol{(\lambda y.\ T_i(x,y))))}$	

Die in (81) ermittelte typenlogische Übersetzung von (80) enthält die für die **λ**-Konversion einschlägige Konstellation '**(λx.α) (β)**' zweimal. Zum Einen hat die Gesamtformel die Gestalt '**(λQ.α)(β)(γ)**'; hier lässt sich die Konversion auf dem unterstrichenen Teil durchführen, denn **β** ist die Formel **M(*i*)**, in der ***i*** als einzige freie Variable vorkommt, die aber im **α**-Teil nirgends gebunden wird. Zum Anderen ist das Argument **γ** selbst von der Form '**(λx. (λQ.α') (β') (γ))**', wobei die Konversionsregel beim doppelt unterstrichenen Teil greift – und wieder anwendbar ist, weil **β'** die Formel **F*i*** ist und ***i*** in **α'** nicht gebunden wird. Reduziert man die einfach unterstrichene Formel, ergibt sich (82a); führt man die Konversion auf dem doppelt unterstrichenen Teil aus, bekommt man (82b):

104 Die doppelten Klammern um den mit '**λx.**' beginnenden Teil ergeben sich aus der Syntax der Typenlogik: der Lambda-Term beginnt und endet nach (21*Abs*) mit einer Klammer; und als Argument von '*QN*' muss er nach (21*App*) noch einmal mit Klammern umschlossen werden. Aber meistens lassen wir diese Doppelklammerungen weg.

(82a) $(\lambda P.\ \neg\ (\exists x)\ [\mathbf{M}_i(x) \wedge \neg\ P(x)])$
$((\lambda x.\ (\lambda Q.\ (\lambda P.\ (\exists x)\ [Q(x) \wedge P(x)]))\ (\mathbf{F}i)\ (\lambda y.\ \mathbf{T}i(x,y))))$

(b) $(\lambda Q.\ (\lambda P.\ \neg\ (\exists x)\ [Q(x) \wedge \neg\ P(x)]))\ \ (\mathbf{M}_i)$
$((\lambda x.\ (\lambda P.\ (\exists x)\ [\mathbf{F}_i(x) \wedge P(x)])\ (\lambda y.\ \mathbf{T}i(x,y))))$

(82a) ist insgesamt von der Form '**(λ*x*. α) (β)**'. Des Weiteren enthält das Argument (wie man leicht nachprüft) keine freie Variable außer ***i***, und ***i*** wird wieder nirgends gebunden; das Gleiche gilt für die erste Zeile von (82b), bei der das Argument die Übersetzung $\mathbf{M}_i$ von **Mann** ist. Also lässt sich wieder die Lambda-Konversion anwenden, und wir erhalten in beiden Fällen:

(83) $(\lambda P.\ \neg\ (\exists x)\ [\mathbf{M}_i(x) \wedge \neg\ P(x)])\ ((\lambda x.\ (\lambda P.\ (\exists x)\ [\mathbf{F}_i(x) \wedge P(x)])\ \ (\lambda y.\ \mathbf{T}i(x,y))))$

Dass das Ergebnis jedes Mal dasselbe ist, darf nicht verwundern; die alternativen Konversionen, die auf das Endergebnis von (81) angewandt wurden, spielten sich in zwei separaten Teilen der Formel ab: die Gesamtformel hatte die Gestalt '**(λ*x*. α) (β) (γ)**', und einmal haben wir zuerst den unterstrichenen Teil reduziert und danach das Argument **γ**, beim anderen Mal sind wir umgekehrt vorgegangen. Am Ergebnis (83) ändert das natürlich nichts.

(83) lässt sich wieder auf zwei alternative Weisen reduzieren. Zunächst hat die Gesamtformel die Gestalt '**(λ*P*. α) (β)**', wobei das Argument **β** wieder keine freien Variablen außer dem notorisch ungebundenen ***i*** enthält. Die Anwendung der Lambda-Konversion ergibt dafür:

(84) $\neg\ (\exists x)\ [\mathbf{M}_i(x) \wedge \neg\ \underline{(\lambda x.\ (\lambda P.(\exists x)\ [\mathbf{F}_i(x) \wedge P(x)])(\lambda y.\mathbf{T}_i(x,y)))}\ (x)]$

Der Übersichtlichkeit halber haben wir dabei die Stelle, an die das ursprüngliche Argument **β** für die Variable ***P*** eingesetzt wurde, unterstrichen. Man sieht deutlich, dass dieser unterstrichene Teil selbst wieder von der Form '**(λ*x*. α)** ' ist und ihm das Argument ***x*** unmittelbar folgt. Es liegt also die für die Eigen-Konversion (58*) einschlägige Konstellation vor, die auf eine Streichung des Präfixes '**λ*x*.**' und des Arguments '**(*x*)**' hinausläuft:

(85) $\neg\ (\exists x)\ [\mathbf{M}_i(x) \wedge \neg\ (\lambda P.(\exists x)\ [\mathbf{F}_i(x) \wedge P(x)])(\lambda y.\mathbf{T}_i(x,y)))]$

Bevor wir weiter machen, sei darauf hingewiesen, dass die Variablenbedingung nur die Freiheit der Variablen *in* **β** betrifft. Die Tatsache, dass die Variable ***x*** in der Gesamtformel (84) – wie alle Variablen in dieser Formel – gebunden ist, spielt keine Rolle. Denn diese Bindung geschieht von außen (durch den Existenzquantor ganz links[105]), und durch die Bedingung soll verhindert werden, dass ein anderer Operator interveniert und damit die Bindungsbeziehung zwischen dem Existenzquantor und der Variablen zerstört. Aber diese Bindungsbeziehung besteht gerade, weil ***x*** in **β** – und bei Bestehen der Bedingung dann auch in $\alpha[^{x}/_{\beta}]$ – frei ist. Der Punkt wird klar, wenn wir versuchen, (85) weiter zu reduzieren. Denn auch diese Formel enthält wieder eine potenziell einschlägige Konstellation, nämlich den Teil:

(86) $(\lambda P.\ (\exists x)\ [\mathbf{F}_i(x) \wedge P(x)])(\lambda y.\mathbf{T}_i(x,y))$

Das Argument **β** in (86) enthält diesmal drei Variablen, von denen zwei, nämlich ***x*** und ***i***, frei sind. ***i*** bereitet wieder kein Problem, aber ***x*** würde gebunden, wenn man **β** für ***P*** einsetzt: ***P*** steht im **α**-Teil von (86) im Skopus von '**(∃*x*)**' und geriete somit in den Skopus des Existenzquantors. Die Lambda-Konversion ist also auf (86) bzw. (85) nicht anwendbar. Stattdessen müssen wir zuerst eine gebundene Umbenennung vornehmen, um auf diese Weise die Voraussetzung für die Anwendung einer Lambda-Konversion zu schaffen. Da die Variable ***x*** die

[105] Genauer gesagt bindet das in der Notation '**(∃*x*)** …' versteckte Lambda. Aber wir werden die in der Prädikatenlogik übliche (und dort gerechtfertigte) Redeweise von der Bindung der Variablen durch Quantoren übernehmen.

Variablenbedingung verletzt hat, müssen wir also in (86) bzw. (85) die Bindung '$(\exists x)$' so ändern, dass die Bedingung erfüllt ist. Wie bereits erwähnt, kann man dies erreichen, indem man durch eine Variable (gleichen Typs) umbenennt, die nirgends in der Gesamtformel vorkommt – sagen wir einmal: z. Aus (86) wird dann:

(87) $(\lambda P.\ (\exists z)\ [\mathbf{F}_i(z) \wedge P(z)])(\lambda y.\mathbf{T}_i(x,y))$

Jetzt ist offensichtlich die Variablenbedingung erfüllt – das war der Zweck der Übung; denn die in β freie Variable x bleibt nach Anwendung der Lambda-Konversion frei:

(88) $(\exists z)\ [\mathbf{F}_i(z) \wedge (\lambda y.\mathbf{T}_i(x,y))(z)]$

Bevor wir (88) in die Gesamtformel (85) einsetzen, reduzieren wir noch einmal, was diesmal ohne Problem möglich ist: die einzige einschlägige Konstellation enthält nämlich im α-Teil keinen bindenen Operator. Wir erhalten so:

(89) $(\exists z)\ [\mathbf{F}_i(z) \wedge \mathbf{T}_i(x,z)]$

(86) lässt sich also auf (89) reduzieren, und da (86) ein Teil der reduzierten Übersetzung (85) war, kann man ihn auch dort durch das äquivalente, aber kürzere (90) ersetzen:

(90) $\neg\ (\exists x)\ [\mathbf{M}_i(x) \wedge \neg\ (\lambda P.\ (\exists x)\ [\mathbf{F}_i(x) \wedge P(x)])(\lambda y.\mathbf{T}_i(x,y))]$ [= (85)]
$\equiv$ $\neg\ (\exists x)\ [\mathbf{M}_i(x) \wedge \neg\ (\exists z)\ [\mathbf{F}_i(z) \wedge \mathbf{T}_i(x,z)]]$

Beim Übergang von (84) zu (85) hatten wir den hervorgehobenen Lambda-Term reduziert. Doch es gab noch eine weitere potenziell einschlägige Konstellation in dieser Formel, nämlich (86), die ja auch ein Teil von (84) ist. Wir hätten also die soeben vorgeführte Reduktion von (86) ebensogut innerhalb von (85) vornehmen können:

(91) $\neg\ (\exists x)\ [\mathbf{M}_i(x) \wedge \neg\ (\lambda x.\ (\lambda P.\ (\exists x)\ [\mathbf{F}_i(x) \wedge P(x)])(\lambda y.\mathbf{T}_i(x,y)))\ (x)]$ [= (85)]
$\equiv$ $\neg\ (\exists x)\ [\mathbf{M}_i(x) \wedge \neg\ (\lambda x.\ (\lambda P.\ (\exists z)\ [\mathbf{F}_i(z) \wedge P(z)])(\lambda y.\mathbf{T}_i(x,y)))\ (x)]$
$\equiv$ $\neg\ (\exists x)\ [\mathbf{M}_i(x) \wedge \neg\ (\lambda x.\ (\exists z)\ [\mathbf{F}_i(z) \wedge (\lambda y.\mathbf{T}_i(x,y))(z)])\ (x)]$
$\equiv$ $\neg\ (\exists x)\ [\mathbf{M}_i(x) \wedge \neg\ \underline{(\lambda x.\ (\exists z)\ [\mathbf{F}_i(z) \wedge \mathbf{T}_i(x,z)])\ (x)}]$

Ähnlich wie beim Übergang von (84) zu (85) lässt sich an der unterstrichenen Stelle eine Eigen-Konversion durchführen:

(92) $\neg\ (\exists x)\ [\mathbf{M}_i(x) \wedge \neg\ (\lambda x.\ (\exists z)\ [\mathbf{F}_i(z) \wedge \mathbf{T}_i(z,y)])\ (x)]$
$\equiv$ $\neg\ (\exists x)\ [\mathbf{M}_i(x) \wedge \neg\ (\exists z)\ [\mathbf{F}_i(z) \wedge \mathbf{T}_i(x,z)]]$

Ebenso gut hätten wir die letzten beiden Übergänge – Reduktion von 'λx' und 'λy' vertauschen können – wieder mit demselben Ergebnis. Interessanterweise führt die für (83) alternative Reduktion auch zu diesem Ziel. Diese Alternative besteht darin, statt der Gesamtformel den Teil (86) zu reduzieren, der ebenfalls in (83) auftritt. Wir überlassen diese Alternative einer Übungsaufgabe und halten nur fest, dass alle Reduktionen auf dasselbe Ergebnis hinauslaufen. Das ist kein Zufall, sondern ergibt sich aus einer allgemeinen Eigenschaft der Typenlogik, auf die wir hier nicht näher eingehen können. Die Eigenschaft heißt *starke Normalisierbarkeit* und besagt, dass zum einen jede typenlogische Formel auf eine sog. *Normalform* reduziert werden kann, und zum anderen jeweils zwei Normalformen *alphabetische Varianten* voneinander sind. Dabei ist eine Normalform eine Formel, die keine für die Lambda-Konversion (potenziell) einschlägige Konstellation '$(\lambda x.\alpha)\ (\beta)$' enthält; und alphabetische Varianten sind Formeln, die durch (eventuell mehrfache) gebundene Umbenennung ineinander überführt werden können. Der Nachweis der starken Normalisierbarkeit ist sehr komplex und würde hier zu weit führen.[106] Für die indirekte Deutung ist sie nur insofern von Interesse, als wir mit ihr sicher gehen

können, dass die genaue Vorgehensweise bei der Reduktion typenlogischer Formeln unwichtig ist: solange man keine potenziell einschlägige Konstellation der Form '$(\lambda x.\alpha)\ (\beta)$' außer Acht lässt und nötigenfalls gebunden umbenennt, führen ohnehin alle Wege nach Rom – sprich: zur Normalform.

Bevor wir uns abschließend den intensionalen Konstruktionen im Rahmen der indirekten Deutung zuwenden, sei noch auf ein Detail beim Umbenennen gebundener Variablen hingewiesen. Nehmen wir dazu noch einmal den ersten Schritt der obigen Reduktion (90) unter die Lupe:

(93) $\neg\ (\exists x)\ [\mathbf{M}_i(x) \wedge \neg\ \underline{(\lambda P.\ (\exists x)\ [\mathbf{F}_i(x) \wedge P(x)])\ (\lambda y.\mathbf{T}_i(x,y))}\ ]$ [= (85)]

$\equiv$ $\neg\ (\exists x)\ [\mathbf{M}_i(x) \wedge \neg\ \underline{(\lambda P.\ (\exists z)\ [\mathbf{F}_i(z) \wedge P(z)])(\lambda y.\mathbf{T}_i(x,y))}\ ]$

$\equiv$ $\neg\ (\exists x)\ [\mathbf{M}_i(x) \wedge \neg\ (\exists z)\ [\mathbf{F}_i(z) \wedge \mathbf{T}_i(x,z)]]$

Um hier die Lambda-Konversion an der markierten Stelle durchführen zu können, hatten wir zunächst im α-Teil eine gebundene Umbenennung vorgenommen. Stattdessen hätten wir auch wie folgt vorgehen können:

(94) $\neg\ \underline{(\exists x)}\ [\mathbf{M}_i(\underline{x}) \wedge \neg\ (\lambda P.\ (\exists x)\ [\mathbf{F}_i(x) \wedge P(x)])\ (\lambda y.\mathbf{T}_i(\underline{x},y))]$ [= (85)]

$\equiv$ $\neg\ \underline{(\exists z)}\ [\mathbf{M}_i(\underline{z}) \wedge \neg\ (\lambda P.\ (\exists x)\ [\mathbf{F}_i(x) \wedge P(x)])\ (\lambda y.\mathbf{T}_i(\underline{z},y))\ ]$

$\equiv$ $\neg\ (\exists z)\ [\mathbf{M}_i(z) \wedge \neg\ (\exists x)\ [\mathbf{F}_i(x) \wedge \mathbf{T}_i(z,x)]]$

Auch in (94) ist der erste Übergang eine gebundene Umbenennung: die in der ersten Zeile markierte Bindung der Variablen x ist durch eine entsprechende z-Bindung ersetzt worden; da z nirgendwo in der ursprünglichen Formel (85) vorkam, sind in diesem Fall natürlich auch die Voraussetzungen des Prinzips (56) der gebundenen Umbenennung erfüllt. Und mit diesem Übergang sind auch die Voraussetzungen für eine Anwendung der Lambda-Konversion erfüllt; denn die im β-Teil freie Variable x ist durch die Umbenennung verschwunden, und das dafür eingesetzte z ist ja gerade so gewählt worden, dass es im α-Teil nicht vorkommt. Insofern sind (93) und (94) vollkommen parallel; und die Ergebnisse sind in der Tat alphabetische Varianten voneinander. Es gibt aber einen wesentlichen Unterschied zwischen den beiden Vorgehensweisen. Er besteht darin, dass in (93) im α-Teil umbenannt wurde, in (94) dagegen im β-Teil. Beide Vorgehensweisen sind *in diesem Fall* durchaus legitim. Dennoch ist zu bedenken, dass der zweite, in (94) eingeschlagene Weg nur funktioniert, wenn die fragliche Variable – in diesem Fall x – in der Gesamtformel, also oberhalb des zu reduzierenden Teils, gebunden wird, wie dies in (94) durch den Existenzquantor '$(\exists x)$' geschieht. Bleibt dagegen die Variable in der Gesamtformel frei, lässt sich die entsprechende Umbenennung nicht vornehmen – denn umbenannt werden dürfen immer nur *gebundene* Variablen.[107] Die in (93) eingeschlagene Strategie der Umbenennung im α-Teil ist dagegen immer anwendbar. Das liegt ganz einfach daran, dass ein Verstoß gegen die Variablenbedingung der Lambda-Konversion immer von einem Konflikt zwischen einer freien Variablen im β-Teil und einer Bindung im α-Teil herrührt – und eine Umbenennung ebendieser Bindung den Konflikt löst. Ein Beispiel für eine Formel, bei der die Umbenennung im α-Teil erfolgen *muss*, liefert die weiter oben betrachtete Teilformel (86), innerhalb derer wir die Reduktion vorgenommen hatten. Eine Umbenennung (86*) im β-Teil wäre in diesem Fall falsch gewesen und hätte nach der Einsetzung in (85) zu der offenen Gesamtformel (85*) geführt, die offenkundig nicht zur Ausgangsformel (85) äquivalent ist:

---

106 Ein einschlägiges Lehrbuch ist *Introduction to combinators and lambda-calculus* (1986) von J. Roger Hindley und Jonathan Paul Seldin.

107 Das ist kein willkürliches Verbot, sondern wieder ein Reflex der in Abschnitt 5.3 gegebenen Deutung der Typenlogik; denn die Ersetzung freier Variablen erhält im allgemeinen nicht den semantischen Wert.

(85) $\neg\ (\exists x)\ [\mathbf{M}_i(x) \wedge \neg\ (\lambda P.\ (\exists x)\ [\mathbf{F}_i(x) \wedge P(x)])(\lambda y.\mathbf{T}_i(x,y))]$

(85*) $\neg\ (\exists x)\ [\mathbf{M}_i(x) \wedge \neg\ (\exists x)\ [\mathbf{F}_i(x) \wedge (\lambda y.\mathbf{T}_i(\underline{z},y))(x)]]$

(86) $(\lambda P.\ (\exists x)\ [\mathbf{F}_i(x) \wedge P(x)])(\lambda y.\mathbf{T}_i(x,y))$

(86*) $(\lambda P.\ (\exists x)\ [\mathbf{F}_i(x) \wedge P(x)])(\lambda y.\mathbf{T}_i(\underline{z},y))$

In (85*) bleibt die unterstrichene Variable frei, anstatt von dem äußersten Existenzquantor gebunden zu werden. Um ein derartiges Malheur zu vermeiden, wird dringendst dazu geraten, gebundene Umbenennungen zur Vorbereitung von Lambda-Konversionen grundsätzlich nur im α-Teil vorzunehmen.

Die in der Übersetzung von (80) auftretende Konstellation $\neg\ (\exists x)\ [\varphi \wedge \neg\ \psi]$ ist so häufig, dass sie eine eigene Notation verdient. Insgesamt drückt sie aus, dass kein Ding φ erfüllt, ohne auch ψ zu erfüllen – dass also *jedes* Ding, das φ erfüllt, auch ψ erfüllt.[108] In der Prädikatenlogik ist es üblich, diesen Sachverhalt durch eine Kombination zweier logischer Konstanten zum Ausdruck zu bringen. Zum Einen verwendet man den in (95) als Abkürzung definierten *Allquantor*, der zum Ausdruck bringt, dass eine Formel von jedem Indivuum erfüllt wird. Ähnlich wie den Existenzquantor kann man den Allquantor als Formel des Typs ***(et)t*** auffassen:

(95a) $\forall$ ist (eine Abkürzug für) die Formel:
$(\lambda P^{et}.\ \neg\ (\exists x^{e})\ \neg\ P(x))$
des Typs ***(et)t***.

(b) Wenn φ eine Formel des Typs ***t*** ist und ***x*** eine Variable des Typs ***e***, dann ist
$(\forall x)\ \varphi$
(eine Abkürzug für) die Formel:
$\forall((\lambda x.\ \varphi))$.

Wie man (in einer Übungsaufgabe) leicht nachprüft, gilt nach (95) für alle Belegungen *g*:

(96) $[\![\ (\forall x)\ \varphi\ ]\!]^{g} = 1$ gdw. für jedes Individuum *u* gilt: $[\![\ \varphi\ ]\!]^{g[x/u]} = 1$.

(96) rechtfertigt die in der Prädikatenlogik übliche Lesung von '$(\forall x)$ …' als 'für alle *x* gilt' oder 'für jedes *x* gilt:…'. Ähnlich wie im Fall der Disjunktion hätten wir den Allquantor $\forall$ statt als Abkürzung als logische Konstante (des Typ ***(et)t***) einführen können; (96) wäre dann eine unmittelbare Konsequenz der Deutung dieser Konstanten. In jedem Fall lässt sich die zur Debatte stehende Konstellation mit Hilfe des Allquantors äquivalent umschreiben:

(97) $\neg\ (\exists x)\ [\varphi \wedge \neg\ \psi]$
$\equiv$ $\neg\ (\exists x)\ \neg\ \neg\ [\varphi \wedge \neg\ \psi]$
$\equiv$ $(\forall x)\ \neg\ [\varphi \wedge \neg\ \psi]$

Der erste Übergang in (97) basiert auf dem aussagenlogischen Gesetz der *doppelten Negation*, nach dem Formeln der Gestalt $\neg\ \neg\ \varphi$ äquivalent sind zu φ. Das Gesetz ist so trivial, dass wir für den Nachweis nicht einmal eine Übungsaufgabe ansetzen müssen.

Auch für die in (97) verbleibende Konstellation gibt es eine eigene und gängige Abkürzung:

---

108 Der hier informell verwendete Begriff der *Erfüllung* stammt wieder aus der Prädikatenlogik und muss strenggenommen auf eine Belegunge und eine gebundene Variable relativiert werden: ein Individuum *u* erfüllt eine Formel φ *an der Stelle* ***x*** *relativ zu* einer Belegung *g*, falls φ wahr ist, wenn ***x*** für *u* steht – wenn also φ unter der modifizierten Belegung $g[x/u]$ den Wert 1 hat. Im vorliegenden Fall ist ***x*** die einzige freie Variable, so dass wir uns hier nicht so umständlich ausdrücken müssen.

(98) Wenn **φ** und **ψ** Formeln des Typs ***t*** sind, dann ist
[φ → ψ]
(eine Abkürzug für) die Formel:
¬ [φ ∧ ¬ ψ] .

Aus (98) folgt unmittelbar:

(99) **[φ → ψ]**
≡ **¬ [φ ∧ ¬ ψ]**
≡ **[¬ φ ∨ ψ]**

Neben dem Gesetz der doppelten Negation geht in (99) nur die Abkürzungskonvention (64b) für die Disjunktion **∨** ein. Natürlich hätten wir auch für **→** eine eigene Konstante (des Typs ***t(tt)***) einführen können.

Nach (99) besagt eine Formel der Gestalt **[φ → ψ]** , dass entweder **φ** wahr ist oder aber – also *andernfalls* – **ψ** falsch. Diese Beobachtung rechtfertigt *bis zu einem gewissen Grad* die in der Aussagen- und Prädikatenlogik gängige Paraphrase 'wenn **φ**, dann **ψ**'. Der Pfeil **→** bzw. die Wahrheitstafel, für die er steht, wird dabei auch als *materiale Implikation* (oder *Subjunktion*) bezeichnet. Als Semantik des Konditionalsatzes – also der Konstruktion **Wenn ..., [dann] ...** – greift die Deutung im Sinne der in (98) definierten Konstellation allerdings zu kurz, wie wir in Kapitel 10 feststellen werden.

Mit (95) und (98) lassen sich Formeln der Gestalt **¬ (∃*x*) [φ ∧ ¬ ψ]** als **(∀*x*) [φ → ψ]** umschreiben. Wie die oben vorgeführte Übersetzung von (80) zeigt, entsprechen solche Formeln Sätzen der Form [[**Jed-** *N*] *VP*]. Allerdings entspricht dabei – trotz der Bezeichnung 'Allquantor' – das prädikatenlogische Symbol **∀** nicht dem Determinator **jed-**. Denn wie wir in Kapitel 3 gesehen haben, stellt Letzterer (genauer: seine Extension) eine Beziehung zwischen zwei Mengen (bzw. ihren charakeristischen Funktionen) her – die Teilmengenbeziehung. Der Allquantor dagegen macht eine Aussage über eine einzige Menge – dass nämlich jedes Individuum Element dieser Menge ist. Der Unterschied zwischen Allquantor und Determinator schlägt sich entsprechend in ihrem Typ nieder: während die Extension von **jed-** vom Typ ***(et)((et)t)*** ist, handelt es sich beim Allquantor um eine Formel des Typs ***(et)t***. Die Extension von **jed-** ist in dem Sinne *zweistellig*, als sie *zweier* Prädikatsextensionen bedarf, um einen Wahrheitswert zu liefern; der Allquantor ist dagegen in dem Sinn *einstellig*, als er bei Anwendung auf *eine* Prädikatsextension einen Wahrheitswert ergibt. Dennoch lässt sich, wie die obigen Betrachtungen zeigen, die durch den Determinator **jed-** ausgedrückte Teilmengenbeziehung mit Hilfe des einstelligen Allquantors ausdrücken, indem zunächst die beiden in Beziehung zu setzenden Prädikatsextensionen mit Hilfe der materialen Implikation **→** kombiniert werden:

(100) $[\![$ **jed-** *N VP* $]\!]^s$
= $[\![|$ **jed-** *N VP* $|]\!]^g$
= $[\![$ **(∀*x*^{e}) [**$|$ *N* $|$**(*x*) →** $|$ VP $|$**(*x*)]** $]\!]^g$
= $[\![$ **∀** $]\!]^g([\![$ **(λ*x*^{e} [**$|$ *N* $|$**(*x*) →** $|$ VP $|$**(*x*)])** $]\!]^g)$
= $[\![$ **∀** $]\!]^g[\![$ **(λ*x*^{e} [¬$|$ *N* $|$**(*x*) ∨** $|$ VP $|$**(*x*)])** $]\!]^g)$
= $[\![$ **∀** $]\!]^g(\lambda x.\vdash[\![N]\!]^s(x) = 0$ oder $[\![VP]\!]^s(x) = 1\dashv)$
= $[\![$ **∀** $]\!]^g($ $[\lambda P.\lambda Q.\vdash \underline{P(x) = 0 \text{ oder } Q(x) = 1}\dashv]$ $([\![N]\!]^s)([\![VP]\!]^s))$

Die einzelnen Übergänge werden in einer Übungsaufgabe nachvollzogen; vorausgesetzt ist dabei natürlich, dass $g(\boldsymbol{i}) = s$. Die letzte Reformulierung in (100) zeigt, dass die (Teilmengen-) Beziehung, die der Determinator **jed-** zwischen zwei Mengen herstellt, letztlich als Eigenschaft

zwischen einer (unterstrichenen) Kombination dieser beiden Mengen verstanden werden kann, nämlich der Vereinigung der zweiten mit dem Komplement der ersten Menge (wenn wir einmal Mengen mit ihren charakteristischen Funktionen gleichsetzen). Die *Reduzierbarkeit* einer zweistelligen Determinatoren-Extension auf ein einstellige Operation lässt sich auch bei anderen Determinatoren beobachten. So gilt z.B. ganz analog zu (101):

(101) $[\![\textbf{ein-}_{indef}\ N\ VP]\!]^s$
= …
= $[\![\exists]\!]^g(\ [\lambda P.\lambda Q.!P(x) = 1 \text{ und } Q(x) = 1!]\ ([\![N]\!]^s)([\![VP]\!]^s))$

Die durch den indefiniten Artikel ausgedrückte zweistellige Relation (der Überlappung) entpuppt sich also als reduzierbar auf eine Kombination der beiden Prädikatsextensionen; die Kombination ist freilich eine andere als die in (101) zur Reduktion von $[\![\textbf{jed-}]\!]^s$ benötigte – nämlich die Schnittbildung. Doch nicht immer lässt sich eine solche Reduktion angeben. Die Extension von **die meisten** kann nachweislich nicht als einstellige Operation über einer Kombination der beiden Argumente umgeschrieben werden. Die Reduzierbarkeit von $[\![\textbf{jed-}]\!]^s$, $[\![\textbf{ein-}_{indef}]\!]^s$ & Co. ist also eher anekdotischer Natur.[109] Angesichts dessen sind viele SemantikerInnen in den letzten Jahren dazu übergegangen, die prädikatenlogische Existenz- und Allquantifikation zugunsten einer allgemeineren und transparenteren Notation aufzugeben, un schreiben $\textbf{(}\forall x\textbf{:}\ \varphi\textbf{)(}\psi\textbf{)}$, $\textbf{(}\exists x\textbf{:}\ \varphi\textbf{)(}\psi\textbf{)}$ und $\textbf{(MOST}x\textbf{:}\ \varphi\textbf{)(}\psi\textbf{)}$ statt $\textbf{(}\forall x\textbf{)}\ [\varphi \rightarrow \psi]$, $\textbf{(}\exists x\textbf{)}\ [\varphi \wedge \psi]$ bzw. $\textbf{(MOST(}\lambda x.\ \varphi\textbf{)(}\lambda x.\ \psi\textbf{)}$. Auch wenn diese notationelle Innovation was für sich hat, bleiben wir in diesem Skript bei der konservativen Variante.

5.6 Intensionalität

Wir beenden unsere indirekte Reformulierung der in den ersten Kapiteln gegebenen semantischen Analysen mit der Hintikka-Semantik der Einstellungsberichte und erweitern bei dieser Gelegenheit den Bestand der analysierten intensionalen Konstruktionen.

Wie bereits in (2) festgehalten, ist der Extensionstyp eines Einstellungsverbs ***(st)(et)***. Wird ein solches Verb mit einem Komplementsatz gesättigt, ergibt sich die Extension des so entstandenen Prädikats durch Anwendung der Verb*extension* auf die *Intension* des Komplements. Um diese Besonderheit der Bedeutungskombination in der indirekten Deutung darzustellen, muss zunächst eine Möglichkeit gefunden werden, aus einer Formel, die die Extension eines sprachlichen Ausdrucks bezeichnet, eine Bezeichnung seiner Intension zu erstellen; denn die Übersetzung des eingebetteten Satzes ist vom Typ ***t*** und steht für dessen Wahrheitswert. Hier zahlt es sich aus, dass die Übersetzungen im Allgemeinen einen expliziten Bezug auf die jeweilige Situation enthalten – und dass dieser Bezug durch eine Variable hergestellt wird. Denn die Intension eines Ausdrucks *A* ist die Funktion, die jeder Situation *s* des Logischen Raums seine Extension in *s* zuweist:

(102) $[\![A]\!] = \lambda s.\ [\![A]\!]^s$

Da wir davon ausgehen, dass in der typenlogischen Übersetzung $|A|$ von *A* die jeweils betrachtete Situation durch die Variable ***i*** bezeichnet wird, erhält man (für beliebige $s \in LR$) die Extension von *A* in der Situation *s*, wenn man die Belegung der Variablen ***i*** als Wert gerade *s* zuweist:[110]

---

109 Für die Entwicklung der Prädikatenlogik war diese Reduzierbarkeit allerdings offenbar von entscheidender Bedeutung. – Die Nicht-Reduzierbarkeit von **die meisten** folgt aus einem Satz des US-amerikanischen Semantikers Ed Keenan, den dieser in seinem Aufsatz *Natural Language, Sortal Reducibility and Generalized Quantifiers* (1993) bewiesen hat.

110 Wir gehen dabei auch davon aus, dass die anderen Variablen keine Rolle spielen. In den bisher betrachteten Übersetzungen sind außer ***i*** ohnehin immer alle Variablen gebunden; ihr Wert hängt also

(103) $[\![A]\!]^s = [\![\,|A|\,]\!]^{g[\boldsymbol{i}/s]}$

Da (103) für beliebige $s \in LR$ gilt,[111] folgt aus (102) und (103):

(104) $[\![A]\!] = \lambda s.\ [\![\,|A|\,]\!]^{g[\boldsymbol{i}/s]}$

Angesichts der Formulierung (44'') der Deutung des **λ**-Operators ergibt sich jetzt $(\boldsymbol{\lambda i.}\,|A|)$ als Übersetzung der Intension von *A*; denn es gilt:

(105) $\lambda s.\ [\![\,|A|\,]\!]^{g[\boldsymbol{i}/s]} = [\![\,(\boldsymbol{\lambda i.}\,|A|)\,]\!]^g$

Damit erhalten wir die:[112]

(106) *Indirekte Deutung der Anbindung von **dass**-Sätzen als Komplementen*
Wenn *VP* ein Prädikat ist, bestehend aus einem Einstellungsverb *V* und einem Komplementsatz *S* an Objektstelle, dann gilt:
$|VP| = |V|\,(\boldsymbol{(\lambda i.}\,|S|\,\boldsymbol{)})$

Man beachte, dass es für die Regel (106) nicht nur wesentlich ist, dass der Situationsbezug durch eine Variable hergestellt wird – sonst hätten wir ja nicht von ihm abstrahieren können; ebenso wichtig ist, dass es sich immer um dieselbe Variable handelt, nämlich ***i***.

Um die indirekte Formulierung der Hintikka-Semantik abzuschließen, bedarf es nur noch einer Angabe der lexikalischen Übersetzungen von Einstellungsverben. Dafür benennen wir die den Einstellungen zugrundeliegenden Perspektiven *Dox*, *Epi*, *Bou*,… durch geeignete Konstanten **DOX**, **EPI**, **BOU**,… Wie wir in Kapitel 4 gesehen haben, hängt die (doxastische, epistemische, bouletische,…) Perspektive eines Individuums (des Typs ***e***) immer von der betrachten Situation (des Typs ***s***) ab und besteht in einer Menge (des Typs ***st***) von Situationen. Dementsprechend handelt es sich bei den genannten Konstanten um Formeln des Typs ***e(s(st))***, die wie folgt gedeutet werden:

(107d) $[\![\mathbf{DOX}]\!] = \lambda x.\lambda s_0.\lambda s_1.$ ⊢in s_0 ist s_1 eine doxastische Alternative für x⊣
(e) $[\![\mathbf{EPI}]\!] = \lambda x.\lambda s_0.\lambda s_1.$ ⊢in s_0 ist s_1 eine epistemische Alternative für x⊣
(b) $[\![\mathbf{BOU}]\!] = \lambda x.\lambda s_0.\lambda s_1.$ ⊢in s_0 ist s_1 eine bouletische Alternative für x⊣

Dabei handelt es sich bei den doxastischen und epistemischen *Alternativen* (von x in s_0) um die Situationen s_1, in denen sich zu befinden eine Person x aufgrund ihrer Überzeugungen bzw. ihres Wissens nach (in s_0) nicht ausschließen kann, die also mit ihren Überzeugungen bzw. ihrem Wissen (in s_0) konform sind; bouletische Alternativen sind Situationen, die mit ihren Wünschen (in s_0) konform gehen, in denen also alles nach den Wünschen läuft, die x (in s_0) hat. Die in (107) interpretierten Konstanten lassen sich jetzt für die lexikalischen Übersetzungen der Einstellungsverben heranziehen:

(108d) $|\mathbf{meint}| = \boldsymbol{\lambda p^{st}.\lambda x^e.\ (\forall j)\ [DOX(x)(i)(j) \rightarrow p_j]}$
(e) $|\mathbf{weiß}| = \boldsymbol{\lambda p^{st}.\lambda x^e.\ (\forall j)\ [EPI(x)(i)(j) \rightarrow p_j]}$
(b) $|\mathbf{will}| = \boldsymbol{\lambda p^{st}.\lambda x^e.\ (\forall j)\ [BOU(x)(i)(j) \rightarrow p_j]}$

(nach dem Koinzidenzlemma) nicht von der Belegung ab.

111 Diese Voraussetzung ist unabdingbar für den Schluss von (95) und (96) auf (97); denn auch das '*s*' in (95) steht für beliebige Situationen. Diesen Punkt kann man sich klar machen, wenn man (95) und (96) mit verschiedenen (metasprachlichen) Situationsvariablen reformuliert – also eine gebundene Umbenennung in der Metasprache vornimmt.

112 Die Doppeltklammerung ergibt sich daraus, dass nach der in Abschnitt 5.2 formuliterten Syntax der Typenlogik (21) sowohl die Lambda-Abstraktion als auch die Applikation Klammern einführen; solche Dooppelungen werden normalerweise stillschweigend vermieden.

In (108) ist – wie ab jetzt immer – die Variable $\boldsymbol{j}$ vom Typ $\boldsymbol{s}$, und in $\boldsymbol{p_j}$ steht der (untere) Index wieder für die Applikation $\boldsymbol{p(j)}$. Und wie schon zuvor steht $\boldsymbol{(\forall j)}$ … für $\boldsymbol{\forall(\lambda j.\ \varphi)}$, wobei der Allquantor $\boldsymbol{\forall}$ für die Formel $\boldsymbol{(\lambda p^{st}.\ \neg\ (\exists j)\ \neg\ p(j))}$ steht und somit – wie der in ihr verwendete Existenzquantor vom Typ $\boldsymbol{(st)t}$.ist. Strenggenommen handelt es sich also bei diesen All- und Existenzquantoren nicht um die im vorangehenden Abschnitt eingeführten Formeln, die ja vom Typ $\boldsymbol{(et)t}$ waren. Wir schließen uns hier aber der logisch-semantischen Tradition an und machen keinen notationellen Unterschied zwischen den beiden Typen von Quantoren.

Um zu sehen, wie die indirekte Deutung (106) und (108) der Hintikka-Semantik in der Anwendung funktioniert, konstruieren und reduzieren wir die typenlogische Übersetzung eines einfachen Einstellungsberichts:

(109) $|\mathbf{Fritz\ meint,\ dass\ Eike\ einen\ Porsche\ besitzt}|$

mit (67)

$=\ |\mathbf{meint,\ dass\ Eike\ einen\ Porsche\ besitzt}|(|\mathbf{Fritz}|)$

nach (106)[113]

$=\ |\mathbf{meint}|((\lambda i.|\mathbf{Eike\ besitzt\ einen\ Porsche}|))(|\mathbf{Fritz}|)$

mit (66)

$=\ |\mathbf{meint}|((\lambda i.|\mathbf{Eike\ besitzt\ einen\ Porsche}|))(\mathbf{f})$

s. (108d)

$=\ (\lambda p^{st}.\lambda x^{e}.\ (\forall j)\ [\mathbf{DOX}(x)(i)(j) \rightarrow p_j])((\lambda i.|\mathbf{Eike\ besitzt\ einen\ Porsche}|))(\mathbf{f})$

Übungsaufgabe

$\equiv\ (\lambda p^{st}.\lambda x^{e}.\ (\forall j)\ [\mathbf{DOX}(x)(i)(j) \rightarrow p_j])((\lambda i.(\exists y)\ [\mathbf{P}_i(y) \wedge \mathbf{B}_i(\mathbf{e},y)]))(\mathbf{f})$

mit λ-Konversion (58)

$\equiv\ (\lambda x^{e}.\ (\forall j)\ [\mathbf{DOX}(x)(i)(j) \rightarrow (\lambda i.(\exists y)\ [\mathbf{P}_i(y) \wedge \mathbf{B}_i(\mathbf{e},y)])(j)])(\mathbf{f})$

mit λ-Konversion (58)

$\equiv\ (\lambda x^{e}.\ (\forall j)\ [\mathbf{DOX}(x)(i)(j) \rightarrow (\exists y)\ [\mathbf{P}_j(y) \wedge \mathbf{B}_j(j)(\mathbf{e},y)]])(\mathbf{f})$

mit λ-Konversion (58)

$\equiv\ (\forall j)\ [\mathbf{DOX}(\mathbf{f})(i)(j) \rightarrow (\exists y)\ [\mathbf{P}_j(y) \wedge \mathbf{B}_j(\mathbf{e},y)]]$

Man beachte, dass nicht alle Übergänge in (109) Gleichungen (=) sind; bei Anwendung der λ-Konversion und anderer Rechenregeln ergeben sich neue, äquivalente Formeln ($\equiv$).

(109) erfasst übrigens nur eine Lesart des analysierten (Oberflächen-) Satzes. Nach dieser Lesart muss Fritz keine spezifischen Vorstellung über Eikes Besitztum haben. Insbesondere muss er kein bestimmtes Auto im Auge haben, dessen Besitz er Eike zuschreibt. In Kapitel 7 werden wir eine weitere Lesart kennenlernen und analysieren, nach der Fritz *von* einem bestimmten Porsche meint, dass dieser Eike gehört.

Mit der lexikalischen Analyse (108b) können nicht nur finite Komplemente analysiert werden; auch die Verwendung von **wollen** als Infinitiv einbettendes *Kontrollverb* wie in (110) lässt sich mit diesem Lexikoneintrag erfassen :

(110) **Fritz will gewinnen.**

Um (110) im Rahmen der Hintikka-Semantik zu deuten, können wir (vereinfachend) davon ausgehen, dass sich der Infinitiv durch einen **dass**-Satz paraphrasieren lässt, in dem die (oberflächlich vakante) Subjektsposition durch den Matrixsatz ergänzt wird:

(111) *NN* **will** *P* (a)

$\approx$ *NN* **will, dass** *NN P* (b)

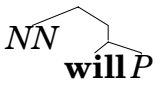
In (111) stehen '*NN*' und '*V*' für einen Eigennamen bzw. ein Prädikat; dabei haben wir die Unterschiede zwischen finiten Prädikaten und Infinitiven vernachlässigt.[114] Der in (111) postu-

[113] Wir setzen nach wie vor voraus, dass der eingebettete **dass**-Satz auf seine Hauptsatzvariante zurückgeführt wird.

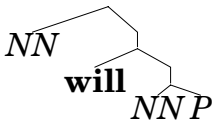
[114] Das heißt vor allem, wir ignorieren den durch die Tempusmorphologie ausgedrückten Zeitbezug.

lierte Synonymie-Effekt – so ist die Quasi-Gleichung zu lesen – lässt sich auf der Grundlage der lexikalischen Deutung (108b) von **wollen** relativ einfach erreichen, wenn man – gängigen syntaktischen Vorstellungen zum Trotz – annimmt, dass der eingebettete Infinitiv auch zugrundeliegend subjektlos ist und somit dem Prädikat entspricht:

(111') (a)



≈ (b)



Nach (111') sollte gelten:

(112) $|\,NN\ \textbf{will}\ P\,|$

≡ $|\,NN\ \textbf{will, dass}\ NN\ P\,|$ wg. (111)

≡ $|\,\underline{\textbf{will}}\,|(\underline{\boldsymbol{\lambda i.}}\,|\,\underline{P}\,|(|\,\underline{NN}\,|))\ (|\,NN\,|)$ mit (67) & (106)

Der unterstrichene Teil entspricht dabei offenbar der Übersetzung des Prädikats von (111b): sein Wert charakterisiert die Menge der Individuen, die wollen, dass *NN* (= der Träger des Namens '*NN*') gewinnt. Doch trotz der (unterstellten) Synonymie zwischen den beiden Einstellungsberichten entspricht die in (112) hervorgehobene Teilformel nicht dem Prädikat von (111a); dessen Extension umfasst die Individuen, die selbst gewinnen wollen: die als Argument des Modalverbs **will** fungierende Proposition hängt bei (111a) vom Subjekt ab. Die folgende Reformulierung der Übersetzung von (110) trägt dem Rechnung:

(113) $|\,NN\ \textbf{will}\ P\,|$

≡ $|\,\textbf{will}\,|(\boldsymbol{\lambda i.}\,|\,P\,|(|\,NN\,|))\ (|\,NN\,|)$ s. (112)

≡ $(\boldsymbol{\lambda x_e.}\,|\,\underline{\textbf{will}}\,|(\underline{\boldsymbol{\lambda i.}}\,|\,\underline{P}\,|(|\,\underline{\boldsymbol{x}}\,|))(\boldsymbol{x}))\ (|\,NN\,|)$ mit **λ**-Konversion (58)

Diesmal entspricht der unterstrichene Teil dem Prädikat von (110): sein Wert charakterisiert die Menge der Individuen, die wollen, dass sie jeweils selbst gewinnen. Die Korrektheit des Übergangs ergibt sich daraus, dass die Übersetzungen von Eigennamen immer Konstanten sind, die also auch keine Variablen enthalten, die beim Konvertieren gebunden werden könnten.

Aus (113) lässt sich unmittelbar eine Deutung der Infinitiveinbettung gewinnen, nach der die Extension des Verbs vom Typ ***(st)(et)*** ist:

(114) *Indirekte Deutung der Infinitiveinbettung bei (Subjekt-) Kontrollverben*
Wenn *VP* ein Prädikat ist, bestehend aus einem Einstellungsverb *V* und einem infinitivischen Prädikat *P* an Objektstelle, dann gilt:
$|\,VP\,| = \boldsymbol{(\lambda x.}\,|\,V\,|\,\boldsymbol{((\lambda i.}\,|\,P\,|\,\boldsymbol{(x))(x)}$

Die angegebene Deutung funktioniert nur bei solchen Verben, die das in ihrem Komplement implizite Subjekt ('PRO') mit ihrem Subjekt im Sinne der (Beinahe-) Synonymie (111) gleichsetzen ('kontrollieren'). Dazu gehören im Deutschen neben den Modalverben (in gewissen Verwendungen) auch Vollverben, die **zu**-Infinitive einbetten: **hoffen**, **versprechen**, **versuchen**. Wie das letzte Beispiel zeigt, betten manche dieser Verben keine **dass**-Sätze ein, lassen sich aber dennoch im Stil von (114) analysieren. Doch nicht jede Infinitiveinbettung kann nach (114) gedeutet werden. Zum Einen wird bei manchen Verben (**anweisen**, **empfehlen**, **raten**,…) das infinitivische Subjekt mit einem Objekt des eingebetteten Verbs gleichgesetzt; in diesem Fall kann man eine zu (114) analoge Deutung der Objekt-Kontroll-Konstruktion angeben, worauf wir hier verzichten. Zum Anderen vollziehen einige wenige Verben gar keine Gleichsetzung, son-

dern verhalten sich (wieder intuitiv gesprochen) so, als gehöre ihr Subjekt ganz dem eingebetteten Infinitv. Zu diesen *Anhebungsverben* gehören im Deutschen neben einigen (Lesarten von) Modalverben unter anderem die Verben **scheinen** und **drohen** (in der unpersönlichen Verwendung). Dass sich Letztere nicht durch (114) erfassen lassen, sieht man am Besten, wenn man ein quantifizierendes Subjekt betrachtet:

(115) **Die meisten Gäste scheinen da zu sein.**

(115) trifft auf eine Situation zu, in der der Sprecher über (nicht näher spezifizierte) Evidenz dafür verfügt, dass sich die Mehrzahl der Zuschauer eines gewissen (nicht näher spezifizierten) Ereignisses bereits am (nicht näher spezifizierten) Ort des Geschehens eingefunden hat. Diese Evidenz könnte beispielsweise im Behängungszustand eines Kleiderständers bestehen: die Tatsache, dass in der Gästegarderobe fünf Mäntel hängen, spricht dafür, dass schon fünf der insgesamt sieben geladenen Gäste da sind. Ein Sprecher, der sich auf diese Evidenz stützt, muss nicht unbedingt in der Lage sein, auch nur eines der genannten Kleidungsstücke seinem Träger zuzuordnen. Er muss nicht einmal in der Lage sein, auch nur über einen einzigen Gast eine qualifizierte Vermutung dahingehend zu äußern, ob dieser Gast schon da ist. Mit anderen Worten: auch wenn (115) zutrifft, muss für keinen einzigen der erwarteten Gäste *NN* der folgende Satz wahr sein,:

(116) ***NN* scheint da zu sein.**

Doch dann kann **scheinen** kein Einstellungsverb (mit Extensionstyp ***(st)(et)***) sein, für das bei Einbettung eines Infinitivs (114) zuständig ist. Denn die in (114) gegebene Übersetzung von Prädikaten der Form '*V* **zu** *P*' kombiniert sich mit dem quantifizierenden Subjekt **die meisten Gäste** gerade so, dass die Wahrheit des quantifizierenden Satzes – nach der üblichen Deutung (77) der Quantifikation – davon abhängt, wie viele entsprechende Prädikationen wahr sind; (115) müsste dementsprechend wahr sein, wenn es mehr wahre als falsche Sätze der Form (116) gäbe (bei denen '*NN*' jeweils für einen Gast steht).[115] Das Beispiel hat gezeigt, dass dem nicht so ist: die Prädikationen (116) können ungeachtet der Wahrheit von (115) allesamt falsch sein.

Der Grund dafür, dass Anhebungsverben wie **scheinen** nicht nach dem Muster (115) verstanden werden, besteht darin, dass sie zwar propositionale Einstellungen ausdrücken, aber keine Einstellungen des Subjekts (genauer: seiner Extension). Im Fall von **scheinen** kann die Rolle des fehlenden Subjekts durch einen 'Experiencer'-Dativ ausgedrückt werden: **ihm scheint, dass…** Dieser Kasus-Anomalie zum Trotz lässt sich dieser Gebrauch von **scheinen** offenkundig im Rahmen der Hintikka-Semantik erfassen. Wir werden hier nicht detailliert auf diese Konstruktion eingehen, gehen aber davon aus, dass ihr wie den anderen in (107) analysierten Einstellungen eine Alternativen-Relation des Typs ***e(s(st))*** zugrundeliegt:

(107s) $[\![\mathbf{EVI}]\!] = \lambda x.\lambda s_0.\lambda s_1$!in s_0 ist s_1 eine evidente Alternative für x⊣

Die *evidenten* Alternativen (für ein Individuum x in einer gegebenen Situation s_0) sind dabei die möglichen Situationen, in denen sich zu befinden x angesichts der x in s_0 verfügbaren Evidenzen nicht ausschließen kann. Was als Evidenz zählt – ob Wahrnehmung, Vorwissen, Hörensagen etc. – lassen wir dabei offen.

Die in (107a) genannte Hintikkasche Einstellungsbeziehung liegt auch der *unpersönlichen* Verwendung von **scheinen** (**Es scheint, dass…**) zugrunde, wobei die Rolle des Einstellungsträgers implizit vom Sprecher oder einer Gruppe, der er angehört, übernommen wird. Da wir fürs Erste (genauer: bis Kapitel 8) Probleme der Kontextabhängigkeit – zu der insbesondere der Sprecherbezug gehört – ausblenden, werden wir der Einfachheit halber unpersönlich von den evidenten Alternativen in einer Situation sprechen und für diese eine Konstante **EVI*** des Typs

115 Wir gehen stillschweigend davon aus, dass jeder Gast genau einen Namen hat!

s(st) bereitstellen:[116]

(107s*) $[\![\mathbf{EVI^*}]\!] = \lambda s_0.\lambda s_1 \vdash s_1$ ist eine evidente Alternative in $s_0 \dashv$

Mit **EVI*** lässt sich die unpersönliche Verwendung von **scheinen** genauso wie ein Einstellungsverb interpretieren, außer dass ihm das (persönliche) Subjekt fehlt. Ein Beispiel sollte genügen:

(117) **|Es scheint, dass die meisten Gäste da sind|**
≡ $\mathbf{(\forall \boldsymbol{j})\ [EVI^*_{\boldsymbol{i}}(\boldsymbol{j}) \rightarrow MOST(G_{\boldsymbol{j}})(D_{\boldsymbol{j}})]}$
≡ **|schein-|(λ*i*. |die meisten Gäste sind da|)**

Die – wenig aufregenden – Details dieser Analyse lassen wir aus. Wichtig ist nur, dass wir in (117) von der folgenden lexikalischen Deutung des unpersönlichen **scheinen** ausgehen:

(118) $\mathbf{|scheint|} = \mathbf{(\lambda \boldsymbol{p}^{st}.\ (\forall \boldsymbol{j})\ [EVI^*_{\boldsymbol{i}}(\boldsymbol{j}) \rightarrow \boldsymbol{p}_{\boldsymbol{j}}])}$

Da nun (115) offenbar dasselbe besagt wie die in (117) analysierte unpersönliche Variante, liegt es nahe, (118) auch der Verwendung von **scheinen** als Anhebungsverb zugrunde zu legen. Die (indirekte) Deutung von (115) muss demnach Folgendes leisten:

(119) **|die meisten Gäste scheinen da zu sein|**
≡ **|scheint|(λ*i*. |die meisten Gäste sind da|)** — s. (117)
≡ **|scheint|(λ*i*. |die meisten Gäste|(|sind da|))** — nach (77)
≡ $\mathbf{|scheint|(\lambda \boldsymbol{i}.\ \underline{MOST(G_{\boldsymbol{i}})}(D_{\boldsymbol{i}}))}$ — mit (63)

Um dieses Ergebnis auf kompositionelle Weise zu erreichen, muss das Prädikat **scheinen da zu sein** mit dem Subjekt **die meisten Gäste** so kombiniert werden, dass Letzteres die unterstrichene Position im Argument von **|scheint|** einnehmen kann. Die Übersetzung des Prädikats muss also der letzten Zeile von (119) gleichen, abzüglich des unterstrichenen Teils. Die dafür erforderlich Differenzbildung kann natürlich wieder per Lambda-Abstraktion vorgenommen werden. Allerdings gilt es dabei, eine Komplikation mit der Situationsvariablen ***i*** zu vermeiden. Die folgende Formel ist nämlich *nicht* äquivalent zu der letzten Zeile in (119):

(120) $\mathbf{(\lambda X_{(et)t}\ |scheint|(\lambda \boldsymbol{i}.\ \boldsymbol{X}(D_{\boldsymbol{i}}))\ (MOST(G_{\boldsymbol{i}}))}$

Den Nachweis der Nicht-Äquivalenz schenken wir uns hier; aber wir nehmen zur Kenntnis, dass der Übergang von (119) zu (120) nicht durch das Gesetz (58) der Lambda-Konversion gedeckt ist: das Argument $\mathbf{MOST(G_{\boldsymbol{i}})}$ (= **|die meisten Gäste|**) enthält die freie Variable ***i***, die beim Einsetzen gebunden wird. Dass (120) als Dekomposition der Übersetzung von (115) nicht funktioniert, zeigt sich im Übrigen auch darin, dass das Argument lediglich die *Extension* des Subjekts benennt. Nach (120) müsste damit jedes extensionsgleiche Subjekt zum selben Wahrheitswert führen, was nicht der Fall ist: wenn die Gäste gerade die Mitglieder des Kaninchenzüchtervereins sind, muss mit dem anscheinenden Eintreffen der Mehrheit der Letzteren nicht unbedingt auch Evidenz dafür vorliegen, dass die meisten Gäste eingetroffen sind. (Ein entsprechendes Szenario zu konstruieren, in dem die extensionale Substitution des Subjekts scheitert, bleibt der LeserInnenschaft überlassen.) Der Beitrag des Subjekts zum Wahrheitswert von (115) besteht also nicht nur in seiner Extension, sondern in seiner Intension. Entsprechend muss die Formel in der letzten Zeile von (119) wie folgt zerlegt werden:

---

116 Objekte dieses Typs kann man offenbar als Beziehungen zwischen Punkten im Logischen Raum verstehen. Solche Beziehungen werden in der Modallogik – der Urform der Hintikka-Semantik – als *Zugänglichkeitsrelationen* bezeichnet.

(121) $|\textbf{scheint}|(\lambda i.\ \textbf{MOST}(\textbf{G}_i)(\textbf{D}_i))$

Eigen-Konversion (58*)

$\equiv \quad |\textbf{scheint}|(\lambda i.\ \underline{(\lambda i.\textbf{MOST}(\textbf{G}_i)}\ (\textbf{D}_i)))$

Lambda-Konversion (58)

$\equiv \quad (\lambda \boldsymbol{X}^{s((et)t)}\ |\textbf{scheint}|(\lambda i.\ \boldsymbol{X}(i)\ (\textbf{D}_i)))\ \underline{(\lambda i.\textbf{M}(\textbf{G}_i))}$

Die unterstrichene Teilformel entspricht dabei der Intension des Subjekts. Entsprechend sollte der Rest der Formel der Übersetzung des Prädikats entsprechen, die sich wiederum aus der Übersetzung (118) des Anhebungsverbs und des eingebetteten Infinitivs ergeben sollte. Letzteres leistet die folgende, jetzt naheliegende Regel:

(122) *Indirekte Deutung der Infinitiveinbettung bei Anhebungsverben*
Wenn *VP* ein Prädikat ist, bestehend aus einem Anhebungsverb *V* und einem infinitivischen Prädikat *P* an Objektstelle, dann gilt:
$|VP| = (\lambda \boldsymbol{X}^{s((et)t)}.|V|\ (\lambda i.\ \boldsymbol{X}(i)(|P|)))$

(122) lässt sich per Applikation mit der Intension des Subjekts verbinden. Allerdings kann dies die Quantifikationsregel (77) nicht leisten; da auch die Prädikation hier nicht in Frage kommt – das Subjekt von (115) ist ja ein Quantor und nicht vom Typ $\boldsymbol{e}$ –, benötigen wir an dieser Stelle offenbar eine dritte Möglichkeit, die Verbindung von Subjekt und Prädikat zu deuten:

(123) *Indirekte Deutung angehobener Subjekte*
Wenn *S* ein Satz ist, an dessen Subjektstelle eine quantifizierende Nominalphrase *QN* steht und dessen Prädikat *VP* mit einer typenlogischen Formel $|VP|$ des Typs $\boldsymbol{s((et)t))t}$ übersetzt wird, dann gilt:
$|S| = |VP|\ (\lambda i.\ |QN|)$

(123) ist insofern eine Regel neuen Typs, als sie explizit auf den Typ der Übersetzung einer der Konstituenten Bezug nimmt. Diese – methodologisch unbedenkliche – Komplikation ließe sich vermeiden, wenn etwa die Prädikate entsprechend syntaktisch markiert wären (etwa durch ein Merkmal [±Raising]). Die Bezeichnung 'angehobene Subjekte' gibt die Intuition wieder, dass die quantifizierenden Subjekte sich so verhalten, als seien aus den untergeordneten Infinitivsätzen herausgehoben worden. Diese Intuition lässt sich auch direkt syntaktisch modellieren, was zu einer alternativen Deutung der Konstruktion führt, die wir in Kapitel 7 kennenlernen werden.[117]

(122) und (123) erfassen nur Anhebungsverben mit quantifizierenden Subjekten und lassen sich damit nicht unmittelbar auf Sätze der Form (116) anwenden. Wir werden diese Lücke im nächsten Kapitel schließen, wo wir eine Methode kennenlernen, Prädikation, Quantifikation und Anhebung auf eine einzige Konstruktion zurückzuführen.

Das Besondere an der Deutung (123) der Anhebungskonstruktion ist der Umstand, dass einerseits die Subjektposition durch einen Quantor besetzt ist, andererseits aber dieses Subjekt (genauer: seine Intension) als Argument des Prädikats (genauer: seiner Extension) gedeutet wird. In Ersterem gleicht die Konstruktion der Quantifikation, in Letzerem dagegen der Prädikation. Diese Kombination, wie wir sie in der Subjektposition von Anhebungsverben vorfinden, trifft man auch in der Objektposition so genannter *opaker* Verben an. Betrachten wir dazu ein Beispiel:

(124) **Fritz sucht ein**$_{indef}$ **Restaurant.**

---

117 Die hier vorgestellte Deutung der Anhebungskonstruktion wurde erstmals von Richard Montague in seinem Aufsatz *The Proper Treatment of Quantifiaction in Ordinary English* (1973) erwähnt, aber nicht im Detail ausgearbeitet. Aus demselben Werk stammt auch die Semantik der Kontrollverben im Stil von (114).

Ganz ähnlich wie bei (115) die Deutung (77) quantifizierender Subjekte versagt hat, führt in diesem Fall die Interpretation (78) quantifizierender Objekte zu Problemen. Wenn nämlich (124) auf eine Situation *s* zutrifft, muss Fritz in dieser Situation kein bestimmtes Lokal suchen; es muss also kein Lokal namens '*L*' geben, so dass auch (125) gilt:

(125) **Fritz sucht *L*.**

Nach der Objektanbindungsregel (78) besagt aber (124), dass für ein Restaurant *L* der Satz (125) zutrifft. Doch (124) kann sogar wahr sein, obwohl es in der gegebenen Situation überhaupt keine Restaurants gibt. Um dennoch zu einer kompositionellen Deutung von (124) zu gelangen, betrachten wir zunächst die folgende annähernde Paraphrase:[118]

(126) **Fritz will ein$_{indef}$ Restaurant finden.**

(126) können wir mit dem oben eingeführten Instrumentarium indirekt deuten. Wie man leicht nachprüft, ergibt sich mit, der Regel (114) für Kontrollverben die folgende (reduzierte) typenlogische Übersetzung von (126):

(127) $(\forall j)\,[\mathbf{BOU(f)}(i)(j) \rightarrow (\exists y)\,[\mathbf{R}_j(y) \wedge \mathbf{F}_j(\mathbf{f},y)]]$

Unter der Annahme, dass (124) mit (126) synonym ist, lässt sich mit Hilfe von (127) eine kompositionelle Deutung des Satzes gewinnen. Dabei gehen wir genauso vor wie bei der Entwicklung der Semantik von **scheinen** in (119) und (121) und isolieren zunächst den Beitrag, den das Objekt zum Prädikat leistet:

(128) $|\,\mathbf{will\ ein}_{indef}\ \mathbf{Restaurant\ finden}\,|$

mit (78) & (114)

$\equiv\quad (\lambda x^e.\,|\,\mathbf{will}\,|\,(\lambda i.\,|\,\mathbf{ein\ Restaurant}\,|\,(\lambda y^e.|\,\mathbf{finden}\,|(x,y)))(x))$

mit (65b)

$\equiv\quad (\lambda x^e.\,|\,\mathbf{will}\,|\,(\lambda i.\,(\lambda P^{et}\,(\exists y)\,[\mathbf{R}_i(y) \wedge P(y)])\,(\lambda y^e.|\,\mathbf{finden}\,|(x,y)))(x))$

Eigen-Konversion (58*)

$\equiv\quad (\lambda x^e.\,|\,\mathbf{will}\,|\,(\lambda i.\,\underline{(\lambda i.}\,\underline{(\lambda P^{et}\,(\exists y)\,[\mathbf{R}_{\underline{i}}(y) \wedge P(y)])}\,)\,(i)\,(\lambda y^e.|\,\mathbf{finden}\,|(x,y)))(x))$

λ-Konversion (58)

$\equiv\quad (\lambda X^{s((et))t}.(\lambda x^e.|\,\mathbf{will}\,|\,(\lambda i.\,X(i)(\lambda y.|\,\mathbf{finden}\,|(x,y)))(x)))(\lambda i.\,(\lambda P^{et}\,(\exists y)[\mathbf{R}_i(y) \wedge P(y)]))$

Annahme

$\equiv\quad |\,\mathbf{sucht\ ein}_{indef}\ \mathbf{Restaurant}\,|$

Der erste Übergang in (128) ergibt sich unmittelbar aus den genannten Übersetzungsregeln (und den üblichen Annahmen zu den lexikalischen Deutungen). Beim zweiten Übergang wird die Übersetzung des Objekts explizit angegeben, um auf die freie Variable ***i*** aufmerksam zu machen. Mit dem dritten Übergang wird – ähnlich wie in (121) – die Intension von **ein Restaurant** als Beitrag zum Gesamt-Prädikat isoliert, was wieder mit einer Eigen-Konversion geschieht. Auf diese Weise wird die folgende Konversion vorbereitet, bei der sonst – ähnlich wie in (120) – das freie ***i*** gebunden würde. Der letzte Übergang macht deutlich, dass wir – ausgehend von der Synonymie der Prädikate in (124) und (126) – die Intension des Objekts des opaken Verbs isoliert haben. Der Rest der Formel ist dann offenbar der Beitrag des Prädikats selbst:

(129) $|\,\mathbf{sucht}\,| = (\lambda X^{s((et))t}.\,(\lambda x^e.|\,\mathbf{will}\,|\,(\lambda i.\,X(i)(\lambda y^e.|\,\mathbf{finden}\,|(x,y)))(x)))$

$\equiv\quad (\lambda X^{s((et))t}.(\lambda x^e.(\forall j)\,[\mathbf{BOU}(x)(i)(j) \rightarrow X(j)(\lambda y.\mathbf{F}_j(x,y))]))$

---

118 Die Idee, opake Verben durch Paraphrasen auf propositionale Einstellungen zurückzuführen, geht auf den US-amerikanischen Logiker und Philosophen Willard Van Orman Quine zurück, der sie in seinem Aufsatz *Quantifiers and Propositional Attitudes* (1956) entwickelt hat. Die kompositionelle Deutung der Opazität stammt von Richard Montague, der sie erstmals in seinem Aufsatz *On the Nature of Certain Philosophical Entities* (1968) angedeutet und später in den in Fn. 15 und 117 genannten Schriften ausgearbeitet hat.

Die in (129) gegebene lexikalische Analyse erfordert offenbar die folgende Übersetzungsregel:

(130) *Indirekte Deutung der Objektanbindung bei opaken Verben*
Wenn *VP* ein Prädikat bestehend aus einem opaken transitiven Verb *V* und einer quantifizierenden Nominalphrase *QN* ist, dann gilt:
$|VP| = |V|(\lambda i.|QN|)$

Analysen wie in (129), in denen explizit auf die Bedeutungen (bzw. Intensionen) anderer lexikalischer Ausdrücke Gebrauch gemacht wird (**will** und **finden**) bezeichnet man auch als *lexikalische Zerlegungen*.

So wie nach (123) das Anhebungsverb (genauer: seine Extension) das quantifizierende Subjekt (genauer: seine Intension) als Argument nimmt, so operiert nach (130) das opake Verb (bzw. seine Extension) auf dem Objekt (bzw. seiner Intension). Zudem gibt es – bei allen Unterschieden in der Länge der Formeln – eine interessante Parallele in den lexikalischen Analysen (118) des Anhebungsverbs **scheinen** und (129) des opaken Verbs **suchen**: in beiden Fällen wird die Verbbedeutung kompositionell auf eine propositionale Einstellung zurückgeführt, die dann mit Hilfe Hintikkascher Perspektiven gedeutet wird. Beim Anhebungsverb **scheinen** war dies die durch das unpersönliche **scheinen** ausgedrückte Beziehung. Beim opaken Verb **suchen** war es die durch **wollen** ausgedrückte bouletische Perspektive. Das ist möglicherweise kein Zufall: nach einer als *Propositionalismus* bezeichneten Auffassung lassen sich alle intensionalen Konstruktionen per Paraphrase auf Satzeinbettungen zurückführen. Anhebungsverben und opake Verben bilden aus dieser Sicht – auf die wir hier nicht weiter eingehen können – nur einen Spezialfall. Für den Propositionalismus spricht jedenfalls, dass sich für so gut wie alle bekannten opaken Verben plausible lexikalische Zerlegungen im Stil von (129) angeben lassen. So lässt sich das in einer Übungsaufgabe betrachtete opake Verb **schulden** durch eine Kombination der Bedeutungen von **müssen** und **geben** analysieren.

Wie bereits weiter oben bemerkt, muss Fritz kein bestimmtes Restaurant suchen, damit der Satz (124) wahr ist. Andererseits kann er durchaus ein bestimmtes Restaurant suchen, was die Wahrheit von (124) garantiert. In diesem Fall spricht man von einer *spezifischen Lesart* (der opaken Objektanbindung). Auf sie werden wir im übernächsten Kapitel zu sprechen kommen. Die in (130) gegebene Analyse opaker Objekte erfasst nur die unspezifische Lesart.

5.7 Alternativen zur zweisortigen Typenlogik

Neben der in diesem Kapitel zur indirekten Deutung herangezogenen zweisortigen Typenlogik gibt es eine Reihe anderer formaler Sprachen, die sich für diesen Zweck mehr oder weniger gut eignen. Zwei von ihnen stellen wir im Folgenden vor. Zunächst jedoch werden wir eine Variante der zweisortigen Typenlogik vorstellen, die man in der Literatur häufig antrifft.

Logische Konstanten

Wir hatten in (31) – (34) in Abschnitt 5.3 vier 'logische' Konstanten eingeführt, die von besonderer Bedeutung für die indirekte Deutung sind, weil sie in den verschiedensten Zusammenhängen Verwendung finden. Zudem haben wir weitere Symbole ($\vee$, $\rightarrow$ und $\forall$) als Abkürzungen für Formeln eingeführt, die mit Hilfe dieser logischen Konstanten gebildet werden können – und die wir ebensogut als eigene Konstanten hätten behandeln können. Indem wir sie als Abkürzungen behandelt haben, haben wir gezeigt, dass sie sich auf die anderen letztgenannten Konstanten – $\wedge$,$\exists$, $\neg$ und $=$ – *reduzieren* lassen: im Prinzip kommt man mit diesen in (31) – (34) definierten logischen Konstanten aus.

Im Prinzip kommt man sogar ohne die in (31) – (34) definierten logischen Konstanten aus. Man muss dazu allerdings die Syntax (21) der Typenlogik um eine weitere Konstruktion ergänzen, nämlich um Gleichungen zwischen Formeln *beliebiger* Typen:

(131) *Syntax der (zweisortigen funktionalen) Typenlogik mit Gleichungen*
Für alle Typen a und b gilt:
(Var) Variablen des Typs a sind Formeln des Typs a.
(Kon) Konstanten des Typs a sind Formeln des Typs a.
(App) Wenn $\boldsymbol{\alpha}$ eine Formel eines Typs $\boldsymbol{(ab)}$ ist und $\boldsymbol{\beta}$ eine Formel des Typs a, dann ist $\boldsymbol{\alpha(\beta)}$ eine Formel des Typs b.
(Abs) Wenn $\boldsymbol{x}$ eine Variable eines Typs a ist und $\boldsymbol{\alpha}$ eine Formel des Typs b, dann ist $\boldsymbol{(\lambda x.\, \alpha)}$ eine Formel des Typs $\boldsymbol{(ab)}$.
(Id) Wenn $\boldsymbol{\alpha}$ und $\boldsymbol{\beta}$ Formeln desselben Typs a sind, dann ist $\boldsymbol{(\alpha = \beta)}$ eine Formel des Typs $\boldsymbol{t}$.

Nur die Klausel *(Id)* ist neu; alles andere ist wie gehabt in (21). Allerdings verzichten wir bei der in (131) gegebenen Version der Typenlogik auf die in (31)–(34) definierten logischen Konstanten; denn diese wollen wir ja als Abkürzungen auffassen.

Man beachte, dass die beiden Formeln in Gleichungen vom selben Typ sein müssen, und dass der resultierende Typ immer $\boldsymbol{t}$ ist; eine Gleichung zwischen Formeln verschiedener Typen ist also nicht interpretierbar, und eine Gleichung zwischen Formeln gleichen Typs steht immer für einen Wahrheitswert. Für den Fall, dass der Typ der beiden gleichgesetzten Formeln $\boldsymbol{e}$ ist, erhalten wir Gleichungen, die wir auch mit der in (34) definierten logischen Konstanten '$\boldsymbol{=}$' des Typs $\boldsymbol{e(et)}$ bilden konnten; in Analogie dazu hätten wir die Klausel *(Id)* in (131) auch streichen können und stattdessen für jeden Typ a eine Konstante '$=_a$' einführen können.

Die Deutung der Typenlogik nach (131) geschieht wie zuvor – also durch die in Abschnitt 5.3 gegebenen Klauseln (28), (29), (35) und (44) (bzw. (44') oder (44'')) – sowie durch eine weitere, offensichtliche semantische Bewertung von Gleichungen:

(132) *Deutung der Identität (Id)*
Wenn g eine Belegung ist, und $\boldsymbol{\alpha}$ und $\boldsymbol{\beta}$ Formeln eines Typs a, dann gilt:
$$[\![\boldsymbol{(\alpha = \beta)}]\!]^g = \text{⊦}[\![\boldsymbol{\alpha}]\!]^g = [\![\boldsymbol{\beta)}]\!]^g\text{⊣}\,.$$

Für den Fall, dass die beiden gleichgesetzten Formeln $\boldsymbol{\varphi}$ und $\boldsymbol{\psi}$ vom Typ $\boldsymbol{t}$ sind, gilt $[\![\boldsymbol{(\alpha = \beta)}]\!]^g = 1–([\![\boldsymbol{\alpha}]\!]^g – [\![\boldsymbol{\beta)}]\!]^g)$. In der Logik heißt diese Wahrheitswertkombination 'materiale Äquivalenz' und wird als '$\boldsymbol{[\alpha \leftrightarrow \beta]}$' notiert. Wie man leicht nachprüft, ist die materiale Äquivalenz äquivalent zur Konjunktion der materialen Implikationen in beiden Richtungen: $\boldsymbol{[\alpha \leftrightarrow \beta] \equiv [[\alpha \rightarrow \beta] \wedge [\beta \rightarrow \alpha]]}$. Auch andere Wahrheitstafeln lassen sich mit Hilfe der Identität definieren, wenn auch auf umständlichere Art und Weise. Wir zeigen dies hier nur für die Negation ($\neg$) und die Konjunktion ($\wedge$).[119] Für die Negation konstruieren wir zunächst eine Gleichung $\bot$, die stets – unter beliebigen Belegungen – den Wahrheitswert 0 erhält. $\bot$ besagt, dass die (charakteristische Funktion der) Menge aller Wahrheitswerte {0,1} gleich der (charakteristischen Funktion der) Einermenge {1} ist, was natürlich nicht stimmt. Die Menge aller Wahrheitswerte ist die Menge der mit sich selbst identischen Wahrheitswerte – $\{0,1\} = \{u \in \{0,1\} \mid u=u\}$ – und lässt sich daher durch den Lambda-Term $(\lambda u.\, (u=u))$ charakterisieren, der sich unmittelbar in die Typenlogik mit Identität (in diesem Fall zwischen Wahrheitswerten) übersetzen lässt. {1} ist die Menge der Wahrheitswerte, die mit dem Wert 1 identisch sind: $\{u \in \{0,1\} \mid u=1\}$; der Wert 1 ist wiederum der Wert jeder Gleichung der Gestalt $\boldsymbol{(\alpha=\alpha)}$. Aufgrund dieser Beobachtungen gelangt man zu der folgenden Festsetzung:

[119] Tatsächlich lassen sich in der Typenlogik mit Identität *beliebige* Wahrheitstafeln ausdrücken; das liegt an der aus der Aussagenlogik bekannten *funktionalen Vollständigkeit* von Konjunktion und Negation, d.h. dem Umstand, dass jede Kombination von Wahrheitswerten auf eine Kombination von $\wedge$ und $\neg$ zurückgeführt werden kann. Einen Beweis dafür findet man z.B. auf S. 38 des Skripts *Formale Grundlagen der Sprachphilosophie*: http://web.uni-frankfurt.de/fb10/zimmermann/FGdSp.pdf.

(133) $\bot := (\lambda u^t.\ (u{=}u)) = (\lambda u.\ (u{=}(u{=}u)))$

Die Nachprüfung, dass in der Tat (für beliebige g) $[\![\bot]\!]^g = 0$, ist der Leserschaft überlassen. Mit (133) lässt sich die Negation definieren; denn für Wahrheitswerte u hat die Aussage, dass u mit 0 identisch ist, immer den von u entgegengesetzten Wahrheitswert:

(134) $\neg := (\lambda u^t.\ (u{=}\bot))$

Die Konjunktion ist komplizierter. Eine Möglichkeit, sie mit den in (131) gegebenen Mitteln auszudrücken, bedient sich einer mengentheoretischen Version des so genannten *Leibnizprinzips*, nach dem beliebige Objekte x und y gilt, dass x mit y identisch ist, wenn x genau dieselben Eigenschaften hat wie y[120]:

(135) Wenn U eine Menge ist und $\{u, v\}\subseteq U$, dann gilt:
$u = v$ genau dann, wenn für alle Teilmengen $X\subseteq U$ gilt: $u\in X$ gdw. $v\in X$.

Das Prinzip (135), das in der Mengenlehre Gültigkeit besitzt, sieht komplizierter aus, als es ist. Interessant ist eigentlich nur die eine Richtung. Denn dass u Element derselben Mengen ist wie v, wenn u und v miteinander identisch sind, ist sowieso klar: wenn $u\in X$, folgt mit $u{=}v$ sofort $v\in X$ und umgekehrt. Wenn aber andererseits – und jetzt kommen wir zu interessanteren Richtung – u und v Elemente derselben Teilmengen von X sind, dann gilt insbesondere: $v\in\{u\}$, denn $\{u\}$ ist eine Teilmenge von U, die u als Element besitzt. Aber $\{u\}$ besitzt auch keine anderen Elemente und somit könnnen wir von $v\in\{u\}$ auf $u = v$ schließen.

Der Zusammenhang zwischen dem mengentheoretischen Leibnizprinzip (135) und der Konjunktion ergibt sich aus der Beobachtung, dass Letztere (wie wir dies bereits im ersten Kapitel getan haben) als Funktion von Paaren von Wahrheitswerten in Wahrheitswerte aufgefasst werden kann – und somit als charakteristische Funktion einer Menge von Wahrheitswerten. Da die Konjunktion für ein Paar (u,v) von Wahrheitswerten gerade dann den Wert 1 liefert, wenn $u = v = 1$, charakterisiert sie die Einermenge $\{(1,1)\}$, also die Menge aller Objekte, die mit dem Paar (1,1) identisch sind: $\{(u,v)\ |\ (u,v) = (1,1)\}$. Nach (135) charakterisiert also die Konjunktion die Menge der Paare (u,v), die Element derselben Mengen von Wahrheitswertpaaren sind wie (1,1):

(136) $\downarrow[\![\mathbf{und}]\!]^s$
$=$ $\{(1,1)\}$
$=$ $\{(u,v)\ |\ (u,v) = (1,1)\}$
$=$ $\{(u,v)\,|$ für alle Mengen X von Wahrheitswertpaaren gilt: (u,v) $\in$X gdw. $(1,1)\in X\}$

Im Rahmen der Typenhierarchie, in der Mengen durch ihre charakteristischen Funktionen und Paarmengen durch Funktionen repräsentiert werden, lässt sich (136) wie folgt reformulieren:

(137) $[\![\mathbf{und}]\!]^s$ = $\lambda v.\lambda u.$ für alle R vom Typ $\boldsymbol{t(t,t)}$ gilt: $R(v)(u) = R(1)(1)$

Die Bedingung rechts von den Lambdas in (137) lässt sich nun tatsächlich durch eine Kombination von Abstraktionen, Applikationen und Identitäten ausdrücken und daher in die Typenlogik mit Identität übertragen. Dazu müssen wir nur noch die Ausdrucksweise ‘für alle R vom Typ a gilt:…’ ausdrücken. Das ist relativ leicht; denn eine Aussage ‘…’ gilt gerade dann für alle Objekte eines Typs a, wenn die Menge der Objekte des Typs a, für die ‘…’ gilt, mit der Menge aller Objekte dieses Typs – also der Menge der selbstidentischen Objekte des Typs a – zusam-

120 Genauer handelt es sich um das oft Gottfried Wilhelm Leibniz (1646–1716) zugeschriebene Prinzip der Identität von Ununterscheidbarem (*principium identiatis indiscernibilium*), das aber bereits im Altertum bekannt war. Der genaue Zusammenhang zwischen diesem Zitat und der typenlogischen Version ist komplex und führt hier zu weit.

menfällt. Damit gelangen wir zunächst zu einer (allgemeinen) Definition des Allquantors über Objekte eines Typs a :

(138) $\forall_a := (\lambda P^{(at)t}.\ ((\lambda x^a.\ P(x)) = (\lambda x.\ (x{=}x))))$

Über die so genannte *Dualität* – die gegenseitige Definierbarkeit von All- und Existenzquantor mit Hilfe der Negation – lässt sich mit (138) eine Reduktion des Existenzquantors für beliebige Typen finden:

(139) $\exists^a := (\lambda P^{(at)t}.\ \neg\forall(\lambda x^a.\neg P(x)))$

Für den Fall, dass $a = \boldsymbol{e}$, ist die in (139) definierte Formel dem in (32) als logische Konstante gedeuteten Existenzquantor nachweislich äquivalent. Entsprechendes gilt für die Formel (138) und die in (95a) eingeführte Version des Allquantors. Für den Fall $a = \boldsymbol{t(tt)}$ lässt sich (138) schließlich für eine typenlogische Version von (137) ausnutzen:

(140) $\wedge := (\lambda v^t.(\lambda u^t.\ (\forall R^{t(tt)})\ (R(v)(u) = R(\neg\bot)(\neg\bot)))$

Damit haben wir gezeigt, dass die vier logischen Konstanten im Rahmen der Typenlogik mit Identität überflüssig sind. Diese Tatsache ist zwar aus logischer Sicht bemerkenswert, für die Praxis der indirekten Deutung aber irrelevant. Dennoch reflektieren die Zurückführungen (133), (134) und (138) – (140) eine Gemeinsamkeit zwischen den logischen Konstanten, die auch aus semantischer Sicht von Interesse ist. Die Tatsache nämlich, dass sie sich mit rein logischen Mitteln – also ohne Rückgriff auf irgendwelche Konstanten – definieren lassen, impliziert zugleich ihre universelle Einsetzbarkeit, unabhängig vom inhaltlichen Zusammenhang. Diese Themenneutralität wird in der Logik als *Invarianz* bezeichnet – ein Begriff, der uns im Zusammenhang mit den Determinatoreninhalten schon begegnet ist und der nun hier allgemeiner gefasst werden soll. Die Grundidee dabei ist, dass die semantischen Werte logischer Konstanten Funktionen sind, die sich immer nur auf strukturelle Aspekte ihrer Argumente beziehen. Ein Charakteristikum dieser strukturellen Aspekte ist, dass sie sich ohne Bezugnahme auf die einzelnen Individuen bestimmen lassen. So kommt es z.B. beim Existenzquantor $\exists_e$ nur darauf an, ob das Argument eine Menge (von Individuen) charakterisiert, die überhaupt irgendein Element enthält, nicht aber darauf, welche Elemente sie enthält. Dass es auf die genaue Identität der einzelnen Individuen ankommt, heißt, dass aus Sicht der logischen Konstanten alle Individuen dieselbe Rolle spielen und folglich miteinander austauschbar sind: vertauscht man die Individuen untereinander, so würde das nichts am Ergebnis einer logischen Operation ändern. Diese intuitive Idee lässt sich mit Hilfe des Begriffs der *Permutation* präzisieren:[121] eine Permutation ist eine Funktion, die zunächst die Individuen untereinander vertauscht und dann auf Objekte komplexer Typen 'fortgesetzt' wird:

(141) *Definition*

(a) Eine Permutation ist eine Funktion π vom Typ $\boldsymbol{ee}$, so dass für alle Individuen x und x' gilt:

- wenn $x \neq x'$, dann ist $\pi(x) \neq \pi(x')$;
- es gibt ein z, so dass $x = \pi(z)$.

(b) Wenn π eine Permutation ist, und a ein extensionaler Typ, dann ist π_a eine Funktion des Typs aa, so dass gilt:

- wenn $a = \boldsymbol{e}$, dann ist $\pi_a = \pi$;
- wenn $a = \boldsymbol{t}$, dann ist $\pi_a = \{(0,0),(1,1)\}$;
- wenn $a = \boldsymbol{(bc)}$, und f eine Funktion des Typs a ist, dann ist:

[121] Die Idee, Permutationen zur Charakterisierung von logischen Konstanten zu verwenden, wurde erstmals in dem 1935 erschienenen Aufsatz *On the Limitations of the Means of Expression of Deductive Theories* von Adolf Lindenbaum und Alfred Tarski vorgeschlagen, ist aber offenbar so natürlich, dass sie in der Folgezeit immer wieder neu entdeckt wurde.

$$\pi_a(f) = \{(\pi_b(x),\pi_b(y)) \mid f(x) = y\}\ .$$

Ein *extensionaler* Typ ist dabei ein solcher, der kein ***s*** enthält; eine Verallgemeinerung des Begriffs der Permutation für beliebige Typen ist zwar möglich, aber umständlich und im gegenwärtigen Zusammenhang verzichtbar.

Nach (141a) muss eine Permutation π nicht nur eindeutig jedem Individuum ein anderes zuweisen – das tut jede Funktion; π darf auch nicht zwei Individuen dasselbe zuordnen, und jedes Individuum muss auch irgendeinem Individuum zugeordnet werden. Stellt man sich Funktionen als Arrangements von Pfeilen vor, die jeweils vom Argument zum Wert führen, so besagen die beiden Bedingungen, dass keine zwei Pfeile zusammenführen dürfen und dass jedes Individuum von einem Pfeil getroffen werden muss.[122] Man beachte, dass diese Definition den (harmlosen) Grenzfall der identischen Abbildung von x auf x nicht ausschließt; auch sie ist eine Permutation im Sinne von (141a).

(141b) beschreibt in erster Linie, was mit einer Funktion eines extensionalen Typs passiert, wenn die Individuen untereinander vertauscht werden: Vertauschtes wird auf Vertauschtes abgebildet. Danach wird z.B. die (charakteristische Funktion der) Menge $M = \{x,y,z\}$ von Individuen auf die (charakteristische Funktion von) $M_\pi = \{\pi(x),\pi(y),\pi(z)\}$ abgebildet. Insbesondere haben M und M_π dieselbe Kardinalität. Da der Existenzquantor nur auf die Kardinalität seiner Argumente achtet, bildet er auch M und M_π auf denselben Wahrheitswert $1 = \pi(1)$ ab. Eine Verallgemeinerung dieser Überlegung zeigt, dass für beliebige Permutationen π gilt: $\pi(\exists^e) = \exists^e$. In diesem Sinn 'merkt' also der Existenzquantor nicht, wenn die Individuen vertauscht werden:

(141c) Ein Objekt X eines extensionalen Typs a ist (*Permutations-*) *invariant*, falls für alle Permutationen π gilt: $\pi_a(X) = X$.

Neben Existenz- und Allquantor erweisen sich alle in Kapitel 3 betrachteten Determinatorenextensionen als invariant, daneben auch alle Wahrheitstafeln und die Identität zwischen Individuen (als Objekt des Typs ***e(et)***). Man kann auch zeigen, dass die Extensionen von Formeln der Typenlogik (mit oder ohne Identität), die keine Konstanten enthalten und keine freien Variablen immer invariant sind. Aus der Tatsache, dass sich alle logischen Konstanten durch solche Formeln definieren lassen, folgt also insbesondere ihre Invarianz.[123]

Inhaltsbasierte Typenlogik
Die indirekte Deutung ist nicht auf die Methode von Extension und Intension beschränkt. Bei geeigneter Modifikation der Typenlogik kann man sie ebensogut für die auf Inhalten basierende Semantik entwickeln. Dazu muss man zunächst den Typenbegriff adaptieren. Wenn man – analog zu unserem Vorgehen in Abschnitt 5.1 – die Inhalte der in den vorangehenden Kapiteln analysierten Ausdrücke Revue passieren lässt, stellt sich rasch heraus, dass die meisten von ihnen, ausgehend von den Propositionen und den Individuen, durch Differenzbildung gewonnene Funktionen waren. Statt der drei Grundtypen benötigt man danach offenbar nur zwei primitive *inhaltsbasierte* Typen, den Typ ***e*** der Individuen und den Typ ***p*** der Propositionen, also der Mengen möglicher Situationen. Die komplexen inhaltsbasierten Typen werden dann wieder als

122 Die erste Bedingung heißt in der Mathematik auch *Injektivität*, die zweite *Surjektivität*. Funktionen, die beide Bedingungen erfüllen heißen *bijektiv*. Eine Permutation ist also eine Bijektion über der Menge der Individuen. Wie man sich leicht überlegt, ist die Umkehrung einer Bijektion π, also die Menge aller Paare $(\pi(x),x)$ auch wieder eine Bijektion.

123 Ein ähnlicher Zusammenhang zwischen Definierbarkeit und Permutationsinvarianz besteht in praktisch allen Logiksprachen (mit Ausnahme solcher, die sog. *Auswahloperatoren* enthalten). Die Umkehrung des Zusammenhangs ist allerdings immer falsch: für jede (abzählbare) Logiksprache gibt es invariante Objekte, die man in ihr nicht definieren kann.

Paare ***(ab)*** aus weniger komplexen Typen a und b gebildet und entsprechen Funktionen von Objekten des Typs a in solche des Typs b. Die Ausdrucksmittel der inhaltsbasierten Typenlogik sind dieselben wie die der zweisortigen Typenlogik. Die Sprache verfügt über zwei Kombinationsmöglichkeiten von Formeln, Applikation und Lambda-Abstraktion, die genau wie in der zweisortigen Logik notiert und interpretiert werden. Die genauen Definitionen können wir uns sparen, weil sie – bis auf die Bezugnahme auf die inhaltsbasierten Typen (anstelle der zweisortigen) – haargenaue Kopien der weiter oben getroffenen Festlegungen wären. Auch im Bereich der Variablen tut sich nichts Wesentliches, d.h. es gibt unendlich viele von jedem (inhaltsbasierten) Typ. Die Auswahl der Konstanten richtet sich weitgehend nach den Erfordernissen der indirekten Deutung, auf die wir gleich zu sprechen kommen. Bei den *logischen* Konstanten muss allerdings der Typ angepasst werden; denn sie stehen für Inhalte statt für Extensionen. Entsprechend ist die Konjunktion (**∧**) in der inhaltsbasierten Typenlogik vom Typ ***p(pp)***, der Existenzquantor (**∃**) hat den Typ ***((ep)p)***, die Negation (**¬**) ist vom Typ ***pp***, und die Identität (**=**) ist eine Konstante des Typs ***e(ep)***. Die Deutungen verstehen sich praktisch von selbst:

(142) $[\![\wedge]\!] = \lambda p.\, \lambda q.\, p \cap q$
(143) $[\![\exists]\!] = \lambda P.\, \{s \in LR \mid P^s \neq \dot{\mathrm{E}}\}$
(144) $[\![\neg]\!] = \lambda p.\, LR \setminus p$
(145) $[\![=]\!] = \lambda x.\, \lambda y.\, \{s \in LR \mid x = y\}$

Die Formulierung (145) ist insofern irreführend, als man üblicherweise davon ausgeht, dass es sich bei der Identität in dem Sinne um eine 'wesenshafte' Beziehung handelt, als ein Individuum nur mit *einem* Individuum – sich selbst – identisch sein kann und es mit diesem einen Individuum auch identisch sein muss. Das soll heißen, dass es z.B. keine mögliche Situation gibt, in der Fritz mit jemand anderem identisch ist als mit sich selbst, Fritz, und dass er in jeder möglichen Situation mit sich selbst, Fritz, identisch ist. Eine Konsequenz aus dieser Auffassung ist, dass Formeln der Gestalt **(α=β)** immer triviale Propositionen ausdrücken; denn entweder bezeichnen **α** und **β** dasselbe Individuum und die Formel trifft auf alle Situationen zu – oder sie bezeichnen verschiedene Individuen und sie trifft auf keine Situation zu. Dementsprechend lässt sich (145) wie folgt umformulieren:

(145') $[\![=]\!] = \P(y)(x) = \begin{cases} LR, \text{ falls } x = y \\ \dot{\mathrm{E}}, \text{ falls } x \neq y \end{cases}$:

Die Übertragung der in den beiden vorangehenden Abschnitten entwickelten Prinzipien der indirekten Deutung auf die inhaltsbasierte Typenlogik ist weder schwierig noch überraschend. Für jeden natürlichsprachlichen Ausdruck A lässt sich eine Formel $\langle A \rangle$ angeben, deren Wert der Inhalt dieses Ausdrucks ist: $[\![\langle A \rangle]\!] = \| A \|$; und die Inhaltskomposition geschieht, dem Vorgehen in den Kapiteln 1 – 4 folgend, stets per Funktionalapplikation.

Sieht man einmal von dem Unterschied zwischen Mengen und ihren charakteristischen Funktionen ab, kann man die inhaltsbasierte Typenlogik als eine *Teilsprache* der zweisortigen Typenlogik auffassen, die über weniger Typen verfügt und bei die logischen Konstanten nur in bestimmten Konstellationen vorkommen. Genauer gesagt lässt sich die inhaltsbasierte Typenlogik in dem Sinn in die zweisortige Typenlogik *einbetten*, als man für jede Formel **α** der ersteren eine (bis auf den genannten Unterschied) äquivalente Formel $\boldsymbol{\alpha}^{+}$ der letzteren angeben kann. Variablen der inhaltsbasierten Logik werden daür durch zweisortige Variablen des entsprechenden Typs übersetzt, wobei der (primitive) Typ ***p*** durch sein (komplexes) zweisortiges Pendant ***(st)*** ersetzt wird. Wenn z.B. ***x*** eine Variable des Typs ***p((ep)p)*** ist, dann ist $\boldsymbol{x}^{+}$ eine Variable des Typs ***(st)((e(st))(st))***. Entsprechend verfährt man bei der Übersetzung nichtlogischer Konstanten. Für die logischen Konstanten setzt man dagegen:

$(142^{+}) \wedge^{+} = (\lambda p^{st}.\ (\lambda q^{st}.\ (\lambda i.\ [p(i) \wedge q(i)]))$

$(143^{+}) \exists^{+} = (\lambda P^{e(st)}.\ (\lambda i.\ (\exists x^{e})\ P(i)(x)))$

$(144^{+}) \neg^{+} = (\lambda p^{st}.\ (\lambda i.\ \neg p(i)))$

$(145^{+}) =^{+} = =$

Die so übersetzten Variablen und Konstanten lassen sich genauso mit den Ausdrucksmitteln der zweisortigen Typenlogik kombinieren, wie sie in der inhaltsbasierten Logik kombiniert werden: $\alpha(\beta)^{+} = \alpha^{+}(\beta^{+})$, und $(\lambda x.\ \alpha)^{+} = (\lambda x^{+}.\ \alpha^{+})$. Wie man leicht nachprüft, ist die $^{+}$-Übersetzung in dem Sinne korrekt, als die semantischen Werte der Formeln dabei erhalten bleiben (sieht man einmal von dem Unterschied zwischen Propositionen und ihren charakteristischen Funktionen ab). Da die $^{+}$-Übersetzung zudem offenkundig 1:1 ist – verschiedene Formeln werden unterschiedlich übersetzt – lassen sich inhaltsbasierte Formeln als Abkürzungen für ihre zweisortigen Übersetzungen auffassen. In diesem Sinn handelt es sich bei der inhaltsbasierten Typenlogik um eine Art Teilsprache der zweisortigen Typenlogik. Insofern ist es auch wenig überraschend, dass die beiden Logiken denselben allgemeinen Gesetzen gehorchen. Insbesondere gelten auch für inhaltsbasierte Formeln die Lambda-Konversion und die gebundene Umbenennung. Auch die hier nicht im Detail erklärte metalogische Eigenschaft der starken Normalisierbarkeit lässt sich für die inhaltsbasierte Typenlogik nachweisen.

Die einzigen nennenswerten Unterschiede zwischen inhaltsbasierter und zweisortiger Typenlogik ergeben sich daraus, dass nicht jeder zweisortige Typ eine direkte inhaltsbasierte Entsprechung im Sinne des vorangehenden Absatzes hat. In vielen Fällen gibt es allerdings eine indirekte Entsprechung. So lässt sich zwar der Typ ***s(e(et))*** (der Intensionen transitiver Verben) nicht durch Ersetzung eines ***p*** aus einem entsprechenden inhaltsbasierten Typ gewinnen. Doch wie wir schon am Ende von Kapitel 2 gesehen haben, besteht ein enger Zusammenhang zwischen den Intensionen und den Inhalten transitiver Verben und somit zwischen dem genannten Typ und dem inhaltsbasierten Typ ***e(ep)***. Dieser Zusammenhang ist ganz allgemeiner Natur: in einem naheliegenden Sinn (der im folgenden Kapitel noch präzisiert und verallgemeinert wird) entsprechen inhaltsbasierte Typen ***(...p)*** zweisortigen Typen ***s(...t)***. Andererseits gibt es natürlich zweisortige Typen, die nicht in dieser Weise inhaltsbasierten Typen entsprechen. So hat z.B. der Intensionstyp ***s((et)((et)t))*** eines Determinators keine direkte Entsprechung im inhaltsbezogenen Bereich. Allerdings enthält der Typ ***(ep)((ep)p))*** natürliche Entsprechungen von Determinatorenintensionen – nämlich Determinatoreninhalte. Auch dieser Art von Entsprechung, bei der den Objekten eines Typs (in diesem Fall: ***s((et)((et)t)***) Objekte eines anderen Typs (***(ep)((ep)p))***) entsprechen, aber nicht umgekehrt, werden wir im nächsten Kapitel systematisch nachgehen. Dabei wird sich herausstellen, dass – von wenigen Ausnahmen abgesehen – die zweisortigen Typen, die in der indirekten Deutung eine Rolle spielen, von den inhaltsbasierten Typen abgedeckt werden. Zwei bemerkenswerte Ausnahmen bilden der Typ ***s(st)***, der in der Deutung indirekter Fragen (in Kapitel 11) eine Rolle spielen wird sowie der Typ ***((se)t)***, der als Extensionsyp von Substantiven wie **Temperatur** und Verben wie **steigen** vorgeschlagen wurde.[124]

[124] Die Idee hinter diesem, auf Richard Montagues in Fn. 117 genannten Aufsatz zurückgehenden Vorschlag ist, dass eine Temperatur eine Funktion ist, die jeder Situation einen Zahlenwert zuweist, und das Steigen der Temperatur eine Eigenschaft einer solchen Funktion ist. Die Motivation dafür bildete ein Beispiel von Barbara Partee: **The temperature is ninety but it's rising**. Montagues Analyse gestattet es, das Kopulaverb **be** als Identität (von Funktionswerten) und das Subjekt als Russellsche Kennzeichnung zu deuten.

Intensionale Typenlogik
Die neben der zweisortigen Typenlogik in der semantischen Fachliteratur am häufigsten zur indirekten Deutung herangezogene Logiksprache ist die von Richard Montague entwickelte *intensionale Typenlogik* (oft auch einfach als *Intensionale Logik* bezeichnet).[125] Ähnlich wie die inhaltsbasierte Typenlogik kommt diese Sprache ohne explzite Verweise auf Situationen aus, erhält dabei aber einen Großteil der Ausdrucksstärke der zweisortigen Typenlogik.

Ausgangspunkt der intensionalen Typenlogik ist ein gegenüber dem zweisortigen modifizierter Typenbegriff, dem die Beobachtung zugrunde liegt, dass derLogische Raum in den Extensionen und Intensionen stets als Definitionsbereich fungiert: es gibt offenbar keine natürlichsprachlichen Ausdrücke, deren Extensionen von einem Typ der Gestalt $\boldsymbol{as}$ sind; ebensowenig scheint es natürlichsprachliche Ausdrücke des Typs $\boldsymbol{s}$ selbst zu geben. In der intensionalen Typenlogik beschränkt man sich daher auf die Typen, bei denen $\boldsymbol{s}$ links von einem anderen Typ steht. Die primitiven *intensionalen Typen* sind dementsprechend nur die Typen $\boldsymbol{e}$ und $\boldsymbol{t}$; und komplexe intensionale Typen werden entweder durch Paarbildung aus intensionalen Typen erzeugt – d.h. ab ist ein intensionaler Typ, wenn a und b intensionale Typen sind – oder durch Voranstellung eines $\boldsymbol{s}$ – d.h. $\boldsymbol{s}a$ ist ein intensionaler Typ, wenn a einer ist. Die Objekte eines intensionalen Typs a werden dabei genauso definiert wie in der zweisortigen Typenlogik: die Objekte des Typs $\boldsymbol{e}$ sind die Individuen; die des Typs $\boldsymbol{t}$ sind die Wahrheitswerte; die Objekte eines intensionalen Typs ab sind die Funktionen, die allen Objekten des (intensionalen) Typs a Objekte des (intensionalen) Typs b zuweisen; und die Objekte des (intensionalen) Typs $\boldsymbol{s}a$ sind die Funktionen, die jeder möglichen Situation ein Objekt des (intensionalen) Typs a zuweisen.

Erwartungsgemäß enthält die intensionale Typenlogik unendlich viele Variablen jedes intensionalen Typs a, die dann – je nach Belegung – für entsprechende Objekte des Typs a stehen. Erwartungsgemäß enthält die intensionale Typenlogik auch hinreichend viele Konstanten jedes intensionalen Typs, aber diese Konstanten werden anders gedeutet als in der zweisortigen Typenlogik. Wie ihre natürlichsprachlichen Pendants haben nämlich die Konstanten der intensionalen Typenlogik sowohl eine Extension als auch eine Intension. Der Typ a der Konstanten – und einer Formel der intensionalen Typenlogik im Allgmeinen – ist dabei ihr *Extensionstyp*; ihre Intension ist entsprechend vom Typ $\boldsymbol{s}a$. Das hat unmittelbare Konsequenzen für die indirekte Deutung: während wir ein Substantiv wie **Tisch** durch eine zweisortige Formel der Gestalt $\mathbf{T}(\boldsymbol{i})$ übersetzt haben (wobei $\mathbf{T}$ eine Konstante des Typs $\boldsymbol{et}$ ist), ist die entsprechende Übersetzung in die intensionale Typenlogik einfach eine Konstante $\mathbf{T}$, deren Extension die Menge der Tische in einer gegebenen Situation s ist. In der intensionalen Typenlogik besitzt jede Formel zwei semantische Werte: eine Extension und eine Intension. Zur Bezeichnung dieser Werte benutzt man dabei die bereits in den ersten Kapiteln für die direkte Deutung verwendete Notation, allerdings mit der zusätzlichen Erwähnung der Belegung: die Extension einer Formel $\boldsymbol{\alpha}$ in einer Situation s und bei einer Belegung g ist $[\![\boldsymbol{\alpha}]\!]^{g,s}$, ihre Intension (bei derselben Belegung) ist $[\![\boldsymbol{\alpha}]\!]^{g}$. Das Verhältnis zwischen Extension und Intension bestimmt sich in der intensionalen Typenlogik wie schon in der direkten Deutung, d.h. es gilt für alle $s \in LR$: $[\![\boldsymbol{\alpha}]\!]^{g,s} = [\![\boldsymbol{\alpha}]\!]^{g}(s)$.

Wie die zweisortige und die inhaltsbasierte Typenlogik verfügt die auch intensionale Logik über die Ausdrucksmittel der Applikation und der $\boldsymbol{\lambda}$-Abstraktion, die (auf der Extensionsebene) genauso funktionieren wie in den anderen genannten Formelsprachen: $[\![\boldsymbol{\alpha(\beta)}]\!]^{g,s} = [\![\boldsymbol{\alpha}]\!]^{g,s}([\![\boldsymbol{\beta}]\!]^{g,s})$, und $[\![\boldsymbol{(\lambda x.\alpha)}]\!]^{g,s}(u) = [\![\boldsymbol{\alpha}]\!]^{g[\boldsymbol{x}/u],s}$. Für den in Abschnitt 5.5 als (147) in die zweisortige Typenlogik übersetzten Satz (146) lässt sich dann eine ganz analoge kompositionelle Übersetzung (148) in die intensionale Typenlogik angeben:

---

125 Die Sprache wurde ursprünglich in Montagues Aufsatz *Universal Grammar* (1970) eingeführt; spätere Darstellungen beziehen sich auf die in *The Proper Treatment of Quantifiaction in Ordinary English* (1973) verwendete Variante.

(146) **Jeder Mann trifft eine Frau.** = (80)

(147) $(\lambda Q.\ (\lambda P.\ \neg\ (\exists x)\ [Q(x) \wedge \neg\ P(x)]))\ \ (\mathbf{M}_i)$ vgl. (81)
$((\lambda x.\ (\lambda Q.\ (\lambda P.\ (\exists x)\ [Q(x) \wedge P(x)]))\ \ (\mathbf{F}_i)\ (\lambda y.\ \mathbf{T}_i(x,y))))$

(148) $(\lambda Q.\ (\lambda P.\ \neg\ (\exists x)\ [Q(x) \wedge \neg\ P(x)]))\ \ (\underline{\mathbf{M}})$
$((\lambda x.\ (\lambda Q.\ (\lambda P.\ (\exists x)\ [Q(x) \wedge P(x)]))\ \ (\underline{\mathbf{F}})\ (\lambda y.\ \underline{\mathbf{T}}(x,y))))$

Die intensionallogische Version unterscheidet sich von der ursprünglichen zweisortigen Übersetzung nur an den beiden unterstrichenen Stellen, wo sie die durch die Variable ***i*** explizit gemachte Bezugnahme auf eine Situation durch die den intensionallogischen Konstanten innewohnende Situationsabhängigkeit ersetzt. Und ähnlich wie die zweisortige Formel (147) lässt sich auch ihr intensionales Pendant (148) mit einer Reihe von **λ**-Reduktionen äquivalent umformulieren:

(147') $\neg\ (\exists x)\ [\mathbf{M}_i(x) \wedge \neg\ (\exists z)\ [\mathbf{F}_i(z) \wedge \mathbf{T}_i(x,z)]]$ = (90) [≡ (147)]

(148') $\neg\ (\exists x)\ [\mathbf{M}(x) \wedge \neg\ (\exists z)\ [\mathbf{F}(z) \wedge \mathbf{T}(x,z)]]$ ≡ (148)

Das Beispiel suggeriert, dass die intensionale Logik in der indirekten Deutung zu einfacheren (oder zumindest kürzeren) Formeln führt. Wie wir gleich sehen werden, gilt das allerdings nur für den Bereich der extensionalen Konstruktionen.

Applikation und Abstraktion der intensionalen Typenlogik reichen nicht aus, um etwa die Anbindung eines Komplementsatzes an ein Einstellungsverb darzustellen. Die Extension eines solchen Verbs hat den Typ ***((st)(et))*** und kann sich nicht mit der Übersetzung des eingebetteten Satzes verbinden, die ja den Typ ***t*** hat. Hier wird offenbar eine Formel benötigt, die statt der Extension die Intension des eingebetteten Satzes bezeichnet. In der zweisortigen Typenlogik hatten wir diese Formel durch **λ**-Abstraktion von der Situationsvariablen ***i*** gewonnen; diese Möglichkeit steht in der intensionalen Typenlogik nicht zur Verfügung, denn es gibt dort keine solche Variable. Stattdessen benutzt man an dieser Stelle den sog. *Cap-Operator* ^, der es ganz allgemein gestattet, von einem Ausdruck **α** eines (intensionalen) Typs *a* zu einem Ausdruck ($^{\wedge}\boldsymbol{\alpha}$) des Typs ***s****a* überzugehen, der die Intension von **α** bezeichnet, dessen Extension also **α**s Intension ist: $[\![{}^{\wedge}\boldsymbol{\alpha}]\!]^{g,s} = [\![\boldsymbol{\alpha}]\!]^{g}$.[126]

Vergleicht man die bisher eingeführten Ausdrucksmittel der intensionalen Typenlogik mit denen der zweiortigen, ergibt sich ein einfacher Übersetzungszusammenhang. Grob gesprochen erhält man die zweisortige Entsprechung **α*** einer intensionallogischen Formel **α**, indem man die Situationsabhängigkeit von **α**s Extension in **α*** durch die Variable '***i***' zum Ausdruck bringt:

---

126 Man beachte, dass die Extension von Ausdrücken der Form ($^{\wedge}\boldsymbol{\alpha}$) situationsunabhängig ist; denn die Intension eines Ausdrucks variiert nicht mit dem Logischen Raum. Natürlich gibt es mögliche Situationen *s*, in denen z.B. das deutsche Wort **Tiger** in dem Sinn eine andere Intension hat, als dieselbe Form *in der in s gesprochenen Variante des Deutschen* zur Bezeichnung von Eseln verwendet wird. Doch die Extension des *tatsächlichen* deutschen Worts **Tiger** enthält die Individuen, die Tiger *sind* – und nicht die, die als **Tiger** *bezeichnet werden*; denn dass man ihn **Tiger** nennt, macht aus einem Vertreter der Gruppe Equus asinus keinen Panthera tigris. Wenn wir über die Intension von **Tiger** reden, meinen wir in diesem Zusammenhang die Funktion, die jeder möglichen Situation die Extension des Wortes **Tiger** im tatsächlichen Deutschen zuweist.

(149a)

$\boldsymbol{\alpha}$ [Formel der intensionalen Typenlogik]	$\boldsymbol{\alpha*}$ [zweisortige Entsprechung von $\boldsymbol{\alpha}$]
$\mathbf{c}$ [Konstante eines intensionalen Typs a]	$\mathbf{c^+}(\boldsymbol{i})$ [wobei $\mathbf{c^+}$ eine Konstante d. Typs $\boldsymbol{(sa)}$ ist]
$\boldsymbol{x}$ [Variable eines intensionalen Typs a]	$\boldsymbol{x}$
$\boldsymbol{\alpha(\beta)}$	$\boldsymbol{\alpha*(\beta*)}$
$\boldsymbol{(\lambda x.\alpha)}$	$\boldsymbol{(\lambda x.\alpha*)}$
$(\hat{}\boldsymbol{\alpha})$	$\boldsymbol{(\lambda i.\alpha*)}$

Nach der Tabelle (149a) kann man für jede (bisher erwähnte) Formel $\boldsymbol{\alpha}$ der intensionalen Typenlogik eine zweisortige Formel $\boldsymbol{\alpha*}$ ermitteln, die in jeder Situation s dieselbe Extension hat wie $\boldsymbol{\alpha}$, wenn man $\boldsymbol{i}$ auf s bezieht:[127]

(150) $[\![\boldsymbol{\alpha}]\!]^{g,s} = [\![\boldsymbol{\alpha*}]\!]^{g*[\boldsymbol{i}/s]}$

In (150) ist g eine beliebige Belegung der intensionallogischen Variablen, während g^* eine Belegung aller zweisortigen Variablen ist, die auf den intensionallogischen Variablen mit g übereinstimmt: $g \subseteq g^*$. Da $\boldsymbol{\alpha*}$ außer $\boldsymbol{i}$ nur intensionallogische Variablen enthalten kann, kommt es ohnehin nur auf deren Belegung und den Wert von $\boldsymbol{i}$ an. Erstere wird in (150) durch g geleistet, Letztere durch die links vom Gleichheitszeichen betrachtete (beliebige) Situation s.

Neben dem *Cap-Operator* gibt es in der intensionalen Typenlogik noch ein weiteres, quasi spiegelbildliches Audrucksmittel, das vor allem benötigt wird, um die semantischen Beiträge intensionaler Argumente näher zu spezifizieren. So lässt sich die Extension des Einstellungsverbs **meint** durch eine komplexe Formel der zweisortigen Typenlogik darstellen, die beschreibt, welche Rolle die Intension des Komplementsatzes (= das Argument $\boldsymbol{p}$) spielt, indem diese in den jeweiligen doxastischen Alternativen ($\boldsymbol{j}$) 'ausgewertet' wird; die Auswertung geschieht dabei durch Anwendung der Intension auf die (jeweilige) Situation:

(151) $\boldsymbol{\lambda p_{st}.\lambda x_e.\ (\forall j)\ [DOX(x)(i)(j) \rightarrow p_j]}$ [= |**meint**|, nach (108d)]

Da die intensionale Logik nicht über Variablen zur Benennung von Situationen verfügt, benötigt man, um in ihr Analysen wie (151) zu formulieren, die Möglichkeit, die Anwendung einer Intension auf eine gegebene Situation auszudrücken. Dies geschieht mit dem sog. *Cup-Operator* $\check{}$, der es ganz allgemein gestattet, von einem Ausdruck $\boldsymbol{\alpha}$ eines (intensionalen) Typs $\boldsymbol{sa}$ zu einem Ausdruck $(\check{}\boldsymbol{\alpha})$ des Typs $\boldsymbol{a}$ überzugehen, dessen Extension in einer gegebenen Situation der Wert der durch $\boldsymbol{\alpha}$ bezeichneten Funktion für diese Situation ist: $[\![(\check{}\boldsymbol{\alpha})]\!]^{g,s} = [\![\boldsymbol{\alpha}]\!]^{g,s}(s)$. Man beachte, dass der Cup-Operator nur vor Formeln von Typen der Gestalt $\boldsymbol{sa}$ stehen kann, die selbst die Gestalt $(\hat{}\boldsymbol{\alpha})$ haben können.[128] Offenkundig lässt sich auch der Cup-Operator auf einfache Weise in die zweisortige Typenlogik übertragen:

---

127 $\boldsymbol{\alpha*}$ wird auch als *zweisortige Übersetzung* von $\boldsymbol{\alpha}$ bezeichnet. Der in (150) konstatierte Zusammenhang (sowie die in (149) eingeführter Notation) gehen auf Daniel Gallins Werk *Intensional and Higher-Order Modal Logic* (1975) zurück.

128 – aber nicht müssen: in der intensionallogischen Version (156) von (151) wird er z.B. mit einer Variablen (des Typs $\boldsymbol{st}$) kombiniert

(149b)

α [Formel der intensionalen Typenlogik]	α^* [zweisoritge Entsprechung von α]
$(^\vee\alpha)$	$\alpha^*(i)$

Da Cap und Cup notationelle Varianten von Abstraktion und Applikation sind, gibt es für sie auch ein Gesetz der λ-Konversion, das in der Praxis der indirekten Deutung auf Basis der intensionalen Typenlogik häufig verwendet wird:

(152) *Down-Up-Cancellation*
Wenn α eine (intensionallogische) Formel eines Typs *a* ist, dann gilt.
$^\vee(^\wedge\alpha) \equiv \alpha$

Angesichts der *-Entsprechugen intensionallogischer Formeln der Gestalt $^\vee(^\wedge\alpha)$ erweist sich die Down-Up-Cancellation als Variante der Eigenkonversion; wie man anhand von (149) leicht nachprüft, ist nämlich $^\vee(^\wedge\alpha)^* = (\lambda i.\ \alpha^*)(i)$. Der Umweg über die zweisortigen Entsprechungen erklärt auch, warum es kein spiegelbildliches Gesetz der Up-Down-Cancellation gibt: $^\wedge(^\vee\alpha)^* = (\lambda i.\ \alpha^*(i))$, was nicht unbedingt zu α äquivalent ist, wenn α^* die Variable i enthält.[129]

Die in (151) gegebene Analyse des Einstellungsverbs **meint** lässt sich auch mit dem Cup-Operator nicht ohne Weiteres in die intensionale Typenlogik übertragen, da sie neben der allgegenwärtigen Situationsvariablen i noch eine weitere Variable des Typs s enthält, nämlich 'j'. Allerdings handelt es sich um eine gebundene Variable, die sich prinzipiell durch gebundene Umbenennung wieder in ein (gebundenes) 'i' verwandeln lässt:[130]

(153) $\lambda p^{st}.\lambda x^e.\ (\forall j)\ [\mathbf{DOX}(x)(i)(j) \rightarrow p_j]$

λ-Konversion

$\equiv\quad [\lambda q^{st}.\lambda p^{st}.\lambda x^e.\ (\forall j)\ [q_j \rightarrow p_j]]\ (\mathbf{DOX}(x)(i))$

geb. Umbenennung

$\equiv\quad [\lambda q^{st}.\lambda p^{st}.\lambda x^e.\ (\forall i)\ [q_j \rightarrow p_j]]\ (\mathbf{DOX}(x)(i))$

Die so umformulierte Analyse von **meint** lässt sich nach (149) in der intensionalen Typenlogik ausdrücken, vorausgesetzt diese enthält den Pfeil '$\rightarrow$' und den Allquantor. Ersterer wird wieder durch Kombination von Negation und Konjunktion definiert, die im Unterschied zu den nichtlogischen Konstanten genauso gedeutet werden (und genauso aussehen) wie in der zweisortigen Logik. Entsprechendes gilt vom Allquantor des Types $(et)t$, der wie gewohnt auf Negation und Existenzquantor zurückgeführt werden kann und ebenfalls als $\forall$ notiert wird. Der in (153) verwendete Allquantor des Typs $(st)t$ wird dagegen in der intensionalen Logik als *Notwendigkeitsoperator* (engl. *Box*) bezeichnet[131] ; auch er lässt sich auf einen entsprechenden Existenzquantor zurückführen, der in diesem Zusammenhang als *Möglichkeitsoperator* (*Diamond*) bezeichnet wird. Anders als in der zweisortigen Logik werden die beiden Operatoren anders notiert als die Quantoren:

---

129 Wenn $i \notin Fr(\alpha^*)$, gilt $(\lambda i.\ \alpha^*(i)) \equiv \alpha^*$ nach dem sog. Prinzip der *η-Konversion*.

130 Das (152) zugrunde liegende allgemeine Verfahren zur Elimination gebundener Variablen vom Typ s stammt aus meinem Aufsatz *Intensional Logic and Two-Sorted Type Theory* (1989).

131 Nach einer auf Leibniz zurückgehenden Idee kann Notwendigkeit als Wahrheit in – oder besser: *von* – allen möglichen Welten aufgefasst werden. Dieser Notwendigkeitsbegriff wird in der Modallogik (aus der die Notation (153) stammt) auch als *metaphysische* oder *absolute Notwendikeit* bezeichnet.

(154a) $\Diamond$ ist eine (intensionallogische) Konstante des Typs ***(st)t***, für die gilt:
$[\![\Diamond]\!]^{g,s} = \lambda p.\ /\downarrow p \neq \varnothing/$ vg. (32) in 5.3

= diejenige Funktion des Typs ***(st)t***, die jeder Satzintension p genau dann den Wahrheitswert 1 zuordnet, wenn p nicht die leere Menge charakterisiert vg. (32') in 5.3

(b) Wenn φ eine (intensionallogische) Formel des Typs ***t*** ist, dann wird die Formel $\Diamond(\hat{}\varphi)$ als $\Diamond\varphi$ abgekürzt.

(c) $\Box$ ist (eine Abkürzug für) die Formel: vg. (95) in 5.4
$$(\lambda p^{st} . \neg \Diamond \neg (^{\vee}p))$$
des Typs ***(st)t***.

(d) Wenn φ eine Formel des Typs ***t*** ist, dann ist
$$\Box\varphi$$
(eine Abkürzug für) die Formel:
$$\Box.(\hat{}\varphi)$$

Der Vergleich mit den genannten Definitionen und Notationskonventionen für die prädikatenlogischen Quantoren zeigt, wie in der intensionalen Typenlogik die Variable ***i*** durch Cap- und Cup-Operator simuliert wird. Mit (154) ergeben sich auf dem Hintergrund von (149) die folgenden Konsequenzen für die zweisortige Rückübersetzung der *Modaloperatoren* (= Box & Diamond) als Quantoren des Typs ***(st)t***:

(155a) $(\Diamond\varphi)^* \equiv (\exists i)\ \varphi^*$
(b) $(\Box\varphi)^* \equiv (\forall i)\ \varphi^*$

Mit (156) können wir jetzt die Analyse (151) des Einstellungsverbs **meinen** in der Reformulierung (152) in der intensionale Typenlogik übertragen:

(156) $[\lambda q^{st}.\lambda p^{st}.\lambda x^{e}.\ \Box[^{\vee}q \rightarrow {}^{\vee}p]]\ (^{\vee}\mathbf{DOX}(x))$

Der Vergleich zwischen (151) und (156) zeigt deutlich, dass die Formeln der intensionalen Typenlogik nicht immer kürzer und übersichtlicher sind als ihre zweisortigen Gegenstücke. Sobald nämlich von mehr als einer Situation die Rede ist, kann sich das Weglassen der Situationsvariablen ***i*** offenbar als Komplikation erweisen. Das gilt nicht nur für die Lesbarkeit der Formeln, sondern auch für ihre formalen Eigenschaften. Die intensionale Typenlogik verstößt nämlich gegen eine Reihe grundlegender logischer Gesetze. Dazu gehört das so genannte *Spezialisierungsprinzip*, nach dem man von einer allquantifizierten Formel $(\forall x)\ \varphi$ auf beliebige *Instanzen* $\varphi[^{x}/_{\alpha}]$ schließen kann, in denen die Variable x durch einen Term α desselben Typs ersetzt wird – vorausgesetzt α enthält keine freie Variable, die dabei gebunden würde. Ein Gegenbeispiel zu diesem Prinzip ist die (gültige) intensionallogische Formel$(\forall x^{e})\ (\exists y^{e})\ ,\ \Box(x = y)$ aus der *nicht* $(\exists y^{e})\ \Box\ (\mathbf{c} = y)$ folgt, wenn $\mathbf{c}$ eine Konstante (des Typs e) ist, deren Extension situationsabhängig ist. Die zweisortige *-Entsprechung zeigt, warum dieser Schluss scheitert: der Notwendigkeitsoperator bindet die in der Konstanten $\mathbf{c}$ implizite Situationsvariable i. Auf ähnliche Weise lässt sich auch die Allgemeingültigkeit des Prinzips der λ-Konversion in der intensionalen Typenlogik widerlegen: $[\lambda x^{e}.\ (\exists y^{e})\ \Box(x = y)$ ist nicht äquivalent mit $\Box(\exists y^{e})\ (\mathbf{c} = y)$ .. Die genannten Prinzipien (Spezialisierung und λ-Konversion) gelten nur ein eingeschränkter Form – nämlich dann wenn die *-Entsprechung der einzusetzenden Formel kein freies ***i*** enthält, das der Variablenbedingung – zufällige Bindung bei Einsetzung – widerspricht. Diese eingeschränkte Form der Prinzipien lässt sich auch ohne Bezugnahme auf die zweisortigen *-Entsprechungen der intensionallogischen Formeln formulieren, worauf wir hier

verzichten.[132] Doch auch das so eingeschränkte Prinzip der **λ**-Konversion ist nicht ganz frei von bösen Überraschungen: anders als in der zweisortigen oder in der inhaltsbasierten Typenlogik führen Abfolgen von **λ**-Reduktionen von einer Ausgangsformel nicht immer zum selben Endergebnis.[133] Angesichts dieser formalen Komplikationen emfiehlt es sich, die intensionale Typenlogik zu meiden und dort, wo man ihren Formeln begnet, sie durch ihre *-Entsprechungen zu ersetzen.

Substitutionelle Quantifikation
Die in Abschnitt 5.3 entwickelte Deutung der Variablen ist nicht die einzige Möglichkeit, das Problem der Bindung zu lösen. Anstelle von Belegungen kann man auch *Substitutionen* verwenden. Dazu muss man freilich voraussetzen, dass es für jedes Objekt u, gleich welchen Typs a, einen *Standardnamen* $\mathbf{c}_u$ gibt, eine eindeutig bestimmte Konstante des Typs a. Unter dieser Voraussetzung hängt der semantische Wert $[\![\boldsymbol{\alpha}]\!]$ einer Formel $\boldsymbol{\alpha}$ nicht von einer Belegung ab. Insbesondere ist also stets $[\![\mathbf{c}_u]\!] = u$. Die Interpretation der anderen Konstanten geschieht wie zuvor; die Variablen dagegen erhalten keinen eigenen semantischen Wert (oder einen willkürlich gesetzten). Die Applikation wird ebenfalls so gedeutet wie in der Belegungssemantik, nur ohne Bezug auf die Belegung. Der **λ**-Operator unterliegt dagegen der folgenden Interpretationsklausel:

(157) *Substitutionelle Deutung der Abstraktion*
Wenn $\boldsymbol{x}$ eine Variable eines Typs a ist und $\boldsymbol{\alpha}$ eine Formel eines Typs b, dann ist $[\![\boldsymbol{(\lambda x.\alpha)}]\!]$ diejenige Funktion von Typ (ab), so dass für jedes Objekt u vom Typ a gilt:
$[\![\boldsymbol{(\lambda x^a.\ \alpha)}]\!](u) = [\![\boldsymbol{\alpha}^{[x/\mathbf{c}_u]}]\!]$

Die Notation '$\boldsymbol{\alpha}^{[x/\mathbf{c}_u]}$' steht dabei (wie zuvor) für die Ersetzung der freien Vorkommen von $\boldsymbol{x}$ durch die Konstante $\mathbf{c}_u$, den Standardnamen des Objekts u. Man beachte, dass bei dieser Ersetzung keine ungewollten Bindungen eintreten können, weil $\mathbf{c}_u$ keine Variable enthält.

Die substitutionelle Deutung der Abstraktion ist unabhängig von der hier betrachteten zweisortigen Typenlogik und lässt sich auf so gut wie alle bekannten Formen der Variablenbindung übertragen. Insbesondere lassen sich auch substitutionelle Varianten der inhaltsbezogenen wie der intensionalen Typenlogik.angeben. In der Praxis findet man substitutionelle Deutung allerdings beinahe ausschließlich im Zusammanhang mit prädikatenlogischen Sprachen.

(157) setzt voraus, dass das Ergebnis der Substitution $\boldsymbol{\alpha}^{[x/\mathbf{c}_u]}$ einen semantischen Wert besitzt.Wenn $\boldsymbol{x}$ die einzige freie Variable in $\boldsymbol{\alpha}$ ist, kann man zeigen, dass dies gewährleistet ist – auch wenn $\boldsymbol{\alpha}$ (und damit auch $\boldsymbol{\alpha}^{[x/\mathbf{c}_u]}$) noch weitere **λ**-Operatoren enthalten. Enthält $\boldsymbol{\alpha}$ dagegen weitere freie Variablen, besitzt $\boldsymbol{\alpha}^{[x/\mathbf{c}_u]}$ keinen Wert (bzw. einen zufälligen, aufgrund der willkür-

132 Einer Idee Gallins folgend (vgl. Fußnote 127) kann man den Begriff der *modal geschlossenen* intensionallogischen Formel definieren, das sind solche Formeln, in deren *-Entsprechung $\boldsymbol{i}$ nicht frei vorkommt. Modal geschlossen in diesem Sinn sind (i) alle Variablen, (ii) alle Formelnder Gestalt $\boldsymbol{(\hat{}\alpha)}$, sowie (iii) alle Applikationen $\boldsymbol{\alpha(\beta)}$ und Abstraktionen $\boldsymbol{[\lambda x.\ \alpha]}$, bei denen $\boldsymbol{\alpha}$ und $\boldsymbol{\beta}$ selbst modal geschlossen sind. Das Prinzip der **λ**-Konversion wird dann so eingeschränkt, dass – abgesehen von der üblichen Variablenbedingung – $\boldsymbol{[\lambda x.\ \alpha](\beta)}$ und $\boldsymbol{\alpha^{[\beta/x]}}$ nur dann äquivalent sind, wenn entweder $\boldsymbol{\beta}$ modal geschlossen ist oder (das freie) $\boldsymbol{x}$ in $\boldsymbol{\alpha}$ nie im Skopus des Cap-Operators steht.

133 Diese Tatsache ist seit dem Aufsatz 'λ-Normal Forms in an Intensional Logic for English' (1980) von Joyce Friedman und David Warren bekannt, die die Formel $\boldsymbol{(\lambda x\ P((\lambda y\ \hat{}y)(x)))(c)}$ angeben (wobei $\boldsymbol{P}$ und $\boldsymbol{c}$ Konstanten der Typen $\boldsymbol{e(se)(et)}$ und $\boldsymbol{e}$ sind): sie ist per (eingeschränkter) **λ**-Konversion sowohl auf $\boldsymbol{P((\lambda y\ \hat{}y)(c))}$ als auch auf $\boldsymbol{(\lambda x\ P(\hat{}y))(c)}$ reduzierbar.

lich gesetzten Variablenwerte). Insofern ist die substitutionelle Deutung nur eine Deutung der gschlossenen Formeln – was kein Verlust ist: freie Variablen und offene Formeln besitzen keinen wirklichen Inhalt; auch in der Belegunssemantik haben sie ja nur provisorische, belegungsabhängige Werte. Im Bereich der geschlossenen Formeln stimmen allerdings Belegungs- und Substitutionssemantik miteinander überein.

Nach (157) wird der semantische Wert einer Formel der Gestalt **(λx.α)** nicht aus den Werten ihrer Teile ermittelt. Dies ist nicht einmal dann der Fall, wenn es sich um eine geschlossene Formel handelt. Der Wert hängt vielmehr von allen Sustitutionsinstanzen $\alpha_{[x/c_u]}$ ab, von denen keine ein Teil der Formel ist.[134] Im Gegensatz dazu ist die Belegungssemantik, wie wir in Abschnitt 5.3 gesehen haben, zumindest auf der Ebene der Bedeutungen (= Funktionen von Belegungen in semantische Werte) kompositionell. Ihe Mangel an Kompositionalität mag für den geringen Verbreitungsgrad der ansonsten konzeptuell und technisch einfacheren Substitutionssemantik mit verantwortlich sein. Ein anderer Grund mag in der künstlich wirkenden Annahme von Standardnamen zu sehen sein, auf die die substiutionelle Deutung der Variablenbindung angewiesen ist.[135]

Bemerkungen zu weiteren Logiksprachen
Neben den hier skizzierten Varianten der zweisortigen Typenlogik sind in der sematnischen Literatur eine Reihe weiterer Logiksprachen für die indirekte Deutung natürlicher Sprachen vorgeschlagen bzw. entwickelt worden, deren Darstellung hier zu weit führen würde. Wir beschränken uns auf ein paar kurze Bemerkungen zu einigen dieser Systeme:

- *Partielle Typenlogik*[136]
  Die Objekte komplexer Typen *ab* sind stets die Funktionen, deren Definitionsbereich die Objekte des Typs *a* sind und deren Werte Objekte des Typs *b* sind. Lässt man stattdessen auch partielle Funktionen zu, die nicht (unbedingt) jedem Objekt des Typs *a* einen Wert zuweisen. Ein klassisches Anwendungsgebiet sind *semantische Präsuppositionen* (vgl. Kap. 9), die man für feinkörnigere Deutungen *(a)* des definiten Artikels oder auch *(b)* so genannter *faktiven* Verben wie **wissen** verwenden kann: *(a)* Sätze mit Kennzeichnungen, in denen Existenz- oder Einzigkeitsbedingung nicht erfüllt sind, werden danach als wahrheitswertlos klassifizieren statt (wie nach der Russellschen Analyse) als falsch; *(b)* ebenso erhalten Wissenszuschreibungen, in denen der Komplementsatz falsch ist, keinen Wahrheitswert. Daneben eröffnet sich durch die Verwendung partieller Funktionen die Möglichkeit, Objekte komplexer Typen zugleich auch einfachere Typen zuzuweisen (was sonst aus mengentheoretischen Gründen nicht möglich ist): im Fall von Gerundien und Nominalisierungen könnten so z.B. Prädikatsintensionen – also Objekte des Typs ***(se)t*** – zugleich als Elemente von Substantivextensionen – also Objekte des Typs ***e*** – fungieren.
- *Relationale Typenlogik*[137]
  Eine andere, in gewisser Hinsicht natürlichere Modellierung von Partialität ergibt sich,

134 … abgesehen von dem neurotischen Grenzfall, in dem $Fr(\alpha) = \emptyset$ und somit nicht einmal x frei ist in α.

135 Historisch war wohl der Kompositionalitätsaspekt ausschlaggebend für die Entwicklung der Belegungssemantik als Alternative zu der bereits bekannten Belegungssemantik. Zumindest hat Tarski (vgl. Fn. 94) keinerlei Skrupel im Umgang mit Sprachen beliebiger Kardinalität gehabt.

136 Der Klassiker ist Maxwell J. Cresswells Buch *Logics and Languages* (1973), in dem eine partielle Variante der inhaltsbezogenen Typenlogik verwendet wird.

137 Einschlägige relationale Typenlogiken wurden von Steven Orey (*Model Theory for Higher Order Predicate Calculus*, 1959) und Daniel Gallin (☛ Fn. 127) entwickelt und von dem niederländischen Semantiker Reinhard Muskens in seiner Amsterdamer Dissertation *Meaning and Partiality* (1989) für die indirekte Deutung natürlicher Sprachen verwendet.

wenn man in komplexen Typen beliebige Relationen zwischen den Objekten niedrigerer Typen zusammenfasst. Ausgehend von dem einzigen primitiven *relationalen Typ* **e** definiert man dazu komplexe relationale Typen als (beliebig lange) Listen $(a_1,\ldots,a_n)$ von relationalen Typen. Objekte eines Typs $(a_1,\ldots,a_n)$ sind dann die Relationen zwischen Objekten der Typen $a_1,\ldots,a_n$, d.h. die Funktionen, die jeder möglichen Situation eine Menge von Listen $(x_1,\ldots,x_n)$ zuordnen, wobei x_1 ein Objekt des Typs a_1 ist,... und x_n ein Objekt des Typs a_n. Die inhaltsbasierte Variante der im vorangehenden Absatz gegebenen Deutung des definiten Artikels stellt sich dann z.B. als dasjenige Objekt des Typs **((*e*),(*e*))** heraus, das jeder Situation *s* die Listen (P,Q) von Objekten des Typs **(*e*)** zuweist, so dass $P(s)$ eine Einermenge und zugleich Teilmenge von Q ist. Eine interessante Konsequenz aus der relationalen Vorgehensweise ist es, dass nach ihr die grundlegenden Operationen zur Kombination von Bedeutungen andere sind als in der hier verfolgten funktionalen Analyse. Insbesondere den im nächsten Kapitel betrachteten *Modifikationen* eine Schlüsselrolle zu.

- *Variablenfreie Logik*[138]
  Die Komplikationen mit der Interpretation der Variablenbindung lassen sich umgehen, wenn man die Typenlogik um gewisse logische Konstanten – so genannte *Kombinatoren* – erweitert, die es gestatten, jede (geschlossene) Formel ohne einen einzigen **λ**-Operator äquivalent umzuformulieren. Anekdotisch haben wir solche Umformulierungen schon kennengelernt: die typenlogische Formel **(∀*x*) [P(*x*) → Q(*x*)]** besagt soviel wie eine variablenfreie mengentheoretische Aussage der Form '$P \subseteq Q$'. Eine systematische Elimination aller gebundenen Variablen ist eine diffizile Angelegenheit und erhöht im Allgemeinen nicht gerade die Lesbarkeit der Formeln. Andererseits könnten Einschränkungen in der Verwendung von Kombinatoren eine natürliche Erklärung für die Beschränktheit der Ausdrucksmittel in der natürlichen Sprache erlauben.
- *Dynamische Semantik*[139]
  Unter diesem Stichwort werden einige recht verschiedene Systeme zusammengefasst, denen gemeinsam ist, dass sie sich besonders gut zur systematischen Darstellung bestimmter Pronominalbezüge – so genannter *Esels-* und *Diskursanaphern* – eignen. Standardbeispiele sind: **Jeder Bauer, der einen Esel besitzt, schlägt ihn** und **Ein Bauer besitzt einen Esel. Er schlägt ihn**, wobei sich das doppelt unterstrichene Pronomen jeweils (intuitiv gesprochen) auf das einfach unterstrichene Indefinitum zurück bezieht. Wir werden auf die genannten Phänomene und einige für sie entwickelte Analyseverfahren in Kapitel 10 zu sprechen kommen.

---

138 Die Geschichte der variablenfreien Logik beginnt mit Moses Schönfinkels (vgl. Fn. 86) Aufsatz *Über die Bausteine der mathematischen Logik* (1924). Variablenfreie Logiksprachen wurden in der modernen Semantik vor allem von Pauline Jacobson (u. a. in ihrem Aufsatz *Towards a Variable-Free Semantics*, 1999) zur indirekten Deutung herangezogen.

139 Historisch-bibliografische Hinweise zu diesem Bereich werden in Kapitel 10 gegeben.

Übungsaufgaben zu Kapitel 5

A1 a) Was ist der semantische Typ der in (8) und (9) gegebenen Bedeutungen der koordinierenden Konjunktionen?

b) Formulieren Sie die Kompositionsregel (4) so um, dass sie mit den Analysen (8) und (9) verträglich ist.

A2 a) Geben Sie für die Formeln (19) und (20) Strukturbäume im Stil von (18') an.

b) Eliminieren Sie die notationellen Abkürzungen in (25) und geben Sie für das Resultat einen Strukturbaum an.

A3 Zeigen Sie, dass

(24) $\mathbf{(\lambda Q^{et}.\ (\lambda P^{et}.\ (\exists x^{e})\ [Q(x) \wedge P(x)]))}$

eine Formel des Typs ***(et)((et)t)*** ist.

A4 Stellen Sie $[\![\mathbf{(\lambda x.\ S(i)(x)(y))}]\!]^{g_2}$ im Stil von (41_2) durch 'Ausdünnung' der Tabelle(41) dar.

A5 Reformulieren Sie die Deutung (44') der Abstraktion $\mathbf{(\lambda x_a.\ \alpha)}$ als Kombination der Bedeutungen von durch Kombination von x_a und α.

Tipp: Es genügt, für beliebige Belegungen g und Objekte u des Typs a die modifizierte Belegung $g^{[x/u]}$ aus g, u und der Bedeutung von x – dem Wert in Abhängigkeit von der Belegung – zu konstruieren.

A6 a) Bestimmen Sie den semantischen Wert von

(36) $[\![\mathbf{(\lambda y^{e}.\ (\lambda x^{e}.\ S_i(x)(y)))}]\!]^{g_1}$

ausgehend von der Tabelle 42.

b) Zeigen Sie, dass der Wert von $\mathbf{(\lambda i.\ (\lambda y^{e}.\ (\lambda x^{e}.\ S_i(x)(y))))}$ nicht von der Belegung abhängt.

A7 a) Zeigen Sie, dass $\mathbf{(\lambda x_{e}.\ (x = y))}$ nicht äquivalent ist mit $\mathbf{(\lambda y^{e}.\ x = y[x/y])}$.

b) Zeigen Sie, dass $\mathbf{(\lambda x.(\exists y)\ T_i(x)(y))\ (y)}$ nicht äquivalent ist mit $\mathbf{(\lambda x.(\exists y)\ T_i(y)(y))}$.

A8 Zeigen Sie, dass die oben vor der Lambda-Reduktion von

(86) $\mathbf{(\lambda P.\ (\exists x)\ [F_i(x) \wedge P(x)])(\lambda y.T_i(x,y))}$

vorgenommene gebundene Umbenennung nötig war. Geben Sie dazu eine Belegung an, unter der (86) einen anderen Wert hat als

(86*) $\mathbf{(\exists x)\ [F_i(x) \wedge (\lambda y.T_i(x,y))(x)]}$

A9 Führen Sie die in (91) und (92) gegebene Reduktion in der umgekehrten Reihenfolge (x vor y) durch.

A10 Weisen Sie (96) anhand der Abkürzungskonventionen (95) und der in Abschnitt 5.3 gegebenen Deutung der Typenlogik nach.

A11 Zeigen Sie, dass – unter der im Text genannten Voraussetzung, dass $g(i) = s$ – die Übergänge in (100) korrekt sind. Setzen Sie dabei voraus, dass $[\![\,|N|\,]\!]^{g} = [\![N]\!]^{s}$ und $[\![\,|VP|\,]\!]^{g} = [\![VP]\!]^{s}$.

A12 Zeigen Sie, dass

$\mathbf{|Eike\ besitzt\ einen\ Porsche|}$
$\equiv \mathbf{(\exists y)\ [P_i(y) \wedge B_i(e,y)]}$

A13 Zeigen Sie, dass es sich bei dem ditransitiven Verb **schulden** um ein opakes Verb handelt und geben Sie eine lexikalische Analyse im Stil von (129) an, die auf der im Text erwähnten annähernden Synonmyie mit **geben müssen** basiert.